Schnittstellenspezifikation für die ec-Karte mit Chip

- Version 2.2 -

- **Referenzen**
- **Datenstrukturen und Kommandos**
- **Das electronic-cash System**
- **Die elektronische Geldbörse – Börsenkarte –**
- **Die elektronische Geldbörse – Händlerkarte –**
- **GeldKarte – Ladeterminals –**
- **GeldKarte – Händlersysteme –**
- **GeldKarte – Bankensonderfunktionsterminals –**
- **GeldKarte – Taschenkartenleser –**
- **Anforderungen an die elektromechanischen Eigenschaften von Chipkarten-Terminals**
- **Datei-Übertragungsprotokolle**
- **Dateinamenskonvention für den Dateitransfer mittels FTAM**
- **Funktionsbeschreibung ISO-Dateitransfer für Händler-Einreichungsdateien**
- **Funktionsbeschreibung ZMODEM-Übertragungsprotokoll Version 8-3-87**
- **Einreicherkarte**
- **Key-Management**
- **Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen**

Referenzen

[ANSI 1] American National Standard X3.92 - 1981, Data Encryption Algorithm

[ANSI 2] American National Standard X9.19 - 1986, Financial Institution Retail Message Authentication

[CCIT 1] CCITT V.III: International alphabet No 5/ISO 646: Information processing - ISO 7-bits coded characters set for information interchange

[EC 1] Technischer Anhang zum Vertrag über die Zulassung als Netzbetreiber im electronic-cash-System der deutschen Kreditwirtschaft, Version 2.0, 01.01.1994

[EC 2] Kriterien für die Bewertung und Konstruktion von electronic-cash-Systemen, Version 1.4, 09.10.1991

[EMV 1] Europay International, MasterCard International and Visa International, Integrated Circuit Card Specifications for Payment Systems, Part 1: Electromechanical Characteristics, Logical Interface, and Transmission Protocols, Release 1.1, 31.10.1994

[EN 1] prEN 1546-1, Identification card systems - Inter-sector electronic purse, Part 1: Definitions, concepts and structures, 15.03.1995

[EN 2] prEN 1546-2, Identification card systems - Inter-sector electronic purse, Part 2: Security architecture, 03.07.1995

[EN 3] prEN 1546-3, Identification card systems - Inter-sector electronic purse, Part 3: Data elements and interchanges, 09.12.1994

[EN 4] prENV 1257-2, Identification card systems - Rules for Personal Identification Number handling in intersector environments, Part 2: PIN protection, 31.01.1995

[EN 5] prENV 1855, Identification card systems - Intersector integrated circuit(s) card systems - Tolerance ranges for IC cards, März 1995

[ISO 1] ISO 7812, Identification cards - Numbering system and registration procedure for issuer identifiers, 1987

[ISO 2] ISO 7816 - 1, Identification cards - Integrated circuit(s) cards with contacts, Part 1: Physical characteristics, 1987

[ISO 3] ISO 7816 - 2, Identification cards - Integrated circuit(s) cards with contacts, Part 2: Dimensions and location of the contacts, 1988

[ISO 4] ISO 7816 - 3, Identification cards - Integrated circuit(s) cards with contacts, Part 3: Electronic signals and transmission protocols, 1989

[ISO 4'] ISO 7816 - 3, Amendment 1, Identification cards - Integrated circuit(s) cards with contacts, Part 3: Electronic signals and transmission protocols, AMENDMENT 1: Protocol type T = 1, asynchronous half duplex block transmission protocol, 1992

[ISO 4''] ISO 7816 - 3, Amendment 2, Identification cards - Integrated circuit(s) cards with contacts, Part 3: Electronic signals and transmission protocols, AMENDMENT 2: Revision of protocol type selection, 1994

[ISO 5] ISO 7816 - 4, Identification cards - Integrated circuit(s) cards with contacts, Part 4: Inter-industry commands for interchange, 1995

[ISO 6] ISO 7816 - 5, Identification cards - Integrated circuit(s) cards with contacts, Part 5: Numbering system and registration procedure for application identifiers, 1994

[ISO 7] ISO 7816 - 6, Identification cards - Integrated circuit(s) cards with contacts, Part 6: Inter-industry data elements, DIS 1995

[ISO 8] ISO 3166, Codes for the representation of names of countries, 1993

[ISO 9] ISO 4217, Codes for the representation of currencies and funds, 1990

[ISO 10] ISO 10118 - 2, Information technology - Security techniques - Hash-functions, Part 2: Hash-functions using an n-bit block cipher algorithm, 1994

[ISO 11] ISO 8583, Bank card originated messages - Interchange message specifications - Content for financial transactions, 1987

[ISO 12] ISO 9564-1, Banking - Personal identification number management and security, Part 1: PIN protection principles and techniques, 1991

[ISO 13] ISO 10202-6, Financial transaction cards - Security architecture of financial transaction systems using integrated circuit cards, Part 6: Cardholder verification, 1993

[LIT 1] Schnittstellenspezifikation für die ec-Karte mit Chip, Datenstrukturen und Kommandos, Version 2.2, 22.01.1997

[LIT 2] Schnittstellenspezifikation für die ec-Karte mit Chip, Key-Management, Version 2.2, 22.01.1997

[LIT 3] Schnittstellenspezifikation für die ec-Karte mit Chip, Das electronic cash-System, Version 2.2, 22.01.1997

[LIT 4A] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Börsenkarte, Version 2.2, 22.01.1997

[LIT 4B] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Händlerkarte, Version 2.2, 22.01.1997

[LIT 4C] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Ladezentrale, Version 2.2, 22.01.1997

[LIT 4D] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Ladeterminals, Version 2.2, 22.01.1997

[LIT 4E] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Händlersysteme, Version 2.2, 22.01.1997

[LIT 4F] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Datei-Übertragungsprotokolle, Version 2.2, 22.01.1997

[LIT 4G] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Einreicherkarte, Version 2.2, 22.01.1997

[LIT 4H] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Bankensonderfunktionsterminals, Version 2.2, 22.01.1997

[LIT 4I] Schnittstellenspezifikation für die ec-Karte mit Chip, GeldKarte, Taschenkartenleser, Version 2.2, 22.01.1997

[LIT 5] Schnittstellenspezifikation für die ec-Karte mit Chip, Personalisierung, Version 2.2, 22.01.1997

[LIT 5A] Schnittstellenspezifikation für die ec-Karte mit Chip, Nachladen von Daten und Kommandos, Version 2.2, 22.01.1997

[LIT K] Schnittstellenspezifikation für die ec-Karte mit Chip, Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen, Version 2.2, 22.01.1997

[MFC] MFC Multi Funktionale Chipkarte, Migrationskonzept für das Kreditgewerbe,

GAD/Telekom/IBM 06.04.1994

[SB] Vereinbarung über die Absicherung der ec-PIN in institutseigenen SB-Anwendungen, ZKA, Entwurf vom 04.05.1995

Datenstrukturen und Kommandos

Version 2.2 (22.01.1997).
Ausgabedatum: 17.05.1995

Inhalt

0. Darstellung und Kodierung numerischer Werte

Im folgenden Text werden numerische Werte dezimal oder hexadezimal dargestellt. Die hexadezimale Darstellung ist dadurch gekennzeichnet, daß sie in ' ' eingeschlossen wird.

Beispiel: '30' und 48 stellen denselben numerischen Wert dar.

Zur Kodierung numerischer Werte in Bytefolgen fester Länge werden die folgenden Formate verwendet:

- Binäre Kodierung:

  Hierbei ist jedes Byte der Bytefolge mit einer beliebigen Folge von 8 Bit belegt, wobei das linke Bit in einem Byte das höchstwertige Bit ist. Das linke Byte einer Bytefolge ist das höchstwertige Byte.

  Beispiel: Die Bytefolge '00 11 45 00' ist die binäre Kodierung des Wertes 1131776.

  Im Text wird die binäre Kodierung von variabel belegbaren Werten durch das Zeichen X für Halbbytes in der hexadezimalen Darstellung ausgedrückt.

  Beispiel: 'XX XX' steht für in 2 Byte binär kodierte Werte.

- BCD-Kodierung ohne Vorzeichen,

  Hierbei ist jedes Byte der Bytefolge mit 2 BCD-Ziffern belegt, wobei die linke Ziffer die höherwertige ist. Das linke Byte einer Bytefolge ist das höchstwertige Byte.

  Beispiel: Die Bytefolge '00 11 45 00' ist die BCD-Kodierung des Wertes 114500.

  Im Text wird die BCD-Kodierung von variabel belegbaren Werten durch das Zeichen n für Halbbytes in der hexadezimalen Darstellung ausgedrückt.

  Beispiel: 'nn nn' steht für in 2 Byte BCD-kodierte Werte.

- EBCDIC-Kodierung ohne Vorzeichen.

  Hierbei ist jedes Byte der Bytefolge mit einer EBCDIC-Ziffer belegt. Das linke Byte einer Bytefolge ist das höchstwertige Byte.

  Beispiel: Die Bytefolge 'F0 F1 F5 F0' ist die EBCDIC-Kodierung des Wertes 150.

1. Datenstrukturen

1.1. Datei-Organisation

Daten werden in der ec-Karte logisch in ***Dateien*** strukturiert gespeichert. Es gibt es zwei Arten von Dateien in der ec-Karte gemäß ISO 7816-4

- ***Dedicated File*** (***DF***) oder ***Verzeichnis*** und
- ***Elementary File*** (***EF***) oder ***Datenfeld***.

Im folgenden wird von Dateien gesprochen, wenn sowohl DFs als auch EFs gemeint sind.

Die Dateien der ec-Karte sind logisch in Form einer Baumstruktur angeordnet. Hierbei bildet ein DF die Wurzel dieser Baumstruktur. Es wird als ***Master File*** (***MF***) oder ***Root-Verzeichnis*** bezeichnet und muß vorhanden sein. Jedes DF kann sowohl DFs als auch EFs als Nachfolger haben. EFs können keine Nachfolger haben. Diese logische Datei-Organisation wird im folgenden auch als ***Basis-Struktur*** der ec-Karte bezeichnet.

Die Vorgänger-Nachfolger-Beziehung in der Baumstruktur wird auch als *Enthalten* bezeichnet. Vorgänger-DFs werden *übergeordnete DFs*, Nachfolger-Dateien *untergeordnete Dateien* genannt.

DFs, die dasselbe direkte Vorgänger-DF besitzen, werden als DFs einer ***Ebene*** bezeichnet. Das MF befindet sich hierbei auf Ebene 1. EFs befinden sich logisch auf derselben Ebene wie ihr direktes Vorgänger-DF.

Die ec-Karte muß mindestens drei Ebenen unterstützen.

Die Basis-Struktur darf nur durch Entfernen oder Hinzufügen von Nachfolger-Dateien modifiziert werden.

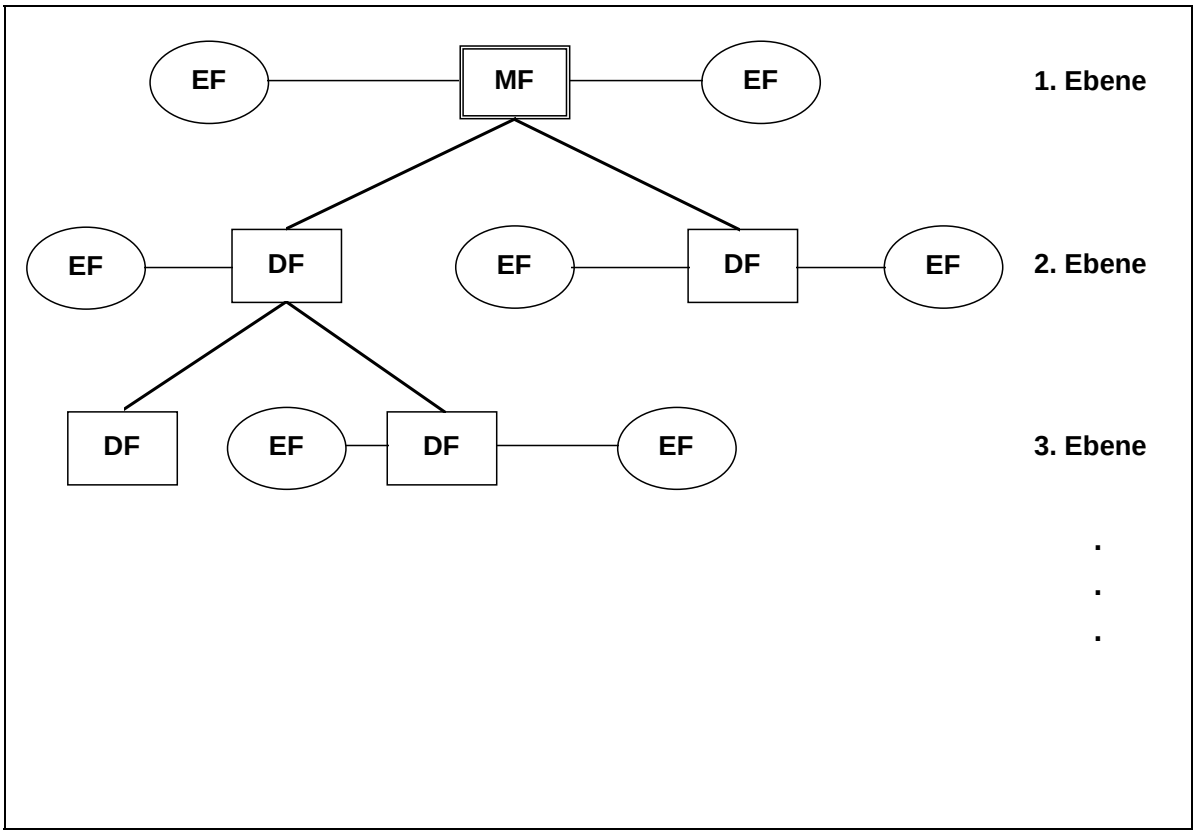


Bild 1: Logische Datei-Organisation der ec-Karte

Eine ***Applikation*** der ec-Karte ist eine logische Gruppierung von Dateien der ec-Karte bestehend aus

- genau einem DF, dem ***Application Definition File*** (***ADF***), und
- einem oder mehreren EFs, den ***Application Elementary File***s (***AEF***s).

Hierbei gelten folgende Regeln:

- Jedes DF, auch das MF, kann ADF einer Applikation sein.
- Ein DF kann das ADF für höchstens eine Applikation sein.
- EFs können AEFs in verschiedenen Applikationen sein (vgl. Kapitel 4. zu der Verträglichkeit mit den Sicherheitsattributen der EFs).

Eine Applikation kann als eine einfache Datei-Organisation betrachtet werden. Das ADF wird als Verzeichnis aufgefaßt, das die AEFs als Datenfelder enthält. Ziel dieser übergeordneten Struktur der ec-Karte ist es, einem Terminal Zugriff auf die Daten einer Applikation zu erlauben, ohne daß

es die interne Basis-Struktur der Karte kennen muß.

Die Gruppierung von Dateien in einer Applikation kann der logischen Datei-Organisation der ec-Karte entsprechen. Dies ist aber nicht notwendig der Fall. Beispielsweise kann es vorteilhaft sein, EFs mit globalen Daten in mehrere Applikationen einzubinden.

Durch ihre Position in der Basis-Struktur der ec-Karte sind die ADFs wiederum in einer oder mehreren Baumstrukturen angeordnet. Durch diese Anordnung der ADFs werden die zugehörigen Applikationen gruppiert und hierarchisiert.

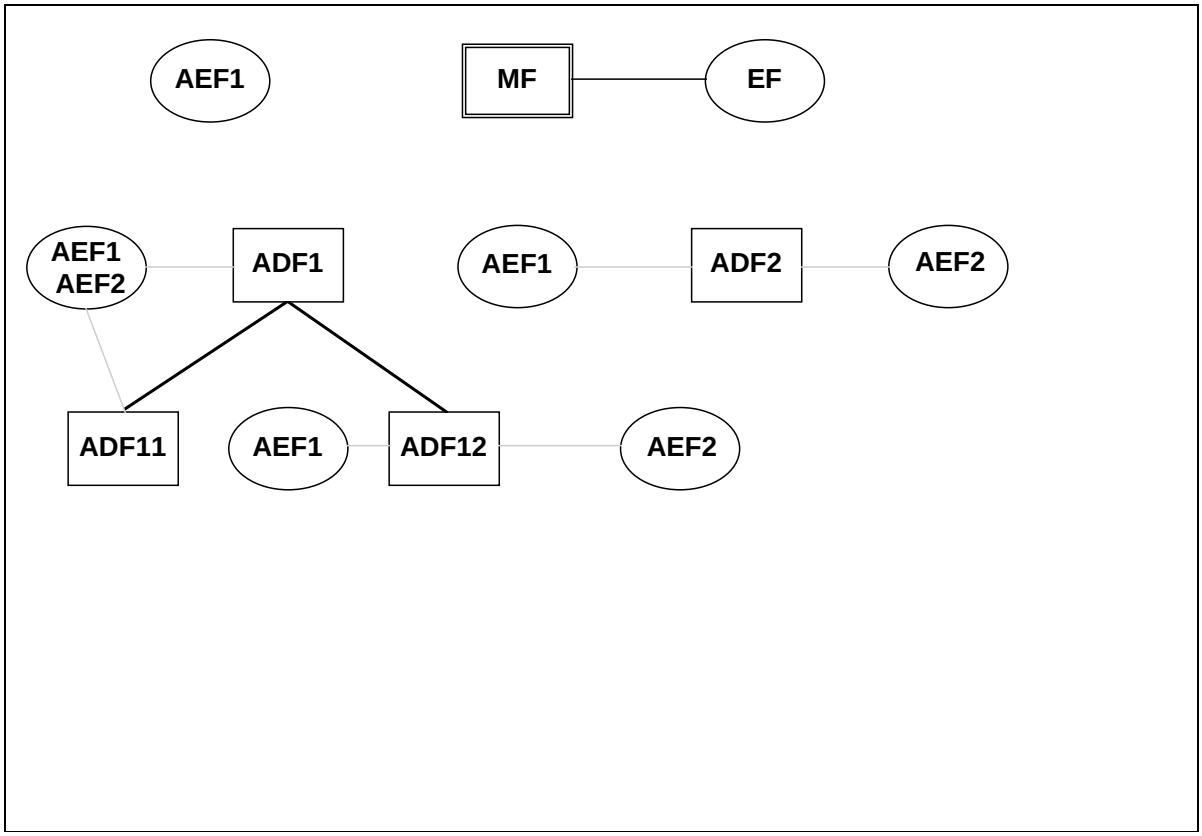


Bild 2: Applikationsbezogene Struktur der ec-Karte

1.2. Datei-Referenzierung

Durch eine implizite oder explizite ***Selektion*** wird eine Datei zur ***aktuellen bzw. selektierten Datei***. Zu jedem Zeitpunkt nach einem ATR (Answer To Reset, vgl. Kapitel 7.)

- gibt es genau ein aktuelles DF, und
- kann es innerhalb des aktuellen DF genau ein aktuelles EF geben.

Nach einem ATR ist automatisch das MF selektiert.

Eine aktuelle Datei kann bei einem Kommandoaufruf an der Schnittstelle der ec-Karte ***implizit referenziert*** werden. Zur ***expliziten Referenzierung*** von Dateien bei einem Kommandoaufruf stehen die im folgenden beschriebenen Möglichkeiten zur Verfügung.

1.2.1. Referenzierung mittels Datei-ID und Pfad

Jede Datei besitzt eine 2 Byte lange, binär kodierte ***Datei-ID***.

Folgende Regeln gelten für die Vergabe der Datei-ID:

- Das MF besitzt die Datei-ID '3F00'. Diese ID darf für keine weitere Datei in einer ec-Karte vergeben werden.
- Alle direkt in einem DF enthaltenen Dateien müssen unterschiedliche Datei-IDs besitzen.
- Die Datei-IDs '3FFF' und 'FFFF' sind reserviert.

Weitere Datei-IDs sind für spezielle EFs der ec-Karte reserviert. Die folgende Tabelle gibt eine Übersicht über die reservierten Datei-IDs. Die speziellen EFs sind in Kapitel 6. erläutert.

Datei-ID	Dateibezeichnung
’3F00’	MF
’0010’	EF_KEY
’0011’	EF_AUT
’0012’	EF_PWD0
’0022’	EF_PWD1
’0013’	EF_KEYD
’0014’	EF_AUTD
’0015’	EF_PWDD0
’0025’	EF_PWDD1
’0016’	EF_FBZ
’0017’	EF_VERSION
’0003’	EF_ID
’0004’	EF_LOG
’0005’	EF_RAND
’2FXX’	RFU
’3FXX’	RFU
’FFFF’	RFU

Durch diese Vergabe von Datei-IDs wird jede Datei in der ec-Karte eindeutig durch einen ***Pfad*** identifiziert. Ein Pfad ist gemäß ISO 7816-4, Kapitel 3.19 und 5.1.2 definiert.

Die Datei-Referenzierung durch Datei-ID oder Pfad wird für den Standard-Betrieb an der Schnittstelle der ec-Karte i. a. nicht benötigt. Sie ist für die Karten-Administration vorgesehen und sollte auch nur in diesem Rahmen verwendet werden.

1.2.2. Referenzierung mittels DF-Namens

Jedes DF der ec-Karte besitzt einen innerhalb der ec-Karte eindeutigen ***DF-Namen***, der eine Länge zwischen 1 und 16 Byte hat und binär kodiert ist.

Der DF-Name eines ADF kann eine ***Application ID*** (***AID***) sein. Die AIDs der Applikationen der ec-Karte sollen national oder international registriert sein, so daß sie gemäß ISO 7816-5 eine Länge zwischen 5 und 16 Byte haben müssen.

Die AID einer Applikation der ec-Karte besteht somit aus einer 5 Byte langen ***Registered Application Provider ID*** (***RID***) und optional aus einer ***Proprietary Application Identifier Extension*** (***PIX***), die maximal 11 Byte lang ist. Die RID der Kategorie "Nationale Registration" wird

in Deutschland von GMD auf Antrag festgelegt. Verschiedene AIDs mit gleicher RID müssen unterschiedliche PIX besitzen.

Die Selektion mittels des Kommandos SELECT FILE und des DF-Namens, insbesondere eines AID, ermöglicht es der externen Welt, auf Applikationen ohne Kenntnis der Datei-Struktur der ec-Karte zuzugreifen.

DFs der ec-Karte, die keine ADFs sind, sollen höchstens im Rahmen der Karten-Administration an der Schnittstelle der ec-Karte durch ihren DF-Namen referenziert werden.

1.2.3. Referenzierung mittels SFI

Jedes AEF besitzt innerhalb einer Applikation eine eindeutige ***Short EF-ID*** (***SFI***), durch die es innerhalb der Applikation referenziert wird. Die SFIs einer Applikation können die Werte zwischen 1 und 30 annehmen. Eine SFI wird binär in 5 Bit kodiert.

Ein EF, das AEF verschiedener Applikationen ist, kann in den unterschiedlichen Applikationen unterschiedliche SFIs besitzen.

Durch die Referenzierung von EFs mittels SFI, können innerhalb einer Applikation Kommandos auf die AEFs der Applikation ausgeführt werden, ohne sie explizit zu selektieren, also ohne ihre Datei-ID oder ihre Position in der Karte kennen zu müssen.

1.3. Applikationskontext

Eine Applikation ist eine Sicht der externen Welt auf die Dateien der ec-Karte, die unabhängig von deren Anordnung im Verzeichnisbaum der Karte ist. Ist eine solche Sicht geöffnet, so befindet sich die Karte in dem entsprechenden ***Applikationskontext.*** Zur internen Ablaufkontrolle muß die ec-Karte nachhalten, ob und in welchem Applikationskontext sie sich befindet.

Der Applikationskontext einer Applikation wird ***geöffnet oder selektiert***, indem das ADF der Applikation selektiert wird.

Der Applikationskontext wird ***verlassen oder geschlossen***, wenn ein von dem ADF der Applikation verschiedenes DF selektiert wird.

Zwischen Selektion und Verlassen des Applikationskontextes der Applikation befindet sich die ec-Karte im Applikationskontext dieser Anwendung. Wenn ein DF selektiert ist, das kein ADF ist, befindet sie sich in keinem Applikationskontext.

Nur solange sich die ec-Karte in dem Applikationskontext der Applikation befindet, sind die innerhalb der Applikation eindeutig definierten SFIs der AEFs dieser Applikation ***gültig***. Diese AEFs können nur dann in Kommandonachrichten durch Angabe ihrer SFI referenziert werden.

Befindet sich die ec-Karte in keinem Applikationskontext, oder ist der angegebene SFI in dem

Applikationskontext nicht definiert, oder ist das EF, auf das sich ein SFI bezieht, gelöscht, müssen Kommandos, die ein EF mittels SFI referenzieren, von der Karte zurückgewiesen werden.

1.4. EF-Strukturen

An der Schnittstelle der ec-Karte werden die folgenden in ISO 7816-4, Kapitel 5.1.3 definierten EF-Strukturen unterstützt:

- ***Formatiertes EF*** mit Record-Struktur, wobei unterschieden werden
  - ***Lineares EF*** mit Records konstanter Länge,
  - ***Zyklisches EF*** mit Records konstanter Länge.

Die maximale Recordlänge eines formatierten EF ist 255 Byte, da in der Kommandonachricht von UPDATE RECORD maximal 255 Byte übergeben werden können (1 Byte L_c). Soll für ein formatiertes EF eine AC vom Typ PRO oder ENC gesetzt werden (vgl. Kapitel 4.2.5.), darf die Recordlänge höchstens 247 Byte betragen, da in der Kommandonachricht der 8 Byte lange MAC zu übertragen ist.

1.5. Daten-Referenzierung

Alle EFs der ec-Karte sind formatiert. Gemäß ISO 7816-4 werden Daten in einem formatierten EF als eine Folge von ***Records*** gespeichert. Es ist nicht vorgesehen, daß auf die Daten der EFs in kleineren Einheiten als Records zugegriffen wird, so daß an der Schnittstelle der ec-Karte nur Referenzierung von Records verwendet wird.

Die Records eines formatierten EF sind durch die ***Recordnummer*** eindeutig indiziert. Die Recordnummer wird binär in 8 Bit kodiert und kann Werte zwischen '01' und 'FE' annehmen. Die Werte '00' und 'FF' sind reserviert. Ein formatiertes EF der ec-Karte kann daher maximal 254 Records enthalten.

Die Recordnummern werden in einem formatierten EF sequentiell wie folgt vergeben:

- In einem linearen EF entspricht die Folge der Recordnummern der Reihenfolge, in der die Records bei der Initialisierung oder Personalisierung bzw. mit dem Kommando APPEND RECORD angelegt werden. Der zuerst angelegte (FIRST) Record ist der Record mit der Nummer '01'. Der zuletzt angelegte (LAST) Record hat die höchste verfügbare Recordnummer.

- In einem zyklischen EF werden die Recordnummern in der umgekehrten Reihenfolge vergeben. Der zuletzt bei der Initialisierung oder Personalisierung bzw. mit dem Kommando APPEND RECORD oder durch ein internes Append der Karte angelegte Record hat die Nummer '01' und ist der FIRST Record. Für die Definition des LAST Record mit der höchsten verfügbaren Recordnummer sind zwei Fälle zu unterscheiden. Wenn das

zyklische EF Speicherplatz für n Records bietet, ist der LAST Record

- der erste angelegte Record, solange höchstens n Records angelegt wurden,
- der zyklische Nachfolger des FIRST Records, wenn mehr als n Records angelegt wurden.

Im folgenden werden die Fälle für ein zyklisches EF mit n = 10 Records illustriert:

6	5	4	3	2	1				

Bisher wurden 6 Records angelegt. Der LAST Record ist der erste angelegte Record. Er hat die Recordnummer 6.

10	9	8	7	6	5	4	3	2	1

Bisher wurden 10 Records angelegt. Der LAST Record ist der erste angelegte Record. Er hat die Recordnummer 10.

2	1	10	9	8	7	6	5	4	3

Bisher wurden 12 Records angelegt. Der LAST Record ist der zyklische Nachfolger des FIRST Records. Er hat die Recordnummer 10.

An der Schnittstelle der ec-Karte erfolgt der Zugriff auf einen Record innerhalb eines formatierten EF immer mittels Referenzierung durch die Recordnummer. Die Angabe einer Recordnummer, die größer als die des LAST Records ist, führt hierbei zu einem Abbruch. Wurde insbesondere in einem EF noch kein Record bei der Initialisierung oder Personalisierung bzw. mittels APPEND RECORD oder eines internen Appends der ec-Karte angelegt, ist für dieses EF kein Zugriff auf Records mittels Referenzierung durch Recordnummer möglich.

2. Sicherheitsalgorithmen

Zur Bereitstellung von Sicherheitsdiensten, die an der Schnittstelle der ec-Karte benötigt werden, müssen die Kommandos auf kryptographische Algorithmen in der ec-Karte zurückgreifen, die im

folgenden beschrieben werden.

2.1. DES

Als Basisverschlüsselungsalgorithmus wird der Data Encryption Standard (DES) verwendet, wie er in ANSI X3.92-1981 spezifiziert ist.

Durch den DES werden Klartextblöcke der Länge 8 Byte unter Verwendung eines 56 Bit langen Schlüssels zu 8 Byte langen Chiffretextblöcken verschlüsselt. Derselbe Schlüssel wird sowohl zum Verschlüsseln als auch zum Entschlüsseln verwendet.

DES-Schlüssel werden gewöhnlich durch 8 Byte dargestellt. Die 7 höherwertigen Bit pro Byte enthalten die eigentliche Schlüsselinformation. Das niedrigwertige Bit ist ein Paritätsbit und wird auf ungerade Parität gesetzt, d.h. jedes Byte enthält eine ungerade Anzahl von Einsen.

Für die DES-Verschlüsselung eines 8 Byte Klartextblockes x und die DES-Entschlüsselung eines 8 Byte Chiffretextblocks y unter dem 8 Byte Schlüssel K wird die folgende Notation verwendet:

DES-Verschlüsselung: $eK(x)$

DES-Entschlüsselung: $dK(y)$

Für jeden 8 Byte Block x gilt

$$eK(dK(x)) = dK(eK(x)) = x$$

2.2. Triple-DES

Für besonders sicherheitsrelevante Objekte wird der Triple-DES verwendet. Der Triple-DES verschlüsselt wie der einfache DES 8 Byte lange Klartextblöcke zu 8 Byte langen Chiffretextblöcken. Er verwendet dabei aber einen doppelt langen, d. h. 16 Byte langen Schlüssel.

Sei KK ein 16 Byte langer Schlüssel. Für die Triple-DES-Verschlüsselung eines 8 Byte Klartextblockes x und die Triple-DES-Entschlüsselung eines 8 Byte Chiffretextblockes y mit KK wird die folgende Notation verwendet:

Triple-DES-Verschlüsselung: $e*KK(x)$

Triple-DES-Entschlüsselung: $d*KK(y)$

Der Triple-DES setzt sich aus einer dreifachen Anwendung der Basisroutine DES zusammen. Sei KK_l die linke und KK_r die rechte Hälfte des doppelt langen Schlüssels KK. Das Verfahren ist wie folgt definiert:

$$e*KK(x) = eKK_l(dKK_r(eKK_l(x)))$$

$$d^*KK(y) = dKK_l(eKK_r(dKK_l(y)))$$

Für jeden 8 Byte Block x gilt daher

$$e^*KK(d^*KK(x)) = d^*KK(e^*KK(x)) = x$$

2.3. Verschlüsselung

Je nach Länge des verwendeten Schlüssels werden zwei Verfahren zur Sicherstellung der Vertraulichkeit von Nachrichten realisiert. Zum einen ist das die ***DES CBC-Mode*** Ver- und Entschlüsselung und zum anderen die ***Triple-DES CBC-Mode*** Ver- und Entschlüsselung.

2.3.1. DES CBC-Mode

Eingabe-Daten für den DES im CBC-Mode sind

- ein 8 Byte langer Schlüssel K,
- ein 8 Byte langer ***ICV*** (***Initial Chaining Value***) und
- eine Nachricht x.

Sei $x_1|...|x_n$ die Nachricht x aufgeteilt in 8 Byte lange Blöcke x_i. Eine Nachrichtenlänge, die ein Vielfaches von 8 Byte ist, erhält man ggf. durch Rechtsauffüllen mit '00'-Bytes (Padding).

Die Verschlüsselung mit DES im CBC-Mode ist rekursiv wie folgt definiert:

$$y_0 = ICV$$

$$y_i = eK(y_{i-1} \text{ XOR } x_i) \text{ für } i = 1,...,n$$

$y_1|...|y_n$ sind dann die zu der Nachricht gehörigen Chiffretextblöcke.

Bei der Entschlüsselung geht man in der umgekehrten Reihenfolge vor:

$$y_0 = ICV$$

$$x_i = dK(y_i) \text{ XOR } y_{i-1} \text{ für } i = 1,...,n$$

Für die DES CBC-Mode Ver- und Entschlüsselung von Nachrichten wird die folgende Notation verwendet:

DES CBC-Mode Verschlüsselung: eK(ICV,x)

DES CBC-Mode Entschlüsselung: dK(ICV,y)

Offensichtlich gilt

eK(ICV,dK(ICV,x)) = dK(ICV,eK(ICV,x)) = x

2.3.2. Triple-DES CBC-Mode

Eingabe-Daten für den Triple-DES im CBC-Mode sind

- ein 16 Byte langer Schlüssel KK,
- ein 8 Byte langer ICV und
- eine Nachricht x.

Bei der Triple-DES CBC-Mode Ver- und Entschlüsselung verfährt man analog zu dem Verfahren für den DES im CBC-Mode. Anstelle einer DES Ver- bzw. Entschlüsselung mit einfach langem Schlüssel K tritt eine Triple-DES Ver- bzw. Entschlüsselung mit doppelt langem Schlüssel KK.

Für die Triple-DES CBC-Mode Ver- und Entschlüsselung von Nachrichten werden folgende Notationen verwendet:

Triple-DES CBC-Mode Verschlüsselung: e*KK(ICV,x)

Triple-DES CBC-Mode Entschlüsselung: d*KK(ICV,y)

Die obigen, für den einfachen DES im CBC-Mode geltenden Relationen übertragen sich sinngemäß.

2.4. MAC-Bildung

Ein ***Message Authentication Code*** (***MAC***) wird zu einer Nachricht unter Verwendung eines kryptographischen Schlüssels gebildet und mit der Nachricht versandt, um für den Empfänger die Integrität der Nachricht überprüfbar zu machen.

Je nach Länge des verwendeten Schlüssels werden in der ec-Karte zwei verschiedene, auf dem DES beruhende Algorithmen zur MAC-Bildung realisiert (vgl. [ANSI 2]), die beide einen 8 Byte langen MAC erzeugen.

Das erste Verfahren verwendet nur einfach lange, d. h. 8 Byte lange Schlüssel und generiert den einen ***einfachen MAC***.

Der zweite Algorithmus verwendet doppelt lange, d. h. 16 Byte lange Schlüssel und generiert einen

Retail MAC.

Beide Verfahren können sowohl im ***CBC-Mode*** (***CBC-MAC***) als auch im ***CFB-Mode*** (***CFB-MAC***) betrieben werden.

2.4.1. Einfacher MAC

2.4.1.1. Einfacher CBC-MAC

Eingabe-Daten für den einfachen CBC-MAC sind

- ein 8 Byte langer Schlüssel K,
- der feste 8 Byte lange ICV = '00..00' und
- eine Nachricht x.

Die Berechnung des einfachen CBC-MAC zu einer Nachricht verläuft in ähnlicher Weise wie die DES CBC-Mode Verschlüsselung aus Kapitel 2.3.1. Die einzelnen Chiffretextblöcke stellen hierbei mit Ausnahme des letzten Blockes nur Zwischenergebnisse dar. Dieser letzte Ausgabeblock bildet den einfachen CBC-MAC:

Sei $x_1|...|x_n$ die Nachricht x aufgeteilt in 8 Byte Blöcke $x_{i.}$ Eine Nachrichtenlänge, die ein Vielfaches von 8 Byte ist, erhält man ggf. durch Rechtsauffüllen mit '00'-Bytes (Padding).

Der einfache CBC-MAC ist rekursiv wie folgt definiert:

$$y_0 = ICV = '00..00'$$

$$y_i = eK(y_{i-1} \text{ XOR } x_i) \text{ für } i = 1,...,n$$

y_n ist der zu der Nachricht gehörige einfache CBC-MAC.

Für die Bildung des einfachen CBC-MAC über eine Nachricht x mit Schlüssel K und Initial Chaining Value ICV = '00..00' wird die folgende Notation verwendet

CBC-MAC Generierung: mK(x)

2.4.1.2. Einfacher CFB-MAC

Eingabe-Daten für den einfachen CFB-MAC sind

- ein 8 Byte langer Schlüssel K,

- ein 8 Byte langer ICV und
- eine Nachricht x.

Der einfache CFB-MAC zu einer Nachricht x und einem ICV wird berechnet, indem zunächst aus der Nachricht x durch Voranstellen des ICV die um 8 Byte längere Nachricht

$$x' = ICV|x$$

gebildet wird. Zu dieser Nachricht x' wird mit dem Schlüssel K der einfache CBC-MAC

$$y = mK(x')$$

berechnet. y ist der zu der Nachricht x und dem ICV gehörige einfache CFB-MAC. Die schrittweise Berechnung des einfachen CFB-MAC Mode läßt sich demnach folgendermaßen darstellen:

Sei $x_1|...|x_n$ die Nachricht x aufgeteilt in 8 Byte Blöcke x_i. Eine Nachrichtenlänge, die ein Vielfaches von 8 Byte ist, erhält man ggf. durch Rechtsauffüllen mit '00'-Bytes (Padding).

Der einfache CFB-MAC ist rekursiv wie folgt definiert:

$$y_0 = eK(ICV)$$

$$y_i = eK(y_{i-1} \text{ XOR } x_i) \text{ für } i = 1,...,n$$

y_n ist der zu der Nachricht gehörige einfache CFB-MAC.

Für die Bildung des einfachen CFB-MAC über eine Nachricht x mit Schlüssel K und Initial Chaining Value ICV wird die folgende Notation verwendet

CFB-MAC Generierung: mK(ICV,x)

2.4.2. Retail MAC

2.4.2.1. Retail CBC-MAC

Eingabe-Daten für den Retail CBC-MAC sind

- ein 16 Byte langer Schlüssel KK,
- der feste 8 Byte lange ICV = '00..00' und
- eine Nachricht x.

Zur Bildung des Retail CBC-MAC über eine in n Blöcke der Länge 8 Byte aufgeteilte Nachricht x wird zunächst als Zwischenresultat ein einfacher CBC-MAC über die ersten n-1 Blöcke berechnet. Dabei verwendet man als Schlüssel die linke Schlüsselhälfte KK_l des doppelt langen Schlüssels KK. Anschließend erfolgt eine Triple-DES-Verschlüsselung mit Schlüssel KK über die XOR-Summe

des Zwischenergebnisses mit dem letzten Nachrichtenblock. Der so erhaltene 8 Byte Ausgabeblock ist der Retail CBC-MAC:

Sei KK_l die 8 Byte lange linke Hälfte von KK und $x_1|...|x_n$ die Nachricht x aufgeteilt in 8 Byte lange Blöcke x_i. Eine Nachrichtenlänge, die ein Vielfaches von 8 Byte ist, erhält man ggf. durch Rechtsauffüllen mit '00'-Bytes (Padding).

Der Retail CBC-MAC ist wie folgt definiert:

$$y = mKK_l(x_1|...|x_{n-1})$$

$$y' = e*KK(y \text{ XOR } x_n)$$

y' ist der zu der Nachricht gehörige Retail CBC-MAC.

Für die Generierung des Retail CBC-MAC über eine Nachricht x mit Schlüssel KK und Initial Chaining Value ICV = '00..00' wird die folgende Notation verwendet

Retail CBC-MAC Generierung: m*KK(x)

2.4.2.2. Retail CFB-MAC

Eingabe-Daten für den Retail CFB-MAC sind

- ein 16 Byte langer Schlüssel KK,
- ein 8 Byte langer ICV und
- eine Nachricht x.

Der Retail CFB-MAC zu einer Nachricht x und einem ICV wird berechnet, indem zunächst aus der Nachricht x durch Voranstellen des ICV die um 8 Byte längere Nachricht

$$x' = ICV|x$$

gebildet wird. Zu dieser Nachricht x' wird mit dem Schlüssel KK der Retail CBC-MAC

$$y = m*KK(x')$$

berechnet. y ist der zu der Nachricht x und dem ICV gehörige Retail CFB-MAC.

Für die Bildung des Retail CFB-MAC über eine Nachricht x mit Schlüssel KK und Initial Chaining Value ICV wird die folgende Notation verwendet

Retail CFB-MAC Generierung: m*KK(ICV,x)

2.5. Schlüsselableitung

Zur Ableitung eines kartenindividuellen Schlüssels aus Kartenidentifikationsdaten (CID) und einem Masterkey (Key Generating Key KGK) wird der folgende Algorithmus verwendet:

Ein kartenindividueller Schlüssel KK von 16 Byte Länge wird aus

- KGK (16 Byte),
- CID mit '00' auf das nächste Vielfache von 8 Byte Länge gepaddet und
- dem öffentlich bekannten Initialwert

  I = '52 52 52 52 52 52 52 52 25 25 25 25 25 25 25 25' (16 Byte)

zu

KK = P(d*KGK(H(I,CID))).

berechnet. Hierbei bezeichnen

- **P** die Funktion "Parity Adjustment", die wie folgt definiert ist:

  Sei $b_1,..,b_8$ die Darstellung eines Byte als Folge von 8 Bit. Dann setzt P das niedrigstwertige Bit b_8 jedes Byte auf ungerade Parität, d. h. b_8 wird in jedem Byte so gesetzt, daß es eine ungerade Anzahl von 1 enthält.

- **d*KGK** die Triple-DES Entschlüsselung mit dem Schlüssel KGK (vgl. Abschnitt 2.2.), wobei die beiden 8 Byte langen Blöcke H_1 und H_2 von $H(I,CID) = H_1|H_2$ einzeln entschlüsselt werden (ECB-Mode):

  $d*KGK(H(I,CID)) = d*KGK(H_1) \mid d*KGK(H_2)$.

- **H** die in ISO 10118-2 ([ISO 10]) definierte Hash-Funktion, die Werte X mit einer Länge, die ein Vielfaches von 8 Byte ist, mittels des Startwertes I (gemäß Anhang A von ISO 10118-2) auf einen Wert von 16 Byte Länge abbildet. Es werden die um das Parity Adjustment P erweiterten Transformationen u (Ad10) und u' (Ad 01) aus Anhang A von ISO 10118-2 verwendet. H ist rekursiv definiert:

  - Sei $X = x_1|...|x_n$ die Zerlegung des Wertes X in 8 Byte lange Blöcke und $L_0|R_0$ die Zerlegung des vorgegebenen Startwerts I in zwei 8 Byte Blöcke.
  - eK(X) ist die DES-Verschlüsselung eines 8 Byte Wertes mit einem 8 Byte Schlüssel.
  - $\oplus$ sei die bitweise Addition modulo 2 (XOR).
  - Die Transformationen Ad10 und Ad01 transformieren 8 Byte Werte K wie folgt:

    Sei $K = k_1,...,k_{64}$ die Darstellung von K als Folge von 64 Bit. Dann ist

$Ad10(K) = [P](k_1,1,0,k_4,...,k_{64})$

$Ad01(K) = [P](k_1,0,1,k_4,...,k_{64})$

[P]: Das Parity Adjustment in Ad10 und Ad01 ist optional. Wenn vor der DES-Verschlüsselung eK(X) keine Paritätsprüfung des Schlüssels K erfolgt, kann P entfallen.

- Dann errechnet sich $L_i|R_i$ aus $L_{i-1}|R_{i-1}$ und x_i wie folgt:

  L_{i-1} bestehe aus den Bits $l_1,...,l_{64}$ und R_{i-1} bestehe aus den Bits $r_1,...,r_{64}$,

  Schritt 1: $L'_i := Ad10(L_{i-1}) = [P](l_1,1,0,l_4,...,l_{64})$

  $R'_i := Ad01(R_{i-1}) = [P](r_1,0,1,r_4,...,r_{64})$

  Schritt 2: $A_i = A_{i[links]}|A_{i[rechts]} = eL'_i(x_i) \oplus x_i$.

  $B_i = B_{i[links]}|B_{i[rechts]} = eR'_i(x_i) \oplus x_i$.

  Schritt 3: $L_i = A_{i[links]}|B_{i[rechts]}$

  $R_i = B_{i[links]}|A_{i[rechts]}$

- $L_n|R_n$ ist dann der Hash-Wert von X unter H:

  $H(I,X) = L_n|R_n$

Soll ein kartenindividueller Schlüssel K von 8 Byte Länge aus KGK, CID und I berechnet werden, wird zuerst der 16 Byte lange Schlüssel KK bestimmt und die linke Hälfte von KK als K verwendet.

2.6. Zufallszahlengenerator

Die ec-Karte benötigt zur Abwicklung kryptographischer Operationen intern erzeugte Zufallszahlen der Länge 8 Byte. Hierzu muß ein interner Zufallszahlengenerator in der ec-Karte implementiert werden, der bei Bedarf, beispielsweise bei der Ausführung des Kommandos GET CHALLENGE (vgl. Kapitel 8.), eine 8 Byte lange, binär kodierte Zufallszahl bereitstellt. Von Zufallszahlen wird gesprochen, wenn die Folge z_1, z_2,... der generierten Werte gute Zufallseigenschaften hat.

Eine Zufallszahl z_i ist von ihrer Erzeugung durch den Zufallszahlengenerator bis zu der Beendigung der Ausführung des nächsten Kommandos, das die ec-Karte nach der Erzeugung des z_i erhält, ***gültig***. Dann muß dieses z_i ***ungültig*** gesetzt werden, auch wenn das z_i bei der Ausführung nicht verwendet wird. Auf diese Weise wird sichergestellt, daß eine Zufallszahl z_i nur bei der Durchführung desjenigen Kommandos verwendet werden kann, das unmittelbar auf die Erzeugung

von z_i durch den Zufallszahlengenerator folgt.

Beispiel:

Als (Pseudo-)Zufallszahlengenerator kann der DES im CBC-Mode mit 8 Byte langem Schlüssel K (vgl. Kapitel 2.3.1.) und ICV = '00..00' eingesetzt werden.

Als Schlüssel K kann hierbei ein 8 Byte langer, geheimer Wert aus einem speziellen EF, dem EF_RAND im MF der ec-Karte, verwendet werden. Als 8 Byte langer Startwert z_0 kann ein Wert verwendet werden, der bei der Initialisierung einer ec-Karte an eine feste Adresse im EEPROM der Karte geschrieben wird.

Benötigt die ec-Karte eine Zufallszahl, so bildet sie dann

$$z_1 = eK('00..00', z_0)$$

als Zufallszahl und überschreibt z_0 mit z_1.

Danach bildet die Karte, immer wenn eine Zufallszahl benötigt wird, sukzessive Zufallszahlen z_2,z_3, ...

$$z_i = eK('00..00', z_{i-1}),\ i > 1$$

und überschreibt die zuvor verwendete Zufallszahl z_{i-1} mit der neu erzeugten Zufallszahl z_i.

2.7. ICV-Handhabung

Wenn bei der Ausführung eines Kommandos eine Ver- bzw. Entschlüsselung oder CFB-MAC-Bildung ausgeführt werden soll, muß dem entsprechenden Sicherheitsalgorithmus DES CBC-Mode, Triple-DES CBC-Mode, einfacher CFB-MAC oder Retail CFB-MAC als Eingabe-Datum ein ICV übergeben werden. Durch die Implementierung des jeweiligen Kommandos wird definiert, auf welche Weise der ICV bei der Kommando-Ausführung bestimmt wird. Beispielsweise kann ein konstanter 8 Byte langer Wert vorgeschrieben werden (vgl. die Kommandos EXTERNAL AUTHENTICATE, INTERNAL AUTHENTICATE, die immer den ICV '00..00' zur Verschlüsselung verwenden), oder der ICV ist das Ergebnis einer Berechnung bzw. Auswahl des Kommandos aus Kommandodaten und/oder Kartendaten.

Zur Bestimmung eines ICV können insbesondere die von der Karte gespeicherten Werte

- durch den Zufallszahlengenerator erzeugte Zufallszahl oder
- durch die externe Welt mittels des Kommandos PUT DATA übergebener und von der Karte gespeicherter 8 Byte langer Wert

verwendet werden. Die externe Welt kann die Erzeugung und Ausgabe einer Zufallszahl mit dem

Kommando GET CHALLENGE anfordern.

Allerdings darf eine Zufallszahl nur dann als ICV verwendet werden, wenn sie gültig (vgl. vorhergehendes Kapitel) ist, so daß eine Zufallszahl nur dann durch ein Kommando als ICV verwendet werden kann, wenn dieses Kommando unmittelbar auf die Erzeugung der Zufallszahl folgt.

Ein mittels PUT DATA übergebener und von der Karte gespeicherter Wert ist von dem Zeitpunkt der Übergabe bis zur Beendigung der Ausführung des nächsten auf PUT DATA folgenden Kommandos ***gültig***. Dann wird er ***ungültig*** und darf nicht mehr als ICV verwendet werden. Ein mit PUT DATA übergebener Wert kann also nur durch ein Kommando als ICV verwendet werden, das unmittelbar auf PUT DATA folgt.

Insbesondere kann es nicht vorkommen, daß einem Kommando sowohl eine gültige Zufallszahl als auch ein gültiger mit PUT DATA übergebener Wert zur Verwendung als ICV zur Verfügung stehen.

Wenn durch ein Kommando eine Zufallszahl bzw. ein mit PUT DATA übergebener Wert als ICV verwendet werden muß, aber zum Zeitpunkt der Ausführung keine gültige Zufallszahl bzw. kein gültiger mit PUT DATA übergebener Wert vorhanden ist, wird das Kommando abgebrochen.

2.8. Schlüsselauswahl

Die für Verschlüsselung und MAC-Bildung verwendbaren kryptographischen Schlüssel der ec-Karte sind in speziellen EFs (vgl. Kapitel 6.) abgelegt. Diese speziellen EFs können in der ec-Karte mehrfach vorkommen. Jedes DF kann

- höchstens ein Schlüssel-EF **EF_KEY** mit bis zu 254 kryptographischen Schlüsseln, die in dem EF_KEY jeweils durch eine **logische Schlüsselnummer KID** zwischen ’00’ und ’FF’ eindeutig identifiziert sind,

- höchstens ein Authentikations-Schlüssel-EF **EF_AUT** mit bis zu 254 kryptographischen Schlüsseln, die in dem EF_AUT jeweils durch eine **logische Schlüsselnummer KID** zwischen ’00’ und ’FF’ eindeutig identifiziert sind,

enthalten.

Wenn ein DF ein EF_KEY bzw. ein EF_AUT enthält, muß es auch jeweils ein zugehöriges Deskriptor-EF, d.h. ein **EF_KEYD** bzw. **EF_AUTD** enthalten, in dem die nicht geheimen Zusatzinformationen zu den Schlüsseln bzw. Authentikations-Schlüsseln, ebenfalls indiziert durch die logische Schlüsselnummer, enthalten sind.

Diese speziellen EFs werden in dem jeweiligen DF durch ihre Datei-ID eindeutig identifiziert.

In einem EF_KEY ist ein Schlüssel mit logischer Schlüsselnummer KID vorhanden, wenn in dem EF_KEY ein Record angelegt ist, der KID und den entsprechenden Schlüssel enthält. In einem EF_KEY, in dem noch kein Record angelegt wurde, ist kein Schlüssel vorhanden.

Die in dem EF_KEY bzw. einem EF_AUT des MF enthaltenen Schlüssel werden als ***globale***

Schlüssel bzw. als ***globale Authentikations-Schlüssel*** der ec-Karte bezeichnet.

Enthält ein DF, das nicht das MF ist, ein EF_KEY bzw. ein EF_AUT, werden die enthaltenen Schlüssel als ***DF-spezifische Schlüssel*** bzw. als ***DF-spezifische Authentikations-Schlüssel*** bezeichnet.

Die Schlüssel in einem EF_KEY mit den Schlüsselnummern '00' bis '03' werden als ***Einzelschlüssel*** bezeichnet. Die übrigen Schlüssel werden in zwei Gruppen eingeteilt. Die ***Schlüsselgruppe 1*** besteht aus den Schlüsseln mit den logischen Schlüsselnummern zwischen '04' und '0E'. Die ***Schlüsselgruppe 2*** umfaßt alle Schlüssel mit Schlüsselnummer > '0E'.

Durch die Verwendung von Schlüsselgruppen ist es möglich, daß zwar in dem EF_KEY eines DF der ec-Karte mehrere oder alle Schlüssel einer Gruppe vorhanden sind, ein Terminal aber nur über einen dieser Schlüssel verfügen muß, um die Kommunikation mit der Karte kryptographisch absichern zu können. Auf diese Weise kann die Verbreitung globaler Schlüssel in Terminals eingeschränkt werden.

Die Schlüssel in einem EF_AUT werden nicht in Gruppen eingeteilt. Authentikations-Schlüssel sind immer Einzelschlüssel.

2.8.1. Such-Algorithmus

Der Algorithmus der ec-Karte zur Auswahl eines Schlüssels oder Authentikations-Schlüssels arbeitet anhand der drei Parameter

- globaler/DF-spezifischer Schlüssel,
- Schlüsselnummer KID und
- Datei der ec-Karte.

Ein globaler Schlüssel oder ein globaler Authentikations-Schlüssel ist durch die Angabe einer Schlüsselnummer eindeutig referenziert, so daß er anhand der Parameter "globaler Schlüssel" und "KID" sofort aufgefunden werden kann, sofern der Schlüssel oder Authentikations-Schlüssel mit der Schlüsselnummer KID im EF_KEY des MF bzw. im EF_AUT des MF vorhanden ist. In diesem Fall ist der Parameter "Datei der ec-Karte" redundant.

Da es in der ec-Karte i. a. mehrere DFs gibt, ist die Angabe der Parameter "DF-spezifischer Schlüssel" und "KID" aber keine eindeutige Referenzierung eines Schlüssels oder Authentikations-Schlüssels der ec-Karte. Die Angabe wird erst eindeutig interpretierbar durch den zusätzlichen Parameter "Datei der ec-Karte".

Zu den Parametern

- DF-spezifischer Schlüssel,
- Schlüsselnummer KID und

- Datei der ec-Karte

wird auf die folgende Weise ein DF-spezifischer Schlüssel bzw. Authentikations-Schlüssel gesucht:

Wenn nach einem Schlüssel aus einem EF_KEY gesucht wird und wenn **KID** einen Wert **zwischen ’00’ und ’03’** hat, wird nach dem ***schlüsselnummer-relevanten DF*** zu der Datei und der Schlüsselnummer KID gesucht.

Das schlüsselnummer-relevante DF zu einer Datei und einer vorgegebenen Schlüsselnummer KID ist wie folgt definiert:

- Ist die Datei ein DF, aber nicht das MF und enthält sie ein EF_KEY, in dem ein Schlüssel mit Schlüsselnummer KID enthalten ist, so ist sie schlüsselnummer-relevantes DF für sich selbst und KID.

- Andernfalls ist das erste übergeordnete DF in der Datei-Struktur der ec-Karte, das nicht das MF ist und das ein EF_KEY enthält, in dem ein Schlüssel mit der Schlüsselnummer KID enthalten ist, das schlüsselnummer-relevante DF zu der Datei und KID.

  Wenn es ein solches DF nicht gibt, existiert kein schlüsselnummer-relevantes DF zu der Datei und KID.

Wird zu der Datei und KID ein schlüsselnummer-relevantes DF gefunden, dann ist in dem EF_KEY dieses DF der gesuchte DF-spezifische Schlüssel enthalten. Andernfalls gibt es zu der Datei und KID keinen DF-spezifischen Schlüssel.

Wenn nach einem Schlüssel aus einem EF_KEY gesucht wird und wenn **KID** einen **Wert > ’03’** hat, wird zunächst nach der **DF-spezifischen Schlüsselgruppe** gesucht, zu der der Schlüssel mit der Schlüsselnummer KID gehört. Hierzu wird nach dem ***schlüsselgruppen-relevanten DF*** zu der Datei und KID gesucht.

Das schlüsselgruppen-relevante DF zu einer Datei und einer Schlüsselnummer KID ist wie folgt definiert:

- Ist die Datei ein DF, aber nicht das MF und enthält sie ein EF_KEY, in dem mindestens ein Schlüssel aus der Schlüsselgruppe enthalten ist, zu der auch der Schlüssel mit der Schlüsselnummer KID gehört, so ist sie schlüsselgruppen-relevantes DF für sich selbst und KID.

- Andernfalls ist das erste übergeordnete DF in der Datei-Struktur der ec-Karte, das nicht das MF ist und das ein EF_KEY enthält, in dem mindestens ein Schlüssel aus der Schlüsselgruppe enthalten ist, zu der auch der Schlüssel mit der Schlüsselnummer KID gehört, das schlüsselgruppen-relevante DF zu der Datei und KID.

  Wenn es ein solches DF nicht gibt, existiert kein schlüsselgruppen-relevantes DF zu der Datei und KID.

Die Suche nach dem schlüsselgruppen-relevanten DF zu einer Datei wird also abgebrochen, wenn

ein DF mit einem EF_KEY gefunden ist, das einen oder mehrere Schlüssel aus der Schlüsselgruppe enthält, zu der der Schlüssel mit der vorgegebenen Nummer gehört, während nach dem schlüsselnummer-relevanten DF gesucht wird, bis ein DF gefunden ist, das ein EF_KEY mit dem Einzelschlüssel mit der vorgegebenen Nummer enthält.

Wird zu der Datei und KID die DF-spezifische Schlüsselgruppe im EF_KEY des schlüsselgruppen-relevanten DF zu der Datei und KID gefunden, muß geprüft werden, ob in der Schlüsselgruppe der gesuchte Schlüssel mit der Schlüsselnummer KID auch enthalten ist. Nur in diesem Fall gibt es zu der Datei und KID einen DF-spezifischen Schlüssel. Wird kein schlüsselgruppen-relevantes DF gefunden oder enthält das EF_KEY des schlüsselgruppen-relevanten DF keinen Schlüssel mit der Schlüsselnummer KID, gibt es keinen DF-spezifischen Schlüssel zu der Datei und KID.

Wenn nach einem Schlüssel aus einem EF_AUT gesucht wird, wird immer nach dem ***authentikations-schlüsselnummer-relevanten DF*** zu der Datei und der Schlüsselnummer KID gesucht.

Das authentikations-schlüsselnummer-relevante DF zu einer Datei und einer vorgegebenen Schlüsselnummer KID ist wie folgt definiert:

- Ist die Datei ein DF, aber nicht das MF und enthält sie ein EF_AUT, in dem ein Authentikations-Schlüssel mit Schlüsselnummer KID enthalten ist, so ist sie authentikations-schlüsselnummer-relevantes DF für sich selbst und KID.

- Andernfalls ist das erste übergeordnete DF in der Datei-Struktur der ec-Karte, das nicht das MF ist und das ein EF_AUT enthält, in dem ein Authentikations-Schlüssel mit der Schlüsselnummer KID enthalten ist, das authentikations-schlüsselnummer-relevante DF zu der Datei und KID.

  Wenn es ein solches DF nicht gibt, existiert kein authentikations-schlüsselnummer-relevantes DF zu der Datei und KID.

Wird zu der Datei und KID ein authentikations-schlüsselnummer-relevantes DF gefunden, dann ist in dem EF_AUT dieses DF der gesuchte DF-spezifische Authentikations-Schlüssel enthalten. Andernfalls gibt es zu der Datei und KID keinen DF-spezifischen Authentikations-Schlüssel.

Im folgenden wird der Such-Algorithmus für Schlüssel anhand der in Bild 3 **beispielhaft** dargestellten Speicherung von Schlüsseln (repräsentiert durch ihre Schlüsselnummer) in einer Datei-Struktur tabellarisch erläutert.

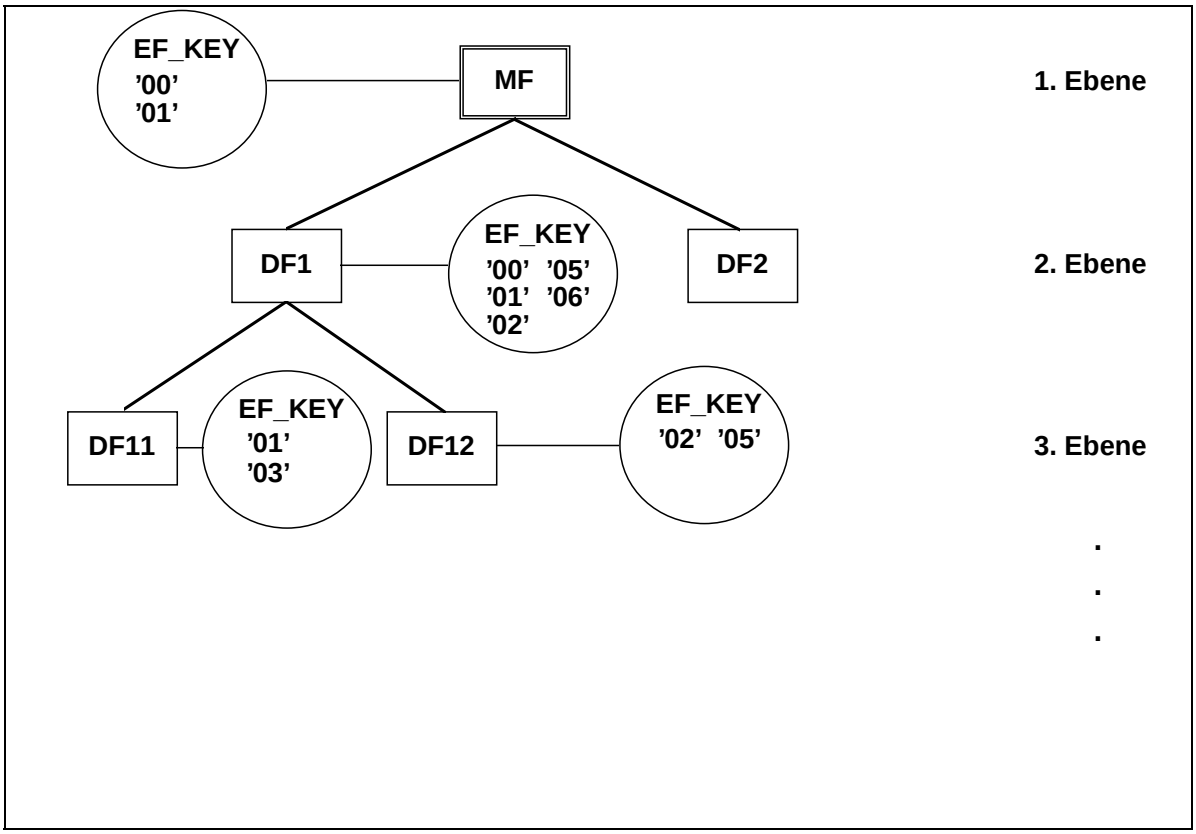


Bild 3: Speicherung von Schlüsseln in der Datei-Struktur einer ec-Karte

Der Einfachheit halber sind in Bild 3 keine EFs außer den EF_KEYs dargestellt. Für ein EF führt der Such-Algorithmus zu demselben Schlüssel wie für das DF, in dem das EF direkt enthalten ist.

Die Einträge der folgenden Tabelle zeigen, in welchem DF das EF_KEY enthalten ist, in dem der Schlüssel gefunden wird, der mit den Parametern

- globaler Schlüssel,
- Schlüsselnummer KID und
- DF aus der Datei-Struktur in Bild 3

mittels des Such-Algorithmus gefunden wird.

DF KID	MF	DF1	DF2	DF11	DF12
globaler Einzelschlüssel					
'00'	MF	MF	MF	MF	MF
'01'	MF	MF	MF	MF	MF
'02'	-	-	-	-	-
'03'	-	-	-	-	-
globale Schlüsselgruppe 1					
'04'	-	-	-	-	-
'05'	-	-	-	-	-
'06'	-	-	-	-	-
'07'-'0E'	-	-	-	-	-
globale Schlüsselgruppe 2					
'0F'-'FF'	-	-	-	-	-

Tabelle 1: Ergebnis des Such-Algorithmus für globale Schlüssel

Die Einträge der folgenden Tabelle zeigen, in welchem DF das EF_KEY enthalten ist, in dem der Schlüssel gefunden wird, der mit den Parametern

- DF-spezifischer Schlüssel,
- Schlüsselnummer KID und
- DF aus der Datei-Struktur in Bild 3

mittels des Such-Algorithmus gefunden wird.

DF KID	MF	DF1	DF2	DF11	DF12
DF-spezifischer Einzelschlüssel					
'00'	-	DF1	-	DF1	DF1
'01'	-	DF1	-	DF11	DF1
'02'	-	DF1	-	DF1	DF12
'03'	-	-	-	DF11	-
DF-spezifische Schlüsselgruppe 1					
'04'	-	-	-	-	-
'05'	-	DF1	-	DF1	DF12
'06'	-	DF1	-	DF1	-[1)]
'07'-'0E'	-	-	-	-	-
DF-spezifische Schlüsselgruppe 2					
'0F'-'FF'	-	-	-	-	-

Tabelle 2: Ergebnis des Such-Algorithmus für DF-spezifische Schlüssel

Erläuterung:

1) Das schlüsselgruppen-relevante DF zu DF12 und allen Schlüsselnummern zwischen '04' und '0E' ist DF12 selbst und dieses enthält keinen Schlüssel mit der Nummer '04' oder einer Nummer zwischen '06' und '0E'.

2.8.2. Mechanismus zur Schlüsselauswahl

Wenn bei der Ausführung eines Kommandos zur Durchführung einer kryptographischen Operation ein Schlüssel oder Authentikations-Schlüssel benötigt wird, wird zunächst mittels des oben beschriebenen Algorithmus nach dem (Authentikations-)Schlüssel gesucht. Wenn hierbei kein Schlüssel gefunden wird oder wenn Struktur oder Recordlänge des EF_KEY bzw. EF_AUT nicht der Spezifikation entsprechen (vgl. Kapitel 6.2. und 6.4.), wird das Kommando abgebrochen.

Wenn der gewünschte Schlüssel gefunden wird, kann die ec-Karte einen Paritätscheck des Schlüssels durchführen. Werden hierbei Fehler festgestellt, bricht die ec-Karte das in Bearbeitung befindliche Kommando ab.

Wenn ein 8 Byte langer Schlüssel benötigt wird, werden Byte 2 - 9 des entsprechenden Records des EF_KEY bzw. EF_AUT verwendet.

Wenn zwar der gewünschte (Authentikations-)Schlüssel gefunden wird, aber zu dem EF_KEY bzw. EF_AUT, in dem der Schlüssel enthalten ist, kein EF_KEYD bzw. EF_AUTD in dem entsprechenden DF existiert, oder in dem zugehörigen EF_KEYD bzw. EF_AUTD keine Zusatzinformationen unter der KID des Schlüssels gespeichert sind, oder diese Zusatzinformationen inkonsistent sind, oder Struktur oder Recordlänge des EF_KEYD bzw.

EF_AUTD nicht der Spezifikation entsprechen (vgl. Kapitel 6.3.2. und 6.5.2.), wird das Kommando abgebrochen.

Wenn der zu einem Schlüssel gehörende Fehlbedienungszähler im jeweiligen EF_KEYD den Wert '00' hat, wird das Kommando abgebrochen.

2.9. Paßwortauswahl

Die für die Karteninhaber-Authentikation verwendbaren geheimen Paßwörter der ec-Karte sind in speziellen EFs (vgl. Kapitel 6.) abgelegt. Diese speziellen EFs können in der ec-Karte mehrfach vorkommen:

Jedes DF kann

- höchstens ein Paßwort-EF **EF_PWD0**, mit genau einem Paßwort, dem Paßwort mit der **Paßwortnummer 0**, und
- höchstens ein Paßwort-EF **EF_PWD1**, mit genau einem Paßwort, dem Paßwort mit der **Paßwortnummer 1**

enthalten.

Ein in Record '01' eines EF_PWDx des MF enthaltenes Paßwort wird als ***globales Paßwort*** der ec-Karte bezeichnet.

Enthält ein DF, das nicht das MF ist, ein EF_PWD0 und/oder ein EF_PWD1, wird das jeweils in Record '01' enthaltene Paßwort als ***DF-spezifisches Paßwort*** bezeichnet.

Wenn ein DF ein EF_PWDx enthält, muß es auch ein zugehöriges Deskriptor-EF, d. h. ein **EF_PWDDx** enthalten, in dessen Record '01' nicht geheime Zusatzinformationen zu dem jeweiligen Paßwort gespeichert sind.

Der Fehlbedienungszähler zu einem in Record '01' eines EF_PWD1 enthaltenen Paßworts befindet sich im Record '01' des jeweils zugehörigen EF_PWDD1. Der **gemeinsame** Fehlbedienungszähler zu allen in den EF_PWD0 der ec-Karte enthaltenen Paßwörtern befindet sich im Record '01' des EF_FBZ im MF.

Die genannten speziellen EFs werden in dem jeweiligen DF durch ihre Datei-ID eindeutig identifiziert.

2.9.1. Such-Algorithmus

Der Algorithmus der ec-Karte zur Auswahl eines Paßworts arbeitet anhand der drei Parameter

- globales/DF-spezifisches Paßwort,

- Paßwortnummer x (0 oder 1) und

- Datei der ec-Karte.

Ein globales Paßwort ist durch die Angabe einer Paßwortnummer eindeutig referenziert, so daß es anhand der Parameter "globales Paßwort" und "x" sofort aufgefunden werden kann, sofern das entsprechende EF_PWDx im MF vorhanden ist und ein Paßwort enthält. In diesem Fall ist der Parameter "Datei der ec-Karte" redundant.

Da es in der ec-Karte i. a. mehrere DFs gibt, ist die Angabe der Parameter "DF-spezifisches Paßwort" und "x" aber keine eindeutige Referenzierung eines DF-spezifischen Paßworts der ec-Karte. Die Angabe wird erst eindeutig interpretierbar durch den zusätzlichen Parameter "Datei der ec-Karte".

Zu den Parametern

- Paßwortnummer x und

- Datei der ec-Karte

wird auf die folgende Weise ein DF-spezifisches Paßwort gesucht:

Zunächst wird nach dem ***paßwortnummer-relevanten DF*** zu der Datei und der Paßwortnummer x gesucht.

Das paßwortnummer-relevante DF zu einer Datei der ec-Karte und einer vorgegebenen Paßwortnummer x ist wie folgt definiert:

- Ist die Datei ein DF, aber nicht das MF und enthält sie ein EF_PWDx, so ist sie paßwortnummer-relevantes DF für sich selbst und die angegebene Paßwortnummer x.

- Andernfalls ist das erste übergeordnete DF in der Datei-Struktur der ec-Karte, das nicht das MF ist und das ein EF_PWDx enthält, das paßwortnummer-relevante DF zu der Datei und der Paßwortnummer x.

  Wenn es ein solches DF nicht gibt, existiert kein paßwortnummer-relevantes DF zu der Datei und der Paßwortnummer x.

Wird zu der Datei und x ein paßwortnummer-relevantes DF gefunden, wird geprüft, ob in dem EF_PWDx dieses DF ein Paßwort enthalten ist. Dies ist dann das gesuchte Paßwort. Wird kein paßwortnummer-relevantes DF gefunden oder enthält das EF_PWDx des paßwortnummer-relevanten DF kein Paßwort, gibt es kein DF-spezifisches Paßwort zu der Datei und der Paßwortnummer x.

Im folgenden wird der Such-Algorithmus für Paßwörter anhand der in Bild 4 **beispielhaft** dargestellten Datei-Struktur tabellarisch erläutert.

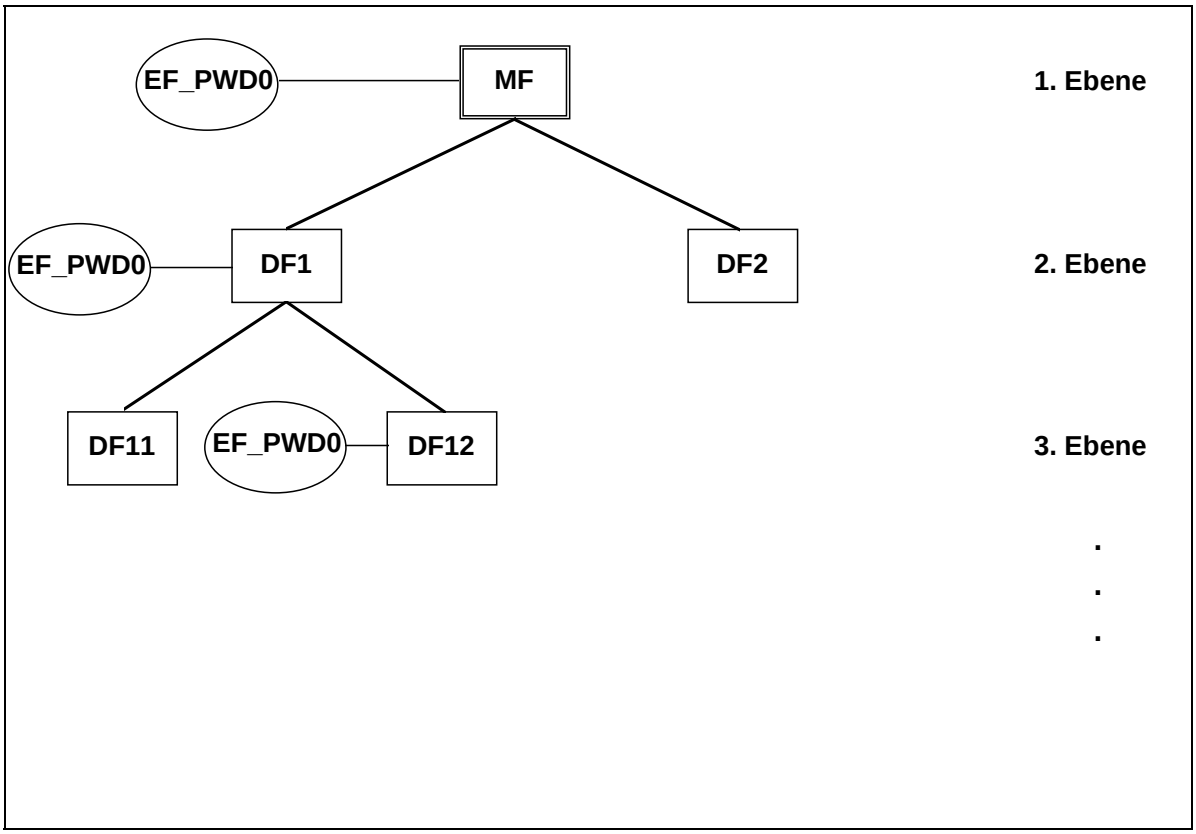


Bild 4: EF_PWDs in der Datei-Struktur einer ec-Karte

Der Einfachheit halber sind in diesem Bild keine EFs außer den EF_PWD0s dargestellt. Für ein EF führt der Such-Algorithmus zu demselben Paßwort wie für das DF, in dem das EF direkt enthalten ist.

Die Einträge der folgenden Tabelle zeigen, in welchem DF das gefundene EF_PWDx enthalten ist, das mit den Parametern

- globales Paßwort,
- Paßwortnummer x und
- DF aus der Datei-Struktur in Bild 4

mittels des Such-Algorithmus gefunden wird.

DF x	MF	DF1	DF2	DF11	DF12
0	MF	MF	MF	MF	MF
1	-	-	-	-	-

Tabelle 3: Ergebnis des Such-Algorithmus für globale Paßwörter

Die Einträge der folgenden Tabelle zeigen, in welchem DF das gefundene EF_PWDx enthalten ist,

das mit den Parametern

- DF-spezifisches Paßwort,
- Paßwortnummer x und
- DF aus der Datei-Struktur in Bild 4

mittels des Such-Algorithmus gefunden wird.

DF x	MF	DF1	DF2	DF11	DF12
0	-	DF1	-	DF1	DF12
1	-	-	-	-	-

Tabelle 4: Ergebnis des Such-Algorithmus für DF-spezifische Paßwörter

2.9.2. Mechanismus zur Paßwortauswahl

Wenn bei der Ausführung eines Kommandos ein Paßwort benötigt wird, wird zunächst mittels des oben beschriebenen Algorithmus nach dem Paßwort gesucht. Wenn hierbei kein Paßwort gefunden wird oder wenn Struktur oder Recordlänge des EF_PWDx nicht der Spezifikation entsprechen (vgl. Kapitel 6.6.), wird das Kommando abgebrochen.

Wenn zwar das gewünschte Paßwort gefunden wird, aber die zu dem Paßwort gehörigen Zusatzinformationen nicht gefunden werden können, wird das Kommando ebenfalls abgebrochen. Das ist dann der Fall,

- wenn für ein in einem EF_PWD0 enthaltenes Paßwort kein EF_PWDD0 in dem entsprechenden DF existiert oder leer ist, oder wenn das EF_FBZ im MF nicht existiert oder leer ist oder die Struktur oder Recordlänge des EF_FBZ nicht der Spezifikation entsprechen (vgl. Kapitel 6.8.),
- wenn für ein in einem EF_PWD1 enthaltenes Paßwort kein EF_PWDD1 in dem entsprechenden DF existiert oder leer ist oder die Struktur oder Recordlänge des EF_PWDD1 nicht der Spezifikation entsprechen (vgl. Kapitel 6.7.).

Wenn der zu einem Paßwort gehörende Fehlbedienungszähler im jeweils zugehörigen EF_PWDD1 bzw. im EF_FBZ den Wert '00' hat, wird das Kommando ebenfalls abgebrochen.

3. Secure Messaging

An der Schnittstelle der ec-Karte können die Sicherheitsmechanismen

- MAC-Bildung oder

- MAC-Bildung und Verschlüsselung

verwendet werden, um die Sicherheitsdienste

- Nachweis der Integrität der Daten oder
- Nachweis der Integrität und Sicherstellung der Vertraulichkeit der Daten

von Kommandonachrichten von einem Terminal an die Karte und/oder Antwortnachrichten von der Karte an ein Terminal bereitzustellen.

Ob ein Sicherheitsmechanismus, welcher Sicherheitsmechanismus und mit welchen Parametern verwendet werden soll, hängt ab von dem jeweiligen Kommando und von den Sicherheitsattributen der Datei, auf die mit dem Kommando zugegriffen werden soll (vgl. Kapitel 4. und 8.).

Die Formate der durch die im folgenden beschriebenen Sicherheitsmechanismen erzeugten Kommando- und Antwortnachrichten entsprechen nicht den in Kapitel 5.6 von ISO 7816-4 definierten. Da es sich somit um ein proprietäres Secure Messaging der ec-Karte handelt, wird das rechte Halbbyte des CLA-Byte eines Kommandos an die ec-Karte, das mit Secure Messaging durchgeführt wird, zu '4' kodiert.

3.1. Nachweis der Integrität von Nachrichten

Um die Integrität von Kommandonachrichten an die ec-Karte oder Antwortnachrichten von der ec-Karte nachprüfbar zu machen, wird über die jeweilige Nachricht ein CFB-MAC berechnet und mit der Nachricht versandt. Hierbei wird bei Verwendung eines 8 Byte langen Schlüssels ein einfacher CFB-MAC (vgl. Kapitel 2.4.1.2.), bei Verwendung eines 16 Byte langen Schlüssels ein Retail CFB-MAC (vgl. Kapitel 2.4.2.2.) berechnet. Welcher Schlüssel zu verwenden ist, ergibt sich aus dem jeweiligen Kontext.

Die Struktur von Kommando- und Antwortnachrichten an der Schnittstelle der ec-Karte wird in Kapitel 8. beschrieben.

3.1.1. MAC-Bildung für Kommandonachrichten

Die folgenden Schritte sind bei der CFB-MAC-Bildung mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

über Kommandonachrichten durch das Terminal durchzuführen:

1. Sei CLA|INS|P1|P2|[L_c]|[Datenfeld]|[L_e] die Kommandonachricht, über die ein CFB-MAC zu bilden ist.

- Ist L_c nicht vorhanden, so wird hinter P2 ein Feld $L_{c'}$ mit dem Wert '08' eingefügt.

- Ist L_c vorhanden, so wird es durch $L_c' = L_c + '08'$ ersetzt.

Die so bearbeitete Nachricht CLA|INS|P1|P2|$L_{c'}$|[Datenfeld]|[L_e] wird mit x bezeichnet.

Hat x eine Gesamtlänge in Byte, die kein Vielfaches von 8 ist, so wird die Nachricht auf eine Byte-Länge, die ein Vielfaches von 8 ist, durch Anhängen von maximal 7 '00'-Byte (Filler) aufgefüllt.

Die so erzeugte Nachricht wird mit x' bezeichnet.

2. Berechnung des 8 Byte langen CFB-MAC zu x':

   MAC = mK(ICV,x') für einen 8 Byte langen Schlüssel gemäß Kapitel 2.4.1.2. oder

   MAC = m*KK(ICV,x') für einen 16 Byte langen Schlüssel gemäß Kapitel 2.4 .2.2.

3. Aus der Nachricht x wird durch Anhängen des MAC an das Datenfeld die Kommandonachricht CLA|INS|P1|P2|$L_{c'}$|[Datenfeld]|MAC|[L_e] generiert und an die ec-Karte gesandt.

Die folgenden Schritte sind bei der Prüfung des CFB-MAC einer Kommandonachricht mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und

- einem 8 Byte langen ICV

durch die ec-Karte durchzuführen:

1. Erzeugen einer Nachricht y aus der empfangenen Kommandonachricht durch Entfernen der letzten 8 Byte des Datenfeldes,

2. Falls notwendig, Auffüllen der Nachricht y mit maximal 7 Filler-Byte '00' zu einer Nachricht y' mit einer Byte-Länge, die ein Vielfaches von 8 ist,

3. Berechnung des 8 Byte langen CFB-MAC zu y':

   MAC' = mK(ICV,y') für einen 8 Byte langen Schlüssel gemäß Kapitel 2.4.1.2. oder

   MAC' = m*KK(ICV,y') für einen 16 Byte langen Schlüssel gemäß Kapitel 2.4.2.2.

4. Vergleich des berechneten MAC' mit den letzten 8 Byte des Datenfeldes aus der empfangenen Kommandonachricht.

   Sind die Werte gleich, so ist die Integrität der Kommandonachricht nachgewiesen.

   Sind die Werte verschieden, so dekrementiert die ec-Karte den dem Schlüssel

zugeordneten Fehlbedienungszähler um '01' und weist das Kommando zurück.

3.1.2. MAC-Bildung für Antwortnachrichten

Für Antwortnachrichten zu einem Kommando wird von der ec-Karte nur dann ein MAC gebildet, wenn das Datenfeld der Nachricht nicht leer ist.

Bei dem Wert des Längen-Bytes L_e in der Kommandonachricht sollte berücksichtigt werden, daß die Antwortdaten um einen 8 Byte langen MAC ergänzt werden.

Die folgenden Schritte sind bei der CFB-MAC-Bildung über eine Antwortnachricht mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

durch die ec-Karte durchzuführen:

1. Sei Datenfeld|SW1 SW2 die Antwortnachricht, über die ein CFB-MAC zu bilden ist.

   Hat das Datenfeld eine Gesamtlänge in Byte, die kein Vielfaches von 8 ist, so wird es auf eine Byte-Länge, die ein Vielfaches von 8 ist, durch Anhängen von maximal 7 Filler-Byte '00' aufgefüllt.

   Das so erweiterte Datenfeld wird mit x' bezeichnet.

2. Berechnung des 8 Byte langen CFB-MAC zu x':

   MAC = mK(ICV,x') für einen 8 Byte langen Schlüssel gemäß Kapitel 2.4.1.2. oder

   MAC = m*KK(ICV,x') für einen 16 Byte langen Schlüssel gemäß Kapitel 2.4.2.2.

3. Aus dem Datenfeld der ursprünglichen Antwortnachricht wird durch Anhängen des MAC an das Datenfeld das Datenfeld' = Datenfeld|MAC generiert. Als Antwortnachricht wird Datenfeld'|SW1 SW2 an das Terminal gesandt.

Die folgenden Schritte sind bei der Prüfung des MAC einer Antwortnachricht mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

durch das Terminal durchzuführen:

1. Entnehmen des Datenfeld' aus der empfangenen Antwortnachricht und Erzeugen eines

Wertes y durch Entfernen der letzten 8 Byte des Datenfeld',

2. Falls notwendig, Auffüllen des Wertes y mit maximal 7 Filler-Byte '00' zu einem Wert y' mit einer Byte-Länge, die ein Vielfaches von 8 ist,

3. Berechnung des 8 Byte langen CFB-MAC zu y':

   MAC' = mK(ICV,y') für einen 8 Byte langen Schlüssel gemäß Kapitel 2.4.1.2. oder

   MAC' = m*KK(ICV,y') für einen 16 Byte langen Schlüssel gemäß Kapitel 2.4.2.2.

4. Vergleich des berechneten MAC' mit den letzten 8 Byte des Datenfeld' aus der empfangenen Antwortnachricht.

   Sind die Werte gleich, so ist die Integrität der Antwortdaten nachgewiesen.

3.2. Nachweis der Integrität und Sicherstellung der Vertraulichkeit von Nachrichten

Um die Integrität von Kommandonachrichten an die ec-Karte oder Antwortnachrichten von der ec-Karte nachprüfbar zu machen und die Vertraulichkeit der Daten sicherzustellen, wird die Nachricht im CFB-MAC-gesichert und CBC-verschlüsselt. Die verschlüsselte Nachricht wird dann versandt. Hierbei wird bei Verwendung eines 8 Byte langen Schlüssels ein einfacher CFB-MAC (vgl. Kapitel 2.4.1.2.) berechnet und mittels des DES im CBC-Mode verschlüsselt (vgl. Kapitel 2.3.1.) Bei Verwendung eines 16 Byte langen Schlüssels wird ein Retail CFB-MAC (vgl. Kapitel 2.4.2.2.) gebildet und mittels des Triple-DES im CBC-Mode verschlüsselt (vgl. Kapitel 2.3.2.). Welche Parameter zu verwenden ist ergibt, sich aus dem jeweiligen Kontext.

3.2.1. MAC-Bildung und Verschlüsselung für Kommandonachrichten

Die folgenden Schritte sind bei der CFB-MAC-Bildung und CBC-Verschlüsselung mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

von Kommandonachrichten durch das Terminal durchzuführen:

1. Sei CLA|INS|P1|P2|[L_c]|[Datenfeld]|[L_e] die Kommandonachricht.

   Aus dieser Nachricht wird wie in Kapitel 3.1.1. beschrieben die um den CFB-MAC ergänzte Nachricht CLA|INS|P1|P2|$L_{c'}$|[Datenfeld]|MAC|[L_e] generiert.

2. Für den Teil $L_{c'}$|[Datenfeld]|MAC|[L_e] der Nachricht wird ein Padding gemäß ISO 7816-4 durchgeführt. Dazu wird immer ein Byte '80' an diesen Nachrichtenteil angehängt. Hat das Ergebnis $L_{c'}$|[Datenfeld]|MAC|[L_e]|'80' eine Länge in Byte, die kein Vielfaches von 8 ist, so

wird es auf eine Byte-Länge, die ein Vielfaches von 8 ist, durch Anhängen von maximal 7 Filler-Byte '00' aufgefüllt.

Die so ergänzten Daten x = $L_{C'}$|[Datenfeld]|MAC|[L_e]|'80'|[Filler] werden mit demselben Schlüssel verschlüsselt, mit dem bereits der MAC gebildet wurde:

ENC = eK(ICV,x) für einen 8 Byte langen Schlüssel gemäß Kapitel 2.3.1. oder

ENC = e*KK(ICV,x) für einen 16 Byte langen Schlüssel gemäß Kapitel 2.3.2.

Hierbei wird derselbe ICV wie bei der MAC-Bildung verwendet.

3. CLA|INS|P1|P2|ENC werden an die ec-Karte gesandt.

Die folgenden Schritte sind bei der Entschlüsselung mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

einer Kommandonachricht durch die ec-Karte durchzuführen:

1. Erzeugen eines Wertes y aus der empfangenen Nachricht durch Entfernen der Bytes CLA|INS|P1|P2,

   Prüfen, ob y eine Byte-Länge hat, die ein Vielfaches von 8 ist. Ist das nicht der Fall, wird das Kommando abgebrochen.

2. Entschlüsselung von y:

   x' = dK(ICV,y) für einen 8 Byte langen Schlüssel gemäß Kapitel 2.3.1. oder

   x' = d*KK(ICV,y) für einen 16 Byte langen Schlüssel gemäß Kapitel 2.3.2.

3. Prüfen des Padding, indem von rechts nach links in x' das erste von '00' verschiedene Byte gesucht wird. Wird dieses nicht nach höchstens 7 Byte '00' gefunden oder hat es nicht den Wert '80', wird das Kommando abgebrochen.

   Die Filler-Byte '00' und das Byte '80' werden entfernt und CLA|INS|P1|P2 an den Anfang gestellt, um die ursprüngliche Kommandonachricht zu erhalten.

Nach positiv verlaufenen kommandospezifischen Prüfungen der gewonnenen Kommandonachricht wird die MAC-Prüfung aus Kapitel 3.1.1. für die Nachricht unter Verwendung desselben ICV und Schlüssels durchgeführt.

3.2.2. MAC-Bildung und Verschlüsselung für Antwortnachrichten

Zu Antwortnachrichten wird von der ec-Karte nur dann ein MAC gebildet und Verschlüsselung durchgeführt, wenn das Datenfeld der Nachricht nicht leer ist.

Bei dem Wert des Längen-Bytes L_e in der Kommandonachricht sollte berücksichtigt werden, daß die Antwortdaten um einen 8 Byte langen MAC ergänzt werden und anschließend ein Padding gemäß ISO 7816-4 durchgeführt wird.

Die folgenden Schritte sind bei der MAC-Bildung und Verschlüsselung mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

für eine Antwortnachricht durch die ec-Karte durchzuführen:

1. Sei Datenfeld|SW1 SW2 die Antwortnachricht.

   Aus dieser Nachricht wird wie in Kapitel 3.1.2. beschrieben ein um den MAC ergänztes Datenfeld Datenfeld|MAC generiert. Für Datenfeld|MAC wird ein Padding gemäß ISO 7816-4 durchgeführt. Dazu wird immer ein Byte '80' an diese Daten angehängt. Hat das Ergebnis Datenfeld|MAC|'80' eine Länge in Byte, die kein Vielfaches von 8 ist, so wird es auf eine Byte-Länge, die ein Vielfaches von 8 ist, durch Anhängen von maximal 7 Filler-Byte '00' aufgefüllt.

2. Die so ergänzten Daten x = Datenfeld|MAC|'80'|[Filler] werden mit demselben Schlüssel verschlüsselt, mit dem bereits der MAC gebildet wurde:

   ENC = eK(ICV,x) für einen 8 Byte langen Schlüssel gemäß Kapitel 2.3.1. oder

   ENC = e*KK(ICV,x) für einen 16 Byte langen Schlüssel gemäß Kapitel 2.3.2.

   Hierbei wird derselbe ICV wie bei der MAC-Bildung verwendet.

3. ENC|SW1 SW2 wird an das Terminal gesandt.

Die folgenden Schritte sind bei der Entschlüsselung mit

- einem Schlüssel K (8 Byte lang) oder KK (16 Byte lang) und
- einem 8 Byte langen ICV

einer Antwortnachricht durch das Terminal durchzuführen:

1. Auswerten des Returncodes SW1-SW2.

2. Prüfen, ob das Datenfeld (ENC) der Antwortnachricht eine Byte-Länge hat, die ein Vielfaches von 8 ist.

3. Entschlüsselung des Datenfeldes der Antwortnachricht:

   x' = dK(ICV,Datenfeld) für einen 8 Byte langen Schlüssel gemäß Kapitel 2.3.1. oder

   x' = d*KK(ICV,Datenfeld) für einen 16 Byte langen Schlüssel gemäß Kapitel 2.3.2.

4. Von rechts nach links wird in x' das erste von '00' verschiedene Byte gesucht. Wird dieses nach höchstens 7 Filler-Byte '00' gefunden und hat es den Wert '80', werden die Filler-Byte '00' und das Byte '80' entfernt, um das ursprüngliche Datenfeld der Antwortnachricht zurückzugewinnen.

Nach positiv verlaufenen kommandospezifischen Prüfungen der gewonnenen Antwortnachricht wird die MAC-Prüfung aus Kapitel 3.1.2. für die Nachricht unter Verwendung desselben ICV und Schlüssels durchgeführt.

4. Sicherheits-Architektur der ec-Karte

Auf die in einer personalisierten ec-Karte gespeicherte Information kann ausschließlich mittels der Kommandos der ec-Karte zugegriffen werden.

Für die ec-Karte wird zwischen Standardkommandos, Administrationskommandos und Ergänzungskommandos unterschieden.

Als ***Standardkommandos*** werden Inter Industry Commands nach ISO 7816-4 bezeichnet, die für den regulären Betrieb der ec-Karte zur Verfügung stehen.

Administrationskommandos sind solche, die zur Administration und Wartung der ec-Karte verwendet werden können. Sie dienen

- der Wartung von bereits geladenen Anwendungen,
- dem Nachladen und Einrichten von neuen Anwendungen und Kommandos sowie
- der Ersetzung von Kommandos, um Fehler beheben zu können.

Alle weiteren Kommandos werden als ***Ergänzungskommandos*** bezeichnet. Die für die ec-Karte bisher definierten Ergänzungskommandos dienen der Abwicklung der Anwendungen electronic cash und elektronische Geldbörse des deutschen Kreditgewerbes. Sie sind in den Spezifikationsdokumenten [LIT 3], [LIT 4A] und [LIT 4B] beschrieben.

Zentral für die Sicherheitsarchitektur der ec-Karte ist der Begriff ***Sicherheitsattribut***. Hierbei wird unterschieden zwischen ***implizit*** und ***explizit festgelegten*** Sicherheitsattributen.

Für jedes Kommando der ec-Karte ist festzulegen, unter welchen Bedingungen bzw. in welcher Weise das Kommando ausgeführt werden darf. Insbesondere muß für jede Datei der ec-Karte festgelegt werden, ob und unter welchen Voraussetzungen die Kommandos der ec-Karte auf diese Datei zugreifen dürfen. Diese Festlegungen werden als **Sicherheitsattribute** des jeweiligen Kommandos bezeichnet.

Sicherheitsattribute, die allein im jeweiligen Kommandocode realisiert werden, werden als **implizit festgelegte** Sicherheitsattribute bezeichnet. Implizit festgelegte Sicherheitsattribute sind fest in das jeweilige Kommando hineinprogrammiert und können nach der Einbringung des Kommandocodes in die ec-Karte nicht mehr verändert werden.

Ein Sicherheitsattribut wird für ein Kommando **explizit festgelegt**, wenn

- beim Anlegen einer Datei eine ***Access Condition (AC)*** oder ***Zugriffsbedingung*** (vgl. Kapitel 4.2.) festgelegt wird und

- im Kommandocode sichergestellt wird, daß die AC der Datei vor der Ausführung des Kommandos ausgewertet wird und der weitere Ablauf des Kommandos von dem Ergebnis der Auswertung abhängt.

Auf diese Weise ist es möglich, erst beim Anlegen von Dateien der ec-Karte zu entscheiden, unter welcher Bedingung ein Kommando ausgeführt werden darf bzw. unter welcher Bedingung ein Kommando auf eine Datei zugreifen darf.

Eine AC ist immer an eine Datei gebunden, kann aber auf verschiedene Weise durch die Kommandos der ec-Karte genutzt werden. Beispielsweise kann durch die AC einer Datei festgelegt werden, unter welcher Bedingung ein einzelnes Kommando oder eine Gruppe von Kommandos genau auf diese Datei zugreifen dürfen. Durch die AC einer Datei kann aber auch bestimmt werden, unter welcher Voraussetzung ein Kommando überhaupt ausgeführt werden darf oder auf andere Dateien zugreifen darf.

ACs können daher als dateigebundene Variablen für Sicherheitsattribute bezeichnet werden.

4.1. Sicherheitszustand

Ein Sicherheitsattribut kann für ein Kommando der ec-Karte festlegen, daß das Kommando nur unter der Bedingung ausgeführt werden darf bzw. auf eine Datei zugreifen darf, daß sich die ec-Karte in einem bestimmten ***Sicherheitszustand*** befindet. Nur wenn das der Fall ist, wird das Kommando in der vorgeschriebenen Weise ausgeführt.

Die ec-Karte verwaltet hierzu die folgenden Sicherheitszustände:

- ***globaler Sicherheitszustand***,

- ***DF-spezifischer Sicherheitszustand*** und

- ***kommandospezifischer Sicherheitszustand***.

Um einen der Sicherheitszustände zu verändern, muß sich die externe Welt gegenüber der ec-Karte durch Nachweis der Kenntnis von karteninhaber-spezifischen Daten und/oder eines geheimen Schlüssels authentisieren. Bei den karteninhaber-spezifischen Daten kann es sich prinzipiell um beliebige, den Karteninhaber authentisierende Daten, beispielsweise um die Binärdarstellung eines biometrischen Merkmals handeln. Zur Zeit wird aber ausschließlich ein Paßwort zur Karteninhaber-Authentikation verwendet. Daher wird im folgenden nur der Begriff Paßwort verwendet.

4.1.1. Globaler und DF-spezifischer Sicherheitszustand

In Abhängigkeit davon, welche der speziellen EFs (vgl. Kapitel 6.)

- **EF_PWD0**, mit genau einem Paßwort, dem Paßwort mit der **Paßwortnummer 0**,

- **EF_PWD1**, mit genau einem Paßwort, dem Paßwort mit der **Paßwortnummer 1**, und

- **EF_KEY** mit bis zu 254 kryptographischen Schlüsseln, die in dem EF_KEY jeweils durch eine **logische Schlüsselnummer KID** zwischen ’00’ und ’FF’ eindeutig identifiziert sind,

in einem DF enthalten sind und welche Schlüssel in einem enthaltenen EF_KEY vorhanden sind, kann sich die externe Welt **bezogen auf dieses DF** gegenüber der ec-Karte mittels der Kommandos VERIFY und EXTERNAL AUTHENTICATE (vgl. Kapitel 8.) authentisieren:

- Enthält das DF ein EF_PWD0, kann sich die externe Welt durch korrekte Angabe des Paßwortes mit der Paßwortnummer 0 mittels des Kommandos VERIFY, bezogen aus dieses DF, authentisieren.

- Enthält das DF ein EF_PWD1, kann sich die externe Welt durch korrekte Angabe des Paßwortes mit der Paßwortnummer 1 mittels des Kommandos VERIFY, bezogen auf dieses DF, authentisieren.

- Enthält das DF ein EF_KEY, in dem ein Schlüssel mit logischer Schlüsselnummer KID vorhanden ist, kann sich die externe Welt mittels des Kommandos EXTERNAL AUTHENTICATE unter Verwendung dieses Schlüssels, bezogen auf dieses DF, authentisieren.

Jedem DF, das ein EF_PWD0 und/oder ein EF_PWD1 und/oder ein EF_KEY enthält, ist ein **Sicherheitszustand** zugeordnet, durch den nachgehalten werden kann, welche der möglichen Authentikationen der externen Welt, bezogen auf dieses DF, bereits erfolgreich stattgefunden

haben.

Der Sicherheitszustand, der dem MF zugeordnet ist, wird als **globaler Sicherheitszustand** bezeichnet. Ein Sicherheitszustand, der einem DF zugeordnet ist, das nicht das MF ist, wird als **DF-spezifischer Sicherheitszustand** bezeichnet.

Durch den einem DF zugeordneten Sicherheitszustand muß nur für die vier logischen Schlüsselnummern '00' bis '03' **pro Einzelschlüssel** nachgehalten werden, ob eine Authentikation mittels EXTERNAL AUTHENTICATE mit diesem Schlüssel stattgefunden hat. Für die Schlüssel mit Schlüsselnummer > '03' muß durch den dem DF zugeordneten Sicherheitszustand nur **pro Schlüsselgruppe** nachgehalten werden, ob ein EXTERNAL AUTHENTICATE mit einem beliebigen Schlüssel aus der jeweiligen Gruppe stattgefunden hat. Hierbei besteht die ***Schlüsselgruppe 1*** aus den Schlüsseln mit den logischen Schlüsselnummern zwischen '04' und '0E'. Die ***Schlüsselgruppe 2*** umfaßt alle Schlüssel mit Schlüsselnummer > '0E' (vgl. Kapitel 2.8.).

Durch die Einteilung in Gruppen soll es ermöglicht werden, daß zwar in dem EF_KEY eines DF der ec-Karte mehrere oder alle Schlüssel einer Gruppe vorhanden sind, ein Terminal, mit dem die Karte kommuniziert, aber nur über einen dieser Schlüssel verfügen muß, um sich, bezogen auf dieses DF, mit dem Kommando EXTERNAL AUTHENTICATE authentisieren zu können.

Durch den einem DF zugeordneten Sicherheitszustand müssen also maximal 8 Authentikationen nachgehalten werden. Er kann repräsentiert werden durch maximal 8 Flags, die jeweils gesetzt werden, wenn die entsprechende Authentikation erfolgreich abgeschlossen wurde.

Bild 5: Repräsentation des Sicherheitszustandes eines DF

Die Flags des Sicherheitszustandes eines DF werden nur verwendet, wenn die entsprechenden geheimen Informationen in diesem DF vorhanden sind. Flags für die Schlüsselgruppen 1 und/oder 2 müssen verwendet werden, wenn mindestens ein Schlüssel aus der jeweiligen Gruppe in dem EF_KEY dieses DF vorhanden ist.

Die ec-Karte muß mindestens die Sicherheitszustände des aktuellen DF und aller übergeordneten DFs (inklusive MF) gleichzeitig nachhalten können.

Durch ein RESET der ec-Karte werden alle Sicherheitszustände der ec-Karte gelöscht.

Durch das Kommando SELECT FILE mit DF-Name (P1 = '04') werden die Sicherheitszustände des selektierten DF selbst und aller DFs, die Nachfolger-DFs des selektierten DF sind, zurückgesetzt.

4.1.1.1. Sicherheitsrelevante Wirkung des Kommandos VERIFY

Mit dem Kommando VERIFY (vgl. Kapitel 8.) kann sich die externe Welt gegenüber der ec-Karte durch Angabe eines Paßwortes authentisieren. In den Kommando-Parametern wird spezifiziert, ob das in den Kommandodaten übergebene Paßwort

- global oder DF-spezifisch ist und
- die Paßwortnummer 0 oder 1 hat.

Durch den in Kapitel 2.9.2. beschriebenen Mechanismus zur Paßwortauswahl wird mit den Parametern

- "globales/DF-spezifisches Paßwort" aus den Kommandodaten,
- "Paßwortnummer x" aus den Kommandodaten und
- bei Aufruf von VERIFY aktuelles DF

das durch das Kommando referenzierte Paßwort gesucht.

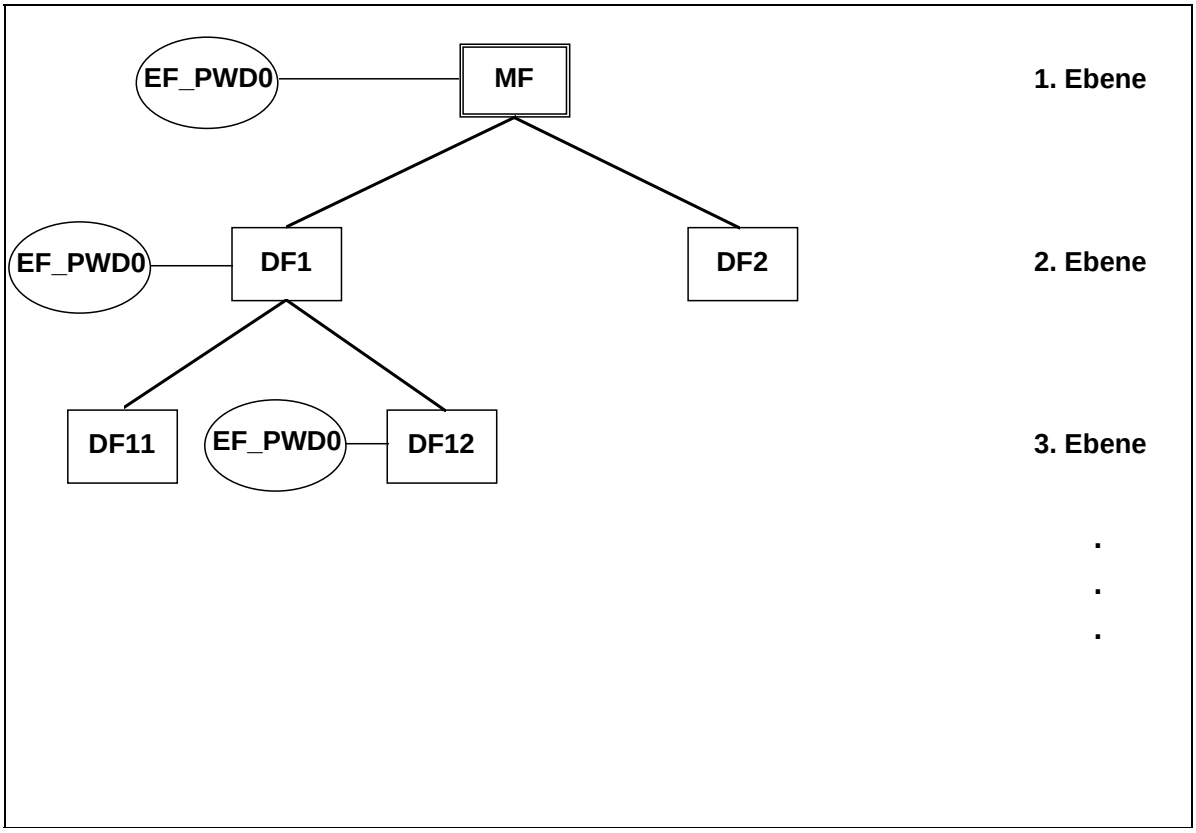
Wenn der Mechanismus zur Paßwortauswahl erfolgreich war, wird das mit dem Kommando übergebene Paßwort mit dem gefundenen Paßwort verglichen.

Verläuft dieser Vergleich positiv, so wird der Sicherheitszustand geändert, der dem MF bzw. DF zugeordnet ist, in dessen EF_PWDx sich das gefundene Paßwort befindet: Das der Paßwortnummer zugeordnete Flag des Sicherheitszustandes wird gesetzt. Der dem Paßwort zugeordnete Fehlbedienungszähler im EF_FBZ bzw. im zugehörigen EF_PWDD1 wird auf den Initialwert gesetzt.

Geht die Prüfung negativ aus, wird das entsprechende Flag des Sicherheitszustandes zurückgesetzt und der dem Paßwort zugeordnete Fehlbedienungszähler um 1 dekrementiert. Der externen Welt wird der aktuelle Stand des Fehlbedienungszählers mitgeteilt.

Durch das Kommando VERIFY wird also nur dann der Sicherheitszustand des bei Aufruf des Kommandos aktuellen DF geändert, wenn dieses ein EF_PWDx mit einem Paßwort enthält.

Im folgenden wird anhand des in Kapitel 2.9. enthaltenen Bild 4 die Wirkung des Kommandos VERIFY tabellarisch erläutert.



Die Einträge der folgenden Tabelle zeigen das DF und das Flag des Sicherheitszustandes, der diesem DF zugeordnet ist, die durch die Ausführung von VERIFY betroffen sind, wenn VERIFY mit den Parametern

- globales Paßwort und
- Paßwortnummer x

aufgerufen wird und

- DF aus der Datei-Struktur in Bild 4 aktuell ist.

aktuelles DF x in Kommando	MF	DF1	DF2	DF11	DF12
0	MF PWD 0	MF PWD 0	MF PWD 0	MF PWD 0	MF PWD 0
1	-	-	-	-	-

Tabelle 5: Durch VERIFY betroffener globaler Sicherheitszustand

Erläuterung:

PWD x Flag des Sicherheitszustandes, das dem Paßwort mit Paßwortnummer x zugeordnet ist.

Die Einträge der folgenden Tabelle zeigen das DF und das Flag des Sicherheitszustandes, der diesem DF zugeordnet ist, die durch die Ausführung von VERIFY betroffen sind, wenn VERIFY mit den Parametern

- DF-spezifisches Paßwort und
- Paßwortnummer x

aufgerufen wird und

- DF aus der Datei-Struktur in Bild 4 aktuell ist.

aktuelles DF x in Kommando	MF	DF1	DF2	DF11	DF12
0	-	DF1 PWD 0	-	DF1 PWD 0	DF12 PWD 0
1	-	-	-	-	-

Tabelle 6: Durch VERIFY betroffener DF-spezifischer Sicherheitszustand

Erläuterung:

PWD x Flag des Sicherheitszustandes, das dem Paßwort mit Paßwortnummer x zugeordnet ist.

4.1.1.2. Sicherheitsrelevante Wirkung des Kommandos EXTERNAL AUTHENTICATE

Mit dem Kommando EXTERNAL AUTHENTICATE (vgl. Kapitel 8.) kann sich die externe Welt gegenüber der ec-Karte authentisieren, indem sie eine unmittelbar zuvor von der ec-Karte mit dem Kommando GET CHALLENGE abgeholte und in der Karte gespeicherte Zufallszahl verschlüsselt

in den Kommandodaten an die Karte übergibt. In den Kommando-Parametern wird spezifiziert,

- ob der verwendete Schlüssel global oder DF-spezifisch ist und
- welche Schlüsselnummer KID der verwendete Schlüssel hat.

Durch den in Kapitel 2.8.2. beschriebenen Mechanismus zur Schlüsselauswahl wird mit den Parametern

- "globaler/DF-spezifischer Schlüssel" aus den Kommandodaten,
- "KID" aus den Kommandodaten und
- bei Aufruf von EXTERNAL AUTHENTICATE aktuelles DF

der durch das Kommando referenzierte Schlüssel gesucht.

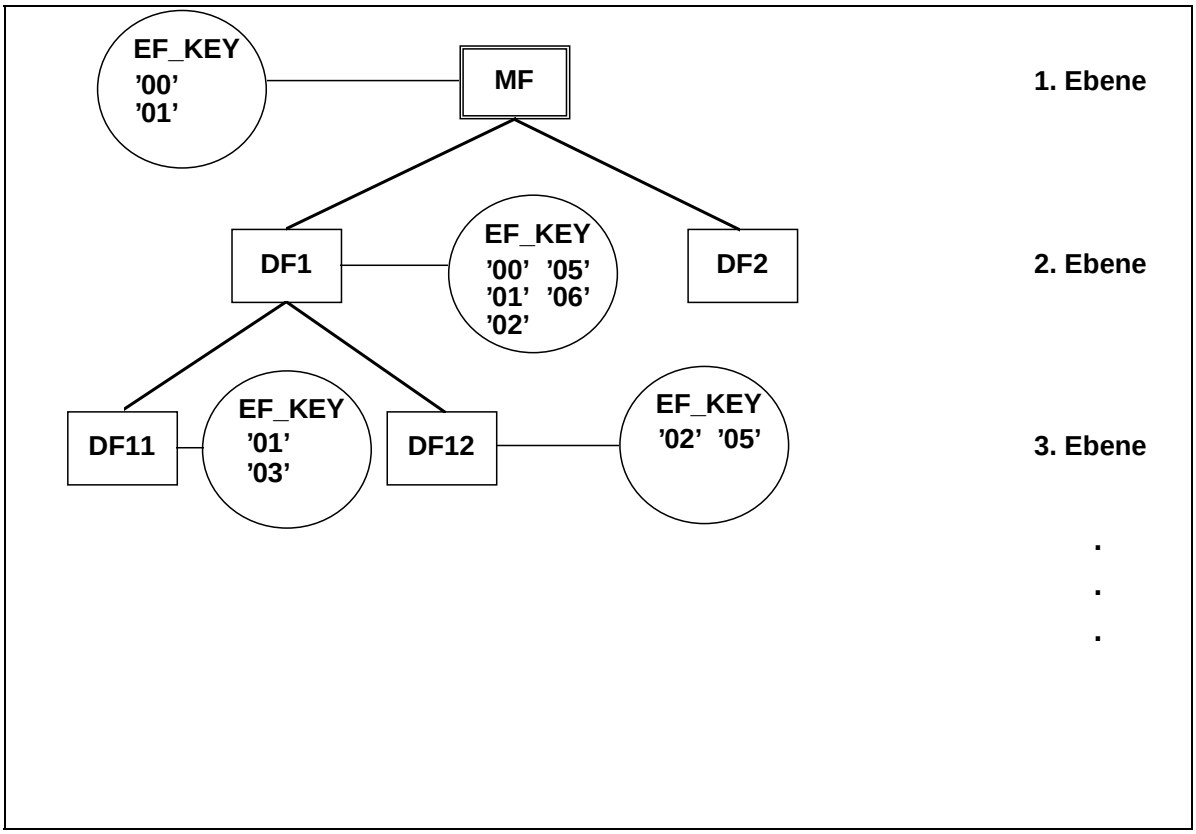
Wenn der Mechanismus zur Schlüsselauswahl erfolgreich war und der für den Schlüssel durch die Algorithmus-ID im zugehörigen EF_KEYD festgelegte Verschlüsselungsalgorithmus durch EXTERNAL AUTHENTICATE verwendet werden darf (vgl. Kapitel 6.3.2.), wird die gespeicherte Zufallszahl unter Verwendung des ICV '00..00' mit diesem Schlüssel verschlüsselt und das Chiffrat mit dem im Kommando übergebenen Chiffrat verglichen.

Verläuft der Vergleich positiv, so wird der Sicherheitszustand geändert, der dem MF bzw. DF zugeordnet ist, in dessen EF_KEY sich der gefundene Schlüssel befindet: Das Flag, das der Schlüsselnummer (Schlüsselnummer zwischen '00' und '03') bzw. der Schlüsselgruppe, zu der die Schlüsselnummer gehört (Schlüsselnummer > '03'), zugeordnet ist, wird gesetzt.

Geht die Prüfung negativ aus, wird das Flag des Sicherheitszustandes zurückgesetzt und der dem Schlüssel zugeordnete Fehlbedienungszähler im zugehörigen EF_KEYD um 1 dekrementiert. Der externen Welt wird der aktuelle Stand des Fehlbedienungszählers mitgeteilt.

Durch das Kommando EXTERNAL AUTHENTICATE wird also nur dann der Sicherheitszustand des bei Aufruf des Kommandos aktuellen DF geändert, wenn dieses ein EF_KEY mit einem Schlüssel mit der angegebenen Schlüsselnummer enthält.

Im folgenden wird anhand des in Kapitel 2.8. enthaltenen Bild 3 die Wirkung des Kommandos EXTERNAL AUTHENTICATE tabellarisch erläutert.



Die Einträge der folgenden Tabelle zeigen das DF und das Flag des Sicherheitszustandes, der diesem DF zugeordnet ist, die durch die Ausführung von EXTERNAL AUTHENTICATE betroffen sind, wenn EXTERNAL AUTHENTICATE mit den Parametern

- globaler Schlüssel und
- Schlüsselnummer KID

aufgerufen wird und

- DF aus der Datei-Struktur in Bild 3 aktuell ist.

aktuelles DF KID in Kommando	MF	DF1	DF2	DF11	DF12
globaler Einzelschlüssel					
'00'	MF KID '00'	MF KID '00'	MF KID '00'	MF KID '00'	MF KID '00'
'01'	MF KID '01'	MF KID '01'	MF KID '01'	MF KID '01'	MF KID '01'
'02'	-	-	-	-	-
'03'	-	-	-	-	-
globale Schlüsselgruppe 1					
'04'	-	-	-	-	-
'05'	-	-	-	-	-
'06'	-	-	-	-	-
'07'-'0E'	-	-	-	-	-
globale Schlüsselgruppe 2					
'0F'-'FF'	-	-	-	-	-

Tabelle 7: Durch EXTERNAL AUTHENTICATE betroffener globaler Sicherheitszustand

Erläuterung:

KID 'XX' Flag des Sicherheitszustandes, das dem Einzelschlüssel mit Schlüsselnummer 'XX' (zwischen '00' und '03') zugeordnet ist.

Die Einträge der folgenden Tabelle zeigen das DF und das Flag des Sicherheitszustandes, der diesem DF zugeordnet ist, die durch die Ausführung von EXTERNAL AUTHENTICATE betroffen sind, wenn EXTERNAL AUTHENTICATE mit den Parametern

- DF-spezifischer Schlüssel und
- Schlüsselnummer KID

aufgerufen wird und

- DF aus der Datei-Struktur in Bild 3 aktuell ist.

aktuelles DF KID in Kommando	MF	DF1	DF2	DF11	DF12
DF-spezifischer Einzelschlüssel					
'00'	-	DF1 KID '00'	-	DF1 KID '00'	DF1 KID '00'
'01'	-	DF1 KID '01'	-	DF11 KID '01'	DF1 KID '01'
'02'	-	DF1 KID '02'	-	DF1 KID '02'	DF12 KID '02'
'03'	-	-	-	DF11 KID '03'	-
DF-spezifische Schlüsselgruppe 1					
'04'	-	-	-	-	-
'05'	-	DF1 SG 1	-	DF1 SG 1	DF12 SG 1
'06'	-	DF1 SG 1	-	DF1 SG 1	-
'07'-'0E'	-	-	-	-	-
DF-spezifische Schlüsselgruppe 2					
'0F'-'FF'	-	-	-	-	-

Tabelle 8: Durch EXTERNAL AUTHENTICATE betroffener DF-spezifischer Sicherheitszustand

Erläuterung:

KID ’XX’ Flag des Sicherheitszustandes, das dem Einzelschlüssel mit Schlüsselnummer 'XX' (zwischen '00' und '03') zugeordnet ist.

SG n Flag des Sicherheitszustandes, das der Schlüsselgruppe n (1 oder 2) zugeordnet ist.

4.1.2. Kommandospezifischer Sicherheitszustand

Der kommandospezifische Sicherheitszustand zeigt an, ob die Prüfung eines MAC über Daten der Kommandonachricht eines Kommandos zu einem positiven Ergebnis geführt hat.

Zur Prüfung des MAC verwendet die ec-Karte den Schlüssel, der durch das Sicherheitsattribut der Datei (vgl. das folgende Kapitel), auf die zugegriffen werden soll, für das auszuführende Kommando bestimmt ist.

Der kommandospezifische Sicherheitszustand ist nur für die Zeit der Ausführung dieses Kommandos gültig. Er ist unabhängig vom globalen und DF-spezifischen Sicherheitszustand.

4.2. Zugriffsbedingungen (ACs)

Beim Anlegen einer Datei der ec-Karte können eine oder mehrere **Zugriffsbedingungen** oder **Access Conditions (AC)** für die Datei festgelegt werden. Für das Kommando oder die Kommandos, die eine für eine Datei festgelegte AC bei ihrer Ausführung auswerten, wird hierdurch bestimmt, unter welcher Bedingung und in welcher Weise das Kommando ausgeführt werden darf. Insbesondere läßt sich mittels ACs der Zugriff von Kommandos auf Dateien dateispezifisch regeln.

ACs, die die Verwendung eines globalen Paßwortes oder eines globalen Schlüssels erfordern, werden als ***globale ACs*** bezeichnet. ACs, die die Verwendung eines DF-spezifischen Paßwortes oder eines DF-spezifischen Schlüssels erfordern, werden als ***DF-spezifische ACs*** bezeichnet.

Im folgenden werden die 10 vordefinierten ***Basis-AC***s der ec-Karte erläutert. Die folgende Beschreibung bezieht sich immer auf ein Paar bestehend aus einer Datei, für die die jeweilige AC festgelegt ist, und einem Kommando, das vor seinem Zugriff auf die Datei die festgelegte AC auswertet.

4.2.1. ALW

Der Zugriff des Kommandos auf die Datei ist immer erlaubt.

4.2.2. NEV

Der Zugriff des Kommandos auf die Datei ist nie erlaubt.

4.2.3. Typ PWD

Durch eine AC vom Typ PWD wird festgelegt, daß der Zugriff des Kommandos auf die Datei nur erlaubt ist, wenn zuvor eine Authentikation der externen Welt durch Angabe eines Paßwortes mittels des Kommandos VERIFY stattgefunden hat.

Durch eine AC vom Typ PWD wird festgelegt, ob das für die Authentikation zu verwendende Paßwort

- global oder DF-spezifisch ist und
- die Paßwortnummer 0 oder 1 hat.

Wenn ein **globales Paßwort** mit der Paßwortnummer x festgelegt wird, wird die AC als **PWD_G x** notiert. Wenn ein **DF-spezifisches Paßwort** mit der Paßwortnummer x festgelegt wird, wird die AC als **PWD_D x** notiert.

Durch den in Kapitel 2.9. beschriebenen Such-Algorithmus zur Paßwortauswahl wird mit den

Parametern

- "globales/DF-spezifisches Paßwort" aus der AC,
- "Paßwortnummer x" aus der AC und
- Datei, für die die AC festgelegt ist,

durch die AC und die Datei ein Paßwort eindeutig bestimmt.

Wird durch den Such-Algorithmus kein Paßwort bestimmt, ist das Kommando zurückzuweisen.

Vor der Ausführung des Kommandos wird geprüft, ob die AC für die Datei ***erfüllt*** ist, d. h. ob das der Paßwortnummer x zugeordnete Flag des Sicherheitszustandes, der dem MF bzw. DF zugeordnet ist, in dessen EF_PWDx das durch die AC und den Such-Algorithmus bestimmte Paßwort enthalten ist, anzeigt, daß die Authentikation stattgefunden hat.

Ist das nicht der Fall, wird das Kommando abgelehnt.

Es ist zu beachten, daß ein zu verwendendes DF-spezifische Paßwort durch eine AC und die Datei in Abhängigkeit von der Position der Datei in der Datei-Struktur der ec-Karte festgelegt ist aber unabhängig davon definiert ist, welches DF zum Zeitpunkt der Ausführung des Kommandos aktuell ist.

Ein EF, dem für ein Kommando PWD_D x als AC zugewiesen ist, sollte nur dann als AEF in eine Applikation eingebunden werden, wenn der Such-Algorithmus aus Kapitel 2.9. für das EF und die Paßwortnummer x und für das ADF der Applikation und die Paßwortnummer x dasselbe DF-spezifische Paßwort findet. Nur wenn dies der Fall ist, ist es möglich, mittels des Kommandos VERIFY die AC für das EF zu erfüllen, wenn das ADF aktuelles DF ist.

4.2.4. Durch ACs referenzierte Schlüssel

Die im folgenden beschriebenen ACs erfordern die Verwendung eines Schlüssels. Durch die AC wird festgelegt, ob der zu verwendende Schlüssel

- global oder DF-spezifisch ist und
- welche logische Schlüsselnummer KID er hat.

Wenn die **logische Schlüsselnummer KID zwischen '00' und '03'** liegt, muß der Einzelschlüssel mit dieser Nummer verwendet werden.

Wenn die **logische Schlüsselnummer KID > '03'** ist, wird nicht die Verwendung des Einzelschlüssels mit dieser Nummer, sondern die Verwendung eines Schlüssels aus einer der beiden Schlüsselgruppen festgelegt. Hierbei legt die Nummer '04' die Verwendung eines Schlüssels aus der Schlüsselgruppe 1 (Schlüsselnummern zwischen '04' und '0E'), die Nummer '0F' die Verwendung eines Schlüssels aus der Schlüsselgruppe 2 (Schlüsselnummer > '0E') fest.

Durch den in Kapitel 2.8. beschriebenen Such-Algorithmus zur Schlüsselauswahl wird mit den Parametern

- "globaler/DF-spezifischer Schlüssel" aus der AC,
- "Schlüsselnummer KID" ('00' bis '03', '04' und '0F') aus der AC und
- Datei, für die die AC festgelegt ist,

durch die AC und die Datei ein Schlüssel (KID zwischen '00' und '03') bzw. eine Schlüsselgruppe (KID > '03') eindeutig bestimmt.

Wird durch den Such-Algorithmus kein Schlüssel bzw. keine Schlüsselgruppe gefunden, ist das entsprechende Kommando zurückzuweisen.

Es ist zu beachten, daß der zu verwendende Schlüssel bzw. die zu verwendende Schlüsselgruppe durch eine AC und die Datei in Abhängigkeit von der Position der Datei in der Datei-Struktur der ec-Karte festgelegt ist, aber unabhängig davon definiert ist, welches DF zum Zeitpunkt der Ausführung des Kommandos aktuell ist.

4.2.5. Typ PRO und ENC

Allgemeine Beschreibung

Die ACs vom Typ PRO und ENC erfordern für den Zugriff des Kommandos auf die Datei MAC-Bildung zum Nachweis der Integrität (AC vom Typ **PRO**) bzw. MAC-Bildung und Verschlüsselung zum Nachweis der Integrität und zur Sicherstellung der Vertraulichkeit (AC vom Typ **ENC**).

Durch diese ACs wird nicht festgelegt, welche Daten der Kommandonachricht und/oder der Antwortnachricht abzusichern sind. Dies wird erst im Rahmen der Spezifikation des jeweiligen Kommandos definiert.

Nur wenn sich die AC vom Typ PRO oder ENC auf die Kommandonachricht beziehen, handelt es sich um Zugriffsbedingungen im eigentlichen Sinn, da sich dann die externe Welt dadurch authentisieren muß, daß sie die Kommandonachricht mit einem korrekten MAC versieht. In diesem Fall ist die Ausführung des Kommandos von dem kommandospezifischen Sicherheitszustand abhängig.

Der bei einer AC vom Typ PRO für die MAC-Bildung bzw. bei einer AC vom Typ ENC für die MAC-Bildung und Verschlüsselung zu verwendende Schlüssel bzw. die entsprechende Schlüsselgruppe wird durch die AC festgelegt.

Ist ein globaler Schlüssel mit Schlüsselnummer KID zu verwenden, wird die AC als **PRO_G KID bzw. ENC_G KID** notiert. Ist ein DF-spezifischer Schlüssel mit Schlüsselnummer KID zu

verwenden, wird die AC als **PRO_D KID bzw. ENC_D KID** notiert. In beiden Fällen kann KID die Werte '00' bis '03' zur Referenzierung von Einzelschlüsseln, '04' zur Referenzierung der Schlüsselgruppe 1 oder '0F' zur Referenzierung der Schlüsselgruppe 2 haben.

Wenn durch eine AC vom Typ PRO oder ENC für eine Datei und ein Kommando die Verwendung eines Schlüssels aus einer Schlüsselgruppe festgelegt wird, muß die tatsächlich verwendete Schlüsselnummer KID' implizit bestimmt oder in der Kommando- bzw. Antwortnachricht enthalten sein. KID' darf dann nicht verschlüsselt sein.

Wenn KID' in der Kommandonachricht enthalten ist, muß durch die Karte vor der Schlüsselauswahl geprüft werden, ob der Schlüssel mit der Nummer KID' zu der durch die AC festgelegten Schlüsselgruppe gehört. Ist das der Fall, muß mittels des in Kapitel 2.8. beschriebenen Mechanismus zur Schlüsselauswahl zur MAC-Bildung bzw. MAC-Bildung und Verschlüsselung der Schlüssel mit den folgenden Parametern ausgewählt werden:

- "globaler/DF-spezifischer Schlüssel" aus der AC,
- "Schlüsselnummer KID' " und
- Datei, für die die AC vom Typ PRO oder ENC festgelegt ist.

Wenn der zu verwendende Schlüssel gefunden wird, führt die ec-Karte alle weiteren Prüfschritte des in Kapitel 2.8.2. beschriebenen Mechanismus zur Schlüsselauswahl durch. Werden alle diese Prüfschritte erfolgreich durchlaufen, aber ist für den eine Schlüssel eine Algorithmus-ID festgelegt, die besagt, daß der Schlüssel nicht zur MAC-Bildung bzw. MAC-Bildung und Verschlüsselung für das Kommando verwendet werden darf, wird das Kommando abgebrochen.

Beschreibung für die Standard- und Administrationskommandos

Für die in Kapitel 8. spezifizierten Standard- und Administrationskommandos der ec-Karte legen die ACs vom Typ PRO und ENC fest, daß Secure Messaging gemäß Kapitel 3. zum Nachweis der Integrität (AC vom Typ **PRO**) bzw. zum Nachweis der Integrität und zur Sicherstellung der Vertraulichkeit (AC vom Typ **ENC**) von Nachrichten zu verwenden ist.

Eine AC vom Typ **PRO** bedeutet für Standard- und Administrationskommandos, daß die Kommando- oder Antwortnachricht des Kommandos bei dem Zugriff auf die Datei, wie in Kapitel 3.1. beschrieben, durch einen CFB-MAC abgesichert wird.

Eine AC vom Typ **ENC** bedeutet für Standard- und Administrationskommandos, daß die Kommando- oder Antwortnachricht des Kommandos bei Zugriff auf die Datei, wie in Kapitel 3.2. beschrieben, durch einen CFB-MAC und CBC-Verschlüsselung mit demselben Schlüssel abgesichert wird.

In Kapitel 8. wird festgelegt, ob die Kommandonachricht oder die Antwortnachricht durch Secure Messaging abzusichern ist.

Wenn die durch eine AC vom Typ PRO oder ENC für ein Standard- oder Administrationskommando festgelegte Schlüsselnummer KID eine Schlüsselgruppe referenziert, wird das Kommando abgewiesen, da in Kommando- und Antwortnachricht kein Datenfeld zum

Einstellen der tatsächlich verwendeten Schlüsselnummer KID' vorhanden ist.

4.2.6. Typ AUT

Durch eine AC vom Typ AUT wird festgelegt, daß der Zugriff des Kommandos auf die Datei nur erlaubt ist, wenn zuvor eine Authentikation der externen Welt mittels des Kommandos EXTERNAL AUTHENTICATE unter Verwendung eines Schlüssels stattgefunden hat.

Der zu verwendende Schlüssel bzw. die entsprechende Schlüsselgruppe wird durch die AC festgelegt.

Ist ein globaler Schlüssel mit Schlüsselnummer KID zu verwenden, wird die AC als **AUT_G KID** notiert. Ist ein DF-spezifischer Schlüssel mit Schlüsselnummer KID zu verwenden, wird die AC als **AUT_D KID** notiert. Hierbei kann KID die Werte '00' bis '03' zur Referenzierung von Einzelschlüsseln, '04' zur Referenzierung der Schlüsselgruppe 1 oder '0F' zur Referenzierung der Schlüsselgruppe 2 haben.

Vor dem Zugriff des Kommandos auf die Datei, für die eine AC vom Typ AUT festgelegt ist, wird geprüft, ob die AC erfüllt ist, d. h. ob das der Schlüsselnummer bzw. Schlüsselgruppe zugeordnete Flag des Sicherheitszustandes, der dem MF bzw. DF zugeordnet ist, in dessen EF_KEY der durch die AC und den Such-Algorithmus bestimmte Schlüssel bzw. die Schlüsselgruppe enthalten ist, anzeigt, daß die Authentikation stattgefunden hat. Ist das nicht der Fall, wird das Kommando abgelehnt. Hierbei ist das der Schlüsselnummer '04' zugeordnete Flag das der Schlüsselgruppe 1 und das der Schlüsselnummer '0F' zugeordnete Flag das der Schlüsselgruppe 2.

Ein EF, dem für ein Kommando AUT_D KID als AC zugewiesen ist, sollte nur dann als AEF in eine Applikation eingebunden werden, wenn der Such-Algorithmus aus Kapitel 2.8. für das EF und die Schlüsselnummer bzw. Schlüsselgruppe und für das ADF der Applikation und die Schlüsselnummer bzw. Schlüsselgruppe denselben DF-spezifischen Schlüssel bzw. dieselbe DF-spezifische Schlüsselgruppe findet. Nur wenn dies der Fall ist, ist es möglich, mittels des Kommandos EXTERNAL AUTHENTICATE die AC für das EF zu erfüllen, wenn das ADF aktuelles DF ist.

4.2.7. Kombinierte ACs

Kombinierte ACs sind aus jeweils zwei Basis-ACs zusammengesetzte ACs. Ist für ein Kommando und für eine Datei eine kombinierte AC festgelegt, darf das Kommando nur ausgeführt werden, wenn beide Basis-ACs erfüllt sind.

Wenn eine kombinierte AC aus ALW und einer weiteren Basis-AC AC1 zusammengesetzt ist, darf das entsprechende Kommando auf die Datei nur zugreifen, wenn AC1 erfüllt ist.

Wenn eine kombinierte AC aus NEV und einer weiteren Basis-AC zusammengesetzt ist, darf das entsprechende Kommando auf die Datei nie zugreifen.

Wenn für ein Standard- oder Administrationskommando und eine Datei eine AC festgelegt ist, die aus zwei ACs vom Typ PRO oder zwei ACs vom Typ ENC oder einer AC vom Typ PRO und einer AC vom Typ ENC zusammengesetzt ist, können diese beiden Basis-ACs im allgemeinen nicht gleichzeitig erfüllt werden (vgl. Kapitel 4.2.5.). Daher darf für ein Standard- oder Administrationskommando eine solche kombinierte AC nicht festgelegt werden. Andernfalls muß das Kommando abgewiesen werden.

4.2.8. Kodierung von ACs

Die Basis-ACs werden in ein Byte kodiert. Die Kodierung des **linken Halbbyte** ist der folgenden Tabelle zu entnehmen.

AC	Wert (hex.)
ALW	’0’
RFU	’1’
PWD_G	’2’
PWD_D	’3’
PRO_G	’4’
PRO_D	’5’
ENC_G	’6’
ENC_D	’7’
AUT_G	’8’
AUT_D	’9’
RFU	’A’
RFU	’B’
RFU	’C’
RFU	’D’
RFU	’E’
NEV	’F’

Tabelle 9: Kodierung des linken Halbbyte einer AC

Das rechte Halbbyte

- gibt für ACs vom Typ PRO, ENC und AUT die logische Schlüsselnummer des zu verwendenden Schlüssels an, wobei die Nummern '0', '1', '2', '3', '4' und 'F' verwendet werden,

- gibt für ACs vom Typ PWD die Paßwortnummer an, wobei die Nummern '0' und '1'

verwendet werden, und

- ist für die ACs ALW und NEV zu '0' zu setzen.

Kombinierte ACs werden in zwei Byte kodiert, wobei in jeweils ein Byte eine Basis-AC, wie oben beschrieben, kodiert ist.

4.3. Festlegung von Sicherheitsattributen

Für die **Standardkommandos** der ec-Karte sind folgende **impliziten** Festlegungen für den Zugriff auf **DF**s getroffen:

- SELECT FILE darf auf ein DF zugreifen;
- alle übrigen Standardkommandos dürfen **nie** auf ein DF zugreifen.

Für das **Administrationskommando LOAD COMMAND** wird **implizit** die AC PRO_G '00' festgelegt, das heißt, daß das Administrationskommando nur ausgeführt werden darf, wenn die Integrität der Kommandonachricht durch Mitsenden eines korrekten MAC nachgewiesen wird.

Der MAC wird gemäß Kapitel 3.1.1. gebildet und geprüft. Der zu verwendende Schlüssel ist der Schlüssel mit der logischen Schlüsselnummer '00' aus dem EF_KEY des MF (globaler Schlüssel K_{Card}).

Wenn der Zugriff von Kommandos auf eine Datei der ec-Karte **explizit** durch ACs geregelt werden soll, müssen diese ACs beim Anlegen der Datei festgelegt werden.

Explizit festgelegte ACs bleiben bis zum Löschen einer Datei gültig und sind nicht änderbar.

Ist für eine Datei und ein Kommando keine AC festgelegt, obwohl das gemäß der Kommandospezifikation erforderlich wäre, muß das Kommando abgewiesen werden.

Für die **Standardkommandos** sind die folgenden Festlegungen für den Zugriff auf EFs getroffen:

- Für das Kommando READ RECORD muß für jedes EF eine AC festgelegt werden.
- Für das Kommando UPDATE RECORD muß für jedes EF eine AC festgelegt werden.
- Für das Kommando VERIFY muß für jedes EF_PWDx eine AC festgelegt werden.
- Für die übrigen Standardkommandos und EFs kann keine AC festgelegt werden.

Die **Administrationskommandos**

- CREATE FILE,
- DELETE FILE,
- INCLUDE,
- EXCLUDE und
- APPEND RECORD

werden zu der Gruppe **ADMIN** zusammengefaßt. Für diese Gruppe ist für jede Datei genau eine AC festzulegen.

Für jedes **DF** muß eine AC für die Gruppe ADMIN festgelegt werden. Diese regelt den Zugriff der Kommandos

- CREATE FILE,
- DELETE FILE,
- INCLUDE und
- EXCLUDE

auf Dateien, wenn dieses DF selektiert ist. Insbesondere wird durch diese AC der Zugriff von DELETE FILE auf alle in diesem DF enthaltenen Dateien bestimmt.

Für jedes **EF** muß ebenfalls eine AC für die Gruppe ADMIN festgelegt werden. Diese regelt den Zugriff des Kommandos

- APPEND RECORD

auf dieses EF.

Für **Ergänzungskommandos** sind die folgenden Festlegungen für den Zugriff auf Dateien getroffen:

- Für Ergänzungskommandos kann für jede Datei beim Anlegen der Datei eine AC definiert werden.
- Durch die Implementierung des Kommandos wird bestimmt, wie festgelegte ACs ausgewertet werden und welche Konsequenzen die Auswertung für den Ablauf des Kommandos hat.
- Ist für ein Ergänzungskommando und eine Datei keine AC festgelegt, so müssen die implizit festgelegten Sicherheitsattribute, die den Zugriff des Kommandos auf die Datei regeln, geeignet sein, um die Sicherheit der Daten der Datei zu gewährleisten.

5. Datei-Kontrollinformation

Zu jeder Datei gibt die ec-Karte ***Datei-Kontrollinformation*** als Antwort aus, wenn das Kommando SELECT FILE auf die Datei zugreift. Die Datei-Kontrollinformation zu einer Datei besteht aus Daten, die die Datei identifizieren sowie Struktur und Zugriffsbedingungen der Datei festlegen.

Hierbei handelt es sich um Daten, die beim Anlegen der Datei bzw. für ADFs auch beim Aufbau von Applikationen mittels entsprechender Administrationskommandos oder im Rahmen der Initialisierung an die ec-Karte übergeben werden.

Zusätzlich gibt die Datei-Kontrollinformation zu DFs an, wieviel Speicherplatz in der ec-Karte noch zum Anlegen weiterer Dateien zur Verfügung steht.

Datei-Kontrollinformation wird immer in Form eines zusammengesetztes BER-TLV-Datenobjekts ausgegeben. Die verwendeten BER-TLV-Datenobjekte sind gemäß ISO 7816-4, Anhang D kodiert. Es stehen die drei BER-TLV-Datenobjekte FCP, FMD und FCI gemäß ISO 7816-4, Kapitel 5.1.5 zur Ausgabe von Datei-Kontrollinformation zur Verfügung.

5.1. File Control Parameters (FCP)

Tag	**Länge** (in Byte)	**Wert**
'62'	var.	FCP

Die FCP setzen sich in Abhängigkeit von der Art der Datei aus einer Auswahl von einfachen BER-TLV-Datenobjekten aus der folgenden Liste zusammen. Die Länge der FCP ist die Gesamtlänge dieser BER-TLV-Datenobjekte.

Tag	**Länge** (in Byte)	**Wert**	**anzuwenden auf**
'81'	'02'	freier Speicherplatz in der ec-Karte in Byte	DFs
		für die Nutzdaten allokierter Speicherplatz in Byte	EFs
'82'	'01'	Datei-Deskriptor für DFs	DFs
	'03'	Datei-Deskriptor für EFs	EFs
'83'	'02'	Datei-ID	alle Dateien
'84'	'01' bis '10'	DF-Name (AID für ADFs)	DFs
'86'	var.	ACs	alle Dateien

Die FCP einer Datei haben also folgenden Aufbau:

Tag	**Länge** (in Byte)	**Wert**
'62'	var.	
'81'	'02'	freier Speicherplatz in der ec-Karte in Byte (DFs)
		für die Nutzdaten allokierter Speicherplatz in Byte (EFs)
'82'	'01'	Datei-Deskriptor für DFs
	'03'	Datei-Deskriptor für EFs
'83'	'02'	Datei-ID
'84'	'01' bis '10'	DF-Name (AID)
'86'	var.	ACs

Die Daten der FCP außer der Information über den freien Speicherplatz in der ec-Karte (Tag '81' für DFs) werden der Karte bei dem Anlegen der Datei mit dem Kommando CREATE FILE übergeben oder in die Karte bei der Initialisierung eingebracht. In welcher Form die ec-Karte diese Daten speichert und auf welche Weise sie die Information in den FCP bei einem SELECT FILE der

Datei bereitstellt, wird durch diese Spezifikation nicht festgelegt.

5.2. File Management Data (FMD)

Tag	Länge (in Byte)	Wert
’64’	var.	FMD

Wenn die Datei ein EF ist oder wenn es sich um ein DF handelt, das kein ADF ist, bestehen die FMD aus den zwei Byte ’64 00’.

Wenn die Datei ein ADF ist, informieren die FMD die externe Welt darüber, welche EFs diesem DF als AEFs zugeordnet sind. In diesem Fall enthalten die FMD jeweils einmal pro AEF der Applikation das einfache BER-TLV-Datenobjekt mit der folgenden Struktur:

Tag	Länge (in Byte)	Wert	anzuwenden auf
’85’	var.	SFI des AEF mit Pfad des AEF	ADFs

Die Länge der FMD für ein ADF ist die Gesamtlänge aller dieser einfachen BER-TLV-Datenobjekte.

Die FMD für ein ADF haben also folgenden Aufbau:

Tag	Länge (in Byte)	Wert
’64’	L1+...+Ln+2*n	
’85’	L1	SFI mit Pfad
...	...	...
’85’	Ln	SFI mit Pfad

Die Zuordnung zwischen SFI und Pfad wird mit dem Kommando INCLUDE oder bei der Initialisierung hergestellt. In welcher Form die ec-Karte diese Information speichert und auf welche Weise sie die Information in den FMD bei einem SELECT FILE auf das ADF bereitstellt, wird durch diese Spezifikation nicht festgelegt.

Die Ausgabe der FMD eines ADF muß auch möglich sein, wenn ein zugeordnetes AEF gelöscht oder gelöscht und durch eine andere Datei mit gleicher Datei-ID ersetzt wird, ohne daß die Zuordnung zum ADF aufgehoben wird.

5.3. File Control Information (FCI)

Tag	**Länge** (in Byte)	**Wert**
'6F'	var.	FCI

Wenn die Datei ein EF ist oder wenn es sich um ein DF handelt, das kein ADF ist, enthält die FCI dieselben einfachen BER-TLV-Datenobjekte wie die FCP der Datei. Sie ist also bis auf das Tag '6F' identisch mit den FCP.

Wenn die Datei ein ADF ist, enthält die FCI die einfachen BER-TLV-Datenobjekte aus den FCP des ADF und zusätzlich ein zusammengesetztes BER-TLV-Datenobjekt mit der folgenden Struktur:

Tag	**Länge** (in Byte)	**Wert**	**anzuwenden auf**
'A5'	var.	Informationen über die ACs der AEFs der Applikation	ADFs

Dieses zusammengesetzte BER-TLV-Datenobjekt informiert die externe Welt darüber, welche ACs für die AEFs der Applikation gesetzt sind. Jeweils einmal pro AEF der Applikation enthält dieses Datenobjekt das einfache BER-TLV-Datenobjekt mit dem folgenden Aufbau:

Tag	Länge (in Byte)	Wert	anzuwenden auf
’86’	var.	SFI des AEF mit ACs des AEFs	ADFs

Die Länge der FCI für ein ADF ist die Gesamtlänge aller dieser BER-TLV-Datenobjekte.

Die FCI eines ADF sieht also folgendermaßen aus:

Tag	Länge (in Byte)	Wert
'6F'	var.	
'81'	'02'	freier Speicherplatz in der ec-Karte in Byte
'82'	'01'	'38'
'83'	'02'	Datei-ID des ADF
'84'	'01' bis '10'	DF-Name (AID) des ADF
'86'	'02'	AC des ADF
'A5'	var.	
'86'	var.	SFI mit AC
...	...	...
'86'	var.	SFI mit AC

Die Daten der FCI in den BER-TLV-Datenobjekten mit Tag '81' bis '84' und Tag '86' (außer der Information über den freien Speicherplatz in der ec-Karte in Tag '81' für DFs) werden der Karte bei dem Anlegen des DF mit dem Kommando CREATE FILE übergeben oder in die Karte bei der Initialisierung eingebracht. In welcher Form die ec-Karte diese Daten speichert und auf welche Weise sie die Information in der FCI bei einem SELECT FILE auf das DF bereitstellt, wird durch diese Spezifikation nicht festgelegt.

Die ACs der AEFs der Applikation wurden bei Anlegen oder Initialisieren der einzelnen EFs festgelegt. Es wird ebenfalls nicht festgelegt, wie diese ACs zu speichern sind. Die Zuordnung

zwischen SFI und AEF wird mit dem Kommando INCLUDE oder bei der Initialisierung hergestellt. Auf welche Weise die ec-Karte die ACs der zugeordneten AEFs bei einem SELECT FILE auf das ADF bereitstellt, wird durch diese Spezifikation nicht festgelegt.

Die Ausgabe der FCI eines ADF muß auch möglich sein, wenn ein zugeordnetes AEF gelöscht oder gelöscht und durch eine andere Datei mit gleicher Datei-ID ersetzt wird, ohne daß die Zuordnung zum ADF aufgehoben wird.

5.4. Einfache Datenobjekte der Datei-Kontrollinformation

Im folgenden werden die verwendeten einfachen BER-TLV-Datenobjekte näher erläutert.

Tag ’81’ Speicherplatz

Für DFs wird in Byte angegeben, wieviel freier Speicherplatz in der gesamten ec-Karte noch zum Anlegen von Dateien zur Verfügung steht.

Für EFs wird der für die Nutzdaten des EF allokierte Speicherplatz in Byte angegeben. Der Wert ist immer binär kodiert.

Dieses Datenobjekt muß für jede Datei in den FCP und der FCI als Antwort auf SELECT FILE vorhanden sein.

Tag ’82’ Datei-Deskriptor

Der Datei-Deskriptor für DFs der ec-Karte ist stets ein Byte lang und hat den Wert '38'.

Der Datei-Deskriptor für EFs der ec-Karte ist stets 3 Byte lang und kann die folgenden Werte annehmen:

- '02 41 XX' für lineare EFs,
- '06 41 XX' für zyklische EFs.

Das Byte 'XX' gibt die Länge der Records des EFs an. Es ist binär kodiert.

Dieses Datenobjekt muß für jede Datei in den FCP und der FCI vorhanden sein.

Tag ’83’ Datei-ID

Die Datei-ID muß gemäß den Regeln aus Kapitel 1.2.1. vergeben werden. Sie ist binär kodiert.

Dieses Datenobjekt muß für jede Datei in den FCP und der FCI vorhanden sein.

Tag ’84’ DF-Name

Jedes DF der ec-Karte muß einen DF-Namen erhalten, der gemäß Kapitel 1.2.2. zu vergeben ist. Er ist binär kodiert. Handelt es sich bei dem DF-Namen um eine AID, muß der DF-Name mindestens 5 Byte lang sein.

Dieses Datenobjekt muß für jedes DF in den FCP und der FCI vorhanden sein.

Tag ’86’ ACs

Die für DFs festgelegten ACs werden wie folgt in das Wert-Feld kodiert:

AC für ADMIN (2 Byte)	CLA,INS,AC für weiteres Kommando (4 Byte)	CLA,INS,AC für weiteres Kommando (4 Byte)	....

Die für EFs außer EF_PWDx festgelegten ACs werden wie folgt in das Wert-Feld kodiert:

AC für ADMIN (2 Byte)	AC für READ RECORD (2 Byte)	AC für UPDATE RECORD (2 Byte)	CLA,INS,AC für weiteres Kommando (4 Byte)	CLA,INS,AC für weiteres Kommando (4 Byte)	....

Die für ein EF_PWDx festgelegten ACs werden wie folgt in das Wert-Feld kodiert:

AC für ADMIN (2 Byte)	AC für READ RECORD (2 Byte)	AC für UPDATE RECORD (2 Byte)	AC für VERIFY (2 Byte)	CLA,INS,AC für weiteres Kommando (4 Byte)	....

Jede AC wird gemäß Kapitel 4.2.8. in 2 Byte binär kodiert, so daß immer kombinierte ACs darzustellen sind. Eine Basis-AC kann in 2 Byte kodiert werden, indem die Basis-AC in ein Byte kodiert wird und das andere Byte zu '00' (ALW) gesetzt wird.

Für jede Datei muß dieses Datenobjekt in den FCP oder der FCI enthalten sein.

Die ersten 2 Byte mit der AC für ADMIN müssen für jedes **DF** in diesem Datenobjekt enthalten sein.

Die ersten 6 Byte mit den ACs für ADMIN, READ RECORD und UPDATE RECORD müssen für jedes **EF** in diesem Datenobjekt enthalten sein.

Für jedes **EF_PWDx** müssen mindestens 8 Byte in diesem Datenobjekt enthalten sein: 6 Byte mit den ACs für ADMIN, READ RECORD und UPDATE RECORD und 2 Byte mit der AC für VERIFY.

Folgen weitere Byte, so geben sie die ACs für weitere Kommandos an. Hierbei werden für jedes Kommando 4 Byte benötigt. Das Kommando wird durch die 2 Byte binär kodierten CLA,INS eindeutig identifiziert. Es folgt die festgelegte AC, die ebenfalls, wie oben beschrieben, in 2 Byte kodiert wird.

Am Längen-Byte des Datenobjektes kann abgelesen werden, für wieviele Kommandos ACs festgelegt sind.

Wenn in diesem Datenobjekt an verschiedenen Stellen für dasselbe Kommando ACs festgelegt sind, ist die erste von links maßgeblich.

Die Standard- und Administrationskommandos werten zuerst das linke Byte und anschließend das rechte Byte der kodierten AC aus. Bei der Kodierung der ACs für Standard- und Administrationskommandos sind die folgenden Regeln zu beachten:

- Eine AC vom Typ PRO oder ENC darf keine Schlüsselgruppe referenzieren.
- Kombinationen vom Typ PRO,PRO oder ENC,ENC oder PRO,ENC oder ENC,PRO sind nicht erlaubt.

Die Regeln für die Kodierung der ACs für Ergänzungskommandos sind den entsprechenden Spezifikationen zu entnehmen.

Tag ’85’ SFI eines AEF mit Pfad

Dieses Datenobjekt muß einmal pro AEF der Applikation in den FMD des ADF der Applikation vorhanden sein. In das erste Byte des Wert-Feldes ist die SFI des entsprechenden AEFs wie folgt binär kodiert:

SFI-Kodierung im ersten Byte: 000xxxxx

In den weiteren Byte des Wert-Feldes ist der absolute Pfad des AEF gemäß ISO 7816-4, Kapitel 5.1.1 ohne Datei-ID des MF kodiert.

Tag ’86’ SFI eines AEF mit ACs

Dieses Datenobjekt muß einmal pro AEF der Applikation in dem zusammengesetzten Datenobjekt mit Tag 'A5' in der FCI des ADF der Applikation vorhanden sein. In das erste Byte des Wert-Feldes ist die SFI des entsprechenden AEFs wie folgt binär kodiert:

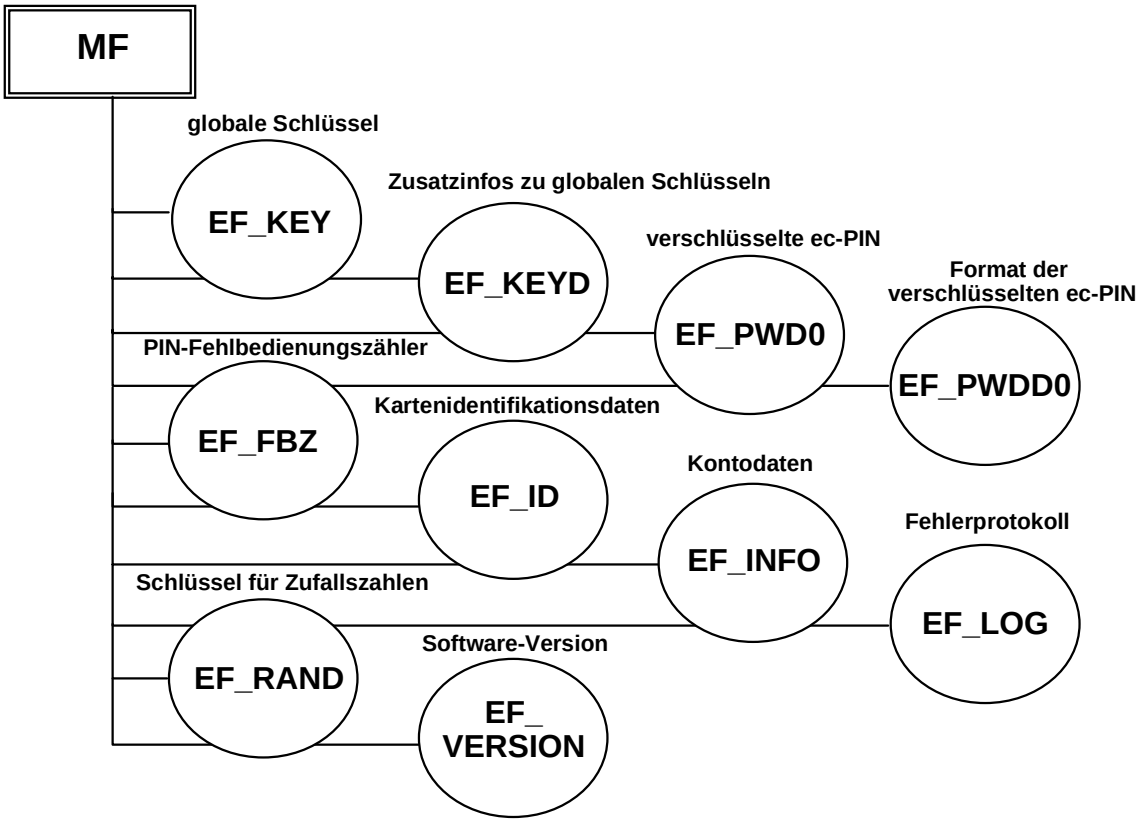
SFI-Kodierung im ersten Byte: 000xxxxx

In den weiteren Byte des Wert-Feldes sind die ACs des entsprechenden AEF kodiert. Die Kodierung ist dieselbe wie die oben für das entsprechende Datenobjekt mit Tag '86' beschriebene.

6. Aufbau, Inhalt und Datei-Kontrollinformation spezieller Dateien

In diesem Kapitel werden FCP, Struktur und Inhalt des MF und der speziellen EFs der ec-Karte beschrieben. Spezielle EFs sind solche, die allgemeine Daten über die ec-Karte und ihre Struktur enthalten bzw. einen gesicherten Zugriff auf die Karte ermöglichen.

Die folgende Grafik gibt einen Überblick über die in diesem Dokument spezifizierten Dateien im MF der ec-Karte.


MF
globale Schlüssel
EF_KEY
Zusatzinfos zu globalen Schlüsseln
EF_KEYD
verschlüsselte ec-PIN
EF_PWD0
Format der verschlüsselten ec-PIN
EF_PWDD0
PIN-Fehlbedienungszähler
EF_FBZ
Kartenidentifikationsdaten
EF_ID
Kontodaten
EF_INFO
Fehlerprotokoll
EF_LOG
Schlüssel für Zufallszahlen
EF_RAND
Software-Version
EF_ VERSION

6.1. MF

Das Anlegen des MF erfolgt mit einer Initialisierungsroutine. Hierbei sind für das MF der ec-Karte die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’11’		
’82’	’01’	’38’	Datei-Deskriptor für DF
'83'	'02'	'3F 00'	Datei-ID des MF
'84'	'04'	'52 4F 4F 54'	DF-Name des MF
'86'	'02'	'00 40'	AC für das MF

Wenn das MF mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FCP ausgegeben:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		
’81’	’02’	’XX XX’	freier Speicherplatz in der ec-Karte in Byte
’82’	’01’	’38’	Datei-Deskriptor für DF
'83'	'02'	'3F 00'	Datei-ID des MF
'84'	'04'	'52 4F 4F 54'	DF-Name des MF
'86'	'02'	'00 40'	AC für das MF

Tag ’81’

In diesen beiden Byte wird der gesamte in der ec-Karte für das Anlegen weiterer Dateien zur Verfügung stehende Speicherplatz angegeben.

Tag ’86’

Für die Kommandogruppe ADMIN (CREATE FILE, DELETE FILE, INCLUDE, EXCLUDE) ist für das MF die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

Da dem MF keine AEFs zugeordnet sind, ist die FCI des MF identisch mit den FCP des MF. Diese wird bei Selektion des MF mittels SELECT FILE und entsprechender Option im Parameterbyte P2 ausgegeben

Wenn das MF mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die FMD '64 00' ausgegeben, da dem MF keine AEFs zugeordnet sind.

Das MF der ec-Karte muß die folgenden speziellen EFs (vgl. die folgenden Kapitel) enthalten:

- ein EF_KEY und zugehöriges EF_KEYD,
- ein EF_PWD0 mit der verschlüsselten ec-PIN und zugehöriges EF_PWDD0,
- das EF_FBZ,
- das EF_ID,
- das EF_INFO,
- das EF_LOG und
- das EF_RAND, wenn der Zufallszahlengenerator; wie in Kapitel 2.6. beschrieben, realisiert wird.

Das MF der ec-Karte enthält kein EF_DIR. Es kann ein EF_VERSION enthalten.

6.2. EF_KEY

EF_KEY bezeichnet ein lineares EF, das kryptographische Schlüssel enthält.

Die ec-Karte muß mindestens ein, kann aber auch mehrere EF_KEY enthalten. Dabei gelten folgende Regeln:

- Jedes DF kann höchstens ein EF_KEY enthalten.
- Das MF muß genau ein EF_KEY enthalten.

- Ein EF_KEY muß die Datei-ID '00 10' besitzen.
- Jedes DF, das ein EF_KEY enthält, muß es auch genau ein zugehöriges EF_KEYD enthalten (vgl. das folgende Kapitel).

In einem EF_KEY können maximal 254 Schlüssel abgelegt werden, die durch ihre ***logische Schlüsselnummer*** mit Werten zwischen '00' und 'FF' in dem EF_KEY eindeutig identifiziert werden.

Das EF_KEY des MF wird als globales EF_KEY bezeichnet. Die enthaltenen Schlüssel sind die globalen Schlüssel der ec-Karte. Ein EF_KEY eines anderen DF wird als DF-spezifisches EF_KEY bezeichnet. Die darin enthaltenen Schlüssel sind die DF-spezifischen Schlüssel dieses DF.

Die in einem EF_KEY abgelegten Schlüssel sind entweder 8 Byte oder 16 Byte lang. Die jeweilige Länge und die Algorithmus-ID, durch die angezeigt wird, für welche Algorithmen der Schlüssel zu verwenden ist, sind im zugehörigen EF_KEYD unter der logischen Schlüsselnummer des Schlüssels abgelegt.

6.2.1. FCP

Für ein EF_KEY sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
'62'	mind. '15'		
'81'	'02'	'XX XX'	allokierter Speicherplatz in Byte 17 x maximale Anzahl Records
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 10'	Datei-ID des EF_KEY
'86'	mind. '06'	'UU VV 00 F0 XX YY..'	ACs für das EF_KEY

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '82'

Das EF_KEY ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag '83'

Die Datei-ID des EF_KEY muß '0010' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_KEY die AC '00 F0' (NEV) festgelegt.

6.2.2. Daten

Ein EF_KEY enthält für jeden enthaltenen Schlüssel einen 17 Byte langen Record, der den folgenden Aufbau hat:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’XX’	logische Schlüsselnummer
2-17	16	'XX..XX'	Schlüssel

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Byte 1 enthält die binär kodierte logische Schlüsselnummer des ab Byte 2 abgelegten Schlüssels. Die Schlüssel sind mit aufsteigender logischer Schlüsselnummer in den Records des EF_KEY zu speichern, wobei Lücken bestehen dürfen, so daß die logische Schlüsselnummer größer sein kann als die Recordnummer.

Byte 2

Ab Byte 2 ist in 16 Byte ein Schlüssel binär kodiert abgelegt. Handelt es sich um einen Schlüssel K der Länge 8 Byte zur Verwendung durch DES-basierte kryptographische Algorithmen, so ist er zu K|K mit sich selbst zu konkatenieren und dieser 16 Byte lange Wert zu speichern.

6.2.3. EF_KEY des MF

Das EF_KEY im MF enthält die globalen Schlüssel der ec-Karte. Dies sind

- der Kartenschlüssel K_{Card} zur Absicherung der Karten-Administration,
- der Schlüssel K_{PIN} für die Absicherung der Übertragung der ec-PIN und

- der Schlüssel K_{INFO} für die Authentikation der externen Welt vor dem Lesen von kontospezifischen Daten und für die Authentikation der ec-Karte gegenüber der externen Welt vor der PIN-Prüfung.

6.2.3.1. FCP

Für das EF_KEY des MF sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 44’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 11’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 10'	Datei-ID des EF_KEY
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_KEY

Tag ’81’

Für das globale EF_KEY ist Speicherplatz für vier Records allokiert, so daß ein weiterer Schlüssel nachträglich mittels APPEND RECORD eingebracht werden kann.

Tag ’86’

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_KEY die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt.

6.2.3.2. Daten

Das EF_KEY enthält drei Records, die jeweils einen globalen Schlüssel mit Schlüsselnummer enthalten.

Logische Schlüsselnummer	Schlüssel
'00'	16 Byte langer K_{Card}
'01'	Konkatenation des 8 Byte langen K_{PIN} mit sich selbst
'02'	Konkatenation des 8 Byte langen K_{INFO} mit sich selbst

6.3. EF_KEYD

Die zu den Schlüsseln eines EF_KEY gehörenden öffentlichen Zusatzinformationen werden in einer separaten Datei EF_KEYD abgelegt, die immer gelesen werden kann. Ein EF_KEYD ist ein lineares EF, das mehrfach in einer ec-Karte enthalten sein kann. Dabei gelten folgende Regeln:

- Jedes DF, das ein EF_KEY enthält, muß auch genau ein EF_KEYD enthalten. Dieses muß die Zusatzinformationen für die Schlüssel aus dem zugehörigen EF_KEY dieses DF enthalten.
- Ein EF_KEYD muß die Datei-ID '00 13' besitzen.

6.3.1. FCP

Für ein EF_KEYD sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
'62'	mind. '15'		
'81'	'02'	'XX XX'	allokierter Speicherplatz in Byte 5 x maximale Anzahl Records
'82'	'03'	'02 41 05'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 13'	Datei-ID des EF_KEYD
'86'	mind. '06'	'UU VV 00 00 XX YY..'	ACs für das EF_KEYD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’82’

Das EF_KEYD ist ein lineares EF, dessen Recordlänge zu 5 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEYD muß '0013' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_KEYD die AC '00 00' (ALW) festgelegt.

6.3.2. Daten

Ein EF_KEYD enthält pro Schlüssel im zugehörigen EF_KEY einen 5 Byte langen Record, der den folgenden Aufbau hat:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’XX’	logische Schlüsselnummer
2	1	'08' oder '10'	Schlüssellänge
3	1	'XX'	Algorithmus-ID
4	1	'FF'	Fehlbedienungszähler
5	1	'XX'	Schlüssel-Version

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Byte 1 enthält die binär kodierte logische Schlüsselnummer des Schlüssels aus dem zugehörigen EF_KEY, zu dem in diesem Record Zusatzinformationen enthalten sind.

Byte 2

Byte 2 gibt die Schlüssellänge des zugehörigen Schlüssels binär kodiert an. Zur Zeit werden nur 8 Byte (Wert '08') oder 16 Byte (Wert '10') lange Schlüssel verwendet.

Die zugehörige Schlüssellänge wird vor Verwendung eines Schlüssels durch die Karte ausgewertet. Insbesondere darf ein Schlüssel nur verwendet werden, wenn Schlüssellänge und

Algorithmus-ID (vgl. Byte 3) konsistent sind.

Byte 3

Den einzelnen Schlüsseln muß ein kryptographischer Algorithmus zugeordnet werden. Die ID des Algorithmus wird in dieses Byte binär kodiert.

Für folgende Algorithmen ist die Kodierung der ID festgelegt:

Algorithmus-ID	Algorithmus	zulässige Schlüssellänge
’01’	Schlüsselableitung für 16 Byte langen Schlüssel vgl. Kapitel 2.5.	'10'
'02'	Schlüsselableitung für 8 Byte langen Schlüssel vgl. Kapitel 2.5.	'10'
'06'	DES (CBC-Mode) vgl. Kapitel 2.3.1.	'08'
'07'	Triple-DES (CBC) vgl Kapitel 2.3.2.	'10'

Die zugehörige Algorithmus-ID wird durch die Karte vor Verwendung eines Schlüssels ausgewertet. Insbesondere darf ein Schlüssel bei der Ausführung eines Kommandos nur dann verwendet werden, wenn dies für die Algorithmus-ID zulässig ist und die Schlüssellänge für die Algorithmus-ID zulässig ist.

Die Algorithmus-ID '01' oder '02' zeigt an, daß der Schlüssel für die Schlüsselableitung reserviert ist. Ohne Ableitung darf er nicht in MAC-Bildung oder Verschlüsselung eingehen. Insbesondere darf er weder durch das Kommando EXTERNAL AUTHENTICATE noch für das Secure Messaging der Standard- und Administrationskommandos verwendet werden.

Byte 4

Byte 4 enthält einen binär kodierten Fehlbedienungszähler, dessen Wert (Startwert 'FF') um 1 vermindert wird, wenn eine Funktion, welche mit diesem Schlüssel arbeitet, fehlerhaft abläuft und die Karte diese Sicherheitsverletzung feststellt. Der Schlüssel wird für jede weitere Verwendung gesperrt, wenn dieser Zähler den Wert '00' erreicht hat.

Byte 5

Byte 5 enthält eine frei wählbare Bezeichnung für den jeweiligen Schlüssel. Dieser Wert ist binär kodiert. Er kann z.B. die Generation eines Schlüssels nach mehrmaligem Schlüsselwechsel anzeigen. Wird von dieser Möglichkeit kein Gebrauch gemacht, so ist dieses Byte mit '00' zu

belegen.

6.3.3. EF_KEYD des MF

Das EF_KEYD im MF enthält die Zusatzinformationen zu den globalen Schlüsseln der ec-Karte.

6.3.3.1. FCP

Für das EF_KEYD des MF sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		
’81’	’02’	’00 14’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 05’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 13'	Datei-ID des EF_KEYD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_KEYD

Tag ’81’

Für das globale EF_KEYD ist Speicherplatz für vier Records allokiert, so daß Zusatzinformationen zu einem weiteren Schlüssel nachträglich mittels APPEND RECORD eingebracht werden können.

Tag ’86’

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_KEYD die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.3.3.2. Daten

Das EF_KEYD enthält drei Records, die jeweils die Zusatzinformation zu einem globalen Schlüssel enthalten.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Fehlbedienungs-zähler	Schlüssel-Version
'00'	'10'	'07'	'FF'	'00'
'01'	'08'	'06'	'FF'	'00'
'02'	'08'	'06'	'FF'	'00'

6.4. EF_AUT

EF_AUT bezeichnet ein lineares EF, das kryptographische Schlüssel für die Authentikation der ec-Karte gegenüber der externen Welt mit dem Kommando INTERNAL AUTHENTICATE enthält.

Die ec-Karte kann mehrere EF_AUTs enthalten. Dabei gelten folgende Regeln:

- Jedes DF kann höchstens ein EF_AUT enthalten.
- Ein EF_AUT muß die Datei-ID '00 11' besitzen.
- Jedes DF, das ein EF_AUT enthält, muß es auch genau ein zugehöriges EF_AUTD enthalten (vgl. das folgende Kapitel).

In einem EF_AUT können maximal 254 Schlüssel abgelegt werden, die durch ihre logische Schlüsselnummer mit Werten zwischen '00' und 'FF' in dem EF_AUT eindeutig identifiziert werden. Die Schlüssel in einem EF_AUT werden als Authentikations-Schlüssel bezeichnet.

Ein EF_AUT des MF wird als globales EF_AUT bezeichnet. Die enthaltenen Schlüssel sind die globalen Authentikations-Schlüssel der ec-Karte. Ein EF_AUT eines anderen DF wird als DF-spezifisches EF_AUT bezeichnet. Die darin enthaltenen Schlüssel sind die DF-spezifischen Authentikations-Schlüssel dieses DF.

Die in einem EF_AUT abgelegten Schlüssel sind entweder 8 Byte lang zur Verwendung durch DES-basierte kryptographische Algorithmen (vgl. Kapitel 2.3.1.) oder 16 Byte lang zur Verwendung durch Triple-DES-basierte kryptographische Algorithmen (vgl. Kapitel 2.3.2.).

Mit dem Kommando INTERNAL AUTHENTICATE (vgl. Kapitel 8.) kann sich die ec-Karte gegenüber der externen Welt authentisieren, indem sie einen in den Kommandodaten übergebenen Wert, unter einem Authentikations-Schlüssel verschlüsselt, in den Antwortdaten übergibt. In den Kommando-Parametern wird spezifiziert,

- ob der von der Karte zu verwendende Authentikations-Schlüssel global oder DF-spezifisch ist und

- welche Schlüsselnummer KID der zu verwendende Authentikations-Schlüssel hat.

Durch den in Kapitel 2.8.2. beschriebenen Mechanismus zur Schlüsselauswahl wird mit den Parametern

- "globaler/DF-spezifischer Schlüssel" aus den Kommandodaten,

- "KID" aus den Kommandodaten und

- bei Aufruf von INTERNAL AUTHENTICATE aktuelles DF

der durch das Kommando referenzierte Authentikations-Schlüssel gesucht.

Wenn der Mechanismus zur Schlüsselauswahl erfolgreich war, wird der in den Kommandodaten übergebene Wert mit dem gefundenen Authentikations-Schlüssel unter Verwendung des ICV '00..00' verschlüsselt und das Chiffrat in der Antwortnachricht an die externe Welt übergeben. Als Verschlüsselungsalgorithmus wird der für den Authentikations-Schlüssel durch die Algorithmus-ID im zugehörigen EF_AUTD festgelegte Algorithmus verwendet.

6.4.1. FCP

Für ein EF_AUT sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	mind. ’15’		
’81’	’02’	’XX XX’	allokierter Speicherplatz in Byte 17 x maximale Anzahl Records
’82’	’03’	’02 41 11’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 11'	Datei-ID des EF_AUT
'86'	mind. '06'	'UU VV 00 F0 XX YY..'	ACs für das EF_AUT

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’82’

Das EF_AUT ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUT muß '0011' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_AUT die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_AUT die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt.

6.4.2. Daten

Ein EF_AUT enthält für jeden enthaltenen Schlüssel einen 17 Byte langen Record, der den folgenden Aufbau hat:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’XX’	logische Schlüsselnummer
2	16	'XX..XX'	Schlüssel

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Byte 1 enthält die binär kodierte logische Schlüsselnummer des ab Byte 2 abgelegten Schlüssels. Die Schlüssel sind mit aufsteigender logischer Schlüsselnummer in den Records des EF_AUT zu speichern, wobei Lücken bestehen dürfen, so daß die logische Schlüsselnummer größer sein kann als die Recordnummer.

Byte 2

Ab Byte 2 ist in 16 Byte ein Schlüssel binär kodiert abgelegt. Handelt es sich um einen Schlüssel K der Länge 8 Byte zur Verwendung durch DES-basierte kryptographische Algorithmen, so ist er zu K|K mit sich selbst zu konkatenieren und dieser 16 Byte lange Wert zu speichern.

6.5. EF_AUTD

Die zu den Schlüsseln eines EF_AUT gehörenden öffentlichen Zusatzinformationen werden in einer separaten Datei EF_AUTD abgelegt, die immer gelesen werden kann. Ein EF_AUTD ist ein lineares EF, das mehrfach in einer ec-Karte enthalten sein kann. Dabei gelten folgende Regeln:

- Jedes DF, das ein EF_AUT enthält, muß auch genau ein EF_AUTD enthalten. Dieses muß die Zusatzinformationen für die Schlüssel aus dem zugehörigen EF_AUT dieses DF enthalten.
- Ein EF_AUTD muß die Datei-ID '00 14' besitzen.

6.5.1. FCP

Für ein EF_AUTD sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	mind. 15’		
’81’	’02’	’XX XX’	allokierter Speicherplatz in Byte 4 x maximale Anzahl Records
’82’	’03’	’02 41 04’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 14'	Datei-ID des EF_AUTD
'86'	mind. '06'	'UU VV 00 00 XX YY..'	ACs für das EF_AUTD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’82’

Das EF_AUTD ist ein lineares EF, dessen Recordlänge zu 4 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUTD muß '0014' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_AUTD die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_AUTD

die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.5.2. Daten

Ein EF_AUTD enthält pro Schlüssel im zugehörigen EF_AUT einen 4 Byte langen Record, der den folgenden Aufbau hat:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'XX'	logische Schlüsselnummer
2	1	'08' oder '10'	Schlüssellänge
3	1	'XX'	Algorithmus-ID
4	1	'XX'	Schlüssel-Version

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Byte 1 enthält die binär kodierte logische Schlüsselnummer des Authentikations-Schlüssels aus dem zugehörigen EF_AUT, zu dem in diesem Record Zusatzinformationen enthalten sind.

Byte 2

Byte 2 gibt in binärer Kodierung die Schlüssellänge des zugehörigen Authentikations-Schlüssels an (Zur Zeit nur die Werte '08' und '10').

Die zugehörige Schlüssellänge wird vor Verwendung eines Authentikations-Schlüssels durch die Karte ausgewertet. Insbesondere darf ein Schlüssel nur verwendet werden, wenn Schlüssellänge und Algorithmus-ID (vgl. Byte 3) konsistent sind.

Byte 3

Den einzelnen Schlüsseln wird jeweils ein kryptographischer Algorithmus zugeordnet. Die ID des Algorithmus wird in dieses Byte binär kodiert.

Für folgende Algorithmen ist die Kodierung der ID festgelegt:

Algorithmus-ID	**Algorithmus**	**zulässige Schlüssellänge**

’06’	DES (CBC-Mode) vgl. Kapitel 2.3.1.	’08’
’07’	Triple-DES (CBC) vgl Kapitel 2.3.2.	’10’

Die Algorithmus-ID wird durch die Karte vor Ausführung des INTERNAL AUTHENTICATE ausgewertet. Das Kommando kann nur dann ausgeführt werden, wenn die Algorithmus-ID einen der spezifizierten Werte hat und die Schlüssellänge für die Algorithmus-ID zulässig ist.

Byte 4

Byte 4 enthält eine frei wählbare Bezeichnung für den jeweiligen Schlüssel. Dieser Wert ist binär kodiert. Er kann z.B. die Generation eines Schlüssels nach mehrmaligem Schlüsselwechsel anzeigen. Wird von dieser Möglichkeit kein Gebrauch gemacht, so ist dieses Byte mit '00' zu belegen.

6.6. EF_PWD0 und EF_PWD1

EF_PWDx (x = 0 oder 1) bezeichnet ein lineares EF, das in Record '01' ein 8 Byte langes Paßwort enthält. Die ec-Karte muß mindestens ein EF_PWD0 enthalten, kann aber auch mehrere EF_PWDx enthalten. Dabei gelten folgende Regeln:

- Jedes DF kann höchstens ein EF_PWD0 enthalten.
- Jedes DF kann höchstens ein EF_PWD1 enthalten.
- Das MF muß genau ein EF_PWD0 enthalten.
- Ein EF_PWD0 muß die Datei-ID '00 12' besitzen.
- Ein EF_PWD1 muß die Datei-ID '00 22' besitzen.
- Das MF muß das EF_FBZ mit dem globalen Fehlbedienungszähler enthalten (vgl. Kapitel 6.8.).
- Jedes DF, das ein EF_PWDx enthält, muß es auch genau ein zugehöriges EF_PWDDx enthalten (vgl. das folgende Kapitel).

Ein EF_PWDx des MF wird als globales EF_PWDx bezeichnet. Das jeweils enthaltene Paßwort ist das globale Paßwort mit der Paßwortnummer x der ec-Karte.

Als globales Paßwort mit der Paßwortnummer 0 enthält die ec-Karte die verschlüsselte ec-PIN des deutschen Kreditgewerbes.

Ein EF_PWDx eines DF, das nicht das MF ist, wird als DF-spezifisches EF_PWDx bezeichnet. Das

jeweils enthaltene Paßwort ist das DF-spezifische Paßwort mit der Paßwortnummer x des DF.

In den DF-spezifischen EF_PWD0 der ec-Karte wird ausschließlich die ec-PIN, eventuell in unterschiedlicher Formatierung oder mit unterschiedlichen ACs für das Kommando VERIFY abgelegt.

6.6.1. FCP

Für ein EF_PWDx sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	mind. ’17’		
’81’	’02’	’00 08’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 08’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 Y2'	Datei-ID des EF_PWDx
'86'	mind. '08'	'SS TT 00 F0 UU VV YY ZZ...'	ACs für das EF_PWDx

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_PWDx enthält maximal einen Record.

Tag ’82’

Das EF_PWDx ist ein lineares EF, dessen Recordlänge auf 8 Byte festgelegt ist.

Tag ’83’

Die Datei-ID eines EF_PWD0 muß '0012' sein.

Die Datei-ID eines EF_PWD1 muß '0022' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_PWDx die AC '00 F0' (NEV) festgelegt.

6.6.2. Daten

Das EF_PWDx enthält einen 8 Byte langen Record, der den folgenden Aufbau hat

Byte	Länge (in Byte)	Wert	Erläuterung
1-8	8	'XX..XX'	Paßwort

Das in diesem Record mit der Recordnummer '01' gespeicherte Paßwort ist das Paßwort mit der Paßwortnummer x, das die ec-Karte als Eingabe-Datum bei einer Paßwort-Verifikation mittels des Kommandos VERIFY erwartet. Dieser Wert ist **immer** 8 Byte lang.

Das von dem Karteninhaber an einem Terminal zur Verifikation einzugebende Paßwort (Karteninhaber-Paßwort) muß durch das Terminal durch Formatierung zu dem hier gespeicherten Paßwort aufbereitet werden, bevor es mittels VERIFY an die Karte übergeben wird. Die Art der Formatierung wird durch das zweite Byte des Records im zugehörigen EF_PWDDx festgelegt.

6.6.3. EF_PWD0 des MF

Das globale EF_PWD0 im MF der ec-Karte enthält in dem 8 Byte langen Record '01' die verschlüsselte ec-PIN.

6.6.3.1. FCP

Für das EF_PWD0 des MF sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’17’		
’81’	’02’	’00 08’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 08’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 12'	Datei-ID des EF_PWD0
'86'	'08'	'00 60 00 F0 00 60 00 61'	ACs für das EF_PWD0

TAG ’86’

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_PWD0 die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt.

Für das Kommando VERIFY wird für das globale EF_PWD0 die AC '00 61' (ENC_G mit Schlüsselnummer '01') festgelegt.

6.6.3.2. Daten

Die ec-PIN ist verschlüsselt im Format "Encrypted Format 0 PIN Block" in dem Record des globalen EF_PWD0 gespeichert. Der Format 0 PIN Block wird nach ISO 9564-1 aus der ec-PIN und Identifikationsdaten des Karteninhabers aus EF_ID und EF_INFO gebildet.

Hierzu wird die ec-PIN wie folgt zu einem 8 Byte langen 'Plain text PIN field' aufbereitet:

C	L	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

Erläuterung:

Jedes Feld repräsentiert ein Halbbyte.

C: Kontroll-Feld, binär kodiert hat immer den Wert '0'

L: PIN-Länge, binär kodiert mögliche Werte von '4' bis 'C'

P: PIN-Ziffer, BCD-kodiert

F:	Filler, binär kodiert	hat immer den Wert 'F'
P/F:	PIN-Ziffer/Filler	Belegung abhängig von der PIN-Länge

Die Identifikationsdaten des Karteninhabers werden wie folgt zu einem 8 Byte langen 'Account number field' aufbereitet:

0	0	0	0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12

Erläuterung:

Jedes Feld repräsentiert ein Halbbyte.

0:	Füll-Ziffer, binär kodiert	hat immer den Wert '0'
A1..A12:	Kontonummer, BCD-kodiert	enthalten sind die 12 niedrigstwertigen Byte der Spur 2-PAN ohne Prüfziffer

Die Werte von A1 bis A12 finden sich wie folgt in den Daten der ec-Karte:

A1,A2:	Byte 4 von EF_ID
A3..A12:	Byte 1-5 von EF_INFO

Es wird die XOR-Summe aus dem 'Plain text PIN field' und dem 'Account number field' gebildet. Der erhaltene Wert ist binär kodiert. Er wird mit **FPIN** bezeichnet.

FPIN wird mit dem Schlüssel K_{PS} (PIN Storage Key) mittels des DES (vgl. Kapitel 2.1.) verschlüsselt. Als Ergebnis erhält man den Wert EPIN:

$$\mathbf{EPIN} = eK_{PS}(FPIN).$$

Dieser binär kodierte Wert wird bei der Personalisierung der ec-Karte in den Record des globalen EF_PWD0 eingestellt.

Der K_{PS} ist nicht in der ec-Karte gespeichert.

6.7. EF_PWDD0 und EF_PWDD1

Zu einem Paßwort mit der Paßwortnummer x gehörende, nicht geheime Zusatzinformationen

werden in Record ’01’ eines separaten EF abgelegt, das mit EF_PWDDx bezeichnet wird. Ein EF_PWDDx ist ein lineares EF, das mehrfach in einer ec-Karte enthalten sein kann. Dabei gelten folgende Regeln:

- Jedes DF, das ein EF_PWD0 enthält, muß auch genau ein EF_PWDD0 enthalten. Dieses muß in Record '01' die Zusatzinformationen für das Paßwort aus dem zugehörigen EF_PWD0 dieses DF enthalten.

- Ein EF_PWDD0 muß die Datei-ID '00 15' besitzen.

- Jedes DF, das ein EF_PWD1 enthält, muß auch genau ein EF_PWDD1 enthalten. Dieses muß die Zusatzinformationen für das Paßwort aus dem zugehörigen EF_PWD1 dieses DF enthalten.

- Ein EF_PWDD1 muß die Datei-ID '00 25' besitzen.

6.7.1. FCP

Für ein **EF_PWDD0** sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	mind. ’15’		
’81’	’02’	’00 03’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 03’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 15'	Datei-ID des EF_PWDDx
'86'	mind. '06'	'UU VV 00 00 XX YY...'	ACs für das EF_PWDDx

Für ein **EF_PWDD1** sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	mind. ’15’		
’81’	’02’	’00 05’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 05’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 25'	Datei-ID des EF_PWDDx
'86'	mind. '06'	'UU VV 00 00 XX YY...'	ACs für das EF_PWDDx

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_PWDDx enthält maximal einen Record.

Tag ’82’

Das EF_PWDD0 ist ein lineares EF, dessen Recordlänge zu 3 Byte festgelegt ist.

Das EF_PWDD1 ist ein lineares EF, dessen Recordlänge zu 5 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_PWDD0 muß '0015' sein.

Die Datei-ID des EF_PWDD1 muß '0025' sein.

Tag ’86’

Für das Kommando READ RECORD wird für jedes EF_PWDDx die AC '00 00' (ALW) festgelegt.

6.7.2. Daten

Ein **EF_PWDD0** enthält einen 3 Byte langen Record, der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’01’	Art der Karteninhaber-Authentikation
2	1	’XX’	Kodierung des Paßwortes
3	1	'XX'	Mindestlänge des Karteninhaber-Paßwortes

Ein **EF_PWDD1** enthält einen 5 Byte langen Record, der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’01’	Art der Karteninhaber-Authentikation
2	1	’XX’	Kodierung des Paßwortes
3	1	'XX'	Mindestlänge des Karteninhaber-Paßwortes
4	1	'XX'	Initialwert des Fehlbedienungszählers in Byte 5
5	1	'XX'	Fehlbedienungszähler (FBZ)

Im folgenden werden die Komponenten der Records näher erläutert:

Byte 1

Byte 1 ist binär kodiert. Dieses Byte wird von der ec-Karte nicht ausgewertet, sondern dient nur der Information der externen Welt.

Durch den enthaltenen Wert wird die Art der Karteninhaber-Authentikation beschrieben, die anhand der im Record des zugehörigen EF_PWDx gespeicherten Daten durchgeführt werden soll.

Zur Zeit ist nur die Paßwort-Verifikation möglich, die durch den Wert '01' kodiert wird.

Byte 2

Byte 2 ist binär kodiert. Durch den enthaltenen Wert wird beschrieben, wie das Paßwort im zugehörigen EF_PWDx kodiert ist. Dieses Byte wird von der ec-Karte nicht ausgewertet, sondern

dient nur der Information der externen Welt.

Das **linke Halbbyte** von Byte 2 spezifiziert die Art der Formatierung, durch die aus dem Karteninhaber-Paßwort das in dem zugehörigen EF_PWDx gespeicherte 8 Byte lange Paßwort erzeugt wird.

Die folgenden Belegungen entsprechen im Aufbau dem rechten Halbbyte des Control Fields (Byte 1) von ISO PIN Block Formaten gemäß ISO 9564-1.

Paßwort Format	Wert
Format 0 PIN Block (siehe ISO 9564-1)	'0'
Format 1 PIN Block (siehe ISO 9564-1)	'1'
Format 2 PIN Block (siehe ISO 10202-6)	'2'
RFU (for allocation by ISO/TC 68)	'3'
RFU (for allocation by ISO/TC 68)	'4'
RFU (for allocation by ISO/TC 68)	'5'
RFU (for allocation by ISO/TC 68)	'6'
RFU (for allocation by ISO/TC 68)	'7'
RFU (for allocation by national standards organization)	'8'
RFU (for allocation by national standards organization)	'9'
RFU (for allocation by national standards organization)	'A'
RFU (for allocation by national standards organization)	'B'
Rechtsauffüllen mit binär 1	'C'
Encrypted Format 0 PIN Block	'D'
RFU (allocated for private use)	'E'
RFU (allocated for private use)	'F'

Das proprietäre Format 'C' ist sowohl für eine BCD-kodierte PIN (maximal 12 Ziffern zwischen '0' und '9') als auch für ein ASCII-kodiertes alphanumerisches Paßwort (maximal 8 Zeichen) anwendbar. In beiden Fällen wird das Karteninhaber-Paßwort mit binär 1 auf 8 Byte aufgefüllt.

Das **rechte Halbbyte** von Byte 2 gibt an, ob das Paßwort binär, BCD- oder ASCII-kodiert ist.

Kodierungsart	Wert
binär	'0'
BCD	'1'
ASCII	'2'
RFU	'3' - 'F'

Bei Speicherung eines verschlüsselten Paßwortes (linkes Halbbyte 'D') oder eines Format n PIN Blocks (linkes Halbbyte '0', '1' oder '2') muß dieses Halbbyte den Wert '0' haben.

Bei Speicherung eines Paßworts im proprietären Format 'C' gibt dieses Halbbyte an, ob es sich um eine PIN (BCD-kodiert) oder um ein alphanumerisches Paßwort (ASCII-kodiert) handelt.

Byte 3

Byte 3 enthält binär kodiert die erforderliche Mindestlänge (in Halbbyte) des Karteninhaber-Paßwortes mit der Paßwortnummer x. Dieses Byte wird von der ec-Karte nicht ausgewertet, sondern dient nur der Information der externen Welt.

Dieses Byte muß mit dem Wert 'FF' belegt werden, wenn das Paßwort verschlüsselt gespeichert ist.

Für ein Paßwort im Format n PIN Block Format ist dieser Wert mindestens zu '04' zu setzen.

Byte 4 (nur EF_PWDD1)

Byte 4 enthält den binär kodierten Initialwert, auf den der Fehlbedienungszähler in Byte 5 nach einer korrekten Paßwort-Prüfung reinitialisiert wird, wenn dieser ungleich dem Initialwert ist.

Byte 5 (nur EF_PWDD1)

Byte 5 enthält den binär kodierten Fehlbedienungszähler (FBZ) des Paßwortes im zugehörigen EF_PWD1, der bei jeder fehlerhaften Paßwort-Prüfung um 1 dekrementiert wird. Hat dieser Zähler den Wert '00' erreicht, ist das Paßwort gesperrt und kann nur durch ein UPDATE RECORD des EF_PWDD1 unter Einhaltung der entsprechenden AC entsperrt werden.

6.7.3. EF_PWDD0 des MF

Das EF_PWDD0 im MF enthält in Record '01' die Zusatzinformationen zu der im globalen EF_PWD0 abgelegten ec-PIN.

6.7.3.1. FCP

Für das globale EF_PWDD0 sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		
’81’	’02’	’00 03’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 03’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 15'	Datei-ID des EF_PWDD0
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_PWDD0

Tag ’86’

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_PWDD0 die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.7.3.2. Daten

Das globale EF_PWDD0 enthält einen 3 Byte langen Record, der die Zusatzinformationen zu der ec-PIN enthält.

Karteninhaber-Authentikation	Kodierung des Paßwortes	Mindestlänge
'01'	'D0'	'FF'

6.8. EF_FBZ

EF_FBZ bezeichnet das lineare EF, das in Record '01' den globalen Fehlbedienungszähler und den zugehörigen Initialwert für die in den EF_PWD0 der ec-Karte, eventuell in unterschiedlicher Formatierung, abgelegte ec-PIN enthält.

Das EF_FBZ ist genau einmal in der ec-Karte enthalten:

- Das MF muß das EF_FBZ enthalten.

- Das EF_FBZ muß die Datei-ID '0016' besitzen.

6.8.1. FCP

Für das EF_FBZ sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 02’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 02’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 16'	Datei-ID des EF_FBZ
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_FBZ

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_FBZ enthält maximal einen Record.

Tag ’82’

Das EF_FBZ ist ein lineares EF, dessen Recordlänge zu 2 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_FBZ muß '0016' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_FBZ die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_FBZ die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.8.2. Daten

Das EF_FBZ enthält in Record '01' einen 2 Byte langen Record, der den globalen Fehlbedienungszähler und den zugehörigen Initialwert für die ec-PIN enthält.

Initialwert des FBZ	FBZ
’03’	’03’

Der Fehlbedienungszähler für die ec-PIN hat den Startwert '03'.

6.9. EF_ID

EF_ID bezeichnet das lineare EF, das in Record '01' die Kartenidentifikationsdaten enthält. Das EF_ID ist genau einmal in einer ec-Karte enthalten:

- Das MF muß das EF_ID enthalten,
- das EF_ID muß die Datei-ID '0003' besitzen.

6.9.1. FCP

Für das EF_ID sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 16	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 16’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 03'	Datei-ID des EF_ID
'86'	'06'	'00 40 00 00 00 F0'	ACs für das EF_ID

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_ID enthält maximal einen Record.

Tag ’82’

Das EF_ID ist ein lineares EF, dessen Recordlänge auf 22 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_ID muß '0003' sein.

Tag ’86’

Für das Kommando APPEND RECORD wird für das EF_ID die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Da das EF_ID maximal einen Record enthält, ist das Kommando höchstens einmal ausführbar.

Für das Kommando READ RECORD wird für das EF_ID die AC '00 00' (ALW) festgelegt.

Für das Kommando UPDATE RECORD wird die AC '00 F0' (NEV) festgelegt.

6.9.2. Daten

Das EF_ID enthält einen 22 Byte langen Record, der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der ec-Karte
13-15	3	'JJ MM TT'	Aktivierungsdatum der ec-Karte
16-17	2	'02 80'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Freier Zähler für die Schlüsselerzeugung

Im folgenden werden die Komponenten des Records näher erläutert:

Die ersten 10 Byte des EF_ID bilden eine ***Kartennummer***, durch die die ec-Karte eindeutig identifiziert wird.

Hierbei sind Byte 1-4 identisch mit den ersten 4 Byte der Spur 2 PAN.

Byte 1

Byte 1 enthält den Branchenhauptschlüssel (international) und ist mit '67' (BCD-kodiert) belegt.

Byte 2-4

Byte 2-4 enthalten die BCD-kodierte Kurz-Bankleitzahl des kartenausgebenden Instituts.

Byte 2 gibt die Institutsgruppe gemäß der folgenden Kodierung an:

Institutsgruppe	Wert
öffentlich-rechtliche und private Banken	'21'
Privat- und Geschäftsbanken	'22'
Sparkassen	'25'
Genossenschaftsbanken	'26' und '29'

Byte 3-4 enthält die 4-stellige Routingnummer, durch die das kartenausgebende Institut innerhalb der Institutsgruppe eindeutig identifiziert wird.

Byte 5-9

Byte 5-9 enthalten eine 10-stellige BCD-kodierte ***individuelle Kartennummer***, durch die die einzelne Karte innerhalb des Nummernkreises des kartenausgebenden Instituts eindeutig identifiziert wird.

Byte 10

Byte 10 enthält im linken Halbbyte eine Prüfziffer, die nach dem Verfahren 'Luhn formula for computing modulus 10' über die Ziffern der ersten 9 Byte berechnet ist.

Das rechte Halbbyte wird zu 'D' gesetzt.

Byte 11-12

Byte 11-12 enthalten das Verfalldatum der Karte im Format JJ MM (BCD-kodiert).

Byte 13-15

Byte 13-15 enthalten das Aktivierungsdatum der Karte im Format JJ MM TT (BCD-kodiert)

Byte 16-17

Byte 16-17 enthalten den BCD-kodierten Ländercode '02 80' nach ISO 3166.

Byte 18-20

Das Währungskennzeichen ist ASCII-kodiert nach ISO 4217.

Byte 21

Durch dieses Byte wird ein Multiplikator festgelegt, der bestimmt, welchen Wert die Einheit eines BCD-kodierten Betrages, bezogen auf die in Byte 18-20 kodierte Währung, hat.

Das Byte ist binär kodiert und kann die folgenden Werte annehmen:

Multiplikator	Kodierung
10^{-2}	'01'
10^{-1}	'02'
10^{0}	'04'
10^{1}	'08'
10^{2}	'10'
10^{3}	'20'

Byte 22

Byte 22 enthält eine binär kodierte Nummer zwischen 0 und 255.

Die Nummer 0 ist unabhängig von Chiptyp und von der Software-Version für Chipkarten, die im Feldversuch in Ravensburg eingesetzt werden, zu verwenden.

Durch die Nummern 1 bis 255 wird angezeigt, welcher Chiptyp mit welcher Version der Software in der Chipkarte verwendet wird. Diese Nummern werden fortlaufend durch den ZKA bei der Zulassung eines Chiptyps zusammen mit einer Software-Version vergeben. Da die jeweilige Nummer in die Ableitung der kartenindividuellen Schlüssel eingeht, kann sie nach der Personalisierung der Chipkarte nicht mehr verändert werden, so daß Änderungen der Software durch Nachladen im Feld an dieser Stelle nicht berücksichtigt werden können.

6.10. EF_INFO

EF_INFO bezeichnet das lineare EF, das in Record '01' kontospezifische Daten des Karteninhabers aus Spur 2 und Spur 3 einer Magnetstreifenkarte enthält.

Das EF_INFO ist im MF enthalten.

Die ACs für das EF_INFO legen fest, daß die enthaltenen Daten nur gelesen werden können, wenn sich die externe Welt vorher mittels EXTERNAL AUTHENTICATE unter Verwendung des globalen Schlüssels K_{INFO} (Schlüsselnummer '02') authentisiert hat. Durch Aufruf des Kommandos READ RECORD mit CLA = '04' fordert das Terminal an, daß die Antwortdaten mit einem mittels K_{INFO} gebildeten MAC versehen werden. Der ICV für die MAC-Bildung wird von dem Terminal vorher mittels PUT DATA an die Karte übergeben. Auf diese Weise authentisiert sich die Karte gegenüber dem Terminal.

Ein Terminal, das diesen Schlüssel nicht besitzt, kann daher nicht über die Kontodaten des Karteninhabers verfügen und damit auch keine PIN-Prüfung vornehmen.

6.10.1. FCP

Für das EF_INFO sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		
’81’	’02’	’00 11’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 11’	Datei-Deskriptor für lineares EF
'83'	'02'	'01 00'	Datei-ID des EF_INFO
'86'	'06'	'00 40 82 42 00 40'	ACs für das EF_INFO

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_INFO enthält maximal einen Record.

Tag ’82’

Das EF_INFO ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_INFO ist '01 00'.

Tag ’86’

Für das Kommando APPEND RECORD wird für das EF_INFO die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Da das EF_INFO maximal einen Record enthält, ist das Kommando höchstens einmal ausführbar.

Für das Kommando READ RECORD wird für das EF_INFO die AC '82 42' (AUT_G und PRO_G mit Schlüsselnummer '02') festgelegt.

Für das Kommando UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.10.2. Daten

Das EF_INFO enthält einen 17 Byte langen Record, der den folgenden Aufbau hat:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1-5	5	'nn..nn'	Kundenkontonummer
6	1	'nD'	Prüfziffer Spur 2
7-8	2	'nn nD'	Service-Code
9-10	2	'0n nn'	Kartenfolgenummer
11-14	4	'nn..nn'	Bankleitzahl kontoführendes Institut
15	1	'nD'	Prüfziffer Spur 3
16	1	'0n'	Freizügigkeitsschlüssel
17	1	'nn'	Kontoart und Benutzungseinschränkung

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1-5

Byte 1-5 enthalten die 10-stellige BCD-kodierte Kundenkontonummer.

Byte 6

Byte 6 enthält die Prüfziffer nach dem Verfahren 'Luhn formula for computing modulus 10' aus Spur 2 und einen Feldseparator 'D'.

Die Prüfziffer aus Spur 2 wird über die Ziffern von

- '67' (Byte 1 aus EF_ID) konkateniert mit
- Kurz-BLZ kartenausgebendes Institut (Byte 2-4 aus EF_ID) konkateniert mit
- Kundenkontonummer (Byte 1-5 aus EF_INFO)

gebildet.

Byte 7-8

In Byte 7-8 ist der Servicecode aus der Spur 2 kodiert.

- Das linke Halbbyte von Byte 7 enthält das BCD-kodierte Funktionskennzeichen.
- Das rechte Halbbyte von Byte 7 enthält das BCD-kodierte Autorisierungskennzeichen.
- Das linke Halbbyte von Byte 8 enthält das BCD-kodierte Servicekennzeichen

Das rechte Halbbyte von Byte 8 enthält den Feldseparator 'D'.

Byte 9-10

Byte 9-10 enthalten rechtsbündig die 3-stellige BCD-kodierte Kartenfolgenummer mit einer vorangestellten '0'.

Byte 11-14

Byte 11-14 enthalten die 8-stellige BCD-kodierte Bankleitzahl des kontoführenden Instituts.

Byte 15

Byte 15 enthält die Prüfziffer nach dem Verfahren 'Luhn formula for computing modulus 10' aus Spur 3 und einen Feldseparator 'D'.

Die Prüfziffer aus Spur 3 wird über die Ziffern von

- '59' konkateniert mit
- BLZ kontoführendes Institut (Byte 11-14 aus EF_INFO) konkateniert mit
- Kundenkontonummer (Byte 1-5 aus EF_INFO)

gebildet.

Byte 16

Byte 16 enthält rechtsbündig den BCD-kodierten Freizügigkeitsschlüssel aus Spur 3 mit einer vorangestellten '0'.

Byte 17

Das linke Halbbyte von Byte 17 enthält BCD-kodiert die Kontoart aus Spur 3.

Das rechte Halbbyte von Byte 17 enthält BCD-kodiert die Benutzungseinschränkung aus Spur 3.

6.11. EF_LOG

EF_LOG bezeichnet ein zyklisches EF, das der internen Fehlerprotokollierung der ec-Karte dient. Die ec-Karte beschreibt immer dann einen Record des EF_LOG, wenn ein Kommando aufgrund einer Fehlersituation abbricht.

Fehler, die aus einem internen Zugriff auf das EF_LOG resultieren, werden weder in das EF_LOG

abgespeichert noch der externen Welt ausgegeben. Die interne Schreibroutine auf das EF_LOG bleibt ohne Funktion, wenn das EF nicht angelegt ist, d.h., es hat keine Auswirkungen auf die Funktionalität der Karte, wenn kein EF_LOG in der ec-Karte vorhanden ist.

Ein EF_LOG darf höchstens einmal in einer ec-Karte enthalten sein:

- Nur das MF kann ein EF_LOG enthalten.
- Das EF_LOG muß die Datei-ID '0004' besitzen.

6.11.1. FCP

Für das EF_LOG sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 1E’	allokierter Speicherplatz in Byte
’82’	’03’	’06 41 0A’	Datei-Deskriptor für zyklisches EF
'83'	'02'	'00 04'	Datei-ID des EF_LOG
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_LOG

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_LOG enthält maximal 3 Records.

Tag ’82’

Das EF_LOG ist ein zyklisches EF, dessen Recordlänge auf 10 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_LOG muß '00 04' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_LOG die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.11.2. Daten

Das EF_LOG enthält 3 Records der Länge 10 Byte, die den folgenden Aufbau haben:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1-2	2	CLA INS	Kommando-ID
3-4	2	SW1-SW2	Fehlercode
5-10	6	'XX..XX'	Pfad

Byte 1-2

Byte 1-2 enthalten CLA und INS zur Identifizierung des Kommandos, bei dessen Ausführung ein Fehler aufgetreten ist.

Byte 3-4

Byte 3-4 enthalten den Returncode des Kommandos SW1-SW2 als Fehlermeldung.

Byte 5-10

In 6 Byte wird der Pfad der bei Ausführung des Kommandos aktuellen Datei eingetragen. Die bei Auftreten eines Fehlers aktuelle Datei ist

- das aktuelle EF, wenn zu diesem Zeitpunkt ein EF selektiert ist,
- andernfalls das aktuelle DF.

Der Pfad wird gemäß ISO 7816-4, Kapitel 5.1.2 ohne die Datei-ID des MF angegeben und rechtsbündig in die Byte 5-10 geschrieben. Ist der Pfad kürzer als 6 Byte, wird links mit 'FFFF' aufgefüllt. Ist er länger als 6 Byte, werden nur die letzten 6 Byte des Pfades gespeichert.

6.12. EF_RAND

EF_RAND bezeichnet ein lineares EF, das in Record '01' den Schlüssel enthält, mit dem Zufallszahlen gemäß dem im Beispiel von Kapitel 2.6. beschriebenen Verfahren gebildet werden

können. Das EF_RAND muß nur dann in der ec-Karte enthalten sein, wenn dieses Beispiel realisiert wird. In diesem Fall sind die folgenden Regeln zu beachten:

- Das EF_RAND ist im MF der ec-Karte enthalten.
- Das EF_RAND muß die Datei-ID '0005' besitzen.

Der 8 Byte lange, kartenindividuelle Schlüssel wird bei der Personalisierung in das EF_RAND geschrieben.

6.12.1. FCP

Für das EF_RAND sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 08’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 08’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 05'	Datei-ID des EF_RAND
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_RAND

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_RAND enthält maximal einen Record.

Tag ’82’

Das EF_RAND ist ein lineares EF, dessen Recordlänge auf 8 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_RAND muß '00 05' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_RAND die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt.

6.12.2. Daten

Das EF_RAND enthält einen Record der Länge 8 Byte, in dem ein binär kodierter 8 Byte Wert abgelegt ist.

6.13. EF_VERSION

EF_VERSION bezeichnet ein lineares EF, das optional im MF der ec-Karte enthalten ist. In diesem EF kann der Kartenherausgeber Informationen zur Software-Version der ec-Karte ablegen. Wenn das EF_VERSION in der ec-Karte enthalten ist, sind die folgenden Regeln zu beachten:

- Das EF_VERSION ist im MF der ec-Karte enthalten.
- Das EF_VERSION muß die Datei-ID '0017' besitzen.

6.13.1. FCP

Für das EF_VERSION sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		
’81’	’02’	’00 08’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 08’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 17'	Datei-ID des EF_VERSION
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_VERSION

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_VERSION enthält maximal einen Record.

Tag ’82’

Das EF_VERSION ist ein lineares EF, dessen Recordlänge auf 8 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_VERSION muß '00 17' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_VERSION die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt.

6.13.2. Daten

Das EF_VERSION enthält einen Record der Länge 8 Byte. Dieser Record wird von der ec-Karte nicht ausgewertet und kann durch den Kartenherausgeber belegt werden, um die aktuelle Software-Version der ec-Karte nachzuhalten.

7. Übertragungsprotokoll

Das Übertragungsprotokoll der Schnittstelle zwischen der ec-Karte mit Chip und dem Chipkartenterminal ist T=1 nach [ISO 4'] und [EMV 1], wobei [EMV 1] Vorrang vor [ISO 4'] hat, mit der zusätzlichen Vorgabe, daß IFSC der ec-Karte mindestens 60 Byte groß sein muß und daß Chaining für I-Blöcke mit LEN<IFSC nicht zulässig ist.

Die folgende Tabelle gibt den ATR der ec-Karte mit Chip an:

TS	T0	TB1	TC1	TD1	TD2	TA3	TB3	T1	T2	...	T15	TCK
’3F’ or ’3B’	’EF’	’00’	’FF’	’81’	’31’	’XX’	’45’	’65’ e	’63’ c	’XX’	’XX’	XOR of Bytes T0 to T15

TS: Inverse oder direkte Konvention

(high voltage = logical 1 oder low voltage = logical 0)
T0: TB1, TC1, TD1 folgen/15 Historical Bytes
TB1: Programmierspannung nicht erforderlich
TC1: Wartezeit zwischen Zeichen mindestens 11 etus
TD1: TA2 bis TC2 fehlen, TD2 vorhanden, Protokoll T=1
TD2: TA3 und TB3 vorhanden, TC3 und TD3 fehlen, Protokoll T=1
TA3: IFSC ist zwischen **60** und 254 (Wertebereich von ’3C’ bis ’FE’)
TB3: BWI = 4, CWI = 5
T1: ’65’
T2: ’63’
T3: IC manufacturer ID
T4: Manufacturer’s IC type ID
T5: Manufacturer’s ROM mask ID
T6: Embedder country code
T7: Embedder country code
T8: Embedder national registration ID
T9: Embedder national registration ID
T10: Chip series number
T11: Chip series number
T12: Chip series number
T13: Chip series number
T14: Manufacturer OS Version (major version number)
T15: Manufacturer OS Version (minor version number)
TCK: XOR über T0 bis T15.

Byte T3 bis T15 sind ebenfalls in den Antwortdaten des Kommandos READ_STATISTICS enthalten (vgl. [LIT 5]).

8. Kommandos

Für die ec-Karte wird zwischen Standardkommandos, Administrationskommandos und Ergänzungskommandos unterschieden.

Als **Standardkommandos** werden Inter Industry Commands nach ISO 7816-4 bezeichnet, die für den regulären Betrieb der ec-Karte zur Verfügung stehen.

Administrationskommandos sind solche, die zur Administration und Wartung der ec-Karte verwendet werden können. Sie dienen

- der Wartung von bereits geladenen Anwendungen,
- dem Nachladen und Einrichten von neuen Anwendungen und Kommandos sowie
- der Ersetzung von Kommandos, um Fehler beheben zu können.

Die für die ec-Karte definierten **Ergänzungskommandos** dienen der Abwicklung der Anwendungen electronic cash und elektronische Geldbörse des deutschen Kreditgewerbes. Sie sind in den Spezifikationsdokumenten [LIT 3], [LIT 4A] und [LIT 4B] beschrieben.

In diesem Kapitel der Spezifikation wird die Klartext-Kommando-Schnittstelle für die Standard- und Administrationskommandos der ec-Karte beschrieben. In Kapitel 3. (Secure Messaging) wird das Verfahren zur Berechnung eines ggf. in Kommando- oder Antwortnachricht enthaltenen MAC definiert.

In Kapitel 3.2. wird der Aufbau einer zusätzlich verschlüsselten Kommando- oder Antwortnachricht erläutert.

Die Standardkommandos READ RECORD, UPDATE RECORD, VERIFY und die Administrationskommandos der Gruppe ADMIN können sowohl mit als auch ohne Secure Messaging verwendet werden. Für diese Kommandos zeigt das CLA-Byte '04' bzw. 'B4' an, daß Secure Messaging verwendet wird. Das CLA-Byte '00' bzw. 'B0' zeigt an, daß kein Secure Messaging verwendet wird.

Wenn für UPDATE RECORD oder ADMIN (APPEND RECORD) und ein EF eine AC vom Typ PRO oder ENC gesetzt ist, muß CLA = '04' verwendet werden. Andernfalls wird das jeweilige Kommando abgewiesen. Wenn für UPDATE RECORD oder ADMIN und ein EF keine AC vom Typ PRO oder ENC gesetzt ist, muß das entsprechende Kommando mit CLA = '00' auf das EF zugreifen, sonst wird es abgewiesen.

Analoges gilt für den Zugriff mittels VERIFY auf ein EF_PWDx und die Ausführung der Kommandos CREATE FILE, DELETE FILE, INCLUDE und EXCLUDE in einem DF.

Wenn für READ RECORD und ein EF eine AC vom Typ ENC gesetzt ist, darf mit READ RECORD nur mit CLA = '04' auf dieses EF zugegriffen werden. Andernfalls wird das Kommando abgewiesen. Wenn für READ RECORD und ein EF keine AC vom Typ PRO oder ENC gesetzt ist, muß das Kommando mit CLA = '00' auf das EF zugreifen, sonst wird es abgewiesen. Nur wenn für READ RECORD und ein EF eine AC vom Typ PRO gesetzt ist, akzeptiert die Karte sowohl CLA = '00' als auch CLA = '04', weist aber alle anderen Werte von CLA zurück.

Die übrigen Standardkommandos akzeptiert die ec-Karte nur ohne Secure Messaging, also nur mit CLA = '00'.

LOAD COMMAND akzeptiert die ec-Karte nur mit Secure Messaging, also nur mit CLA = 'B4'.

Die folgende Tabelle gibt einen Überblick über die Standardkommandos.

Kommando	CLA	INS	Kurzbeschreibung
READ RECORD	'00' '04'	'B2'	Lesen eines Records aus einem EF ohne oder mit Secure Messaging der ec-Karte
UPDATE RECORD	'00' '04'	'DC'	Ändern eines Records in einem EF ohne oder mit Secure Messaging der ec-Karte
PUT DATA	'00'	'DA'	Übergabe von Daten an die Karte
SELECT FILE	'00'	'A4'	Selektieren einer Applikation
VERIFY	'00' '04'	'20'	PIN-Prüfung ohne oder mit Secure Messaging der ec-Karte
INTERNAL AUTHENTICATE	'00'	'88'	Authentikation der Karte oder Applikation gegenüber der externen Welt
EXTERNAL AUTHENTICATE	'00'	'82'	Authentikation der externen Welt gegenüber der Karte
GET CHALLENGE	'00'	'84'	Generieren und Ausgabe einer Zufallszahl

Die folgende Tabelle gibt einen Überblick über die Administrationskommandos.

Kommando	CLA	INS	Kurzbeschreibung
APPEND RECORD	'00' '04'	'E2'	Hinzufügen eines Records in einem EF ohne oder mit Secure Messaging
CREATE FILE	'B0' 'B4'	'E0'	Erstellen eines DF oder EF ohne oder mit Secure Messaging
DELETE FILE	'B0' 'B4'	'E4'	Löschen eines DF oder EF ohne oder mit Secure Messaging
INCLUDE	'B0' 'B4'	'E6'	Zuordnung eines EF zu einer Applikation ohne oder mit Secure Messaging
EXCLUDE	'B0' 'B4'	'E8'	Aufhebung der Zuordnung eines EF zu einer Applikation ohne oder mit Secure Messaging
LOAD COMMAND	'B4'	'EA'	Laden eines Ergänzungskommandos mit Secure Messaging

8.1. Sicherheitsattribute der Kommandos

Wenn in der Spezifikation für ein Kommando und eine Datei eine AC vorgesehen ist, darf das Kommando nur dann auf die Datei zugreifen, wenn die AC vorhanden, korrekt kodiert ist und beide in die zwei Byte der AC kodierten Basis-ACs erfüllt sind. Bei der Ausführung von Standard- und Administrationskommandos wird hierzu zuerst das linke Byte und anschließend das rechte Byte der AC ausgewertet.

Wenn für ein Kommando keine AC gesetzt ist, obwohl dies erforderlich wäre, wenn die Kodierung der AC unzulässig oder fehlerhaft ist oder wenn eine der beiden Basis-ACs nicht erfüllt ist, wird das Kommando demnach abgelehnt.

Im folgenden werden die Regeln für die Kodierung von ACs noch einmal zusammengefaßt:

Jede AC wird in 2 Byte binär kodiert, so daß immer kombinierte ACs darzustellen sind. Hierbei müssen die Basis-ACs gemäß den Regeln aus Kapitel 4.2.8. in ein Byte kodiert sein:

- Wenn das linke Halbbyte einen Wert zwischen '4' und '9' hat, muß das rechte Halbbyte einen Wert zwischen '0' und '4' oder den Wert 'F' haben.

- Wenn das linke Halbbyte einen Wert zwischen '2' und '3' hat, muß das rechte Halbbyte einen Wert zwischen '0' und '1' haben.

- Wenn das linke Halbbyte einen der Werte '0' oder 'F' hat, muß das rechte Halbbyte den Wert '0' haben.

Bei der Kodierung der ACs für Standard- und Administrationskommandos sind zusätzlich die folgenden Regeln zu beachten:

- Eine AC vom Typ PRO oder ENC darf keine Schlüsselgruppe referenzieren, d. h. die Byte '4X', '5X, '6X', '7X' mit X = '4' oder X = 'F' sind nicht erlaubt.

- Kombinationen vom Typ PRO,PRO oder ENC,ENC oder PRO,ENC oder ENC,PRO. d. h. Kombinationen von zwei der Byte '4X', '5X', '6X', '7X' sind nicht erlaubt.

In den folgenden Tabellen wird eine Übersicht über implizite und explizite Sicherheitsattribute der Standard- und Administrationskommandos gegeben.

Standardkommandos

Kommando	Sicherheitsattribute
READ RECORD	Zugriff nur auf EFs. AC des EF für READ RECORD ist einzuhalten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Antwortdaten gemäß CLA-Byte AC ENC: MAC u. Verschlüsselung der Antwortdaten
UPDATE RECORD	Zugriff nur auf EFs. AC des EF für UPDATE RECORD ist einzuhalten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
PUT DATA	KEINE
SELECT FILE	KEINE
VERIFY	Zugriff nur auf EF_PWD0 und EF_PWD1. Zugehöriger FBZ im EF_FBZ bzw. EF_PWDD1 muß größer 0 sein. AC des EF PWDx für VERIFY ist einzuhalten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
INTERNAL AUTHENTICATE	KEINE
EXTERNAL AUTHENTICATE	Nur unmittelbar nach GET CHALLENGE.
GET CHALLENGE	KEINE

Administrationskommandos

Kommando	**Sicherheitsattribute**
CREATE FILE	Darf nicht auf existierende Dateien angewendet werden. Eine Datei kann nur im selektierten DF erstellt werden. AC des selektierten DF für ADMIN ist zu beachten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
DELETE FILE	MF kann nicht gelöscht werden. Nur leere Dateien können gelöscht werden Um eine Datei zu löschen muß das übergeordnete DF selektiert sein. Ein zu löschendes EF darf zusätzlich selektiert sein. AC des selektierten DF für ADMIN ist zu beachten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
APPEND RECORD	Zugriff ist nur auf EFs möglich. AC des EF für ADMIN ist zu beachten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
INCLUDE	AC des selektierten DF für ADMIN ist zu beachten: AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
EXCLUDE	SFI muß im selektierten DF definiert sein. AC des selektierten DF für ADMIN ist zu beachten. AC PWD: vorherige Paßwort-Prüfung AC AUT: vorherige externe Authentikation AC PRO: MAC über Kommandonachricht AC ENC: MAC u. Verschlüsselung der Kommandonachricht
LOAD COMMAND	Implizit abgesichert über AC PRO_G '00', d.h. es ist eine MAC-Bildung über die Kommandodaten erforderlich mit dem K_{CARD} im EF_KEY des MF.

8.2. Kommandostruktur

Die ec-Karte akzeptiert Kommando-APDUs entsprechend ISO 7816-4, 5.3.1:

Header				Body		
CLA	INS	P1	P2	L_c	Data	L_e

L_c und L_e haben immer eine Länge von 1 Byte. Dies bedeutet, daß L_c Werte von 1 bis 255 und L_e Werte von 1 bis 256 annehmen kann, wobei der Wert 256 durch '00' dargestellt wird. L_c gibt die Byte-Länge der Kommandodaten an. L_e enthält die erwartete Byte-Länge der Antwortdaten.

Eine Response-APDU der ec-Karte hat den Aufbau (ISO 7816-4, 5.3.3):

Data	SW1	SW2

Die beiden Byte SW1-SW2 werden im folgenden als Returncode des Kommandos bezeichnet.

Die tatsächliche Byte-Länge der Antwortdaten wird mit L_a bezeichnet. L_a kann Werte von 1 bis 256 annehmen, wobei der Wert 256 durch '00' dargestellt wird. Müßten mehr als 256 Byte Antwortdaten ausgegeben werden, wird das Kommando mit einem negativen Returncode beendet.

Abweichend von ISO 7816-4 wird ein Kommando nicht immer durch die ec-Karte abgewiesen, wenn L_e verschieden von L_a ist. Es wird nur dann abgewiesen, wenn L_e vorhanden ist, obwohl keine Antwortdaten zurückgegeben werden sollen, oder wenn L_e fehlt, obwohl Antwortdaten zurückgegeben werden sollen. Die Antwortdaten werden sonst bei $L_e \neq L_a$ mit dem positiven Returncode '61' L_a zurückgegeben.

8.3. Returncodes

Die folgende Tabelle gibt einen Überblick über die Returncodes, die von den Kommandos aufgrund der bereits in den vorherigen Kapiteln zu **Sicherheitsmechanismen** und zur **Sicherheitsarchitektur** der ec-Karte beschriebenen Fehlerfälle zurückgegeben werden müssen.

SW1	SW2	Bedeutung	Kapitel

’66’	’01’	keine gültige Zufallszahl vorhanden	2.7.
'66'	'02'	kein ICV mit PUT DATA übergeben	2.7.
'66'	'03'	keine AC gesetzt	4.3.
'66'	'04'	unzulässige oder fehlerhaft kodierte AC	4.2.7., 4.2.8., 5., 8.1.
'66'	'05'	Secure Messaging in CLA falsch	8.
'66'	'11'	durch eine AC vom Typ PRO oder ENC für ein Kommando und eine Datei referenzierter Schlüssel nicht gefunden	2.8., 4.2.4.
'66'	'12'	Paritätsfehler eines Schlüssels	2.8.2.
'66'	'13'	Zusatzinformationen (EF_KEYD, EF_AUTD) zu einem Schlüssel nicht gefunden	2.8.2.
'66'	'14'	der Fehlbedienungszähler eines durch eine AC vom Typ PRO oder ENC für ein Kommando und eine Datei referenzierten Schlüssels ist 0	2.8.2.
'66'	'15'	Schlüssellänge/Algorithmus-ID eines Schlüssel unzulässig	2.8.2., 4.1.1.2., 4.2.5., 6.3.2., 6.4.2.
'66'	'22'	Zusatzinformationen (EF_PWDDx, EF_FBZ) zu einem Paßwort nicht gefunden	2.9.2.
'66'	'81'	Kein Zugriff wegen AC NEV	4.2.2.
'66'	'82'	AC vom Typ AUT für ein Kommando und eine Datei nicht erfüllt	4.2.6.
'69'	'82'	AC vom Typ PWD für ein Kommando und eine Datei nicht erfüllt	4.2.3.
'69'	'83'	der Fehlbedienungszähler eines in der Kommandonachricht des EXTERNAL AUTHENTICATE angegebenen Schlüssels ist 0, oder der Fehlbedienungszähler eines in der Kommandonachricht des VERIFY angegebenen Paßworts ist 0	2.8.2., 4.1.1.2., 2.9.2.. 4.1.1.1.
'69'	'87'	Byte-Länge eines Chiffrats ist kein Vielfaches von 8, oder falsches Padding	3.2.1.

’69’	’88’	MAC falsch	3.1.1.
’6A’	’88’	ein in der Kommandonachricht von EXTERNAL AUTHENTICATE oder INTERNAL AUTHENTICATE angegebener Schlüssel nicht gefunden, oder ein in der Kommandonachricht von VERIFY angegebenes Paßwort nicht gefunden	2.8., 4.1.1.2., 6.4., 2.9., 4.1.1.1.

Die folgende Tabelle enthält den Returncode, der von den Kommandos VERIFY und EXTERNAL AUTHENTICATE zurückgegeben werden muß, wenn eine Authentikation fehlgeschlagen ist.

SW1	**SW2**	**Bedeutung**	**Kapitel**
'63'	'CX'	Authentikation mittels VERIFY oder EXTERNAL AUTHENTICATE nicht erfolgreich, Stand des FBZ nach Dekrementierung ist X	4.1.1.1., 4.1.1.2.

Die folgenden Returncodes werden verwendet, um die **erfolgreiche Ausführung** eines Kommandos anzuzeigen:

SW1	**SW2**	**Bedeutung**
’90’	’00’	OK
’61’	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

L_a kann Werte von 1 bis 256 annehmen kann, wobei der Wert 256 durch ’00’ dargestellt wird.

Der folgende Returncode zeigt an, daß ein Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene **Warnung** zu beachten ist:

SW1	**SW2**	**Bedeutung**
’63’	’CX’	Use of internal retry routine (Counter ’X’, valued from 0 to 15)

Der Returncode '63 CX' "Use of internal retry routine" gibt bei Kommandos, die Schreibvorgänge beinhalten (Ausnahmen: VERIFY und EXTERNAL AUTHENTICATE), an, daß die Karte einen Schreibvorgang mehrfach durchführen mußte, um das Kommando erfolgreich abzuschließen. Die Warnung weist auf interne Speicherprobleme der Karte hin und dient zu Diagnosezwecken.

Alle Kommandos, die die im folgenden vorgeschriebenen Prüfungen durchführen, geben im **Fehlerfall** die angegebenen Returncodes zurück.

SW1	**SW2**	**Bedeutung**
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'67'	'00'	Wrong length wird ausgegeben, wenn der Wert von L_c falsch ist, oder wenn L_e fehlt, obwohl es vorhanden sein muß, oder wenn L_e vorhanden ist, obwohl es fehlen muß
'67'	'81'	Antwortdaten länger als 256 Byte
'69'	'81'	Command incompatible with file organisation
'69'	'85'	Conditions of use not satisfied
'69'	'86'	Command not allowed (no current EF) wird ausgegeben, wenn kein EF selektiert ist, obwohl auf das aktuelle EF zugegriffen werden soll
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn das Format der Kommandodaten nicht der Spezifikation entspricht oder Kommandodaten außerhalb des spezifizierten Wertebereichs liegen (keine Längenfehler der gesamten Kommandodaten)
'6A'	'82'	File not found wird ausgegeben, wenn eine Datei, auf die das Kommando zugreifen will, nicht vorhanden ist
'6A'	'83'	Record not found wird ausgegeben, wenn auf einen Record mit Recordnummer größer als die des LAST Record zugegriffen werden soll
'6A'	'84'	Not enough memory space
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

Die Beschreibungen der einzelnen Kommandos enthalten Tabellen mit Returncodes, in denen aufgelistet wird, welche der oben genannten Returncodes das Kommando zurückgeben kann und wie sie kommandospezifisch zu interpretieren sind.

8.4. Standardkommandos

8.4.1. READ RECORD

8.4.1.1. Übersicht

Mit dem Kommando READ RECORD wird ein Record aus einem EF linearer oder zyklischer Struktur gelesen. Mit einem Kommandoaufruf wird der in P1 angegebene Record des in P2 angegebenen EF ausgelesen.

Zur Referenzierung des EF, aus dem der Record gelesen wird, muß in P2 das aktuell selektierte EF oder ein SFI angegeben werden. Wird das aktuell selektierte EF angegeben obwohl kein EF selektiert ist, wird das Kommando mit dem Returncode '69 86' abgebrochen. Wird ein SFI angegeben, durch den für das selektierte DF kein AEF referenziert wird, wird das Kommando mit dem Returncode '6A 82' abgebrochen.

Die Ausführung des Kommandos wird durch die für das referenzierte EF und READ RECORD gesetzte AC und das CLA-Byte gesteuert.

Ist eine AC vom Typ PRO festgelegt und CLA = '04', ist über die Antwortdaten ein MAC mit dem durch die AC referenzierten Schlüssel (vgl. Kapitel 4.2.1) durch die Karte zu berechnen und an die Antwortdaten anzuhängen. Die MAC-Berechnung erfolgt gemäß Kapitel 3.1.2. Ist eine AC vom Typ PRO festgelegt und wird CLA = '00' verwendet, wird auf die MAC-Sicherung der Antwortdaten verzichtet.

Eine AC vom Typ ENC bedeutet, daß über die Antwortdaten ein MAC zu berechnen und an die Antwortdaten anzuhängen ist und anschließend die Antwortnachricht zu verschlüsseln ist. Für beide Operationen wird der durch die AC referenzierte Schlüssel verwendet. Die MAC-Berechnung und die Verschlüsselung erfolgt gemäß Kapitel 3.2.2.. Bei einer AC vom Typ ENC wird das Kommando READ RECORD bei Verwendung von CLA = '00' mit dem Returncode '66 05' abgebrochen.

Der ICV für die MAC-Bildung bzw. Verschlüsselung muß zuvor mittels des Kommandos PUT DATA an die Karte übergeben werden. Andernfalls wird das Kommando mit dem Returncode '66 02' abgebrochen. Der mittels PUT DATA übergebene ICV darf nur einmal verwendet werden.

Wenn keine AC vom Typ PRO oder ENC für das referenzierte EF und READ RECORD gesetzt ist, wird das Kommando bei Verwendung von CLA = '04' mit dem Returncode '66 05' abgelehnt.

Ist für das EF und READ RECORD eine AC vom Typ PWD festgelegt, ist die Ausführung von READ RECORD an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von READ RECORD an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC

referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu verwenden ist.

Müßten mehr als 256 Byte Antwortdaten ausgegeben werden, beispielsweise bei einer Recordlänge von mehr als 248 Byte und einer AC vom Typ PRO oder ENC, wird das Kommando mit dem Returncode '67 81' abgelehnt.

8.4.1.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	'00' bzw. '04'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'B2'	Kommandocode
P1	1	'XX'	Recordnummer
P2	1	'XX'	Reference Control Byte
L_c	0	-	Keine Kommandodaten
Data	0	-	Keine Kommandodaten
L_e	1	'XX'	Recordlänge (ggf. + 8 Byte für MAC)

P1

Der Parameter P1 gibt die Recordnummer des Records an, der gelesen werden soll. Die Recordnummern '00' und 'FF' dürfen nicht verwendet werden.

P2

Im Parameter P2 ist das Referenzkontrollbyte in folgender Weise kodiert:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	0	-	-	-	aktuell selektiertes EF
x	x	x	x	x	-	-	-	Short File Identifier (SFI) zwischen '01' und '1E'
-	-	-	-	-	1	0	0	Referenzierung durch Recordnummer in P1

8.4.1.3. Antwortnachricht

Das Kommando READ RECORD gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	X	'XX..XX'	Recorddaten, ggf. MAC
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Antwortdaten:

Die Klartext-Antwortdaten haben folgenden Aufbau:

Record	MAC über den Record (falls CLA = '04' und AC entsprechend gesetzt)

Returncodes

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'02'	kein ICV mit PUT DATA übergeben
'66'	'03'	keine AC für READ RECORD und das angegebene EF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für READ RECORD und das angegebene EF gesetzt
'66'	'05'	AC vom Typ ENC für READ RECORD und das angegebene EF gesetzt, aber CLA = '00' oder AC nicht vom Typ PRO oder ENC und CLA = '04'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für READ RECORD und das angegebene EF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für READ RECORD und das angegebene EF referenzierten Schlüssels
'66'	'13'	Zusatzinformationen (EF_KEYD) zu einem durch eine AC vom Typ PRO oder ENC für READ RECORD und das angegebene EF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für READ RECORD und das angegebene EF referenzierten Schlüssels ist 0

SW1	SW2	Bedeutung
’66’	’15’	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für READ RECORD und das angegebene EF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für READ RECORD und das angegebene EF gesetzt
'66'	'82'	AC vom Typ AUT für READ RECORD und das angegebene EF nicht erfüllt
'67'	'00'	Wrong length (Byte-Länge der Kommandonachricht ≠ 5)
'67'	'81'	Antwortdaten zu lang
'69'	'82'	AC vom Typ PWD für READ RECORD und das angegebene EF nicht erfüllt
'69'	'86'	Command not allowed (no current EF) wird ausgegeben, wenn kein EF selektiert ist, obwohl auf das aktuelle EF zugegriffen werden soll
'6A'	'82'	File not found wird ausgegeben, wenn durch einen angegebenen SFI für das aktuelle DF kein AEF referenziert wird
'6A'	'83'	Record not found wird ausgegeben, wenn auf einen Record mit Recordnummer größer als die des LAST Record zugegriffen werden soll
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.2. UPDATE RECORD

8.4.2.1. Übersicht

Mit dem Kommando UPDATE RECORD erfolgt ein Update von Datenfeldern mit linearer oder zyklischer Struktur. Mit einem Kommandoaufruf kann genau ein Record des jeweiligen Datenfeldes verändert werden. Es ist jeweils genau die für ein EF festgelegte Recordlänge einzuhalten. Dies bedeutet, daß

- L_c die Recordlänge enthalten muß, wenn keine AC vom Typ PRO oder ENC für das EF gesetzt ist,
- L_c die Recordlänge + 8 enthalten muß, wenn eine AC vom Typ PRO oder ENC für das EF gesetzt ist, und
- L_c immer die Länge der Kommandodaten enthalten muß und kein L_e vorhanden sein darf.

Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen.

Mit dem Kommando UPDATE RECORD können einem Datenfeld **keine neuen Records** hinzugefügt werden, dies kann nur mittels APPEND RECORD erfolgen.

Zur Referenzierung der Datei, in die der Record geschrieben werden soll, wird in P2 das aktuell selektierte EF oder ein SFI angegeben. Wird das aktuell selektierte EF angegeben, obwohl kein EF

selektiert ist, wird das Kommando abgebrochen, und es wird der Returncode ’69 86’ zurückgegeben. Wird ein SFI angegeben, durch den für das selektierte DF kein AEF referenziert wird, wird der Returncode '6A 82' ausgegeben.

Zur Referenzierung des Records wird in P1 die Recordnummer angegeben.

Die Kommandoausführung bedingt die Erfüllung der AC für das EF, das in P2 angegeben wird, und UPDATE RECORD. Im Falle einer AC vom Typ PRO bedeutet dies, daß ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen und an die Kommandodaten anzuhängen ist.

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 mit einem MAC versehen und verschlüsselt wird.

Der ICV für die MAC-Bildung bzw. Verschlüsselung muß in beiden Fällen zuvor mittels des Kommandos GET CHALLENGE von der Karte geholt worden sein. Andernfalls wird das Kommando mit dem Returncode '66 01' abgebrochen. Der ICV darf nur einmal verwendet werden.

Wenn eine AC vom Typ PRO oder ENC für das referenzierte EF und UPDATE RECORD gesetzt ist, wird das Kommando bei Verwendung von CLA = '00' mit dem Returncode '66 05' abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das referenzierte EF und UPDATE RECORD gesetzt ist, wird das Kommando bei Verwendung von CLA = '04' mit dem Returncode '66 05' abgelehnt.

Ist für das EF eine AC vom Typ PWD festgelegt, ist die Ausführung von UPDATE RECORD an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von UPDATE RECORD an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu verwenden ist.

8.4.2.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'00' bzw. '04'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'DC'	Kommandocode
P1	1	'XX'	Recordnummer
P2	1	'XX'	Reference Control Byte
L_c	1	'XX'	Recordlänge (ggfs. + 8 Byte für MAC)
Data	X	'XX..XX'	Record optional MAC über die Kommandonachricht (Länge 8 Byte)
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 gibt die Recordnummer an. Die Recordnummern '00' und 'FF' dürfen nicht verwendet werden.

P2

Im Parameter P2 ist das Referenzkontrollbyte kodiert, das die Datei- und die Record-Referenzierung angibt, entsprechend der folgenden Tabelle:

Referenzkontrollbyte:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	0	-	-	-	aktuell selektiertes EF
x	x	x	x	x	-	-	-	Short File Identifier (SFI) von '01' bis '1E'
-	-	-	-	-	1	0	0	Referenzierung durch Recordnummer

Data

Data enthält die Recorddaten und ggf. den angehängten MAC über die Kommandonachricht.

8.4.2.3. Antwortnachricht

Das Kommando UPDATE RECORD gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
’63’	’CX’	Use of internal retry routine (Counter ’X’, valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
’64’	’00’	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für UPDATE RECORD und das angegebene EF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für UPDATE RECORD und das angegebene EF gesetzt
'66'	'05'	AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF gesetzt, aber CLA = '00' oder AC nicht vom Typ PRO oder ENC und CLA = '04'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF referenzierter Schlüssel nicht gefunden

SW1	SW2	Bedeutung
’66’	’12’	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF referenzierten Schlüssels
'66'	'13'	Zusatzinformationen (EF_KEYD) zu einem durch eine AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für UPDATE RECORD und das angegebene EF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für UPDATE RECORD und das angegebene EF gesetzt
'66'	'82'	AC vom Typ AUT für UPDATE RECORD und das angegebene EF nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)
'69'	'82'	AC vom Typ PWD für UPDATE RECORD und das angegebene EF nicht erfüllt
'69'	'86'	Command not allowed (no current EF) wird ausgegeben, wenn kein EF selektiert ist, obwohl auf das aktuelle EF zugegriffen werden soll
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch
'6A'	'82'	File not found wird ausgegeben, wenn durch einen angegebenen SFI für das aktuelle DF kein AEF referenziert wird
'6A'	'83'	Record not found wird ausgegeben, wenn auf einen Record mit Recordnummer größer als die des LAST Record zugegriffen werden soll
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.3. PUT DATA

8.4.3.1. Übersicht

Das Kommando PUT DATA wird verwendet, um der ec-Karte den ICV für eine MAC-Berechnung oder Verschlüsselung mitzuteilen, die sie bei der Ausführung des nächsten Kommandos durchzuführen hat. Die Karte speichert den übergebenen Wert. Der gespeicherte Wert ist vom Zeitpunkt der Übergabe durch PUT DATA bis zu der Beendigung der Ausführung des nächsten Kommandos, das die ec-Karte nach der Übergabe des Wertes erhält, ***gültig***. Dann muß der Wert ***ungültig*** gesetzt werden, auch wenn er bei der Ausführung nicht verwendet wird. Auf diese Weise wird sichergestellt, daß ein mit PUT DATA übergebener Wert nur bei der Durchführung desjenigen Kommandos als ICV verwendet werden kann, das unmittelbar auf die Übergabe folgt.

Das Kommando PUT DATA ist immer ausführbar.

8.4.3.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	’00’	Kommandoklasse ohne Secure Messaging
INS	1	’DA’	Kommandocode
P1	1	’01’	applikationsabhängige Definition der Kommandodaten
P2	1	'00'	Daten werden als ICV übergebenen.
L_c	1	'08'	Länge der übergebenen Daten
Data	8	'XX..XX'	Daten
L_e	0	-	keine Antwortdaten

P1

Der Wert '01' bedeutet, daß die Definition der durch PUT DATA übergebenen Daten applikationsabhängig ist. Die Definition wird durch P2 festgelegt.

P2

Der Wert '00' bedeutet, daß die Daten als ICV für die CFB-MAC-Berechnung bzw. CFB-MAC-Berechnung und CBC-Verschlüsselung bei der Ausführung von Kommandos übergeben werden.

8.4.3.3. Antwortnachricht

Das Kommando PUT DATA gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
	0	-	keine Antwortdaten
	1	'XX'	Statusbyte 1
	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde
'66'	'05'	CLA = '04'
'67'	'00'	Wrong length ($L_c \neq$ '08'; L_e vorhanden)
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.4. SELECT FILE

8.4.4.1. Übersicht

Das Kommando SELECT FILE dient zur Selektion eines Verzeichnisses (DF) oder eines Datenfeldes (EF) und der Ausgabe der entsprechenden Datei-Kontrollinformation.

ACs sind für dieses Kommando nicht zu erfüllen. Durch SELECT FILE kann jedoch der globale oder DF-spezifische Sicherheitszustand der ec-Karte verändert werden (vgl. Kapitel 4.1.1.). Durch die Selektion eines ADF wird der entsprechende Applikationskontext geöffnet (vgl. Kapitel 1.3.). Durch ein SELECT FILE mit DF-Namen werden alle Sicherheitszustände des neu selektierten DFs und die aller Nachfolger-DFs gelöscht.

Durch den Parameter P2 wird gesteuert, ob als Datei-Kontrollinformation die FCI, die FCP oder die FMD (vgl. Kapitel 5.) in der Antwortnachricht zurückgegeben werden sollen. Es kann ebenfalls angegeben werden, daß keine Datei-Kontrollinformation gewünscht ist.

Falls P1 = '00' angegeben wird, so wird das MF selektiert, und es wird die Datei-Kontrollinformation gemäß der Option in P2 ausgegeben. Die Kommandonachricht enthält in diesem Fall keine Daten.

Falls P1 = '01', '02' oder '03' ist, so wird je nach angegebener Datei-ID ein DF oder ein EF selektiert, und es wird die Datei-Kontrollinformation gemäß der Option in P2 ausgegeben. Dabei werden folgende Fälle unterschieden:

P1 = '01' : Die Kommandodaten enthalten die Datei-ID eines DF, und die Karte selektiert das dem aktuellen DF untergeordnete DF mit dieser ID. Falls im aktuellen DF kein DF mit der angegebenen Datei-ID gefunden wird, so wird dies mit dem Returncode '6A 82' mitgeteilt.

P1 = '02': Die Kommandodaten enthalten die Datei-ID eines EF, und die Karte selektiert das EF im aktuellen DF mit dieser ID. Falls im aktuellen DF kein EF mit der angegebenen Datei-ID gefunden wird, so wird dies mit dem Returncode '6A 82'

mitgeteilt.

P1 = ’03’: Die Kommandodaten sind leer, und die Karte selektiert das dem aktuellen DF übergeordnete DF. Falls das MF aktuell ist, wird dies mit dem Returncode '6A 82' mitgeteilt.

Falls P1 = '04' ist, enthalten die Kommandodaten einen DF-Namen. Ein DF-Name kann eine AID sein, bestehend aus RID und PIX (vgl. Kapitel 1.2.2.).

Bei Angabe eines DF-Namens selektiert die ec-Karte das angegebene DF, falls dieses existiert, und gibt die Datei-Kontrollinformation gemäß der Option in P2 aus. Andernfalls antwortet die ec-Karte mit dem Returncode '6A 82'.

Das Ergebnis der Längenprüfung hängt von den Werten von P1 und P2 ab. Für P1 = '00' und '03' darf kein L_c vorhanden sein. Für P1 = '01', '02' muß L_c = '02' sein und der Länge der Kommandodaten entsprechen. Für P1 = '04' muß L_c zwischen '01' und '10' liegen und der Länge der Kommandodaten entsprechen. Für P2 = '00', '04', '08' muß L_e vorhanden sein. Für P2 = '0C' darf L_e nicht vorhanden sein. Andernfalls wird der Returncode '67 00' ausgegeben.

L_e, insbesondere L_e = '00', wird wie in Kapitel 8.2. beschrieben behandelt. Wenn die tatsächliche Länge L_a der Antwortdaten L_e entspricht, wird der Returncode '90 00' zurückgegeben. Andernfalls wird L_a durch den Returncode '61 L_a' angezeigt.

Müßten mehr als 256 Byte Antwortdaten ausgegeben werden, wird das Kommando mit dem Returncode '67 81' abgelehnt.

Wenn das Kommando nicht erfolgreich ausgeführt werden kann,

- muß das bei Aufruf des Kommandos aktuelle DF selektiert bleiben,
- muß ein bei Kommandoaufruf eventuell aktuelles EF selektiert bleiben und
- dürfen sich die Sicherheitszustände der ec-Karte nicht ändern.

8.4.4.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	’00’	Kommandoklasse ohne Secure Messaging
INS	1	’A4’	Kommandocode
P1	1	’0X’	Selection Control Byte
P2	1	’0X’	Selection Option Byte
L_c	0 oder 1	- oder ’01’ bis ’10’	Länge der Kommandodaten
Data	0 bis 16	- oder 'XX..XX'	Kommandodaten
L_e	0 oder 1	- oder 'XX' (auch '00')	nicht vorhanden oder erwartete Länge der Datei-Kontrollinformation

P1

Die Kodierung des Selection Control Byte ist in der folgenden Tabelle definiert:

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**Bedeutung**
0	0	0	0	0	0	x	x	Selektion durch Datei-ID
-	-	-	-	-	-	0	0	Selektiere MF
-	-	-	-	-	-	0	1	Selektiere untergeordnetes DF
-	-	-	-	-	-	1	0	Selektiere EF im aktuellen DF
-	-	-	-	-	-	1	1	Selektiere das übergeordnete DF
0	0	0	0	0	1	0	0	Selektion durch DF-Namen

P2

Die Kodierung des Selection Option Byte ist in der folgenden Tabelle definiert:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	x	x	0	0	Option für Datei-Kontrollinformation
-	-	-	-	0	0	-	-	Ausgabe der FCI als Antwortdaten
-	-	-	-	0	1	-	-	Ausgabe der FCP als Antwortdaten
-	-	-	-	1	0	-	-	Ausgabe der FMD als Antwortdaten
-	-	-	-	1	1	-	-	Keine Antwortdaten

Data

Die Kommandodaten können folgende Referenzen auf Dateien beinhalten:

Referenz	Länge
DF-Name (AID)	1 bis 16 Byte
Datei-ID	2 Byte
leer	0 Byte

8.4.4.3. Antwortnachricht

Das Kommando SELECT FILE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	X	- oder 'XX..XX'	Datei-Kontrollinformation
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Es wird die Datei-Kontrollinformation wie in Kapitel 5. beschrieben zurückgegeben.

Returncodes

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der Ausführung ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'05'	CLA = '04'
'67'	'00'	Wrong length (L_c vorhanden oder falsch; L_e nicht vorhanden; L_e vorhanden)
'67'	'81'	Antwortdaten zu lang
'6A'	'82'	File not found wird ausgegeben, wenn die Datei, auf die das Kommando zugreifen soll, nicht vorhanden ist
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.5. VERIFY

8.4.5.1. Übersicht

Mit dem Kommando VERIFY kann sich der Karteninhaber gegenüber der ec-Karte oder einer Applikation durch Angabe eines Paßwortes authentisieren. Im Parameter P2 wird angegeben, ob das in den Kommandodaten übergebene Paßwort

- global (Bit 8 = 0) oder DF-spezifisch (Bit 8 = 1) ist und
- die Paßwortnummer 0 (Bit 1 = 0) oder 1 (Bit 1 = 1) hat.

Durch den in Kapitel 2.9. beschriebenen Mechanismus zur Paßwortauswahl wird mit den Parametern

- "globales/DF-spezifisches Paßwort" aus P2,
- "Paßwortnummer x" aus P2 und
- bei Aufruf von VERIFY aktuelles DF

das durch das Kommando referenzierte Paßwort gesucht.

Wenn das referenzierte Paßwort nicht gefunden wird, wird das Kommando mit dem Returncode '6A 88' abgebrochen.

Wenn das Paßwort gefunden wird, wird geprüft, ob der zugehörige Fehlbedienungszähler (FBZ) verschieden von 0 ist. Der Fehlbedienungszähler befindet sich für ein Paßwort in einem EF_PWD0 im EF_FBZ im MF und für ein Paßwort in einem EF_PWD1 im zugehörigen EF_PWDD1 in demselben DF. Wird das EF_FBZ bzw. das EF_PWDD1 nicht gefunden oder enthält es den FBZ nicht, wird das Kommando mit dem Returncode '66 22' abgebrochen. Hat der FBZ den Wert 0, wird

das Kommando mit dem Returncode ’69 83’ abgebrochen.

Anschließend wird die für das EF_PWDx, in dem das Paßwort abgelegt ist, und das Kommando VERIFY gesetzte AC ausgewertet.

Wenn eine AC vom Typ PWD und/oder AUT für das EF_PWDx und VERIFY gesetzt ist, wird geprüft, ob diese AC erfüllt ist.

Ist eine AC vom Typ PRO gesetzt, bedeutet dies, daß durch das Terminal ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden (vgl. Kapitel 4.2.1).

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 durch das Terminal mit einem MAC zu versehen und zu verschlüsseln ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden.

In beiden Fällen muß vorher mit GET CHALLENGE von der Karte ein ICV angefordert worden sein, der in die MAC-Berechnung bzw. in die Verschlüsselung einzubeziehen ist.

Ist eine AC vom Typ PRO oder ENC gesetzt und wurde nicht unmittelbar zuvor das Kommando GET CHALLENGE ausgeführt, wird das Kommando mit dem Returncode '66 01' abgebrochen.

Andernfalls werden die kryptographischen Operationen ausgeführt, die zur Erfüllung der AC vom Typ PRO oder ENC erforderlich sind. Eventuell negative Ergebnisse des Secure Messaging bei AC vom Typ PRO oder ENC führen zum Abbruch mit dem entsprechenden Returncode. Die Längenprüfung für die Kommandodaten wird vor der MAC-Prüfung nach der durch eine AC vom Typ ENC vorgeschriebenen Entschlüsselung durchgeführt. Wenn L_c nicht den Wert '10' hat und/oder die Kommandodaten nicht 16 Byte lang sind und/oder L_e vorhanden ist, wird das Kommando mit dem Returncode '67 00' abgebrochen.

Ist **keine** AC vom Typ ENC oder PRO gesetzt, muß VERIFY mit CLA = '00' ohne Secure Messaging ausgeführt werden. Andernfalls wird das Kommando mit dem Returncode '66 05' abgebrochen. Wird VERIFY mit CLA = '00' verwendet, obwohl eine AC vom Typ ENC oder PRO gesetzt ist, wird das Kommando ebenfalls mit dem Returncode '66 05' abgelehnt.

Ist keine AC vom Typ PRO oder ENC gesetzt, werden die Kommandodaten nur einer Längenprüfung unterzogen. Wenn L_c nicht den Wert '08' hat und/oder die Kommandodaten nicht 8 Byte lang sind und/oder L_e vorhanden ist, wird das Kommando mit dem Returncode '67 00' abgebrochen.

Das in der Kommandonachricht übertragene Paßwort wird mit dem im EF_PWDx gespeicherten Referenzwert verglichen.

Verläuft dieser Vergleich positiv, so wird der Sicherheitszustand geändert, der dem MF bzw. DF zugeordnet ist, in dessen EF_PWDx sich das EF_PWDx befindet: Das der Paßwortnummer zugeordnete Flag des Sicherheitszustandes wird gesetzt.

Der dem Paßwort zugeordnete Fehlbedienungszähler im EF_FBZ bzw. im zugehörigen EF_PWDD1 wird auf den in demselben EF enthaltenen Initialwert gesetzt.

Geht die Prüfung negativ aus, wird das entsprechende Flag des Sicherheitszustandes zurückgesetzt und der dem Paßwort zugeordnete Fehlbedienungszähler um 1 dekrementiert. Danach wird der Returncode '63 CX' zurückgegeben. X ≥ 0 gibt hierbei den Wert des FBZ, wenn dieser kleiner als 16 ist; ansonsten wird in X der Wert 'F' zurückgegeben.

Bricht das Kommando aus einem anderen Grund als negativ verlaufenem Paßwortvergleich ab, wird der FBZ des Paßworts nicht dekrementiert.

PIN-Prüfung mittels des Kommandos VERIFY

Das Paßwort im EF_PWD0 des MF repräsentiert in der ec-Karte die verschlüsselt abgelegte PIN des deutschen Kreditgewerbes. Für das EF_PWD0 im MF ist die AC ENC_G '01' gesetzt. Die Datei EF_KEY im MF enthält unter der logischen Schlüsselnummer '01' den Schlüssel K_{PIN}, der zur MAC-Bildung und Verschlüsselung der VERIFY-Kommandonachricht verwendet wird. Der Fehlbedienungszähler für die PIN ist mit dem Wert 3 initialisiert, d.h. es sind maximal zwei fehlerhafte Eingaben erlaubt. Eine weitere fehlerhafte Eingabe setzt den FBZ auf Null, und VERIFY bricht mit dem Returncode '63 C0' ab. Jede weitere Eingabe von VERIFY endet mit dem Returncode '69 83'. Ein Zurücksetzen des Fehlbedienungszählers ist dann nur noch durch ein mit dem Schlüssel K_{Card} abgesichertes UPDATE des EF_FBZ im MF möglich.

8.4.5.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	'00' bzw. '04'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'20'	Kommandocode
P1	1	'00'	fester Wert
P2	1	'XX'	Reference Control Byte
L_c	1	'08' bzw. '10'	Länge Kommandodaten
Data	X	'XX..XX'	Paßwort, ggf. MAC über Kommandonachricht
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 ist immer auf ’00’ zu setzen.

P2

Im Parameter P2 ist das Referenzkontrollbyte kodiert, das die Auswahl DF-spezifisch/global des relevanten EF_PWDx und die Nummer x des EF_PWDx angibt. Die folgende Tabelle definiert die Kodierung für das Referenzkontrollbyte:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	-	-	-	-	-	-	-	Das Paßwort befindet sich im EF_PWDx des MF
1	-	-	-	-	-	-	-	Das Paßwort befindet sich in einem DF-spezifischen EF_PWDx
-	x	x	-	-	-	-	-	immer 0 0, sonst RFU.
-	-	-	0	0	0	0	0	Referenz von EF_PWD0
-	-	-	0	0	0	0	1	Referenz von EF_PWD1

Data

Data enthält das Paßwort und eventuell den angehängten MAC über die Kommandonachricht.

8.4.5.3. Antwortnachricht

Das Kommando VERIFY gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Authentication failed, Further allowed retries in 'X'

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
’64’	’00’	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für das referenzierte EF_PWDx und VERIFY gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für das referenzierte EF_PWDx und VERIFY gesetzt
'66'	'05'	AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx gesetzt, aber CLA = '00' oder AC nicht vom Typ PRO oder ENC und CLA = '04'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx referenzierter Schlüssel nicht gefunden

SW1	SW2	Bedeutung
’66’	’12’	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx referenzierten Schlüssels
'66'	'13'	Zusatzinformationen (EF_KEYD) zu einem durch eine AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für VERIFY und das angegebene EF_PWDx referenzierten Schlüssels ist unzulässig
'66'	'22'	Zusatzinformationen (EF_PWDDx, EF_FBZ) zu dem angegebenen Paßwort nicht gefunden
'66'	'81'	AC NEV für VERIFY und das angegebene EF_PWDx gesetzt
'66'	'82'	AC vom Typ AUT für VERIFY und das angegebene EF_PWDx nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)
'69'	'82'	AC vom Typ PWD für VERIFY und das angegebene EF_PWDx nicht erfüllt
'69'	'83'	FBZ des angegebenen Paßworts ist 0
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6A'	'88'	Angegebenes Paßwort nicht gefunden
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.6. INTERNAL AUTHENTICATE

8.4.6.1. Übersicht

Mit dem Kommando INTERNAL AUTHENTICATE kann sich die ec-Karte gegenüber der externen Welt authentisieren, indem sie einen in den Kommandodaten übergebenen 8 Byte langen Wert unter einem Authentikations-Schlüssel verschlüsselt in den Antwortdaten übergibt.

Zur Ausführung von INTERNAL AUTHENTICATE sind keine ACs zu erfüllen.

Im Parameter P2 wird angegeben, ob der von der Karte zu verwendende Authentikations-Schlüssel global (Bit 8 = 0) oder DF-spezifisch (Bit 8 = 1) ist.

In der Kommandonachricht wird außerdem festgelegt, welche Schlüsselnummer KID der zu verwendende Authentikations-Schlüssel hat. Ist die Nummer kleiner als 31, kann sie in die 5 niedrigwertigen Bit von P2 kodiert werden. Durch die Belegung der 5 niedrigwertigen Bit von P2 mit dem Wert 31 wird angezeigt, daß die Schlüsselnummer in das erste Byte der Kommandodaten kodiert ist.

Das Ergebnis der Längenprüfung hängt von dem Wert von P2 ab. Für P2 ≠ xxx11111 (binär) muß L_c = '08' sein und der Länge der Kommandodaten entsprechen. Für P2 = xxx11111 (binär) muß L_c = '09' sein und der Länge der Kommandodaten entsprechen. L_e muß immer vorhanden sein. Andernfalls wird der Returncode '67 00' ausgegeben.

Durch den in Kapitel 2.8. beschriebenen Mechanismus zur Schlüsselauswahl wird mit den Parametern

- "globaler/DF-spezifischer Schlüssel" aus P2,
- "KID" aus P2 bzw. Byte 1 der Kommandodaten und
- bei Aufruf von INTERNAL AUTHENTICATE aktuelles DF

der durch das Kommando referenzierte Authentikations-Schlüssel gesucht.

INTERNAL AUTHENTICATE bricht mit dem Returncode '6A 88' ab, wenn kein Schlüssel mit der angegebenen Nummer gefunden wird. Das Kommando bricht mit dem Returncode '66 13' ab, wenn keine Zusatzinformationen zu dem Schlüssel im zugehörigen EF_AUTD gefunden wird, es bricht mit dem Returncode '66 15' ab, wenn Schlüssellänge und Algorithmus-ID im zugehörigen EF_AUTD nicht konsistent sind, und es bricht mit dem Returncode '66 12' ab, wenn ein Paritätscheck negativ ausfällt.

Wenn der Mechanismus zur Schlüsselauswahl erfolgreich war, wird der in den Kommandodaten übergebene Wert mit dem gefundenen Authentikations-Schlüssel unter Verwendung des ICV '00..00' verschlüsselt und das Chiffrat in der Antwortnachricht an die externe Welt übergeben. Als Verschlüsselungsalgorithmus wird der für den Authentikations-Schlüssel durch die Algorithmus-ID im zugehörigen EF_AUTD festgelegte Algorithmus verwendet.

8.4.6.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	’00’	Kommandoklasse ohne Secure Messaging
INS	1	’88’	Kommandocode
P1	1	’00’	fester Wert
P2	1	’XX’	Reference Control Byte
L_c	1	’08’ bzw. ’09’	Länge der Kommandodaten, je nach Kodierung von P2
Data	8	'XX..XX'	Zufallszahl RND (8 Byte)
	9	'XX..XX'	logische Schlüsselnummer (1 Byte), Zufallszahl RND (8 Byte)
L_e	1	'08'	8 Byte verschlüsselte Zufallszahl.

P1

Der Parameter P1 ist immer auf ’00’ zu setzen.

P2

Der Parameter P2 (Referenzkontrollbyte) gibt den zu verwendenden Schlüssel an:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	-	-	-	-	-	-	-	Der Schlüssel befindet sich im EF_AUT des MF.
1	-	-	-	-	-	-	-	Der Schlüssel befindet sich in dem EF_AUT des schlüsselnummer-relevanten DF zum aktuellen DF
-	x	x	-	-	-	-	-	immer 0 0, sonst RFU
-	-	-	x	x	x	x	x	< '1 1 1 1 1': logische Schlüsselnummer 0 bis 30 = '1 1 1 1 1': logische Schlüsselnummer im Datenfeld

8.4.6.3. Antwortnachricht

Das Kommando INTERNAL AUTHENTICATE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	8	'XX..XX'	Autorisierungsdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'05'	CLA = '04'
'66'	'12'	Paritätsfehler des angegebenen Schlüssels
'66'	'13'	Zusatzinformationen (EF_AUTD) zum angegebenen Schlüssel nicht gefunden
'66'	'15'	Schlüssellänge/Algorithmus-ID des angegebenen Schlüssels unzulässig
'67'	'00'	Wrong length (L_c falsch; L_e nicht vorhanden)
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6A'	'88'	angegebenen Schlüssel nicht gefunden
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.7. EXTERNAL AUTHENTICATE

8.4.7.1. Übersicht

Mit dem Kommando EXTERNAL AUTHENTICATE kann sich die externe Welt gegenüber der ec-Karte authentisieren, indem sie eine unmittelbar zuvor von der ec-Karte mit dem Kommando GET CHALLENGE abgeholte und in der Karte gespeicherte Zufallszahl verschlüsselt in den Kommandodaten an die Karte übergibt.

Zur Ausführung von EXTERNAL AUTHENTICATE sind keine ACs zu erfüllen.

Wurde nicht unmittelbar zuvor das Kommando GET CHALLENGE ausgeführt, wird das Kommando mit dem Returncode '66 01' abgebrochen.

Im Parameter P2 wird angegeben, ob der von der Karte zu verwendende Authentikations-Schlüssel global (Bit 8 = 0) oder DF-spezifisch (Bit 8 = 1) ist.

In der Kommandonachricht wird außerdem festgelegt, welche Schlüsselnummer KID der zu verwendende Authentikations-Schlüssel hat. Ist die Nummer kleiner als 31, kann sie in die 5 niedrigwertigen Bit von P2 kodiert werden. Durch die Belegung der 5 niedrigwertigen Bit von P2 mit dem Wert 31 wird angezeigt, daß die Schlüsselnummer in das erste Byte der Kommandodaten kodiert ist.

Das Ergebnis der Längenprüfung hängt von dem Wert von P2 ab. Für P2 ≠ xxx11111 (binär) muß L_c = '08' sein und der Länge der Kommandodaten entsprechen. Für P2 = xxx11111 (binär) muß L_c = '09' sein und der Länge der Kommandodaten entsprechen. L_e darf nicht vorhanden sein. Andernfalls wird der Returncode '67 00' ausgegeben.

Durch den in Kapitel 2.8. beschriebenen Mechanismus zur Schlüsselauswahl wird mit den

Parametern

- "globaler/DF-spezifischer Schlüssel" aus P2,
- "KID" aus P2 bzw. Byte 1 der Kommandodaten und
- bei Aufruf von EXTERNAL AUTHENTICATE aktuelles DF

der durch das Kommando referenzierte Schlüssel gesucht.

EXTERNAL AUTHENTICATE bricht mit dem Returncode '6A 88' ab, wenn kein Schlüssel mit der angegebenen Nummer gefunden wird. Das Kommando bricht mit dem Returncode '66 13' ab, wenn keine Zusatzinformationen zu dem Schlüssel im zugehörigen EF_KEYD gefunden werden, es bricht mit dem Returncode '66 15' ab, wenn Schlüssellänge und Algorithmus-ID des Schlüssels inkonsistent sind oder wenn die Algorithmus-ID anzeigt, daß der Schlüssel nicht durch EXTERNAL AUTHENTICATE verwendet werden darf, und es bricht mit dem Returncode '66 12' ab, wenn ein Paritätscheck negativ ausfällt.

Falls der zu dem Schlüssel gehörige FBZ im EF_KEYD den Wert 0 hat, bricht das Kommando mit dem Returncode '69 83' ab.

Wenn der Mechanismus zur Schlüsselauswahl erfolgreich war, wird die gespeicherte Zufallszahl unter Verwendung des ICV '00..00' mit diesem Schlüssel verschlüsselt und das 8 Byte lange Chiffrat mit dem in den Kommandodaten übergebenen Chiffrat verglichen. Als Verschlüsselungsalgorithmus wird der für den Schlüssel durch die Algorithmus-ID im zugehörigen EF_KEYD festgelegte Algorithmus verwendet.

Verläuft der Vergleich positiv, so wird der Sicherheitszustand geändert, der dem MF bzw. DF zugeordnet ist, in dessen EF_KEY sich der gefundene Schlüssel befindet: Das Flag, das der Schlüsselnummer (Schlüsselnummer zwischen '00' und '03') bzw. der Schlüsselgruppe, zu der die Schlüsselnummer gehört (Schlüsselnummer > '03'), zugeordnet ist, wird gesetzt.

Geht der Vergleich negativ aus, wird das Flag des Sicherheitszustandes zurückgesetzt und der dem Schlüssel zugeordnete Fehlbedienungszähler im zugehörigen EF_KEYD um 1 dekrementiert. Das Kommando endet in diesem Fall mit dem Returncode '63 CX'. Hierbei gibt $X \geq 0$ den Wert des FBZ an, wenn dieser kleiner als 16 ist; ansonsten wird in X der Wert 'F' zurückgegeben.

8.4.7.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'00'	Kommandoklasse ohne Secure Messaging
INS	1	'82'	Kommandocode
P1	1	'00'	fester Wert
P2	1	'XX'	Reference Control Byte
L_c	1	'08' bzw. '09'	Länge der Kommandodaten, je nach Kodierung von P2
Data	8	'XX..XX'	Verschlüsselte Zufallszahl ENCRND'
	9	'XX..XX'	logische Schlüsselnummer (1 Byte), Verschlüsselte Zufallszahl
L_e	-	-	Keine Antwortdaten

P1

Der Parameter P1 ist immer auf ’00’ zu setzen.

P2

Der Parameter P2 (Referenzkontrollbyte) gibt den zu verwendenden Schlüssel an:

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**Bedeutung**
0	-	-	-	-	-	-	-	Der Schlüssel befindet sich im EF_KEY des MF
1	-	-	-	-	-	-	-	Der Schlüssel befindet sich in dem EF_KEY, das im schlüsselnummer- oder schlüsselgruppen-relevanten DF zum aktuellen DF enthalten ist
-	x	x	-	-	-	-	-	immer 0 0, sonst RFU
-	-	-	x	x	x	x	x	< '1 1 1 1 1': logische Schlüsselnummer 0 bis 30 = '1 1 1 1 1': logische Schlüsselnummer im Datenfeld

8.4.7.3. Antwortnachricht

Das Kommando EXTERNAL AUTHENTICATE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Authentication failed, further allowed retries in 'X'

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'05'	CLA = '04'
'66'	'12'	Paritätsfehler des angegebenen Schlüssels
'66'	'13'	Zusatzinformationen (EF_KEYD) zum angegebenen Schlüssel nicht gefunden
'66'	'15'	Schlüssellänge/Algorithmus-ID des angegebenen Schlüssels inkonsistent oder Schlüssel für EXTERNAL AUTHENTICATE nicht zulässig
'67'	'00'	Wrong length (L_c falsch; L_e vorhanden)
'69'	'83'	FBZ des angegebenen Schlüssels ist 0
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6A'	'88'	angegebenen Schlüssel nicht gefunden
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.4.8. GET CHALLENGE

8.4.8.1. Übersicht

Mit diesem Kommando fordert die externe Welt von der ec-Karte eine Zufallszahl (RND) der Länge 8 Byte als Challenge an. Das Kommando GET CHALLENGE wird immer dann benötigt, wenn Eingabe-Daten für die Karte im unmittelbar auf GET CHALLENGE folgenden Kommando verschlüsselt oder mit einem MAC versehen übertragen werden sollen. GET CHALLENGE gibt als Challenge den neu berechneten Wert einer Zufallsfolge aus, die mittels des internen Zufallszahlengenerators (vgl. Kapitel 2.6.) generiert wird.

Wenn keine Zufallszahl erzeugt werden kann, beispielsweise weil ein hierzu benötigtes EF_RAND fehlt oder leer ist, wird das Kommando mit dem Returncode '69 85' abgewiesen.

Die erzeugte Zufallszahl wird gespeichert und ist vom Zeitpunkt der Erzeugung bis zu der Beendigung der Ausführung des nächsten Kommandos, das die ec-Karte nach GET CHALLENGE erhält, ***gültig***. Dann muß die Zufallszahl ***ungültig*** gesetzt werden, auch wenn sie bei der

Ausführung nicht verwendet wird. Auf diese Weise wird sichergestellt, daß eine bei der Durchführung von GET CHALLENGE erzeugte Zufallszahl nur bei der Ausführung desjenigen Kommandos verwendet werden kann, das unmittelbar auf GET CHALLENGE folgt.

ACs sind für das Kommando nicht zu erfüllen.

8.4.8.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	'00'	Kommandoklasse ohne Secure Messaging
INS	1	'84'	Kommandocode
P1	1	'00'	fester Wert
P2	1	'00'	fester Wert
L_c	0	-	Länge der Kommandodaten
Data	0	-	Kommandodaten
L_e	1	'08'	Länge der Antwortdaten

8.4.8.3. Antwortnachricht

Das Kommando GET CHALLENGE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	8	Zufallszahl	Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'05'	CLA = '04'
'67'	'00'	Wrong length (L_c vorhanden; L_e nicht vorhanden)
'69'	'85'	Benötigte Daten nicht vorhanden (z. B. EF_RAND)
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5. Administrationskommandos

8.5.1. APPEND RECORD

8.5.1.1. Übersicht

Ein Hinzufügen von Records zu EFs mit linearer oder zyklischer Struktur erfolgt mit dem Kommando APPEND RECORD. Mit einem Kommandoaufruf kann genau ein Record entweder an das Ende eines linearen EF hinzugefügt werden oder als Record mit Recordnummer '01' eines zyklischen EF geschrieben werden (vgl. Kapitel 1.2.).

Um nach der Erstellung eines EF mit CREATE FILE ein Record mit UPDATE RECORD in das EF schreiben zu können, ist es erforderlich, daß mit einem Aufruf von APPEND RECORD dieser Record zuvor angelegt wurde.

Es ist jeweils genau die für ein EF festgelegte Recordlänge einzuhalten. Dies bedeutet, daß

- L_c die Recordlänge enthalten muß, wenn keine AC vom Typ PRO oder ENC für das EF gesetzt ist,
- L_c die Recordlänge + 8 enthalten muß, wenn eine AC vom Typ PRO oder ENC für das EF gesetzt ist, und
- L_c immer die Länge der Kommandodaten enthalten muß und kein L_e vorhanden sein darf.

Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen

Die Referenzierung des EF, in das der Record hinzugefügt wird, kann durch vorherige Selektion des EF oder durch Angabe eines SFI erfolgen. Wird kein SFI angegeben (P2 = '00') und ist kein EF selektiert, so wird das Kommando abgebrochen und der Returncode '69 86' zurückgegeben. Wird

ein SFI angegeben, durch den für das selektierte DF kein AEF referenziert wird, wird der Returncode '6A 82' ausgegeben.

Wenn ein lineares EF bereits 254 Records enthält, wird das Kommando mit dem Returncode '69 81' abgelehnt. Wenn für ein zyklisches EF Speicherplatz für mindestens 254 Records allokiert wurde, wird bei der 255ten Ausführung von APPEND RECORD in einem solchen EF der erste mit APPEND RECORD angelegte Record überschrieben.

Wenn der für das EF allokierte Speicherplatz nicht ausreicht, um den Record anzulegen, wird das Kommando mit dem Returncode '6A 84' abgelehnt.

Die Kommandoausführung bedingt die Erfüllung der für die Kommandogruppe ADMIN und das EF, das in den Kommandoparametern angegeben wird (SFI), festgelegten AC. Im Falle einer AC vom Typ PRO bedeutet dies, daß ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen und an die Kommandodaten anzuhängen ist.

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 mit einem MAC versehen und verschlüsselt wird.

Der ICV für die MAC-Bildung bzw. Verschlüsselung muß in beiden Fällen zuvor mittels des Kommandos GET CHALLENGE von der Karte geholt worden sein. Er darf nur einmal verwendet werden.

Wenn eine AC vom Typ PRO oder ENC für das referenzierte EF und ADMIN gesetzt ist, wird APPEND RECORD bei Verwendung von CLA = '00' mit dem Returncode '66 05' abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das referenzierte EF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = '04' mit dem Returncode '66 05' abgelehnt.

Ist für das EF eine AC vom Typ PWD festgelegt, ist die Ausführung von APPEND RECORD an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von APPEND RECORD an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu verwenden ist.

8.5.1.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'00' bzw. '04'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'E2'	Kommandocode
P1	1	'00'	fester Wert
P2	1	'XX'	Reference Control Byte
L_c	1	'XX'	Recordlänge, ggf. + 8 Byte für MAC
Data	X	'XX..XX'	Record, ggf. MAC über Kommandonachricht
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 ist immer mit '00' kodiert.

P2

Das Referenzkontrollbyte ist gemäß folgender Tabelle kodiert:

Referenzkontrollbyte:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
0	0	0	0	0	0	0	0	selektiertes EF
x	x	x	x	x	0	0	0	SFI

Data

Data enthält die Recorddaten und den angehängten MAC.

8.5.1.3. Antwortnachricht

Das Kommando APPEND RECORD gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für ADMIN und das angegebene EF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für ADMIN und das angegebene EF gesetzt
'66'	'05'	AC vom Typ PRO oder ENC für ADMIN und das angegebene EF gesetzt, aber CLA = '00' oder AC nicht vom Typ PRO oder ENC und CLA = '04'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels

SW1	SW2	Bedeutung
’66’	’13’	Zusatzinformationen zu einem durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für ADMIN und das angegebene EF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für ADMIN und das angegebene EF gesetzt
'66'	'82'	AC vom Typ AUT für ADMIN und das aktuelle DF nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)
'69'	'81'	EF enthält bereits 254 Records
'69'	'82'	AC vom Typ PWD für ADMIN und das aktuelle DF nicht erfüllt
'69'	'86'	Command not allowed (no current EF) wird ausgegeben, wenn kein EF selektiert ist, obwohl auf das aktuelle EF zugegriffen werden soll
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch
'6A'	'82'	File not found wird ausgegeben, wenn die Datei, auf die das Kommando zugreifen will, nicht vorhanden ist
'6A'	'84'	Speicherplatz reicht nicht aus
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5.2. CREATE FILE

8.5.2.1. Übersicht

Das Kommando erlaubt das Anlegen eines Verzeichnisses (DF) oder Datenfeldes (EF) innerhalb des aktuellen DF nach der Personalisierung einer ec-Karte. Die Ausführung des Kommandos ist an die AC des bei Aufruf des Kommandos aktuellen DF für die Gruppe der Administrationskommandos (ADMIN) gebunden.

Ist für das aktuelle DF eine AC vom Typ PRO gesetzt, bedeutet dies, daß durch das Terminal ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden (vgl. Kapitel 4.2.1).

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 durch das Terminal mit einem MAC zu versehen und zu verschlüsseln ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden.

In beiden Fällen muß vorher mit GET CHALLENGE von der Karte ein ICV angefordert worden sein, der in die MAC-Berechnung bzw. in die Verschlüsselung einzubeziehen ist.

Wenn eine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das

Kommando bei Verwendung von CLA = ’B0’ mit dem Returncode ’66 05’ abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B4' mit dem Returncode '66 05' abgelehnt.

Ist für das aktuelle DF eine AC vom Typ PWD festgelegt, ist die Ausführung von CREATE FILE an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von CREATE FILE an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu verwenden ist.

Mit dem Kommando CREATE FILE werden die File Control Parameters der Datei (vgl. Kapitel 5.)

- für die Nutzdaten zu allokierender Speicherplatz (nur für EFs),
- Datei-Deskriptor,
- Datei-ID,
- DF-Name (nur für DFs) und
- ACs

an die Karte übertragen, die dann die Datei entsprechend anlegt.

Die in den Kommandodaten anzugebende Größe des für die Nutzdaten eines EF benötigten Speicherplatzes läßt sich durch die externe Welt nach der Formel

geplante Anzahl Records x Recordlänge

berechnen.

Ein DF wird nicht dadurch zu einem ADF, daß als DF-Name ein AID angegeben wird. Um dies zu erreichen, ist zusätzlich die Ausführung des Kommandos INCLUDE erforderlich. Der Datenbereich eines EFs wird immer mit dem Wert '00' initialisiert.

Beim Anlegen von EFs werden 2 Modi unterschieden. Mit dem Parameterwert P1 = '00' werden die Records des EF, unabhängig von den Pagegrenzen des EEPROMS, angelegt.

Erfolgen auf ein EF zur Laufzeit häufig Schreibzugriffe, kann dieses, um die physikalische Datensicherheit zu erhöhen und Seiteneffekte auf benachbarte Felder zu vermeiden, so angelegt werden, daß jeder Record auf den Anfang einer Page angelegt wird. Diese Funktionalität beim Anlegen eines EF wird durch das CREATE FILE mit dem Parameter P1 = '01' ausgewählt.

L_c muß die Länge der Kommandodaten enthalten. Der Umfang der Kommandodaten ist abhängig vom anzulegenden DF bzw. EF und der für das aktuelle DF und CREATE FILE gesetzten AC. Wenn für das aktuelle DF und CREATE FILE eine AC vom Typ PRO oder ENC gesetzt ist, muß L_c = Länge der FCP + 10 sein. Wenn eine andere AC gesetzt ist, muß L_c = Länge der FCP + 2 sein. Die Länge der FCP ist in Byte 2 der Kommandodaten enthalten. Es darf kein L_e vorhanden sein. Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen.

Das Kommando CREATE FILE führt eine Plausibilitätsprüfung der übergebenen Daten durch:

- Die TLV-Struktur wird in Abhängigkeit von P2 daraufhin geprüft, ob die vorgeschriebenen Tags in der richtigen Reihenfolge verwendet werden und ob die Längen korrekt sind.
- Es wird geprüft, ob die Längen im spezifizierten Bereich liegen. Insbesondere müssen
  - die ACs für ein DF (P2 = '00') eine Länge 2 + k*4 mit $k \geq 0$ haben,
  - die ACs für ein EF (P2 = '01') eine Länge 6 + k*4 mit $k \geq 0$ haben und
  - die ACs für ein EF_PWDx (P2 = '01' und Datei-ID = '0012' oder '0022') eine Länge 8 + k*4 mit $k \geq 0$ haben.
- Es wird geprüft, ob die folgenden Werte im spezifizierten Bereich liegen (vgl. Kapitel 5.):
  - Der Datei-Deskriptor für P2 = '00' muß den Wert '38' haben.
  - Der Datei-Deskriptor für P2 = '01' muß einen der Werte '02 41 XX' oder '06 41 XX' haben.
  - Die Datei-ID darf keinen der Werte '3F00', '3FFF' oder 'FFFF' haben.

Bei festgestellten Inkonsistenzen wird das Kommando mit dem Returncode '6A 80' abgebrochen.

Es wird geprüft, ob die übergebenen Werte mit der bestehenden Datei-Organisation der ec-Karte kompatibel sind. Wenn die Datei-ID im aktuellen DF bereits existiert oder wenn der DF-Name in der ec-Karte bereits existiert oder wenn ein DF angelegt werden soll, dies aber unterhalb der Ebene des aktuellen DF nicht mehr möglich ist, wird das Kommando mit dem Returncode '69 81' abgebrochen.

Wenn der Speicherplatz zum Anlegen der Datei nicht ausreicht, wird das Kommando mit dem Returncode '6A 84' abgebrochen.

8.5.2.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'B0' bzw. 'B4'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'E0'	Kommandocode
P1	1	'XX'	Parameter 1 (Beachtung der Pagegrenzen)
P2	1	'XX'	Parameter 2 (EF oder DF)
L_c	1	'XX'	Länge der Daten, ggf. + 8 Byte für MAC
Data	X	'XX..XX'	Informationen zu DF oder EF, ggf. MAC
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 gibt an, ob bei dem Anlegen der Datei Pagegrenzen beachtet werden sollen.

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x	x	x	x	x	-	immer 0 ... 0, sonst RFU
-	-	-	-	-	-	-	0	Pagegrenzen werden nicht berücksichtigt
-	-	-	-	-	-	-	1	Pagegrenzen werden berücksichtigt, die Daten der Datei beginnen mit einer neuen Page

P2

Der Parameter P2 gibt an, ob ein DF oder ein EF angelegt werden soll:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x	x	x	x	x	-	immer 0 ... 0, sonst RFU
-	-	-	-	-	-	-	0	Die anzulegende Datei ist ein DF
-	-	-	-	-	-	-	1	Die anzulegende Datei ist ein EF

Data

Es werden die FCP gemäß Kapitel 5. zu der Datei übergeben:

Falls P2 = 'xxxxxxx0':

Tag	Länge (in Byte)	Wert	Format
'62'	'XX'		
'82'	'01'	Datei-Deskriptor '38'	binär
'83'	'02'	Datei-ID	binär
'84'	'01' bis '10'	DF-Name (AID)	binär
'86'	'XX'	ACs	binär

Falls P2 = 'xxxxxxx1':

Tag	Länge (in Byte)	Wert	Format
’62’	’XX’		
’81’	’02’	zu allokierender Speicherplatz in Byte	binär
'82'	'03'	Datei-Deskriptor für EFs	binär
'83'	'02'	Datei-ID	binär
'86'	'XX'	ACs	binär

8.5.2.3. Antwortnachricht

Das Kommando CREATE FILE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	’XX’	Statusbyte 1
SW2	1	’XX’	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für ADMIN und das aktuelle DF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für ADMIN und das aktuelle DF
'66'	'05'	AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF gesetzt, aber CLA = 'B0' oder AC nicht vom Typ PRO oder ENC und CLA = 'B4'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu einem durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für ADMIN und das aktuelle DF gesetzt
'66'	'82'	AC vom Typ AUT für ADMIN und das aktuelle DF nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)

SW1	SW2	Bedeutung
'69'	'81'	Datei-ID im aktuellen DF oder DF-Name in der Karte bereits vorhanden oder aktuelles DF bereits auf unterster Ebene
'69'	'82'	AC vom Typ PWD für ADMIN und das aktuelle DF nicht erfüllt
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch
'6A'	'80'	Kommandodaten fehlerhaft
'6A'	'84'	Speicherplatz reicht nicht aus
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5.3. DELETE FILE

8.5.3.1. Übersicht

Das DELETE FILE Kommando löscht ein einzelnes Datenfeld (EF) oder Verzeichnis (DF) und überschreibt den von einem gelöschten EF vormals belegten Speicherplatz mit '00'. Mindestens dann, wenn die Dateien der ec-Karte in umgekehrter Reihenfolge ihres Anlegens gelöscht werden, muß der frei gewordene Speicherplatz zum Anlegen neuer Dateien zur Verfügung stehen.

Die Ausführung des Kommandos zum Löschen einer Datei ist an die AC des übergeordneten DF dieser Datei für die Gruppe der Administrationskommandos (ADMIN) gebunden.

Ist für das übergeordnete DF eine AC vom Typ PRO gesetzt, bedeutet dies, daß durch das Terminal ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden (vgl. Kapitel 4.2).

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 durch das Terminal mit einem MAC zu versehen und zu verschlüsseln ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden.

In beiden Fällen muß vorher mit GET CHALLENGE von der Karte ein ICV angefordert worden sein, der in die MAC-Berechnung bzw. in die Verschlüsselung einzubeziehen ist.

Wenn eine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B0' mit dem Returncode '66 05' abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B4' mit dem Returncode '66 05' abgelehnt.

Ist für ADMIN und das übergeordnete DF eine AC vom Typ PWD festgelegt, ist die Ausführung von DELETE FILE an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von DELETE FILE an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu verwenden ist.

Zum Löschen eines DF muß das übergeordnete Verzeichnis selektiert sein. Insbesondere ist das Löschen des MF mit dem Kommando nicht möglich. Das zu löschende DF wird durch die Datei-ID bezeichnet. Ein DF kann nur dann gelöscht werden, wenn es keine weiteren DFs oder EFs enthält und wenn es kein ADF ist. Enthält das DF noch weitere Dateien, wird das Kommando mit dem Returncode '69 81' abgebrochen. Ist das DF ein ADF, d. h. sind dem DF noch EFs als AEFs zugeordnet, wird das Kommando mit dem Returncode '69 85' abgebrochen.

Um ein EF zu löschen, muß das dem EF übergeordnete DF selektiert sein. Zusätzlich kann das EF selektiert sein. Das zu löschende EF wird durch die Datei-ID bezeichnet. Falls ein EF gelöscht werden soll, das aktuell selektiert ist, braucht die Datei-ID nicht angegeben zu werden. Ein EF kann nur dann gelöscht werden, wenn alle Records mit dem Wert '00' belegt sind.

Kann eine zu löschende Datei nicht gefunden werden, so wird das Kommando abgebrochen, und es wird der Returncode '6A 82' zurückgegeben. Wird keine Datei-ID angegeben (P2 = '00') und ist

kein EF selektiert, so wird das Kommando abgebrochen und der Returncode ’69 86’ zurückgegeben.

Das Ergebnis der Längenprüfung hängt von dem Wert von P2 und der für das aktuelle DF und DELETE FILE gesetzten AC ab. Hat P2 den Wert '00' und ist keine AC vom Typ PRO oder ENC gesetzt, muß die Kommandonachricht eine Länge von 4 Byte haben. Hat P2 den Wert '00' und ist eine AC vom Typ PRO oder ENC gesetzt, muß L_c den Wert '08' haben, müssen die Kommandodaten eine Länge von 8 Byte haben und darf kein L_e vorhanden sein. Hat P2 den Wert '01' und ist keine AC vom Typ PRO oder ENC gesetzt, muß L_c den Wert '02' haben, müssen die Kommandodaten eine Länge von 2 Byte haben und darf kein L_e vorhanden sein. Hat P2 den Wert '01' und ist eine AC vom Typ PRO oder ENC gesetzt, muß L_c den Wert '0A' haben, müssen die Kommandodaten eine Länge von 10 Byte haben und darf kein L_e vorhanden sein. Andernfalls wird der Returncode '67 00' ausgegeben.

8.5.3.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	’B0’ bzw. ’B4’	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	’E4’	Kommandocode
P1	1	’00’	Parameter 1
P2	1	’00’ oder ’01’	Parameter 2
L_c	0 oder 1 1	- bzw. ’08’ ’02’ bzw. ’0A’	ggf. + 8 Byte für MAC Länge Datei-ID, ggf + 8 Byte für MAC
Data	0, 2, 8, oder 10	- oder 'XX..XX'	ggf. Datei-ID, ggf. MAC über Kommandonachricht
L_e	0	-	keine Antwortdaten

P1

Parameter P1 ist immer mit ’00’ zu kodieren.

P2

P2 = '00': Löscht das aktuell selektierte EF.

P2 = '01': Löscht die angegebene Datei.

Data

Die Kommandodaten können eine 2 Byte lange Datei-ID enthalten.

8.5.3.3. Antwortnachricht

Das Kommando DELETE FILE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für ADMIN und das aktuelle DF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für ADMIN und das aktuelle DF gesetzt
'66'	'05'	AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF gesetzt, aber CLA = 'B0' oder AC nicht vom Typ PRO oder ENC und CLA = 'B4'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu einem durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für ADMIN und das aktuelle DF gesetzt
'66'	'82'	AC vom Typ AUT für ADMIN und das aktuelle DF nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)
'69'	'81'	zu löschendes DF enthält noch Dateien oder zu löschendes EF enthält nicht nur '00'
'69'	'82'	AC vom Typ PWD für ADMIN und das aktuelle DF nicht erfüllt
'69'	'85'	DF ist ADF, dem noch AEFs zugeordnet sind
'69'	'86'	Command not allowed (no current EF) wird ausgegeben, wenn kein EF selektiert ist, obwohl das aktuelle EF gelöscht werden soll
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch

SW1	SW2	Bedeutung
'6A'	'82'	File not found wird ausgegeben, wenn die Datei, deren Datei-ID angegeben wird, nicht existiert
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5.4. INCLUDE

8.5.4.1. Übersicht

Das Kommando INCLUDE dient dazu, eine Verknüpfung festzulegen zwischen einem DF, einem SFI und einem Pfad, der eindeutig ein EF bezeichnet. Damit wird eine Applikation aufgebaut, innerhalb deren Kontext EFs mittels SFI referenzierbar sind. Insbesondere wird ein DF mittels INCLUDE als ADF qualifiziert, sobald dem DF das erste EF erfolgreich zugeordnet wurde.

Das Kommando INCLUDE setzt voraus, daß das entsprechende DF selektiert ist.

Es können folgende spezifische Fehlersituationen auftreten:

- Die TLV-Struktur der Kommandodaten ist fehlerhaft, oder das erste Byte des Wert-Feldes enthält einen Wert > 31 (Returncode '6A 80').
- Es existiert keine Datei zu dem angegebenen Pfad (Returncode '6A 82').
- Die Datei mit dem angegebenen Pfad ist ein DF (Returncode '69 81').
- Der SFI ist reserviert ('00' oder '1F') oder im aktuellen Kontext bereits vergeben (Returncode '69 85').

Wenn zum Einbinden eines EFs mittels INCLUDE Speicherplatz benötigt wird und dieser nicht ausreicht, wird das Kommando mit dem Returncode '6A 84' abgebrochen.

Für das Kommando ist die AC für die Kommandogruppe ADMIN des selektierten DF zu erfüllen:

Ist für das selektierte DF eine AC vom Typ PRO gesetzt, bedeutet dies, daß durch das Terminal ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden (vgl. Kapitel 4.2.1).

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 durch das Terminal mit einem MAC zu versehen und zu verschlüsseln ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden.

In beiden Fällen muß vorher mit GET CHALLENGE von der Karte ein ICV angefordert worden sein, der in die MAC-Berechnung bzw. in die Verschlüsselung einzubeziehen ist.

Wenn eine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B0' mit dem Returncode '66 05' abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B4' mit dem Returncode '66 05' abgelehnt.

Ist für das selektierte DF eine AC vom Typ PWD festgelegt, ist die Ausführung von INCLUDE an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von INCLUDE an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel bzw. ein Schlüssel aus der durch die AC referenzierten Schlüsselgruppe zu

verwenden ist.

L_c muß die Länge der Kommandodaten enthalten. Der Umfang der Kommandodaten ist abhängig vom einzubindenden EF und der AC für das aktuelle DF und INCLUDE. Wenn für das aktuelle DF und INCLUDE eine AC vom Typ PRO oder ENC gesetzt ist, muß L_c = Länge des TLV-Objektes + 10 sein. Wenn eine andere AC gesetzt ist, muß L_c = Länge des TLV-Objektes + 2 sein. Die Länge des TLV-Objektes ist in Byte 2 der Kommandodaten enthalten. Es darf kein L_e vorhanden sein. Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen.

8.5.4.2. Kommandonachricht

Command APDU

	Länge	**Inhalt**	**Beschreibung**
CLA	1	'B0' bzw. 'B4'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'E6'	Kommandocode
P1	1	'00'	Parameter 1
P2	1	'00'	Parameter 2
L_c	1	'XX'	Datenlänge, ggf. + 8 Byte für MAC
Data	X	'XX..XX'	Kommandodaten, ggf. MAC über Kommandonachricht
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 ist immer mit '00' kodiert.

P2

Der Parameter P2 ist immer mit '00' kodiert.

Data

Es werden folgende Daten, eventuell zusammen mit dem angehängten MAC, übergeben:

Tag	Länge (in Byte)	Wert
'85'	'XX'	SFI eines AEF mit Pfad

In das erste Byte des Wert-Feldes ist die SFI des entsprechenden AEFs wie folgt binär kodiert:

SFI-Kodierung im ersten Byte: 000xxxxx

In den weiteren Byte des Wert-Feldes ist der absolute Pfad des AEF gemäß ISO 7816-4, Kapitel 5.1.1 ohne Datei-ID des MF kodiert.

8.5.4.3. Antwortnachricht

Das Kommando INCLUDE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für ADMIN und das aktuelle DF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für ADMIN und das aktuelle DF gesetzt
'66'	'05'	AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF gesetzt, aber CLA = 'B0' oder AC nicht vom Typ PRO oder ENC und CLA = 'B4'
'66'	'11'	durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels
'66'	'82'	AC vom Typ AUT für ADMIN und das aktuelle DF nicht erfüllt
'66'	'13'	Zusatzinformationen zu einem durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für ADMIN und das aktuelle DF gesetzt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; L_e vorhanden)
'69'	'81'	Es soll ein DF zugeordnet werden
'69'	'82'	AC vom Typ PWD für ADMIN und das aktuelle DF nicht erfüllt
'69'	'85'	SFI reserviert oder bereits vergeben
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch

SW1	SW2	Bedeutung
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn die TLV-Struktur der Kommandodaten fehlerhaft ist oder Byte 1 des Wert-Feldes (SFI) falsch kodiert ist.
'6A'	'82'	File not found wird ausgegeben, wenn das angegebene EF nicht vorhanden ist
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6A'	'84'	Speicherplatz reicht nicht aus
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5.5. EXCLUDE

8.5.5.1. Übersicht

Das Kommando EXCLUDE dient dazu, eine mittels INCLUDE bewirkte Verknüpfung zwischen einem EF und einem DF aufzuheben.

Ein DF verliert seine Qualifikation als ADF, sobald die letzte Verknüpfung eines EFs zu ihm aufgehoben wurde.

Das Kommando EXCLUDE setzt voraus, daß das entsprechende ADF mittels SELECT FILE selektiert ist.

Falls zu dem angegebenen SFI für das selektierte DF keine Verknüpfung existiert, wird das Kommando mit dem Returncode '6A 88' abgebrochen.

Für das Kommando ist die für das selektierte DF und die Kommandogruppe ADMIN definierte AC zu erfüllen:

Ist für das selektierte DF eine AC vom Typ PRO gesetzt, bedeutet dies, daß durch das Terminal ein MAC über die Kommandonachricht gemäß Kapitel 3.1.1 zu berechnen ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden (vgl. Kapitel 4.2.1).

Eine AC vom Typ ENC legt fest, daß die Kommandonachricht gemäß Kapitel 3.2.1 durch das Terminal mit einem MAC zu versehen und zu verschlüsseln ist. Hierfür ist der in der AC referenzierte Schlüssel zu verwenden.

In beiden Fällen muß vorher mit GET CHALLENGE von der Karte ein ICV angefordert worden sein, der in die MAC-Berechnung bzw. in die Verschlüsselung einzubeziehen ist.

Wenn eine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B0' mit dem Returncode '66 05' abgebrochen. Wenn keine AC vom Typ PRO oder ENC für das aktuelle DF und ADMIN gesetzt ist, wird das Kommando bei Verwendung von CLA = 'B4' mit dem Returncode '66 05' abgelehnt.

Ist für das selektierte DF eine AC vom Typ PWD festgelegt, ist die Ausführung von EXCLUDE an die erfolgreiche Prüfung des durch die AC referenzierten Paßworts gebunden.

Ist eine AC vom Typ AUT festgelegt, so ist die Ausführung von EXCLUDE an die vorherige erfolgreiche Authentikation mit EXTERNAL AUTHENTICATE gebunden, wobei der in der AC referenzierte Schlüssel zu verwenden ist.

Wenn für das aktuelle DF und EXCLUDE keine AC vom Typ PRO oder ENC gesetzt ist, muß die Kommandonachricht eine Länge von 4 Byte haben. Wenn eine AC vom Typ PRO oder ENC gesetzt ist, muß L_c den Wert '08' haben, der auch der Länge der Kommandodaten entsprechen muß. L_e darf nicht vorhanden sein. Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen.

8.5.5.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'B0' bzw. 'B4'	Kommandoklasse ohne Secure Messaging mit Secure Messaging der ec-Karte
INS	1	'E8'	Kommandocode
P1	1	'00'	Parameter 1
P2	1	'XX'	Parameter 2: SFI
L_c	0 oder 1	- oder '08'	ggf. Länge des MAC
Data	0 oder 8	- oder 'XX..XX'	ggf. MAC über die Kommandonachricht
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 ist immer mit '00' kodiert.

P2

Das Referenzkontrollbyte ist gemäß folgender Tabelle kodiert:

Referenzkontrollbyte:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x	x	x	0	0	0	SFI

8.5.5.3. Antwortnachricht

Das Kommando EXCLUDE gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	Inhalt	Beschreibung
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC für ADMIN und das aktuelle DF gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC für ADMIN und das aktuelle DF
'66'	'05'	AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF gesetzt, aber CLA = 'B0' oder AC nicht vom Typ PRO oder ENC und CLA = 'B4'
'66'	'11'	durch eine AC durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu einem Schlüssel durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten nicht gefunden
'66'	'14'	FBZ eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID eines durch eine AC vom Typ PRO oder ENC für ADMIN und das aktuelle DF referenzierten Schlüssels ist unzulässig
'66'	'81'	AC NEV für ADMIN und das aktuelle DF gesetzt
'66'	'82'	AC vom Typ AUT für ADMIN und das aktuelle DF nicht erfüllt
'67'	'00'	Wrong length (L_c falsch, insbesondere bei fehlendem MAC; $L_e$vorhanden)
'69'	'82'	AC vom Typ PWD für ADMIN und das aktuelle DF nicht erfüllt
'69'	'87'	Byte-Länge des Chiffrats kein Vielfaches von 8 oder falsches Padding
'69'	'88'	MAC falsch
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben, insbesondere bei falscher Kodierung von P2 (SFI)
'6A'	'88'	Zum SFI aus P2 existiert keine Verknüpfung
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

8.5.6. LOAD COMMAND

8.5.6.1. Übersicht

Das Kommando LOAD COMMAND dient dazu, Ergänzungskommandos als neue Kommandos in

die ec-Karte nachzuladen oder bereits vorhandene Kommandos zu ersetzen. LOAD COMMAND bietet eine einheitliche Schnittstelle zu einer herstellerspezifischen Kommando-Nachladeprozedur und abstrahiert damit von den je nach Karte unterschiedlichen technischen Abläufen, die zum Nachladen erforderlich sind.

Als Kommandodaten werden CLA und INS des zu ladenden Kommandos sowie eine transparente Bytefolge übergeben. LOAD COMMAND kann für das gleiche Kommando mehrfach wiederholt werden. Mit dem letzten Aufruf wird der Abschluß der Ladeprozedur angezeigt. Danach ist das geladene Kommando betriebsbereit und kann mit den angegebenen CLA und INS aufgerufen werden. Wird ein Kommando nachgeladen, das auf der Karte bereits vorhanden ist (d.h. CLA und INS des neuen Kommandos stimmen mit dem vorhandenen überein), so wird mit Abschluß der Nachladeprozedur das alte Kommando durch das neue ersetzt. Das alte Kommando ist danach nicht mehr ausführbar.

L_c muß der Länge der Kommandodaten entsprechen (mindestens '0A', da die Kommandodaten immer CLA, INS und MAC enthalten müssen). L_e darf nicht vorhanden sein. Andernfalls wird das Kommando mit dem Returncode '67 00' abgebrochen.

Die Ausführung von LOAD COMMAND ist an die AC PRO_G '00' gebunden, das heißt, es wird eine MAC-Berechnung und -Prüfung mit dem Schlüssel K_{Card} (logische Schlüsselnummer '00') durchgeführt, der im EF_KEY des MF gespeichert ist. Die ec-Karte interpretiert die letzten 8 Byte der Kommandodaten immer als MAC.

Für die MAC-Berechnung bzw. dessen Prüfung durch die Karte muß zuvor mittels GET CHALLENGE ein ICV angefordert worden sein.

Im dem Fehlerfall, daß bei der Fortsetzung eines Ladevorganges die Parameter CLA und INS des zu ladenden Kommandos nicht mit den Parametern der vorherigen Aufrufe von LOAD COMMAND übereinstimmen, wird das Kommando abgebrochen, und es wird der Returncode '6A 80' zurückgegeben.

Mit dem Returncode '6A 80' wird ebenfalls abgebrochen, falls bei Ausführung von LOAD COMMAND festgestellt wird, daß die Kommandodaten keinen ausführbaren Kommandocode enthalten.

Wenn der Speicherplatz nicht ausreicht, um das jeweilige Kommando nachzuladen, wird LOAD COMMAND mit dem Returncode '6A 84' abgebrochen.

8.5.6.2. Kommandonachricht

Command APDU

	Länge	Inhalt	Beschreibung
CLA	1	'B4'	Kommandoklasse mit Secure Messaging
INS	1	'EA'	Kommandocode
P1	1	'XX'	Parameter 1
P2	1	'00'	Parameter 2
L_c	1	'XX'	Länge der Daten, 8 Byte MAC
Data	X	'XX..XX'	Daten, MAC
L_e	0	-	keine Antwortdaten

P1

Der Parameter P1 gibt an, ob der Ladevorgang für das Kommando beendet ist oder ob er in einem weiteren LOAD COMMAND-Aufruf fortgesetzt wird:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x	x	x	x	x	-	0...0, alle anderen Werte sind RFU
-	-	-	-	-	-	-	0	Ladevorgang ist beendet.
-	-	-	-	-	-	-	1	Ladevorgang wird fortgesetzt.

P2

Der Parameter P2 wird zu '00' kodiert.

Data

Die Kommandodaten haben folgende Struktur und Inhalt:

CLA	INS	transparente Daten	MAC
1 Byte	1 Byte	X Byte	8 Byte

CLA und INS sind das Class- und das Instruction-Byte des zu ladenden Kommandos.

8.5.6.3. Antwortnachricht

Das Kommando LOAD COMMAND gibt folgende Antwortnachricht zurück:

Response APDU

	Länge	**Inhalt**	**Beschreibung**
Data	0	-	keine Antwortdaten
SW1	1	'XX'	Statusbyte 1
SW2	1	'XX'	Statusbyte 2

Returncodes

Der folgende Returncode zeigt an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	**SW2**	**Bedeutung**
'90'	'00'	OK

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
'64'	'00'	No precise diagnosis wird ausgegeben, wenn bei der **Ausführung** ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'65'	'81'	Memory failure wird ausgegeben, wenn die **Ausführung** fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'05'	CLA = '00'
'66'	'11'	K_{Card} nicht gefunden
'66'	'12'	Paritätsfehler des K_{Card}
'66'	'13'	Zusatzinformationen zu K_{Card} nicht gefunden
'66'	'14'	FBZ des K_{Card} ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID des K_{Card} ist unzulässig
'67'	'00'	Wrong length (L_c falsch; L_e vorhanden)
'69'	'88'	MAC falsch
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn das Format der Kommandodaten nicht der Spezifikation entspricht oder Kommandodaten außerhalb des spezifizierten Wertebereichs liegen (keine Längenfehler der gesamten Kommandodaten)
'6A'	'84'	Speicherplatz reicht nicht aus
'6A'	'86'	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported

Das electronic-cash System

Version 2.2
22.01.1997

Die Dokumente zum electronic-cash System liegen z. Zt. nicht in elektronischer Form auf dieser CD-ROM vor.

Sie sind zu beziehen bei:

GZS Gesellschaft für Zahlungssysteme mbH

Unternehmensbereich Lizenzen

Administration electronic cash

Solmsstr. 2-26

60486 Frankurt

Tel.: 069 / 7922-0

Fax: 069 / 7922- 4575

oder über den jeweiligen Federführer des Zentralen Kreditausschuß.

Die elektronische Geldbörse – Börsenkarte –

Version 2.2
22.01.1997, Ausgabedatum: 08.08.1995

Inhalt

1. Daten der elektronischen Geldbörse

Mit ***Börsenkarte*** wird eine Chipkarte bezeichnet, deren Struktur der Daten, Sicherheitsarchitektur sowie Standard- und Administrationskommandos der Spezifikation in [LIT 1] entsprechen und die die Applikation elektronische Geldbörse bestehend aus dem Applikationsverzeichnis (ADF) und den zugeordneten AEFs sowie die in Kapitel 2. spezifizierten Ergänzungskommandos der elektronischen Geldbörse enthält. Das ADF der Applikation elektronische Geldbörse wird mit DF_BÖRSE bezeichnet.

Die Ergänzungskommandos der elektronischen Geldbörse greifen explizit auf die folgenden EFs zu:

- EF_BETRAG,
- EF_BÖRSE,
- EF_LSEQ,
- EF_BSEQ,
- EF_LLOG,
- EF_BLOG.

Diese müssen sich im DF_BÖRSE befinden. Da die Ergänzungskommandos der Applikation elektronische Geldbörse nur ausgeführt werden können, wenn das DF_BÖRSE aktuell ist, ist für die Ergänzungskommandos das Auffinden dieser EFs unabhängig von der Position des DF_BÖRSE in der Börsenkarte und von für das DF_BÖRSE vergebenen SFIs möglich.

In Abhängigkeit davon, welche ACs für die Ergänzungskommandos der elektronischen Geldbörse gesetzt sind, müssen in der Börsenkarte globale und/oder DF-spezifische EF_KEYs und EF_PWDx mit den jeweiligen Deskriptor-EFs vorhanden sein, damit die für die Ergänzungskommandos gesetzten ACs erfüllt werden können.

Im folgenden werden zunächst spezielle Zugriffsbedingungen für Ergänzungskommandos der elektronischen Geldbörse erläutert, die für die Standard- und Administrationskommandos in [LIT 1] nicht definiert sind. Anschließend werden Aufbau, Datei-Kontrollinformation und Inhalt des DF_BÖRSE und der darin enthaltenen EFs für zwei Typen von Börsenkarten beschrieben. Bei diesen beiden Typen handelt es sich um

- **Wertkarten** ohne Kontobezug und ohne PIN und
- **kontobezogene Börsenkarten** mit PIN.

Durch die Kodierung des ersten Byte im EF_BÖRSE im DF_BÖRSE wird angezeigt, um welchen Typ von Börsenkarte (**Kartentyp**) es sich handelt.

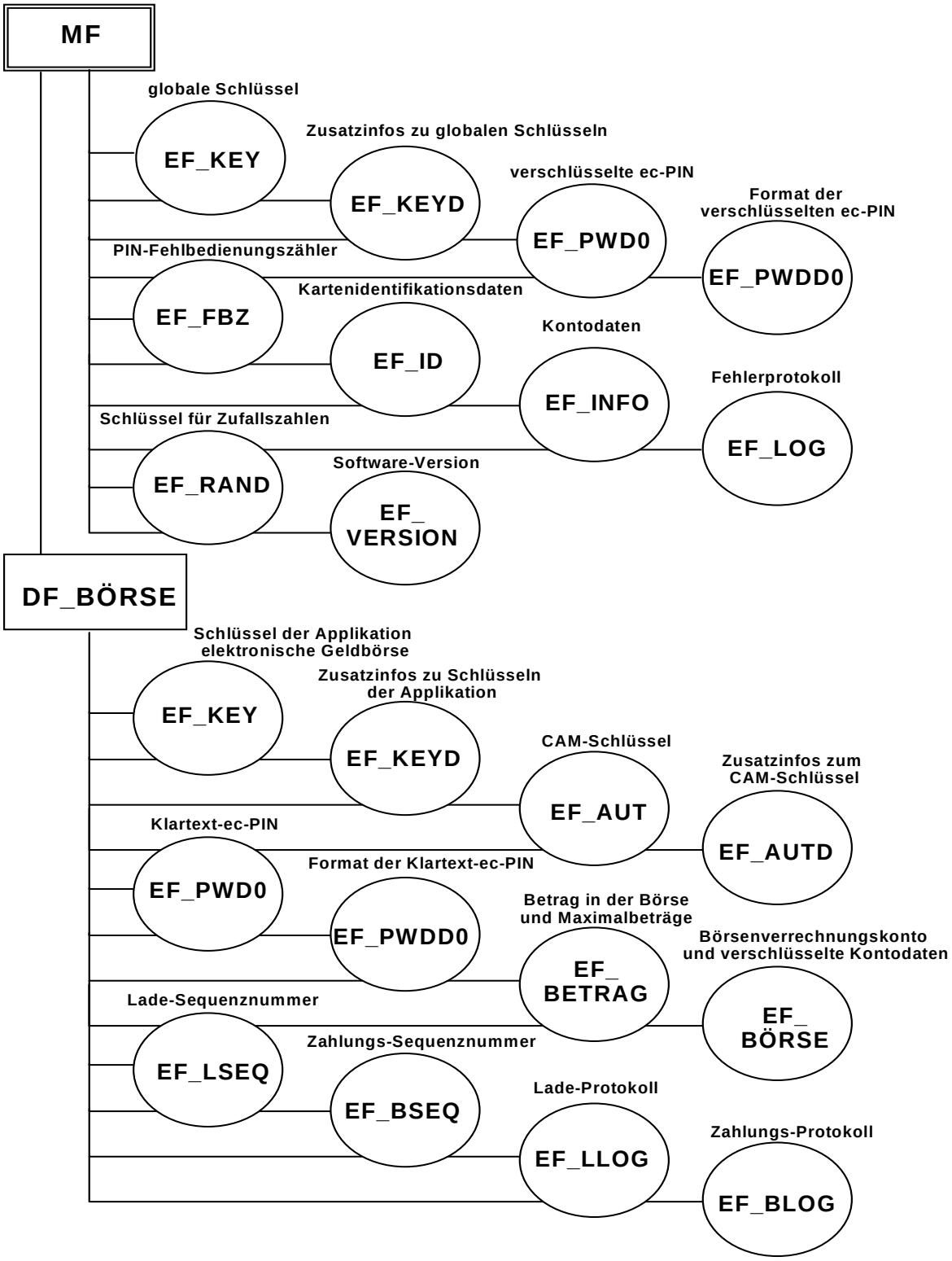
Bei beiden Kartentypen ist das DF_BÖRSE direkt im MF enthalten. Die für die Applikation elektronische Geldbörse relevanten DF-spezifischen Schlüssel sind im EF_KEY abgelegt, das direkt im DF_BÖRSE enthalten ist. Das DF-spezifische EF_PWD0, das in kontobezogenen Börsenkarten die PIN im Format "Format 2 PIN Block" enthält, ist ebenfalls direkt im DF_BÖRSE enthalten.

Die Spezifikation der Ergänzungskommandos der elektronischen Geldbörse in Kapitel 2. setzt diese spezielle Kartenkonfiguration mit den hier gewählten ACs nicht voraus. Die Ergänzungskommandos müssen auch dann korrekt funktionieren, wenn

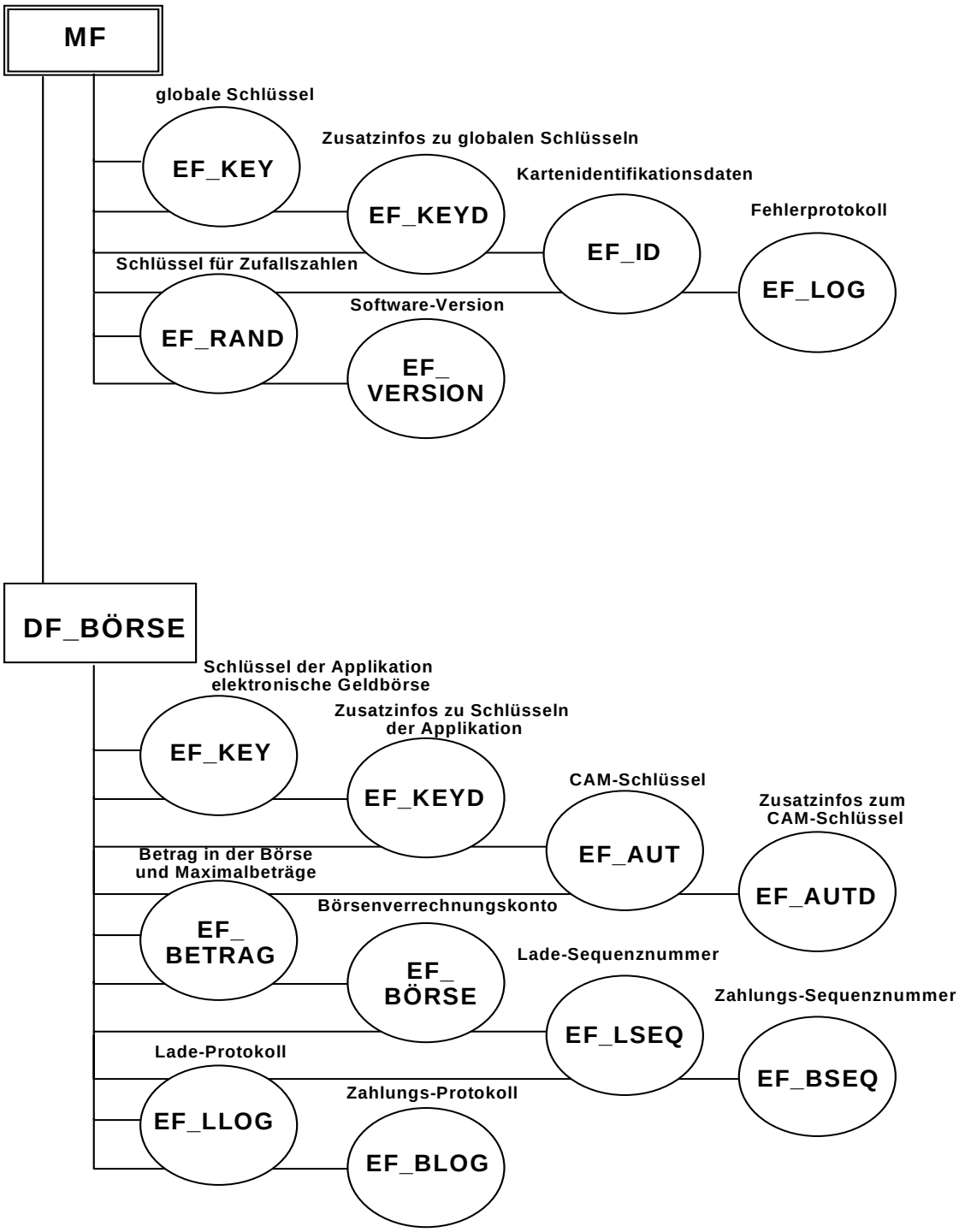
- keine SFIs definiert sind,
- das DF_BÖRSE nicht direkt im MF enthalten ist,
- für die Ergänzungskommandos andere, im Rahmen der Spezifikation zulässige ACs gesetzt sind,
- sich durch ACs referenzierte DF-spezifische Schlüssel nicht im EF_KEY des DF_BÖRSE, sondern eines übergeordneten DF befinden,
- sich das für den Kartentyp kontobezogene Börsenkarte relevante DF-spezifische EF_PWD0 nicht im DF_BÖRSE befindet.

Das MF und die im MF enthaltenen EFs der beiden Typen von Börsenkarten entsprechen der Spezifikation Kapitel 6. von [LIT 1].

Die folgende Grafik gibt eine Übersicht über die Dateien einer kontobezogenen Börsenkarte.



Die folgende Grafik gibt eine Übersicht über die Dateien einer Wertkarte.



1.1. Zugriffsbedingungen für Ergänzungskommandos der elektronischen Geldbörse

Der Zugriff eines Ergänzungskommandos auf Daten der elektronischen Geldbörse wird durch die AC geregelt, die für EF_BETRAG in DF_BÖRSE und das jeweilige Kommando festgelegt ist. Wenn für ein Kommando eine Basis-AC vom Typ PRO festgelegt ist, ist für die Ausführung einiger Kommandovarianten zusätzlich der Kartentyp (Wertkarte oder kontobezogene Börsenkarte)

maßgeblich, der durch die Kodierung des ersten Byte im EF_BÖRSE im DF_BÖRSE angezeigt wird.

Für die Ergänzungskommandos der elektronischen Geldbörse kann ein zusätzlicher Typ von Basis-ACs (**Typ ZERT**) verwendet werden, der für die Standard- und Administrationskommandos nicht definiert ist. Im folgenden werden das Zusammenwirken der AC PRO mit dem Kartentyp sowie der zusätzliche AC-Typ ZERT erläutert.

1.1.1. AC vom Typ PRO und Kartentyp

Wenn für ein Ergänzungskommando der elektronischen Geldbörse die Basis-AC PRO festgelegt ist und der Aufruf einer Kommandovariante mit CLA = 'E4' erfolgt, muß die Kommando- und/oder Antwortnachricht der Kommandovariante mit einem MAC gemäß der Beschreibung des Secure Messaging in Kapitel 3.1 in [LIT 1] versehen werden.

In der Spezifikation der jeweiligen Kommandovariante wird festgelegt, ob die Kommandonachricht oder die Antwortnachricht oder beide Nachrichten mit einem MAC zu versehen sind.

Wenn für ein Ergänzungskommando der elektronischen Geldbörse die Basis-AC PRO festgelegt ist, kann es vom Kartentyp abhängen, ob der Aufruf einer Kommandovariante mit CLA = 'E0' (ohne Secure Messaging) zulässig ist und welche Bedingung erfüllt sein muß, damit die Kommandovariante bei Aufruf mit CLA = 'E0' ausgeführt werden darf.

In der Spezifikation der jeweiligen Kommandovariante wird festgelegt, ob der Aufruf mit CLA = 'E0' zulässig ist und welche Bedingung dann erfüllt sein muß.

Wenn es sich um eine kontobezogene Börsenkarte handelt, ist eine mögliche Bedingung, daß zuvor eine Karteninhaberauthentikation durch Angabe eines Paßwortes mit dem Kommando VERIFY stattgefunden hat.

Die **Karteninhaberauthentikation gegenüber einer kontobezogenen Börsenkarte** muß dann immer mit dem Paßwort, das durch den Such-Algorithmus aus Kapitel 2.9 in [LIT 1] mit den Parametern

- DF-spezifisches Paßwort,
- Paßwortnummer 0 und
- EF_BETRAG in DF_BÖRSE

eindeutig bestimmt ist **oder** mit dem globalen Paßwort aus dem EF_PWD0 des MF erfolgt sein. Hierbei wird zuerst geprüft, ob das DF-spezifische Paßwort korrekt angegeben wurde.

1.1.2. Kodierung des Kartentyps

Der Kartentyp wird durch die Kodierung von Byte 1 des EF_BÖRSE festgelegt. Hierbei bedeutet

'00': kontobezogene Börsenkarte,

'FF': Wertkarte ohne Kontobezug.

Bei kontobezogenen Börsenkarten ist Byte 11-27 des EF_BÖRSE mit unter K_{LD} verschlüsselten Konto- und Kartendaten belegt.

Bei Wertkarten enthalten Byte 11-27 des EF_BÖRSE '00..00'.

1.1.3. Typ ZERT

Basis-ACs vom Typ ZERT erfordern für den Zugriff des Ergänzungskommandos auf Daten der elektronischen Geldbörse, daß Kommando- und/oder Antwortnachricht ein ***Zertifikat*** über Daten der Nachricht enthalten.

Bei einem Zertifikat handelt es sich immer um einen (Retail-)CBC-MAC, der gemäß Kapitel 2.4.2 in [LIT 1] berechnet wird.

In der Spezifikation der jeweiligen Kommandovariante wird festgelegt, über welche Kommando- und/oder Antwortdaten das Zertifikat zu rechnen ist. Das Zertifikat steht in der jeweiligen Nachricht immer direkt hinter den gesicherten Daten.

Durch die AC wird festgelegt, welcher Schlüssel zur Zertifikatsberechnung zu verwenden ist. Ist ein globaler Schlüssel mit Schlüsselnummer KID zu verwenden, wird die AC als **ZERT_G KID** notiert. Ist ein DF-spezifischer Schlüssel mit Schlüsselnummer KID zu verwenden, wird die AC als **ZERT_D KID** notiert. In beiden Fällen kann KID die Werte '00' bis '03' zur Referenzierung von Einzelschlüsseln, '04' zur Referenzierung der Schlüsselgruppe 1 oder '0F' zur Referenzierung der Schlüsselgruppe 2 haben.

1.1.4. Referenzierte Schlüssel

Wenn durch eine AC vom Typ ZERT ein Schlüssel referenziert wird, ist dieser auf dieselbe Weise aufzufinden wie ein Schlüssel, der durch die für die Standard- und Administrationskommandos bereits definierten ACs referenziert wird (vgl. Kapitel 4.2.4 in [LIT 1]):

Wenn die logische Schlüsselnummer KID zwischen '00' und '03' liegt, muß der Einzelschlüssel mit dieser Nummer verwendet werden.

Wenn die logische Schlüsselnummer KID > '03' ist, wird die Verwendung eines Schlüssels aus

einer der beiden Schlüsselgruppen festgelegt. Hierbei legt die Nummer '04' die Verwendung eines Schlüssels aus der Schlüsselgruppe 1 (Schlüsselnummern zwischen '04' und '0E'), die Nummer '0F' die Verwendung eines Schlüssels aus der Schlüsselgruppe 2 (Schlüsselnummer > '0E') fest.

Wenn durch die AC die Verwendung eines Schlüssels aus einer Schlüsselgruppe festgelegt wird, muß die tatsächlich verwendete Schlüsselnummer KID' der Kommandonachricht des Ergänzungskommandos entnommen werden.

Durch den in Kapitel 2.8. in [LIT 1] beschriebenen Such-Algorithmus zur Schlüsselauswahl wird mit den Parametern

- "globaler/DF-spezifischer Schlüssel" aus der AC,
- "Schlüsselnummer KID" ('00' bis '03') aus der AC

  bzw. "Schlüsselnummer KID"' aus der Kommandonachricht und
- Datei, für die die AC festgelegt ist (hier: EF_BETRAG),

durch die AC und die Datei EF_BETRAG bzw. durch die AC, die Datei EF_BETRAG und die Schlüsselnummer KID' aus der Kommandonachricht ein Schlüssel eindeutig bestimmt.

1.1.5. Kodierung der ACs

Die Basis-ACs vom Typ ZERT für die Ergänzungskommandos der elektronischen Geldbörse werden in ein Byte kodiert. Die Kodierung des linken Halbbyte ist der folgenden Tabelle zu entnehmen:

AC	Wert (hex.)
ZERT_G	’A’
ZERT_D	’B’

Das rechte Halbbyte gibt die logische Schlüsselnummer des zu verwendenden Schlüssels ('0', '1', '2', '3') bzw. der Schlüsselgruppe ('4' und 'F') an.

Die AC für ein Ergänzungskommando der elektronischen Geldbörse wird in die FCP des EF_BETRAG gemäß Kapitel 5.4 in [LIT 1] in 2 Byte kodiert. Eine Basis-AC vom Typ ZERT wird hierbei durch die Kombination mit der Basis-AC ALW dargestellt.

Zur Identifikation des jeweiligen Ergänzungskommandos in den ACs des EF_BETRAG wird immer CLA = 'E0' und das entsprechende INS-Byte verwendet. Dies ist auch dann der Fall, wenn das Kommando ebenfalls mit CLA = 'E4' aufgerufen werden kann.

1.2. ADF der Applikation elektronische Geldbörse

Für das ADF der Applikation elektronische Geldbörse (DF_BÖRSE) sind beim Anlegen die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'16'		Tag und Länge für FCP
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A2 00'	Datei-ID des DF_BÖRSE
'84'	'09'	'D2 76 00 00 25 45 50 01 00'	DF-Name (AID) des DF_BÖRSE
'86'	'02'	'00 40'	AC für das DF_BÖRSE

Wenn das DF_BÖRSE mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FCP ausgegeben:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'1A'		Tag und Länge für FCP
'81'	'02'	'XX XX'	freier Speicherplatz in der Chipkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A2 00'	Datei-ID des DF_BÖRSE
'84'	'09'	'D2 76 00 00 25 45 50 01 00'	DF-Name (AID) des DF_BÖRSE
'86'	'02'	'00 40'	AC für das DF_BÖRSE

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

In zwei Byte wird der in der Chipkarte für das Anlegen weiterer Dateien zur Verfügung stehende freie Speicherplatz in Byte angegeben.

Tag '86'

Für die Kommandogruppe ADMIN wird für das DF_BÖRSE die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Wenn das DF_BÖRSE selektiert ist, darf also ein CREATE FILE, DELETE FILE, INCLUDE oder EXCLUDE nur ausgeführt werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels

K_{Card} aus dem EF_KEY des MF gebildet ist.

AEFs der elektronischen Geldbörse auf einer kontobezogenen Börsenkarte

Der Applikation elektronische Geldbörse auf einer kontobezogenen Börsenkarte sind 8 Dateien als AEF zuzuordnen:

SFI '16': EF_INFO im MF,

SFI '17': EF_ID im MF,

SFI '18': EF_BETRAG im DF_BÖRSE,

SFI '19': EF_BÖRSE im DF_BÖRSE,

SFI '1A': EF_LSEQ im DF_BÖRSE,

SFI '1B': EF_BSEQ im DF_BÖRSE,

SFI '1C': EF_LLOG im DF_BÖRSE,

SFI '1D': EF_BLOG im DF_BÖRSE

Wenn das DF_BÖRSE einer kontobezogenen Börsenkarte mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FMD mit den Pfaden der AEFs ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_BÖRSE direkt im MF befindet):

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
'64'	'34'		Tag und Länge für FMD
'85'	'03'	'16 01 00'	Pfad für das AEF mit SFI '16' (EF_INFO im MF)
'85'	'03'	'17 00 03'	Pfad für das AEF mit SFI '17' (EF_ID im MF)
'85'	'05'	'18 A2 00 01 04'	Pfad für das AEF mit SFI '18' (EF_BETRAG im DF_BÖRSE)
'85'	'05'	'19 A2 00 01 05'	Pfad für das AEF mit SFI '19' (EF_BÖRSE im DF_BÖRSE)
'85'	'05'	'1A A2 00 01 06'	Pfad für das AEF mit SFI '1A' (EF_LSEQ im DF_BÖRSE)
'85'	'05'	'1B A2 00 01 07'	Pfad für das AEF mit SFI '1B' (EF_BSEQ im DF_BÖRSE)
'85'	'05'	'1C A2 00 01 08'	Pfad für das AEF mit SFI '1C' (EF_LLOG im DF_BÖRSE)
'85'	'05'	'1D A2 00 01 09'	Pfad für das AEF mit SFI '1D' (EF_BLOG im DF_BÖRSE)

Wenn das DF_BÖRSE einer kontobezogenen Börsenkarte mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, wird die folgende FCI mit den ACs der AEFs im zusammengesetzten Datenobjekt mit Tag 'A5' ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_BÖRSE direkt im MF befindet):

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
'6F'	'74'		Tag und Länge für FCI
'81'	'02'	'XX XX'	freier Speicherplatz in der Chipkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A2 00'	Datei-ID des DF_BÖRSE
'84'	'09'	'D2 76 00 00 25 45 50 01 00'	DF-Name (AID) des DF_BÖRSE
'86'	'02'	'00 40'	AC für das DF_BÖRSE
'A5'	'58'		Tag und Länge für ACs der AEFs
'86'	'07'	'16 00 40 82 42 00 40'	AC für das AEF mit SFI '16' (EF_INFO im MF)
'86'	'07'	'17 00 40 00 00 00 F0'	AC für das AEF mit SFI '17' (EF_ID im MF)
'86'	'17'	'18 00 40 00 53 00 F0 E0 30 5F B2 E0 32 00 B2 E0 34 00 B4 E0 36 00 B4'	AC für das AEF mit SFI '18' (EF_BETRAG im DF_BÖRSE)
'86'	'07'	'19 00 40 00 53 00 40'	AC für das AEF mit SFI '19' (EF_BÖRSE im DF_BÖRSE)
'86'	'07'	'1A 00 40 00 53 00 40'	AC für das AEF mit SFI '1A' (EF_LSEQ im DF_BÖRSE)
'86'	'07'	'1B 00 40 00 53 00 40'	AC für das AEF mit SFI '1B' (EF_BSEQ im DF_BÖRSE)
'86'	'07'	'1C 00 40 00 53 00 40'	AC für das AEF mit SFI '1C' (EF_LLOG im DF_BÖRSE)
'86'	'07'	'1D 00 40 00 53 00 40'	AC für das AEF mit SFI '1D' (EF_BLOG im DF_BÖRSE)

AEFs der elektronischen Geldbörse auf einer Wertkarte

Der Applikation elektronische Geldbörse auf einer Wertkarte sind 7 Dateien als AEF zuzuordnen:

SFI '17': EF_ID im MF,

SFI '18': EF_BETRAG im DF_BÖRSE,

SFI '19': EF_BÖRSE im DF_BÖRSE,

SFI '1A': EF_LSEQ im DF_BÖRSE,

SFI '1B': EF_BSEQ im DF_BÖRSE,

SFI '1C': EF_LLOG im DF_BÖRSE,

SFI ’1D’: EF_BLOG im DF_BÖRSE

Wenn das DF_BÖRSE einer Wertkarte mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FMD mit den Pfaden der AEFs ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_BÖRSE direkt im MF befindet):

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’64’	’2F’		Tag und Länge für FMD
'85'	'03'	'17 00 03'	Pfad für das AEF mit SFI '17' (EF_ID im MF)
'85'	'05'	'18 A2 00 01 04'	Pfad für das AEF mit SFI '18' (EF_BETRAG im DF_BÖRSE)
'85'	'05'	'19 A2 00 01 05'	Pfad für das AEF mit SFI '19' (EF_BÖRSE im DF_BÖRSE)
'85'	'05'	'1A A2 00 01 06'	Pfad für das AEF mit SFI '1A' (EF_LSEQ im DF_BÖRSE)
'85'	'05'	'1B A2 00 01 07'	Pfad für das AEF mit SFI '1B' (EF_BSEQ im DF_BÖRSE)
'85'	'05'	'1C A2 00 01 08'	Pfad für das AEF mit SFI '1C' (EF_LLOG im DF_BÖRSE)
'85'	'05'	'1D A2 00 01 09'	Pfad für das AEF mit SFI '1D' (EF_BLOG im DF_BÖRSE)

Wenn das DF_BÖRSE einer Wertkarte mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, wird die folgende FCI mit den ACs der AEFs im zusammengesetzten Datenobjekt mit Tag 'A5' ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_BÖRSE direkt im MF befindet):

Tag	Länge (in Byte)	Wert	Erläuterung
'6F'	'6B'		Tag und Länge für FCI
'81'	'02'	'XX XX'	freier Speicherplatz in der Chipkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A2 00'	Datei-ID des DF_BÖRSE
'84'	'09'	'D2 76 00 00 25 45 50 01 00'	DF-Name (AID) des DF_BÖRSE
'86'	'02'	'00 40'	AC für das DF_BÖRSE
'A5'	'4F'		Tag und Länge für ACs der AEFs
'86'	'07'	'17 00 40 00 00 00 F0'	AC für das AEF mit SFI '17' (EF_ID im MF)
'86'	'17'	'18 00 40 00 53 00 F0 E0 30 5F B2 E0 32 00 F0 E0 34 00 B4 E0 36 00 B4'	AC für das AEF mit SFI '18' (EF_BETRAG im DF_BÖRSE)
'86'	'07'	'19 00 40 00 53 00 40'	AC für das AEF mit SFI '19' (EF_BÖRSE im DF_BÖRSE)
'86'	'07'	'1A 00 40 00 53 00 40'	AC für das AEF mit SFI '1A' (EF_LSEQ im DF_BÖRSE)
'86'	'07'	'1B 00 40 00 53 00 40'	AC für das AEF mit SFI '1B' (EF_BSEQ im DF_BÖRSE)
'86'	'07'	'1C 00 40 00 53 00 40'	AC für das AEF mit SFI '1C' (EF_LLOG im DF_BÖRSE)
'86'	'07'	'1D 00 40 00 53 00 40'	AC für das AEF mit SFI '1D' (EF_BLOG im DF_BÖRSE)

1.3. EF_KEY

Die applikationsspezifischen Schlüssel der Applikation elektronische Geldbörse sind sowohl in kontobezogenen Börsenkarten als auch in Wertkarten im EF_KEY des Applikationsverzeichnisses DF_BÖRSE einer Börsenkarte gespeichert. Dies sind

- ein 16 Byte langer kartenindividueller Schlüssel K_{LD} mit der Schlüsselnummer '02' zur Absicherung des Ladens und Entladens der elektronischen Geldbörse,
- ein 16 Byte langer kartenindividueller Schlüssel K_{CD} mit der Schlüsselnummer '03' zum MAC-gesicherten Lesen von Applikationsdaten,
- 10 kartenindividuelle Schlüssel K_{RD} der Länge 8 Byte aus der Schlüsselgruppe 1 mit den Schlüsselnummern '05' bis '0E' zur Absicherung der Abbuchungs- und Rückbuchungstransaktionen der elektronischen Geldbörse,
- 10 kartenindividuelle Schlüssel K_{LT} der Länge 8 Byte aus der Schlüsselgruppe 2 mit den Schlüsselnummern '0F' bis '18' zur Absicherung der Kommunikation zwischen Börsenkarte und Ladeterminal bei dem Laden gegen andere Zahlungsmittel.

Der Schlüssel K_{LD} ist nur der Börsenkarte und der für sie zuständigen Ladezentrale bekannt. Der

K_{LD} einer elektronischen Geldbörse wird aus einem KGK_{LD} unter Verwendung der Kartenidentifikationsdaten abgeleitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Diese Daten sind im EF_ID des MF der Börsenkarte abgelegt. Die für die Börsenkarte zuständige Ladezentrale verfügt über den KGK_{LD} und leitet den K_{LD} bei Bedarf ab.

Es können durch eine Ladezentrale verschiedene KGK_{LD} verwendet werden. Ein KGK_{LD} wird wie alle daraus abgeleiteten K_{LD} anhand der Schlüssel-Version (Generationsnummer) identifiziert. Die Schlüssel-Version zur logischen Schlüsselnummer '02' im zugehörigen EF_KEYD zeigt an, aus welchem KGK_{LD} der K_{LD} einer Börsenkarte abgeleitet ist. Die verwendete Schlüssel-Version wird der Ladezentrale bei jeder Ladeanfrage mitgeteilt.

Der K_{CD} ist aus einem KGK_{CD} unter Verwendung der Kartenidentifikationsdaten im EF_ID der Börsenkarte abgeleitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Ein Bankensonderfunktionsterminal oder ein Ladeterminal mit Sicherheitsmodul besitzt den KGK_{CD}, aus dem das Terminal mittels der Kartenidentifikationsdaten aus dem EF_ID der Börsenkarte bei Bedarf den K_{CD} ableitet.

Die K_{RD} einer elektronischen Geldbörse sind jeweils aus einem KGK_{RD} unter Verwendung der Kartenidentifikationsdaten im EF_ID der Börsenkarte abgeleitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Jede Händlerkarte besitzt genau einen KGK_{RD} mit einer Schlüsselnummer zwischen '05' und '0E', aus dem sie bei Bedarf mittels der Kartenidentifikationsdaten aus dem EF_ID der Börsenkarte den kartenindividuellen K_{RD} ableitet.

Die K_{LT} einer elektronischen Geldbörse sind jeweils aus einem KGK_{LT} unter Verwendung der Kartenidentifikationsdaten im EF_ID der Börsenkarte abgeleitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Jedes Sicherheitsmodul eines Ladeterminals, an dem gegen andere Zahlungsmittel geladen werden kann, besitzt genau einen KGK_{LT} mit einer Schlüsselnummer zwischen '0F' und '18', aus der es bei Bedarf mittels der Kartenidentifikationsdaten aus dem EF_ID der Börsenkarte den kartenindividuellen K_{LT} ableitet.

1.3.1. FCP

Für das EF_KEY des DF_BÖRSE sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		Tag und Länge für FCP
'81'	'02'	'01 76'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 10'	Datei-ID des EF_KEY
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_KEY

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_KEY enthält maximal 22 Records der Länge 17 Byte, so daß 374 Byte für die Nutzdaten des EF_KEY zu allokieren sind.

Tag ’82’

Das EF_KEY ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEY muß '0010' sein.

Tag ’86’

Für das Kommando READ RECORD wird die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.3.2. Daten

Das EF_KEY im DF_BÖRSE enthält 22 Records mit den DF-spezifischen Schlüsseln des DF_BÖRSE.

Logische Schlüsselnummer	Schlüssel
'02'	16 Byte langer K_{LD}
'03'	16 Byte langer K_{CD}
'05'	Konkatenation des ersten 8 Byte langen K_{RD} mit sich selbst
...	...
'0E'	Konkatenation des zehnten 8 Byte langen K_{RD} mit sich selbst
'0F'	Konkatenation des ersten 8 Byte langen K_{LT} mit sich selbst
...	...
'18'	Konkatenation des zehnten 8 Byte langen K_{LT} mit sich selbst

1.4. EF_KEYD

Das EF_KEYD im DF_BÖRSE enthält die Zusatzinformationen zu den DF-spezifischen Schlüsseln des DF_BÖRSE.

1.4.1. FCP

Für das EF_KEYD sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 6E'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 05'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 13'	Datei-ID des EF_KEYD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_KEYD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_KEYD enthält maximal 22 Records von 5 Byte Länge, so daß 110 Byte für die Nutzdaten des EF_KEYD zu allokieren sind.

Tag ’82’

Das EF_KEYD ist ein lineares EF, dessen Recordlänge zu 5 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEYD muß '0013' sein.

TAG ’86’

Für das Kommando READ RECORD wird die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.4.2. Daten

Das EF_KEYD enthält 22 Records, die die Zusatzinformation zu den DF-spezifischen Schlüsseln des DF_BÖRSE enthalten.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Fehlbedienungs-zähler	Schlüssel-Version
'02'	'10'	'07'	'FF'	'XX'
'03'	'10'	'07'	'FF'	'00'
'05'	'08'	'06'	'FF'	'00'
...	...	...	...	...
'0E'	'08'	'06'	'FF'	'00'
'0F'	'08'	'06'	'FF'	'00'
...	...	...	...	...
'18'	'08'	'06'	'FF'	'00'

1.5. EF_AUT

Das EF_AUT im DF_BÖRSE enthält den 16 Byte langen kryptographischen Schlüssel K_{CAM} für die Authentikation der elektronischen Geldbörse gegenüber der externen Welt mit dem Kommando INTERNAL AUTHENTICATE. Auf diese Weise kann sich ein Bankensonderfunktionsterminal von der Echtheit der elektronischen Geldbörse überzeugen.

Der kartenindividuelle Schlüssel K_{CAM} ist nur der Börsenkarte bekannt. Ein Bankensonderfunktionsterminal kann den KGK_{CAM} besitzen, aus dem es mittels der Kartenidentifikationsdaten aus dem EF_ID im MF der Karte den entsprechenden K_{CAM} ableiten kann.

1.5.1. FCP

Für das EF_AUT des DF_BÖRSE sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
'62'	'15'		Tag und Länge für FCP
'81'	'02'	'00 11'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 11'	Datei-ID des EF_AUT
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_AUT

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

Das EF_AUT enthält maximal einen Record von 17 Byte Länge, so daß 17 Byte für die Nutzdaten

des EF_AUT zu allokieren sind.

Tag ’82’

Das EF_AUT ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUT muß '0011' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_AUT die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_AUT die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.5.2. Daten

Das EF_AUT im DF_BÖRSE enthält einen 17 Byte langen Record mit dem K_{CAM} der elektronischen Geldbörse. Dieser hat die logische Schlüsselnummer '00'.

Logische Schlüsselnummer	Schlüssel
'00'	16 Byte langer K_{CAM}

1.6. EF_AUTD

Das EF_AUTD im DF_BÖRSE enthält die Zusatzinformationen zu dem K_{CAM} der elektronischen Geldbörse.

1.6.1. FCP

Für das EF_AUTD sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 04'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 04'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 14'	Datei-ID des EF_AUTD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_AUTD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_AUTD enthält maximal einen Record von 4 Byte Länge, so daß 4 Byte für die Nutzdaten des EF_AUTD zu allokieren sind.

Tag ’82’

Das EF_AUTD ist ein lineares EF, dessen Recordlänge zu 4 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUTD muß '0014' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_AUTD die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_AUTD die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.6.2. Daten

Das EF_AUTD enthält einen 4 Byte langen Record, der die Zusatzinformationen zu dem K_{CAM} enthält.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Schlüssel-Version
'00'	'10'	'07'	'00'

1.7. EF_PWD0

Das DF-spezifische EF_PWD0 ist nur in kontobezogenen Börsenkarten im DF_BÖRSE enthalten. In Record '01' dieses EF_PWD0 ist dann die ec-PIN im Format "Format 2 PIN Block" abgelegt.

1.7.1. FCP

Für das EF_PWD0 des DF_BÖRSE in kontobezogenen Börsenkarten sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'17'		Tag und Länge für FCP
'81'	'02'	'00 08'	für das EF_PWD0 allokierter Speicherplatz in Byte
'82'	'03'	'02 41 08'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 12'	Datei-ID des EF_PWD0
'86'	'08'	'00 60 00 F0 00 60 00 00'	ACs für das EF_PWD0

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

Das EF_PWD0 enthält maximal einen Record von 8 Byte Länge, so daß 8 Byte für die Nutzdaten des EF_PWD0 zu allokieren sind.

Tag '82'

Das EF_PWD0 ist ein lineares EF, dessen Recordlänge zu 8 Byte festgelegt ist.

Tag '83'

Die Datei-ID des EF_PWD0 muß '0012' sein.

Tag '86'

Für das Kommando READ RECORD wird die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit dem Kommando darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist. Hierbei ist der Schlüssel K_{Card} aus dem EF_KEY des MF zu verwenden.

Für das Kommando VERIFY wird die AC '00 00' (ALW) festgelegt.

1.7.2. Daten

Die ec-PIN ist im Format "Format 2 PIN Block" im Record ’01’ des EF_PWD0 einer kontobezogenen Börsenkarte abgelegt. Der Format 2 PIN Block wird gemäß Anhang A von ISO 10202-6 ([ISO 13]) aus der PIN gebildet. Sie hat eine Mindestlänge von 4 Ziffern und darf maximal 12 Ziffern lang sein. Die 8 Byte des Records sind wie folgt belegt

C	L	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

Erläuterung:

Jedes Feld repräsentiert ein Halbbyte.

C:	Kontroll-Feld, binär kodiert	hat immer den Wert '2'
L:	PIN-Länge, binär kodiert	mögliche Werte von '4' bis 'C'
P:	PIN-Ziffer, BCD-kodiert	
F:	Filler, binär kodiert	hat immer den Wert 'F'
P/F:	PIN-Ziffer/Filler	Belegung abhängig von der PIN-Länge

1.8. EF_PWDD0

Das EF_PWDD0 ist nur in kontobezogenen Börsenkarten im DF_BÖRSE enthalten. Im Record '01' sind die Zusatzinformationen zu der im DF-spezifischen EF_PWD0 abgelegten ec-PIN gespeichert.

1.8.1. FCP

Für das DF-spezifische EF_PWDD0 in kontobezogenen Börsenkarten sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 03'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 03'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 15'	Datei-ID des EF_PWDD0
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_PWDD0

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_PWDD0 enthält maximal einen Record von 3 Byte Länge, so daß 3 Byte für die Nutzdaten des EF_PWDD0 zu allokieren sind.

Tag ’82’

Das EF_PWDD0 ist ein lineares EF, dessen Recordlänge zu 3 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_PWDD0 muß '0015' sein.

Tag ’86’

Für das Kommando READ RECORD wird die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.8.2. Daten

Das EF_PWDD0 des DF_BÖRSE in einer kontobezogenen Börsenkarte enthält im 3 Byte langen Record '01' die Zusatzinformationen zur ec-PIN.

Karteninhaber-authentikation	Kodierung des Paßwortes	Mindestlänge
'01'	'20'	'04'

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Die Karteninhaberauthentikation erfolgt durch Paßwort-Verifikation, die durch den Wert '01' kodiert wird.

Byte 2

Die PIN ist im Format "Format 2 PIN Block" gespeichert.

Byte 3

Die PIN muß mindestens 4 Ziffern lang sein.

1.9. EF_BETRAG

EF_BETRAG bezeichnet das lineare EF, in dessen Record '01'

- der aktuell in der elektronischen Geldbörse enthaltene Betrag,
- der Maximalbetrag für den aktuellen Betrag und
- der maximale Transaktionsbetrag

gespeichert sind.

Durch die ACs, die für das EF_BETRAG und die Ergänzungskommandos der Applikation elektronische Geldbörse gesetzt werden, wird der Zugriff der Ergänzungskommandos auf das EF_BETRAG und die übrigen Daten der Applikation geregelt. Die hierzu zur Verfügung stehenden ACs und deren Bedeutung sind in den Kapiteln 2.3., 2.4. und 2.5. sowie im Kapitel 1.1. erläutert.

Durch die Kommandovarianten **Laden** und **Rückbuchen** kann der aktuelle Betrag in EF_BETRAG erhöht werden. Durch **Entladen** und **Abbuchen** kann er verringert werden. Die Maximalbeträge können bei Ausführung von **Laden** und **Entladen** optional verändert werden.

Da für das Kommando UPDATE RECORD die AC NEV gesetzt ist, kann der aktuelle Betrag der elektronischen Geldbörse nur mittels der genannten Kommandovarianten der Ergänzungskommandos geändert werden.

1.9.1. FCP

Für das EF_BETRAG in einer kontobezogenen Börsenkarte sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'25'		Tag und Länge für FCP
'81'	'02'	'00 09'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 09'	Datei-Deskriptor für lineares EF
'83'	'02'	'01 04'	Datei-ID des EF_BETRAG
'86'	'16'	'00 40 00 53 00 F0 E0 30 5F B2 E0 32 00 B2 E0 34 00 B4 E0 36 00 B4'	ACs für das EF_BETRAG

Für das EF_BETRAG in einer Wertkarte sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'25'		Tag und Länge für FCP
'81'	'02'	'00 09'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 09'	Datei-Deskriptor für lineares EF
'83'	'02'	'01 04'	Datei-ID des EF_BETRAG
'86'	'16'	'00 40 00 53 00 F0 E0 30 5F B2 E0 32 00 F0 E0 34 00 B4 E0 36 00 B4'	ACs für das EF_BETRAG

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

Das EF_BETRAG enthält maximal einen Record von 9 Byte Länge, so daß 9 Byte für die Nutzdaten des EF_BETRAG zu allokieren sind.

Tag '82'

Das EF_BETRAG ist ein lineares EF, dessen Recordlänge auf 9 Byte festgelegt ist.

Tag '83'

Die Datei-ID des EF_BETRAG ist '01 04'.

Tag '86'

Für das Kommando READ RECORD wird für das EF_BETRAG die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Der Record aus EF_BETRAG kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für das Kommando APPEND RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit dem Kommando darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist. Da das EF_BETRAG maximal einen Record enthält, ist dies maximal einmal möglich.

Für das Kommando UPDATE RECORD wird die AC '00 F0' (NEV) festgelegt.

Für das Ergänzungskommando **LADEN** (CLA, INS = 'E0 30') wird die kombinierte AC '5F B2' (PRO_D mit Schlüsselnummer 'F' und ZERT_D mit Schlüsselnummer '02') festgelegt. Die Bedeutung dieser AC ist in Kapitel 1.1. und in der Spezifikation des Kommandos in Kapitel 2.3. beschrieben.

Für das Ergänzungskommando **ENTLADEN** (CLA, INS = 'E0 32') wird für kontobezogene

Börsenkarten die AC '00 B2' (ZERT_D mit Schlüsselnummer '02') festgelegt. Die Bedeutung dieser AC ist in Kapitel 1.1. und in der Spezifikation des Kommandos in Kapitel 2.4. beschrieben.

Das Ergänzungskommando **ENTLADEN** ist für Wertkarten durch die AC NEV gesperrt.

Für die Ergänzungskommandos **ABBUCHEN** (CLA, INS = 'E0 34') und **RÜCKBUCHEN** (CLA, INS = 'E0 36') wird die AC '00 B4' (ZERT_D mit Schlüsselnummer '04') festgelegt. Die Bedeutung dieser AC ist in Kapitel 1.1. und in der Spezifikation der Kommandos in Kapitel 2.5. beschrieben.

1.9.2. Daten

Das EF_BETRAG enthält den 9 Byte langen Record '01', der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1-3

Byte 1-3 enthalten den 6-stelligen BCD-kodierten aktuellen Betrag, der zum Bezahlen mit der elektronischen Geldbörse zur Verfügung steht.

Byte 4-6

Byte 4-6 enthalten den 6-stelligen BCD-kodierten Maximalbetrag, der die Obergrenze für den aktuellen Betrag der elektronischen Geldbörse bildet.

Der Maximalbetrag kann mittels Varianten der Kommandos **LADEN** und **ENTLADEN** durch die zuständige Ladezentrale verändert werden. Eine Veränderung mittels UPDATE RECORD ist nicht möglich.

Byte 7-9

Byte 7-9 enthalten den 6-stelligen BCD-kodierten maximalen Transaktionsbetrag, der die Obergrenze für die einzeln aus der elektronischen Geldbörse abbuchbaren Beträge bildet.

Der maximale Transaktionsbetrag kann mittels Varianten der Kommandos **LADEN** und **ENTLADEN** durch die zuständige Ladezentrale verändert werden. Eine Veränderung mittels UPDATE RECORD ist nicht möglich.

Bemerkung

Durch den Multiplikator in Byte 21 des EF_ID im MF der Börsenkarte wird bestimmt, welchen Wert eine Einheit der BCD-kodierten Beträge bezogen auf die in Byte 18-20 des EF_ID kodierte Währung DEM hat. Zur Zeit ist der Multiplikator 10^{-2}, so daß die Beträge in Pfennigen angegeben sind.

1.10. EF_BÖRSE

EF_BÖRSE bezeichnet das lineare EF, in dessen Record '01' Karten- und Kontodaten gespeichert sind, die durch die elektronische Geldbörse verwendet werden.

Byte 1 des Record '01' enthält eine Kennung, anhand derer der **Kartentyp** der Börsenkarte erkannt werden kann. Hierbei sind zwei Typen zu unterscheiden:

- Kontobezogene Börsenkarten mit PIN und
- Wertkarten ohne Kontobezug und ohne PIN.

In Byte 2-11 des Record '01' im EF_BÖRSE sind für beide Kartentypen die Kontodaten des Börsenverrechnungskontos gespeichert.

Außerdem enthält der Record '01' des EF_BÖRSE in Byte 12-27 einen 16 Byte langen Wert, bei dem es sich bei kontobezogenen Börsenkarten um die Verschlüsselung mittels des K_{LD} von Bankleitzahl und Kontonummer des Kartenkontos sowie Kartenfolgenummer und Freizügigkeitsschlüssel der Karte handelt.

Diese Daten sind in kontobezogenen Börsenkarten identisch mit denen, die im EF_INFO des MF gespeichert sind. Sie werden hier verschlüsselt abgelegt, um das wiederholte Verschlüsseln während des Lade- oder Entladevorgangs zu vermeiden.

Bei Wertkarten sind Byte 12-27 des Record '01' im EF_BÖRSE mit '00' belegt.

1.10.1. FCP

Für das EF_BÖRSE sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 1B'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 1B'	Datei-Deskriptor für lineares EF
'83'	'02'	'01 05'	Datei-ID des EF_BÖRSE
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_BÖRSE

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_BÖRSE enthält maximal einen Record von 27 Byte Länge, so daß 27 Byte für die Nutzdaten des EF_BÖRSE zu allokieren sind.

Tag ’82’

Das EF_BÖRSE ist ein lineares EF, dessen Recordlänge auf 27 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_BÖRSE ist '01 05'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_BÖRSE die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Der Record aus EF_BÖRSE kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.10.2. Daten

Das EF_BÖRSE enthält den 27 Byte langen Record '01', der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Kennung für den Kartentyp '00': kontobezogene Börsenkarte 'FF': Wertkarte ohne Kontobezug
2-5	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Börsenverrechnungskonto
6-10	5	'nn..nn'	Kontonummer Börsenverrechnungskonto
11	1	'nD'	Prüfziffer über Byte 2-10
12-27	16	'XX..XX'	(Triple-)DES-Verschlüsselung im CBC-Mode mit K_{LD} und ICV = '00..00' der 16 Byte 4 Byte Bankleitzahl|5 Byte Kontonummer des Kartenkontos| 2 Byte Kartenfolgenummer|1 Byte Freizügigkeitsschlüssel| '00 00 00 00'

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Byte 1 enthält die binär kodierte Kennung für den Kartentyp. Hierbei bedeutet

'00': kontobezogene Börsenkarte mit PIN, verschlüsselte Konto- und Kartendaten in Byte 11-27

'FF': Wertkarte ohne Kontobezug und ohne PIN, Byte 11-27 enthalten '00'

Byte 2-5

Byte 2-5 enthalten die 8-stellige BCD-kodierte Bankleitzahl des kontoführenden Instituts für das Börsenverrechnungskonto.

Byte 6-10

Byte 6-10 enthalten die 10-stellige BCD-kodierte Kontonummer des Börsenverrechnungskontos.

Byte 11

Byte 11 enthält die Prüfziffer nach dem Verfahren 'Luhn formula for computing modulus 10' über die Ziffern aus Byte 2-10 sowie einen Feldseparator 'D'.

Byte 12-27

Byte 12-27 enthalten

- für kontobezogene Börsenkarten das binär kodierte Chiffrat von
  - 4 Byte BCD-kodierte Bankleitzahl des Kartenkontos,
  - 5 Byte BCD-kodierte Kontonummer des Kartenkontos,

- 2 Byte BCD-kodierte rechtsbündig eingestellte 3-stellige Kartenfolgenummer der Karte mit vorangestellter '0',
- 1 Byte BCD-kodierter rechtsbündig eingestellter 1-stelliger Freizügigkeitsschlüssel der Karte mit vorangestellter '0' und
- 4 Byte Filler '00 00 00 00'

unter K_{LD} im Triple-DES CBC-Mode mit ICV = '00..00',

- für Wertkarten '00..00'.

1.11. EF_LSEQ

EF_LSEQ bezeichnet ein lineares EF, in dessen Record '01' die Sequenznummer LSEQ für die Kommandos **LADEN** und **ENTLADEN** gespeichert ist.

Die Sequenznummer wird mit dem Wert 1 personalisiert und jedesmal durch die Karte um 1 inkrementiert, wenn ein Laden oder Entladen erfolgreich durchgeführt wurde. Der Überlauf der Sequenznummer wird dadurch angezeigt, daß sie den Wert 0 annimmt. Ist dies der Fall, kann die Karte nicht mehr geladen oder entladen werden.

Die Sequenznummer dient dazu, eine Reihenfolgeprüfung für die Lade- und Entladetransaktionen einer Börsenkarte in der Ladezentrale durchzuführen und somit ungerechtfertigtes Laden mittels aufgezeichneter Ladedaten zu verhindern.

1.11.1. FCP

Für das EF_LSEQ sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		Tag und Länge für FCP
'81'	'02'	'00 02'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 02'	Datei-Deskriptor für lineares EF
'83'	'02'	'01 06'	Datei-ID des EF_LSEQ
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_LSEQ

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

Das EF_LSEQ enthält maximal einen Record von 2 Byte Länge, so daß 2 Byte für die Nutzdaten

des EF_LSEQ zu allokieren sind.

Tag ’82’

Das EF_LSEQ ist ein lineares EF, dessen Recordlänge auf 2 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_LSEQ ist '01 06'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_LSEQ die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Der Record aus EF_LSEQ kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.11.2. Daten

Das EF_LSEQ enthält im 2 Byte langen Record '01' die binär kodierte Sequenznummer.

Die Sequenznummer muß mit dem Wert '00 01' personalisiert werden.

1.12. EF_BSEQ

EF_BSEQ bezeichnet ein lineares EF, in dessen Record '01' die Sequenznummer BSEQ für die Kommandos **ABBUCHEN** und **RÜCKBUCHEN** gespeichert ist. Die Sequenznummer dient dazu, Abbuchungs/Rückbuchungstransaktionen einer Börsenkarte eindeutig zu identifizieren, das Wiedereinspielen von Nachrichten bei einer Börsenzahlung zu erkennen und duplizierte Börsenkarten im System zu erkennen.

Die Sequenznummer wird mit dem Wert 1 personalisiert und jedesmal durch die Karte um 1 inkrementiert, wenn die Variante **Abbuchen** des Kommandos **ABBUCHEN** erfolgreich durchgeführt wurde. Der Überlauf der Sequenznummer wird dadurch angezeigt, daß sie den Wert 0 annimmt. Ist dies der Fall, kann keine weitere Abbuchung durchgeführt werden. Ein **Rückbuchen** des letzten abgebuchten Betrages ist aber noch möglich.

1.12.1. FCP

Für das EF_BSEQ sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 02'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 02'	Datei-Deskriptor für lineares EF
'83'	'02'	'01 07'	Datei-ID des EF_BSEQ
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_BSEQ

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_BSEQ enthält maximal einen Record von 2 Byte Länge, so daß 2 Byte für die Nutzdaten des EF_BSEQ zu allokieren sind.

Tag ’82’

Das EF_BSEQ ist ein lineares EF, dessen Recordlänge auf 2 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_BSEQ ist '01 07'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_BSEQ die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Der Record aus EF_BSEQ kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.12.2. Daten

Das EF_BSEQ enthält im 2 Byte langen Record '01' die binär kodierte Sequenznummer.

Die Sequenznummer muß mit dem Wert '00 01' personalisiert werden.

1.13. EF_LLOG

EF_LLOG bezeichnet ein zyklisches EF, in dem die letzten 3 Lade- bzw. Entladetransaktionen einer Börsenkarte protokolliert sind. Wenn bei Ausführung von **Laden einleiten** oder **Entladen einleiten** alle Prüfungen positiv verlaufen sind, schreibt die Börsenkarte durch ein internes Append die Protokolldaten der Lade- oder Entladetransaktion in Record '01' dieses EF. Wenn die Kommandovarianten **Laden einleiten wiederholen, Laden, Entladen einleiten wiederholen** oder **Entladen** erfolgreich ausgeführt wurden, wird der Record '01' durch ein internes Update entsprechend aktualisiert.

Der Record mit der Nummer '01' enthält also immer die Protokolldaten der letzten Lade- bzw. Entladetransaktion. Das Statusbyte des Records zeigt an, welche Kommandovariante bisher ausgeführt wurde und ob alle Schreibvorgänge durchgeführt werden konnten.

1.13.1. FCP

Für das EF_LLOG sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 63'	allokierter Speicherplatz in Byte
'82'	'03'	'06 41 21'	Datei-Deskriptor für zyklisches EF
'83'	'02'	'01 08'	Datei-ID des EF_LLOG
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_LLOG

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_LLOG enthält maximal drei Records von 33 Byte Länge, so daß 99 Byte für die Nutzdaten des EF_LLOG zu allokieren sind.

Tag ’82’

Das EF_LLOG ist ein zyklisches EF, dessen Recordlänge auf 33 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_LLOG ist '01 08'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_LLOG die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Ein Record aus EF_LLOG kann also bei Bedarf MAC-gesichert

mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.13.2. Daten

Das EF_LLOG enthält 3 Records, die jeweils den folgenden Aufbau haben:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'XX'	Statusbyte
2-3	2	'XX XX'	Sequenznummer LSEQ der Lade-/Entladetransaktion
4	1	'XX'	Wiederholungszähler WZ für das **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geladener bzw. entladener Betrag
8-10	3	'nn..nn'	(neuer) aktueller Betrag
11-13	3	'nn..nn'	AS-ID der Ladezentrale
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
25-27	4	JJJJ MM TT	Datum der Lade-/Entladetransaktion
28-31	3	HH MM SS	Uhrzeit der Lade-/Entladetransaktion
32-33	2	'XX XX'	Sequenznummer BSEQ des letzten Abbuchens bzw. Rückbuchens

Im folgenden werden die Komponenten eines Records näher erläutert:

Byte 1

Das Statusbyte ist in der Protokolldatei sowohl für das Laden und Entladen als auch für das Abbuchen und Rückbuchen enthalten. Es zeigt an,

- welche Kommandovariante der Kommandos **LADEN, ENTLADEN** (in EF_LLOG) bzw. **ABBUCHEN, RÜCKBUCHEN** (in EF_BLOG) in diesem Record protokolliert ist und
- in welchem Zustand die jeweilige Kommandovariante beendet wurde.

Das Statusbyte wird wie folgt kodiert:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x						Kommando: b4 b3 b2 von INS
0	0	0						**LADEN**
0	0	1						**ENTLADEN**
0	1	0						**ABBUCHEN**
0	1	1						**RÜCKBUCHEN**
1	x	x						RFU
			x	x	x			Kommandovariante: b8 b7 b6 von P1
						x		Secure Messaging
						0		ohne Secure Messaging
						1		mit Secure Messaging
							x	bei Kommandoende beschriebene Dateien
							0	erste Datei beschrieben
							1	letzte Datei beschrieben

Wenn ein Ergänzungskommando nicht nur protokolliert wird, sondern auch weitere Applikationsdaten verändert, wird zunächst die Protokollierung in Record '01' der Protokolldatei durchgeführt, wobei das Bit b1 des Statusbyte zu 0 gesetzt wird. Anschließend werden die übrigen Schreibvorgänge des Kommandos durchgeführt. Danach wird das Bit b1 des Statusbyte zu 1 gesetzt. Werden außer der Protokollierung keine Schreibvorgänge durch das Kommando durchgeführt, wird das Bit b1 des Statusbyte bei der Protokollierung direkt zu 1 gesetzt.

Wenn das Bit b1 des Statusbyte in Record '01' einer der beiden Protokolldateien bei Aufruf eines Ergänzungskommandos der Börsenkarte den Wert 0 hat, wird das Kommando mit einer Fehlermeldung abgebrochen.

Durch Bit b8..b2 des Statusbyte im jeweiligen Record '01' der beiden Protokolldateien wird angezeigt, welche Kommandovariante (mit oder ohne Secure Messaging) zuletzt protokolliert wurde. Die Ergänzungskommandos der elektronischen Geldbörse werten diese Bit aus, um festzustellen, welche Kommandovariante ausgeführt werden darf. Wird die erforderliche Reihenfolge von Kommandovarianten nicht eingehalten, wird die entsprechende Variante abgebrochen.

Die folgende Tabelle gibt einen Überblick über die möglichen Werte des Statusbyte in Byte 1 der Records in EF_LLOG, wenn das EF_LLOG als erste Datei beschrieben wurde (ST0) und wenn es als letzte Datei beschrieben (ST1) wurde.

Kommando	ST0	ST1
Laden einleiten ohne Secure Messaging	-	'01'
Laden einleiten mit Secure Messaging	-	'03'
Laden einleiten wiederholen ohne Secure Messaging	-	'05'
Laden einleiten wiederholen mit Secure Messaging	-	'07'
Laden ohne Secure Messaging ohne Änderung der Maximalbeträge	'10'	'11'
Laden mit Secure Messaging ohne Änderung der Maximalbeträge	'12'	'13'
Laden ohne Secure Messaging mit Änderung der Maximalbeträge	'14'	'15'
Laden mit Secure Messaging mit Änderung der Maximalbeträge	'16'	'17'
Entladen einleiten	-	'21'
Entladen einleiten wiederholen	-	'25'
Entladen ohne Änderung der Maximalbeträge	'30'	'31'
Entladen mit Änderung der Maximalbeträge	'34'	'35'

Das Statusbyte muß im Record '01' mit dem Wert '13' personalisiert werden.

Byte 2-3

In Byte 2-3 wird die binär kodierte Sequenznummer aus EF_LSEQ gespeichert, die bei der Durchführung der in diesem Record gespeicherten Lade- oder Entladetransaktion durch die Varianten der Kommandos **LADEN** und **ENTLADEN** verwendet wurde. Diese Sequenznummer wird sowohl von der Karte als auch durch die für die Karte zuständige Ladezentrale geführt.

Byte 2-3 in Record '01' werden mit '00 00' personalisiert.

Byte 4

In Byte 4 wird der binär kodierte Wiederholungszähler für die Kommandovarianten **Laden einleiten** und **Entladen einleiten** gespeichert.

Byte 4 in Record '01' wird mit '01' personalisiert.

Byte 5-7

Byte 5-7 enthalten den 6-stelligen BCD-kodierten, bei der Transaktion mittels einer Variante von **LADEN** oder **ENTLADEN** geladenen bzw. entladenen Betrag, dessen Wertigkeit durch Byte 21 des EF_ID im MF bestimmt wird (zur Zeit Pfennige).

Byte 8-10

Byte 8-10 sind immer 6-stellig BCD-kodiert. Wenn in diesem Record das Einleiten einer Transaktion protokolliert ist, enthalten Byte 8-10 den beim Einleiten aktuellen Betrag, andernfalls den durch die jeweilige Kommandovariante des **LADEN** oder **ENTLADEN** berechneten Betrag. Die Wertigkeit dieses Betrags wird durch Byte 21 des EF_ID im MF bestimmt (zur Zeit Pfennige).

Da es nur möglich ist, den gesamten aktuellen Betrag und keine Teilbeträge einer Börsenkarte zu entladen, müssen diese Byte den Wert '00 00 00' haben, wenn es sich um die Protokolldaten eines Entladens handelt.

Byte 11-13

In Byte 11-13 wird die 6-stellige BCD-kodierte AS-ID der an der Lade-/Entladetransaktion beteiligten Ladezentrale eingestellt. Die AS-ID wird aus der entsprechenden Kommandovariante von **LADEN** oder **ENTLADEN** entnommen.

Byte 14-21

Byte 14-21 enthalten die 16-stellige BCD-kodierte Terminal-ID, durch die das Terminal, an dem die Lade- oder Entladetransaktion durchgeführt wurde, eindeutig identifiziert wird. Diese ID wird aus der Kommandonachricht der entsprechenden Variante von **LADEN** oder **ENTLADEN** entnommen.

Byte 22-24

In Byte 22-24 ist die BCD-kodierte Trace-Nummer TSEQ des Terminals, an dem die Lade- oder Entladetransaktion durchgeführt wurde, gespeichert. Diese 3 Byte lange Nummer wird in der Kommandonachricht der jeweiligen Variante von **LADEN** oder **ENTLADEN** an die Karte übergeben.

Byte 25-28

Byte 25-28 enthalten das BCD-kodierte Datum JJJJ MM TT, das in den Kommandodaten der Kommandovariante von **LADEN** oder **ENTLADEN** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Ladeterminal keine Uhr besitzt.

Byte 29-31

Byte 29-31 enthalten die BCD-kodierte Uhrzeit HH MM SS, die in den Kommandodaten der jeweiligen Variante von **LADEN** oder **ENTLADEN** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Ladeterminal keine Uhr besitzt.

Byte 32-33

In Byte 32-33 wird die letzte vor der Ausführung der protokollierten Kommandovariante von **LADEN** oder **ENTLADEN** verwendete binär kodierte Zahlungssequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG eingetragen.

Byte 5-33 des Record ’01’ werden mit ’00’ personalisiert.

1.14. EF_BLOG

EF_BLOG bezeichnet ein zyklisches EF, in dem die letzten 15 Abbuchungs- bzw. Rückbuchungstransaktionen einer Börsenkarte protokolliert sind. Wenn bei Ausführung der Kommandovariante **Abbuchen** alle Prüfungen positiv verlaufen sind, schreibt die Börsenkarte durch ein internes Append die Protokolldaten der Transaktion in dieses EF. Nach positiv

verlaufenen Prüfungen des **Rückbuchen** wird der Vorgang durch ein internes Update im Record '01' des EF_BLOG protokolliert. Wenn zusätzlich alle Schreibvorgänge bei der Kommandoausführung erfolgreich durchgeführt wurden, wird dies durch ein internes Update des Statusbyte in den Protokolldaten des Record '01' vermerkt.

Der Record mit der Nummer '01' enthält also immer die Protokolldaten der letzten Abbuchungs- bzw. Rückbuchungstransaktion, die alle Prüfschritte erfolgreich durchlaufen hat. Das Statusbyte zeigt an, ob auch alle Schreibvorgänge durchgeführt werden konnten.

1.14.1. FCP

Für das EF_BLOG sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		Tag und Länge für FCP
'81'	'02'	'02 2B'	für das EF_BLOG allokierter Speicherplatz in Byte
'82'	'03'	'06 41 25'	Datei-Deskriptor für zyklisches EF
'83'	'02'	'01 09'	Datei-ID des EF_BLOG
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_BLOG

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag '81'

Das EF_BLOG enthält maximal 15 Records von 37 Byte Länge, so daß 555 Byte für die Nutzdaten des EF_BLOG zu allokieren sind.

Tag '82'

Das EF_BLOG ist ein zyklisches EF, dessen Recordlänge auf 37 Byte festgelegt ist.

Tag '83'

Die Datei-ID des EF_BLOG ist '01 09'.

Tag '86'

Für das Kommando READ RECORD wird für das EF_BLOG die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Ein Record aus EF_BLOG kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter

Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.14.2. Daten

Das EF_BLOG enthält 15 Records, die den folgenden Aufbau haben:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'XX'	Statusbyte
2-3	2	'XX XX'	Sequenznummer BSEQ der Abbuchungs-/Rückbuchungstransaktion
4-5	2	'XX XX'	Sequenznummer LSEQ des letzten erfolgreichen Ladens
6-8	3	'nn..nn'	abgebuchter/rückgebuchter Betrag
9-18	10	'nn..nD'	Händlerkartennummer der an der Transaktion beteiligten Händlerkarte
19-22	4	'XX..XX'	Sequenznummer HSEQ der beteiligten Händlerkarte
23-26	4	'XX..XX'	Sequenznummer SSEQ der beteiligten Händlerkarte
27-29	3	'nn..nn'	durch die Abbuchungs-/Rückbuchungstransaktion berechneter aktueller Betrag
30-33	4	JJJJ MM TT	Datum der Abbuchungs-/Rückbuchungstransaktion
34-36	3	HH MM SS	Uhrzeit der Abbuchungs-/Rückbuchungstransaktion
37	1	'XX'	Schlüsselnummer KID des zur Zertifikatsberechnung verwendeten K_{RD}

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Vergleiche Byte 1 des EF_LLOG.

Die folgende Tabelle gibt einen Überblick über die möglichen Werte des Statusbyte in Byte 1 der Records in EF_BLOG, wenn das EF_BLOG als erste Datei beschrieben wurde (ST0) und wenn es als letzte Datei beschrieben (ST1) wurde.

Kommando	ST0	ST1
Abbuchen	'50'	'51'
Rückbuchen	'70'	'71'

Das Statusbyte muß im Record '01' mit dem Wert '71' personalisiert werden.

Byte 2-3

In Byte 2-3 wird die binär kodierte Sequenznummer BSEQ gespeichert, die bei der Durchführung der in diesem Record gespeicherten Abbuchungs/Rückbuchungstransaktion durch **Abbuchen** oder **Rückbuchen** verwendet wurde.

Byte 4-5

In Byte 4-5 wird die binär kodierte Sequenznummer LSEQ gespeichert, die bei der letzten erfolgreichen Lade- oder Entladetransaktion verwendet wurde. Diese Sequenznummer ist Byte 2-3 des Record '01' oder '02' des EF_LLOG zu entnehmen.

Byte 6-8

Byte 6-8 enthalten den 6-stelligen BCD-kodierten, bei der Transaktion mittels **Abbuchen** bzw. **Rückbuchen** abgebuchten bzw. rückgebuchten Betrag, dessen Wertigkeit durch Byte 21 des EF_ID im MF bestimmt wird (zur Zeit Pfennige).

Byte 9-18

Byte 9-18 enthalten die Händlerkartennummer, durch die die Händlerkarte des Terminals, an dem das Abbuchen und eventuell Rückbuchen durchgeführt wurde, eindeutig identifiziert wird. Diese 10 Byte lange ID wird aus der Kommandonachricht von **Abbuchen** oder **Rückbuchen** entnommen.

Byte 19-22

In Byte 19-22 ist die binär kodierte Sequenznummer HSEQ der Händlerkarte, mit der das Abbuchen bzw. Rückbuchen durchgeführt wurde, gespeichert. Diese 4 Byte lange Sequenznummer wird in der Kommandonachricht von **Abbuchen** oder **Rückbuchen** an die Karte übergeben.

Byte 23-26

In Byte 23-26 ist die binär kodierte Summensequenznummer SSEQ der Händlerkarte, die bei dem Abbuchen bzw. Rückbuchen aktuell war, gespeichert. Diese 4 Byte lange Sequenznummer wird in der Kommandonachricht von **Abbuchen** oder **Rückbuchen** an die Karte übergeben.

Byte 27-29

Byte 27-29 enthalten den 6-stelligen BCD-kodierten neuen aktuellen Betrag, der durch **Abbuchen** oder **Rückbuchen** berechnet wurde. Die Wertigkeit dieses Betrags wird durch Byte 21 des EF_ID im MF bestimmt (zur Zeit Pfennige).

Byte 30-33

Byte 30-33 enthalten das BCD-kodierte Datum JJJJ MM TT, das in den Kommandodaten von **Abbuchen** oder **Rückbuchen** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Händlerterminal keine Uhr besitzt.

Byte 34-36

Byte 34-36 enthalten die BCD-kodierte Uhrzeit HH MM SS, die in den Kommandodaten der

Variante von **Abbuchen** oder **Rückbuchen** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Händlerterminal keine Uhr besitzt.

Byte 37

In Byte 37 wird die binär kodierte Schlüsselnummer KID des Schlüssels K_{RD} gespeichert, der bei der Durchführung der gespeicherten Abbuchungs- oder Rückbuchungstransaktion zur MAC-Berechnung über Kommando- und Antwortdaten verwendet wurde.

Byte 2-37 des Record ’01’ werden mit ’00’ personalisiert.

2. Ergänzungskommandos der elektronischen Geldbörse

Für die Applikation elektronische Geldbörse werden die Ergänzungskommandos **LADEN, ENTLADEN, ABBUCHEN, RÜCKBUCHEN** und **ANTWORT WIEDERHOLEN**, teilweise mit verschiedenen Varianten, verwendet.

Als Kommandoklasse CLA wird für die Ergänzungskommandos der elektronischen Geldbörse ohne Secure Messaging 'E0' (auch wenn ein Zertifikat in Kommando- und/oder Antwortdaten enthalten ist) und mit Secure Messaging (nur für **LADEN** und **ANTWORT WIEDERHOLEN**) 'E4' verwendet.

Der Parameter P2 wird immer zu '00' kodiert. Die folgende Tabelle zeigt die Kodierung von INS und P1 für die verschiedenen Kommandovarianten.

Bedeutung	INS	P1							
		b8	b7	b6	b5	b4	b3	b2	b1
immer 0 0 0 0 0, sonst RFU					x	x	x	x	x
LADEN	'30'	x	x	x					
- **Laden einleiten**		0	0	0					
- **Laden einleiten wiederholen**		0	0	1					
- **Laden** ohne Änderung der Maximalbeträge		1	0	0					
- **Laden** mit Änderung der Maximalbeträge		1	0	1					
- alle anderen Werte RFU									
ENTLADEN	'32'	x	x	x					
- **Entladen einleiten**		0	0	0					
- **Entladen einleiten wiederholen**		0	0	1					
- **Entladen** ohne Änderung der Maximalbeträge		1	0	0					
- **Entladen** mit Änderung der Maximalbeträge		1	0	1					
- alle anderen Werte RFU									
ABBUCHEN	'34'	x	x	x					
- **Abbuchen einleiten**		0	0	0					
- **Abbuchen**		1	0	0					
- alle anderen Werte RFU									
RÜCKBUCHEN	'36'	x	x	x					
- **Rückbuchen** des letzten abgebuchten Betrages		1	0	0					
- alle anderen Werte RFU									
ANTWORT WIEDERHOLEN	'38'	x	x	x					
- Ladedaten wiederholen		0	0	0					
- Zahlungsdaten wiederholen		0	0	1					
- alle anderen Werte RFU									

Kommando- und Antwortdaten aller Ergänzungskommandos außer **ANTWORT WIEDERHOLEN** beginnen immer mit einer 1 Byte langen ***Nachrichten-ID***, an der erkannt werden kann, ob Zertifikate unberechtigt vertauscht und wiederverwendet wurden. Die Nachrichten-ID ist in Abhängigkeit vom auszuführenden Kommando analog zum Statusbyte (vgl. Kapitel 1.13.) aufgebaut:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x						Kommando: b4 b3 b2 von INS
0	0	0						**LADEN**
0	0	1						**ENTLADEN**
0	1	0						**ABBUCHEN**
0	1	1						**RÜCKBUCHEN**
1	x	x						RFU
			x	x	x			Kommandovariante: b8 b7 b6 von P1
						x		Secure Messaging
						0		ohne Secure Messaging:
						1		mit Secure Messaging
							x	Nachrichten-ID für
							0	Kommandonachricht
							1	Antwortnachricht

Die folgende Tabelle gibt einen Überblick über die Nachrichten-IDs in Kommandodaten (NIDIN) und Antwortdaten (NIDOUT) der einzelnen Kommandovarianten:

Kommando	NIDIN	NIDOUT
Laden einleiten ohne Secure Messaging	'00'	'01'
Laden einleiten mit Secure Messaging	'02'	'03'
Laden einleiten wiederholen ohne Secure Messaging	'04'	'05'
Laden einleiten wiederholen mit Secure Messaging	'06'	'07'
Laden ohne Secure Messaging ohne Änderung der Maximalbeträge	'10'	'11'
Laden mit Secure Messaging ohne Änderung der Maximalbeträge	'12'	'13'
Laden ohne Secure Messaging mit Änderung der Maximalbeträge	'14'	'15'
Laden mit Secure Messaging mit Änderung der Maximalbeträge	'16'	'17'
Entladen einleiten	'20'	'21'
Entladen einleiten wiederholen	'24'	'25'
Entladen ohne Änderung der Maximalbeträge	'30'	'31'
Entladen mit Änderung der Maximalbeträge	'34'	'35'
Abbuchen einleiten	'40'	'41'
Abbuchen	'50'	'51'
Rückbuchen	'70'	'71'

Für die von verschiedenen Komponenten verwendeten Sequenznummern werden die folgenden Abkürzungen verwendet:

LSEQ: Sequenznummer für **LADEN** und **ENTLADEN** (2 Byte binär) der elektronischen Geldbörse,

WZ: Wiederholungszähler für **Laden einleiten (wiederholen) und Entladen einleiten (wiederholen)** (1 Byte binär) der elektronischen Geldbörse,

TSEQ: Trace-Nummer (3 Byte BCD-kodiert) des Ladeterminals,

BSEQ: Sequenznummer für **ABBUCHEN** und **RÜCKBUCHEN** (2 Byte binär) der elektronischen Geldbörse,

HSEQ: Sequenznummer für Transaktionen (4 Byte binär) der Händlerkarte,

SSEQ: Summensequenznummer (4 Byte binär) der Händlerkarte.

2.1. Returncodes

Die folgenden Tabellen enthalten die Returncodes, die von den Varianten der Ergänzungskommandos bei erfolgreicher Kommandoausführung oder bei Kommandoabbruch verwendet werden.

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Der Returncode '63 CX' "Use of internal retry routine" gibt an, daß die Karte einen Schreibvorgang mehrfach durchführen mußte, um das Kommando erfolgreich abzuschließen. Die Warnung weist auf interne Speicherprobleme der Karte hin und dient zu Diagnosezwecken.

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde.

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
‘64’	‘00’	No precise diagnosis wird ausgegeben, wenn bei der Ausführung des Kommandos ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
‘65’	‘81’	Memory failure wird ausgegeben, wenn die Ausführung fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC
'66'	'05'	Secure Messaging in CLA falsch
'66'	'11'	durch AC (und KID in Kommandodaten) referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler des durch eine AC (und KID in Kommandodaten) referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu dem durch eine AC (und KID in Kommandodaten) referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ des durch eine AC (und KID in Kommandodaten) referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID unzulässig oder fehlerhaft
'66'	'16'	KID in Kommandodaten stimmt nicht mit der durch eine AC referenzierten Schlüsselnummer überein oder ist nicht in der referenzierten Schlüsselgruppe enthalten
'66'	'81'	Kein Zugriff wegen AC NEV
'66'	'88'	Zertifikat falsch

SW1	SW2	Bedeutung
‘67’	‘00’	Wrong length (L_c incorrect or L_e (not) present)
'69'	'82'	Paßwort 0 nicht eingegeben
'69'	'85'	DF_BÖRSE bei Kommandoaufruf nicht selektiert
'69'	'88'	MAC falsch (Secure Messaging)
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn das Format der Kommandodaten nicht der Spezifikation entspricht oder Kommandodaten außerhalb des spezifizierten Wertebereichs liegen
'6A'	'82'	File not found wird ausgegeben, wenn ein EF, auf das das Kommando zugreifen will, nicht vorhanden ist oder Struktur oder Recordlänge des EF nicht der Spezifikation entsprechen
'6A'	'83'	Record not found wird ausgegeben, wenn das Kommando auf einen Record mit Recordnummer größer als die des LAST Record zugreifen müßte
'6A'	'84'	Speicherplatz reicht nicht aus wird ausgegeben, wenn der für eine Log-Datei allokierte Speicherplatz für die Protokollierung mittels eines internen Appends nicht ausreicht
‘6A’	‘86’	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported
'96'	'01'	Schreibvorgänge eines Ergänzungskommandos unvollständig
'96'	'02'	Applikationsdaten falsch kodiert
'96'	'CX'	Überlauf der Sequenznummer (WZ: X = 0, LSEQ: X = 1, BSEQ: X = 2)
'97'	'01'	Transaktionsbetrag 0
'97'	'02'	Transaktionsbetrag zu groß
'9F'	'XX'	Reihenfolge der Kommandos nicht eingehalten, letzte Kommandovariante war XX

2.2. Grundsätze des Kommandoablaufs

Für alle Ergänzungskommandos der elektronischen Geldbörse gilt

- Wenn das DF_BÖRSE (eindeutig identifiziert durch den DF-Namen) bei Kommandoaufruf nicht selektiert ist: Abbruch mit Returncode '69 85'.

- Die EFs, auf die durch die Kommandos explizit zugegriffen wird, befinden sich im aktuellen DF_BÖRSE. Alle Kommandovarianten können nur ausgeführt werden, wenn das DF_BÖRSE bei ihrem Aufruf aktuell ist. Das Auffinden eines EF, auf das zugegriffen werden soll, muß daher unabhängig von der Position des DF_BÖRSE in der Börsenkarte und unabhängig von für das DF_BÖRSE vergebenen SFI möglich sein.

  Wenn ein EF, aus dem im Verlauf der Kommandoausführung Daten gelesen werden sollen oder in das Daten geschrieben werden sollen, nicht gefunden werden kann oder Struktur (linear oder zyklisch) oder Recordlänge des EF nicht der Spezifikation entsprechen: Abbruch mit Returncode '6A 82'.

Durch die internen Lese- und Schreibroutinen können die Fehlercodes '64 00' und '65 81' verursacht werden.

Bei den internen Schreibroutinen wird unterschieden zwischen internem Update und internem Append.

- Das interne Update überschreibt Daten in den angegebenen Record und bricht mit dem Returncode '6A 83' ab, wenn der entsprechende Record nicht angelegt ist.

- Das interne Append muß wie APPEND RECORD die Struktur (linear oder zyklisch) des EF respektieren, dem ein Record hinzugefügt werden soll. Es schreibt immer einen ganzen Record. Bei fehlendem Speicherplatz führt es zu einem Abbruch mit '6A 84'.

Beide Arten von internen Schreibroutinen können zur Warnung '63 CX' führen.

- Falls die Längenangabe L_c nicht der Spezifikation entspricht oder nicht der tatsächlichen Länge der Kommandodaten entspricht, oder falls L_e fehlt, obwohl es gemäß Spezifikation vorhanden sein muß, oder wenn L_e vorhanden ist, obwohl es gemäß Spezifikation fehlen muß: Abbruch mit Returncode '67 00'.

- Wenn ein EF, aus dem im Verlauf der Kommandoausführung Daten gelesen werden sollen, den benötigten Record nicht enthält: Abbruch mit Returncode '6A 83'.

- Falls ein Speicherfehler von der ec-Karte erkannt wird: Abbruch mit Returncode '65 81'.

- Wenn für ein Ergänzungskommando und das EF_BETRAG keine AC gesetzt ist: Abbruch mit Returncode '66 03'

- Wenn für ein Ergänzungskommando und das EF_BETRAG eine AC gesetzt ist, die nicht der Kommandospezifikation entspricht: Abbruch mit Returncode '66 04'

- Wenn für ein Ergänzungskommando die AC NEV gesetzt ist: Abbruch mit Returncode '66 81'.

- Falls ein Schlüssel nicht gefunden wird: Abbruch mit Returncode '66 11'.

- Falls ein Paritätsfehler eines Schlüssels von der ec-Karte erkannt wird: Abbruch mit Returncode '66 12'.

- Falls die Zusatzinformationen zu einem Schlüssel nicht gefunden werden: Abbruch mit Returncode '66 13'.

- Falls der Fehlbedienungszähler eines Schlüssels abgelaufen ist: Abbruch mit Returncode '66 14'.

- Falls Schlüssellänge und Algorithmus-ID zu einem Schlüssel fehlerhaft oder unzulässig (Algorithmus-ID '01' oder '02') sind: Abbruch mit Returncode '66 15'.

- Jeder bei der Ausführung eines Ergänzungskommandos der elektronischen Geldbörse festgestellte MAC-Fehler führt zur Dekrementierung des Fehlbedienungszählers des

betroffenen Schlüssels.

- Die Ausführung jedes Ergänzungskommandos hat zur Folge, daß eine vorher erzeugte Zufallszahl oder ein vorher übergebener ICV ungültig werden.

- Ein ICV, der in den Kommandodaten eines Ergänzungskommandos an die Börsenkarte übergeben wird, ist nach Ausführung des Kommandos ebenfalls nicht mehr verwendbar.

- Durch das Bit b1 des Statusbyte im jeweiligen Record '01' der beiden Protokolldateien EF_LLOG und EF_BLOG im Verzeichnis DF_BÖRSE wird angezeigt, ob alle Schreibvorgänge der protokollierten Kommandovariante durchgeführt wurden. Ist dies der Fall, hat dieses Bit den Wert 1.

  Wenn das Bit b1 des Statusbyte in Record '01' des EF_LLOG oder des EF_BLOG den Wert 0 hat, gilt für **jedes** Ergänzungskommando der elektronischen Geldbörse: Abbruch mit Returncode '96 01'.

- Durch Bit b8..b2 des Statusbyte im jeweiligen Record '01' der beiden Protokolldateien wird angezeigt, welche Kommandovariante mit oder ohne Secure Messaging zuletzt protokolliert wurde. Die Ergänzungskommandos der elektronischen Geldbörse werten diese Bit aus, um festzustellen, welche Kommandovariante ausgeführt werden darf. Wird die erforderliche Reihenfolge von Kommandovarianten nicht eingehalten: Abbruch mit Returncode '9F XX', wobei in 'XX' das Statusbyte aus Record '01' des EF_LLOG (für Kommandovarianten des **LADEN** und **ENTLADEN**) bzw. des EF_BLOG (für Kommandovarianten des **ABBUCHEN** und **RÜCKBUCHEN**) ist.

- Immer wenn ein in einem EF der Börsenkarte gespeicherter Betrag durch ein Ergänzungskommando in Rechen- oder Vergleichsoperationen verwendet wird, wird geprüft, ob der jeweilige Betrag BCD-kodiert ist. Ist das nicht der Fall, wird das Kommando mit dem Returncode '96 02' abgebrochen.

- Alle Kommandovarianten geben bei Abbruch aufgrund eines festgestellten Fehlers (Returncode $\neq$ '90 00', '61 L_a') nur den Fehlercode und keine Antwortdaten aus.

- In der folgenden Beschreibung der Abläufe wird als Returncode bei erfolgreicher Beendung des Kommandos '90 00' ausgegeben.

  Wenn das jeweilige Kommando mit einem nicht spezifizierten Wert in L_e aufgerufen wird, kann das Kommando ebenfalls mit Returncode '61 L_a' beendet werden.

2.3. LADEN

2.3.1. Übersicht

Je nach Belegung des Parameters P1 in der Kommandonachricht kann mit dem Kommando **LADEN**

- ein Laden eingeleitet werden mit
  - Prüfung der Sequenznummer LSEQ oder des Wiederholungszählers WZ auf Überlauf,
  - Überprüfung der ACs und des Kartentyps für die Kommandovariante,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Prüfung ob die Erhöhung des aktuellen Betrags um den gewünschten Ladebetrag den Maximalbetrag nicht übersteigt,
  - Führen eines Wiederholungszählers,
  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat,
- ein Betrag in die elektronische Geldbörse geladen werden mit
  - Überprüfung der ACs für die Kommandovariante,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Prüfung des Zertifikats in den Kommandodaten,
  - Prüfung der Ladesequenznummer LSEQ und des Wiederholungszählers WZ in den Kommandodaten,
  - Prüfung, ob der zu ladende Betrag der beim Einleiten des Ladens angefragte ist oder ob er den Wert 0 hat,
  - Erhöhung des aktuellen Betrags um den Ladebetrag,
  - Inkrementieren der Ladesequenznummer,
  - Protokollierung,
  - optionaler Veränderung des Maximalbetrags für den aktuellen Betrag und/oder des maximalen Transaktionsbetrags der elektronischen Geldbörse,
  - Ausgabe von Antwortdaten.

Durch die AC, die für EF_BETRAG in DF_BÖRSE und **LADEN** festgelegt ist, wird der Zugriff aller Varianten des Kommandos **LADEN** auf Daten der elektronischen Geldbörse geregelt.

Das Kommando **LADEN** kann mit der AC NEV gesperrt werden.

Wenn das Laden nicht gesperrt ist (AC ≠ NEV), darf es nur möglich sein, wenn der entsprechende Ladebetrag von der zuständigen Ladezentrale autorisiert wurde. Hierzu müssen die Daten der Kommandonachricht des **Laden** mit einem Zertifikat der Ladezentrale versehen werden.

Bevor die Ladezentrale ein solches Zertifikat erzeugt, muß sie sich von der Echtheit der Börsenkarte und der für das Laden relevanten Daten überzeugen können. Hierzu muß die Börsenkarte die Antwortdaten des **Laden einleiten** (**wiederholen**) mit einem Zertifikat versehen.

Durch eine AC vom Typ ZERT für EF_BETRAG in DF_BÖRSE und das Kommando **LADEN** muß festgelegt werden, welcher Schlüssel zur Zertifikatsbildung verwendet werden soll. Durch die AC

darf keine Schlüsselgruppe (Schlüsselnummer > '03') angegeben werden.

Der durch diese AC vom Typ ZERT referenzierte kartenindividuelle Einzelschlüssel wird im folgenden als K_{LD} bezeichnet. Die Schlüssel-Version (Generationsnummer) wird KV genannt.

Welches Verfahren zum Laden einer Börsenkarte zulässig ist, hängt vom jeweiligen Kartentyp ab:

- Wertkarten können **nur** gegen andere Zahlungsmittel geladen werden.
- Kontobezogene Börsenkarten können **sowohl** vom Kartenkonto **als auch** gegen andere Zahlungsmittel geladen werden.

Das Laden von Wertkarten und kontobezogenen Börsenkarten gegen andere Zahlungsmittel soll nur möglich sein, wenn sich sowohl Börsenkarte als auch beteiligtes Ladeterminal von der Authentizität der jeweils anderen Komponente und deren Kommando- und Antwortdaten überzeugen kann. Hierzu wird der Lade-Dialog zwischen den beiden Komponenten mit Secure Messaging gemäß Kapitel 3.1 aus [LIT 1] abgesichert. Auf diese Weise wird sichergestellt, daß nur mit einem Sicherheitsmodul ausgerüstete Ladeterminals ein Laden gegen andere Zahlungsmittel durchführen können, daß wiedereingespielte Ladedaten von der Börsenkarte zurückgewiesen werden können und daß ein Ladeterminal authentisch über den Ausgang des Ladens informiert werden kann.

Für EF_BETRAG und **LADEN** ist daher immer eine AC vom Typ PRO festzulegen. Hierbei ist die Referenzierung einer Schlüsselgruppe zulässig. Die Kommandovarianten **Laden einleiten** (**wiederholen**) und **Laden** sind bei Aufruf mit CLA = 'E4' sowohl für die Kommando- als auch für die Antwortnachricht mit Secure Messaging (MAC-Bildung) auszuführen. Die Kommandonachricht enthält dann einen 8 Byte langen Wert als ICV für die MAC-Bildung über die Antwortnachricht und eine Schlüsselnummer KID. Der durch die AC vom Typ PRO bzw. durch die AC und die Schlüsselnummer KID referenzierte Schlüssel, der zur Durchführung des Secure Messaging zu verwenden ist, wird im folgenden mit K_{LT} bezeichnet.

Für Wertkarten (Kartentyp = 'FF') muß immer ein Laden gegen andere Zahlungsmittel durchgeführt werden, so daß für diesen Kartentyp immer ein Kommandoaufruf mit CLA = 'E4' zu erfolgen hat. Der Aufruf mit CLA = 'E0' wird abgewiesen.

Für kontobezogene Börsenkarten (Kartentyp = '00') ist ein Laden am Kartenkonto ohne Secure Messaging (Kommandoaufruf mit CLA = 'E0') erlaubt. Das Laden einer kontobezogenen Börsenkarte vom Kartenkonto mit einem Ladebetrag $\neq 0$ soll aber nur möglich sein, wenn sich der Karteninhaber vor Ausführung des Kommandos gegenüber der Börsenkarte bzw. der Applikation

elektronische Geldbörse durch korrekte Angabe seines Paßworts authentisiert hat. Auf diese Weise wird sichergestellt, daß niemand unbefugt Abbuchungen vom Kartenkonto veranlassen kann.

Bei Aufruf mit CLA = 'E0' muß daher immer eine Karteninhaberauthentikation gegenüber der Börsenkarte stattgefunden haben. Diese muß mit dem Paßwort, das durch den Such-Algorithmus aus Kapitel 2.9 in [LIT 1] mit den Parametern

- DF-spezifisches Paßwort,
- Paßwortnummer 0 und
- EF_BETRAG in DF_BÖRSE

eindeutig bestimmt ist **oder** mit dem globalen Paßwort aus dem EF_PWD0 des MF erfolgt sein. Hierbei wird zuerst geprüft, ob das DF-spezifische Paßwort korrekt angegeben wurde.

Bei jedem **Laden einleiten (wiederholen)** legt die externe Welt für kontobezogene Börsenkarten durch das CLA-Byte fest, welche Art der Absicherung für das **Laden einleiten (wiederholen)** und das zugehörige **Laden** verwendet werden soll. CLA = 'E0' zeigt an, daß die beiden Kommandovarianten mit einem Ladebetrag ≠ 0 nur nach einer Karteninhaberauthentikation durchgeführt werden sollen (Laden am Kartenkonto). CLA = 'E4' zeigt an, daß die Kommandovarianten mit Secure Messaging durchzuführen sind (Laden gegen andere Zahlungsmittel).

Die Kommandovariante **Laden** muß immer auf dieselbe Weise abgesichert sein wie das zugehörige **Laden einleiten (wiederholen)**. Im Statusbyte des EF_LLOG der elektronischen Geldbörse wird immer festgehalten, ob die letzte Kommandovariante mit oder ohne Secure Messaging durchgeführt wurde, so daß sichergestellt werden kann, daß beide Schritte einer Ladetransaktion in derselben Weise durchgeführt werden.

Insgesamt können für **LADEN** die folgenden ACs gesetzt werden:

- NEV,
- Typ ZERT mit Einzelschlüssel kombiniert mit Typ PRO.

Wenn die AC ≠ NEV ist, ist zusätzlich ist für die Ausführung der Kommandovarianten der Kartentyp (kontobezogene Börsenkarte oder Wertkarte) in Byte 1 des EF_BÖRSE auszuwerten.

Die AC für EF_BETRAG und das Kommando **LADEN** wird in den FCP des EF_BETRAG durch CLA, INS = 'E0 30' identifiziert. Sie ist auch einzuhalten, wenn eine Kommandovariante mit CLA = 'E4' aufgerufen wird.

2.3.2. Kommando- und Antwortnachrichten

2.3.2.1. Laden einleiten (wiederholen)

Command APDU ohne Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘30’	INS
3	1	'00' '20'	P1 für **Laden einleiten** P1 für **Laden einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1C’	L_c
6	1	'00' '04'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
7-9	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	‘3A’	L_e

Command APDU mit Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'00' '20'	P1 für **Laden einleiten** P1 für **Laden einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘2D’	L_c
6	1	'02' '06'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
7-9	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-41	8	'XX..XX'	ICV für die MAC-Bildung über die Antwortdaten
42	1	'XX'	Logische Schlüsselnummer KID des K_{LT}
43-50	8	'XX..XX'	(Retail-)CFB-MAC mit K_{LT} über die 48 Byte Byte 1-42|Byte 51|'00 00 00 00 00' mit ICV = Zufallszahl (GET CHALLENGE)
51	1	‘42’	L_e

Response APDU ohne Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'01' '05'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'
59-60	2	'XX XX'	Statusbyte SW1-SW2

Response APDU mit Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'03' '07'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'
59-66	8	'XX..XX'	(Retail-)CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-58|'00 00 00 00 00 00' mit ICV aus Kommandonachricht
67-68	2	'XX XX'	Statusbyte SW1-SW2

2.3.2.2. Laden

Command APDU ohne Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘30’	INS
3	1	'80' 'A0'	P1 für **Laden** ohne Änderung der Maximalbeträge P1 für **Laden** mit Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'10' '14'	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Command APDU mit Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'80' 'A0'	P1 für **Laden** ohne Änderung der Maximalbeträge P1 für **Laden** mit Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘3C’	L_c
6	1	'12' '16'	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49-56	8	'XX..XX'	ICV für die MAC-Bildung über die Antwortdaten
57	1	'XX'	Logische Schlüsselnummer KID des K_{LT}
58-65	8	'XX..XX'	(Retail-)CFB-MAC mit K_{LT} über die 64 Byte Byte 1-57|Byte 66|'00 00 00 00 00 00' mit ICV = Zufallszahl (GET CHALLENGE)
66	1	‘12’	L_e

Response APDU ohne Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’11’ ’15	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'nn..nn'	geladener Betrag
8-10	3	'nn..nn'	neuer aktueller Betrag
11-12	2	‘XX XX’	Statusbyte SW1-SW2

Response APDU mit Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’13’ ’17	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'nn..nn'	geladener Betrag
8-10	3	'nn..nn'	neuer aktueller Betrag
11-18	8	'XX..XX'	(Retail-)CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-10|'00 00 00 00 00 00' mit ICV aus Kommandonachricht
19-20	2	‘XX XX’	Statusbyte SW1-SW2

2.3.3. Ablauf in der Börsenkarte

Durch alle Kommandovarianten sind die Grundsätze des Kommandoablaufs in Kapitel 2.2. einzuhalten.

2.3.3.1. Laden einleiten (P1 = ’00’)

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls die Sequenznummer LSEQ in Record '01' des EF_LSEQ den Wert '00 00' hat:

Abbruch mit Returncode ’96 C1’.

- Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode ’6A 80’.

- Wenn der Kartentyp in Byte 1 des EF_BÖRSE keinen der spezifizierten Werte '00' oder 'FF' hat: Abbruch mit Returncode '96 02'.

- Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '11', '13', '15', '17' (vorher **Laden** beendet) oder '31', '35' (vorher **Entladen** beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

- Wenn das CLA-Byte den Wert 'E0' hat:

  - Falls der Kartentyp in Byte 1 des EF_BÖRSE den Wert 'FF' (Wertkarte) hat: Abbruch mit Returncode '66 05'.

  - Falls der Kartentyp in Byte 1 des EF_BÖRSE den Wert '00' (kontobezogene Börsenkarte) hat und der Ladebetrag ≠ 0 ist, aber weder das durch den Paßwort-Such-Algorithmus mit den Parametern EF_BETRAG, DF-spezifisch, Paßwortnummer 0 bestimmte Paßwort noch das Paßwort mit der Paßwortnummer 0 im MF korrekt eingegeben wurde: Abbruch mit Returncode '69 82'.

- Wenn das CLA-Byte den Wert 'E4' hat:

  - Falls nicht unmittelbar vorher GET CHALLENGE ausgeführt wurde: Abbruch mit Returncode '66 01'.

  - Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{LT} nicht mit der durch die AC vom Typ PRO referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

  - Falls der CFB-MAC mit K_{LT} nicht korrekt ist: Abbruch mit Returncode '69 88'.

- Falls der Ladebetrag in der Kommandonachricht nicht BCD-kodiert ist: Abbruch mit Returncode '6A 80'.

- Falls aktueller Betrag aus Record '01' des EF_BETRAG + Ladebetrag aus den Kommandodaten größer als Maximalbetrag aus Record '01' des EF_BETRAG: Abbruch mit Returncode '97 02'.

2. Das **Laden einleiten** wird in Record '01' des EF_LLOG durch ein internes Append protokolliert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'01' bzw. '03'	Statusbyte Nachrichten-ID + 1
2-3	2	'XX XX'	Sequenznummer LSEQ aus EF_LSEQ
4	1	'01'	Wiederholunszähler WZ für **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geprüfter Ladebetrag aus Kommandonachricht
8-10	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
11-13	3	'XX..XX'	Filler aus Kommandonachricht
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals aus Kommandonachricht
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals aus Kommandonachricht
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{LD} über die folgenden 50 Byte DATA mit 6 Byte Filler '00' gebildet:

   - Byte 1-10 aus Record '01' des EF_LLOG,
   - Byte 2-27 aus Record '01' des EF_BÖRSE,
   - Byte 1-7 aus Record '02' des EF_LLOG,
   - Byte 4-9 aus Record '01' des EF_BETRAG,
   - 1 Byte Schlüssel-Version (Generationsnummer) des K_{LD}.

4. Falls CLA = 'E0' war, werden DATA|ZERT als Antwortdaten mit Returncode '90 00' ausgegeben.

   Falls CLA = 'E4' war, wird über DATA|ZERT mit K_{LT} und ICV aus Kommandonachricht der CFB-MAC berechnet. Als Antwortdaten werden dann DATA|ZERT|CFB-MAC mit Returncode '90 00' ausgegeben.

2.3.3.2. Laden einleiten wiederholen (P1 = '20')

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls der Wiederholungszähler WZ in Byte 4 des Record '01' des EF_LLOG den Wert 'FF' hat: Abbruch mit Returncode '96 C0'.
   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.

- Wenn der Kartentyp in Byte 1 des EF_BÖRSE keinen der spezifizierten Werte '00' oder 'FF' hat: Abbruch mit Returncode '96 02'.

- Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '01', '03', '05', '07' (vorher **Laden einleiten** (**wiederholen**)) oder '21', '25' (vorher **Entladen einleiten** (**wiederholen**)) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

- Wenn das CLA-Byte den Wert 'E0' hat:

  - Falls der Kartentyp in Byte 1 des EF_BÖRSE den Wert 'FF' (Wertkarte) hat: Abbruch mit Returncode '66 05'.

  - Falls der Kartentyp in Byte 1 des EF_BÖRSE den Wert '00' (kontobezogene Börsenkarte) hat und der Ladebetrag $\neq 0$ ist, aber weder das durch den Paßwort-Such-Algorithmus mit den Parametern EF_BETRAG, DF-spezifisch, Paßwortnummer 0 bestimmte Paßwort noch das Paßwort mit der Paßwortnummer 0 im MF korrekt eingegeben wurde: Abbruch mit Returncode '69 82'.

- Wenn das CLA-Byte den Wert 'E4' hat:

  - Falls nicht unmittelbar vorher GET CHALLENGE ausgeführt wurde: Abbruch mit Returncode '66 01'.

  - Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{LT} nicht mit der durch die AC vom Typ PRO referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

  - Falls der CFB-MAC mit K_{LT} nicht korrekt ist: Abbruch mit Returncode '69 88'.

- Falls der Ladebetrag in der Kommandonachricht nicht BCD-kodiert ist: Abbruch mit Returncode '6A 80'.

- Falls aktueller Betrag aus Record '01' des EF_BETRAG + Ladebetrag aus den Kommandodaten größer als Maximalbetrag aus Record '01' des EF_BETRAG: Abbruch mit Returncode '97 02'.

2. Das **Laden einleiten wiederholen** wird in Record '01' des EF_LLOG durch ein internes Update protokolliert. Hierbei

   - wird die Sequenznummer LSEQ unverändert gelassen,

   - wird der Wiederholungszähler WZ um 1 inkrementiert,

   - werden die übrigen Felder mit den aktuellen Werten überschrieben.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'05' bzw. '07'	Statusbyte Nachrichten-ID + 1
2-3	2	'XX XX'	unverändert
4	1	'XX'	um 1 inkrementiert
5-7	3	'nn..nn'	geprüfter Ladebetrag aus Kommandonachricht
8-10	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
11-13	3	'XX..XX'	Filler aus Kommandonachricht
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals aus Kommandonachricht
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals aus Kommandonachricht
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{LD} über die folgenden 50 Byte DATA mit 6 Byte Filler '00' gebildet:

   - Byte 1-10 aus Record '01' des EF_LLOG,
   - Byte 2-27 aus Record '01' des EF_BÖRSE,
   - Byte 1-7 aus Record '02' des EF_LLOG,
   - Byte 4-9 aus Record '01' des EF_BETRAG,
   - 1 Byte Schlüssel-Version (Generationsnummer) des K_{LD}.

4. Falls CLA = 'E0' war, werden DATA|ZERT als Antwortdaten mit Returncode '90 00' ausgegeben.

   Falls CLA = 'E4' war, wird über DATA|ZERT mit K_{LT} und ICV aus Kommandonachricht der CFB-MAC berechnet. Als Antwortdaten werden dann DATA|ZERT|CFB-MAC mit Returncode '90 00' ausgegeben.

2.3.3.3. Laden

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.
   - Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '01', '03', '05', '07' (vorher **Laden einleiten** (**wiederholen**)) hat: Abbruch mit Returncode '9F XX', wobei 'XX'

den Wert des Statusbyte enthält.

- Wenn das CLA-Byte den Wert 'E0' hat:

  - Falls das Statusbyte in Record '01' des EF_LLOG einen der Werte '03' oder '07' hat (vorher **Laden einleiten** (**wiederholen**) mit Secure Messaging): Abbruch mit Returncode '66 05'.

  - Falls der Ladebetrag ≠ 0 ist und weder das durch den Paßwort-Such-Algorithmus mit den Parametern EF_BETRAG, DF-spezifisch, Paßwortnummer 0 bestimmte Paßwort noch das Paßwort mit der Paßwortnummer 0 im MF korrekt eingegeben wurde: Abbruch mit Returncode '69 82'.

- Wenn das CLA-Byte den Wert 'E4' hat:

  - Falls das Statusbyte in Record '01' des EF_LLOG einen der Werte '01' oder '05' hat (vorher **Laden einleiten** (**wiederholen**) ohne Secure Messaging): Abbruch mit Returncode '66 05'.

  - Falls nicht unmittelbar vorher GET CHALLENGE ausgeführt wurde: Abbruch mit Returncode '66 01'.

  - Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{LT} nicht mit der durch die AC vom Typ PRO referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

  - Falls der CFB-MAC mit K_{LT} nicht korrekt ist: Abbruch mit Returncode '69 88'.

- Falls bei **Laden** mit Änderung der Maximalbeträge die Maximalbeträge in der Kommandonachricht nicht BCD-kodiert sind: Abbruch mit Returncode '6A 80'.

- Wenn die Sequenznummer LSEQ aus der Kommandonachricht nicht mit der Sequenznummer LSEQ in Byte 2-3 des Record '01' des EF_LLOG übereinstimmt: Abbruch mit Returncode '6A 80'.

- Wenn der Wiederholungszähler WZ aus der Kommandonachricht nicht mit dem Wiederholungszähler in Byte 4 des Record '01' des EF_LLOG übereinstimmt: Abbruch mit Returncode '6A 80'.

- Wenn der Ladebetrag aus der Kommandonachricht weder mit dem Ladebetrag in Byte 5-7 des Record '01' des EF_LLOG übereinstimmt noch den Wert '00 00 00' hat: Abbruch mit Returncode '6A 80'.

- Wenn das Zertifikat aus der Kommandonachricht falsch ist: Abbruch mit Returncode '66 88'.

2. Der neue aktuelle Betrag wird berechnet:

   neuer aktueller Betrag = aktueller Betrag aus Record '01' des EF_BETRAG + Ladebetrag

aus den Kommandodaten.

3. Das **Laden** wird in Record ’01’ des EF_LLOG durch ein internes Update protokolliert. Hierbei

   - bleiben die Sequenznummer LSEQ, der Wiederholungszähler WZ, Terminal-ID und Sequenznummer des Terminals unverändert,

   - werden die übrigen Felder mit den aktuellen Werten überschrieben.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’10’, ’12’, ’14’ oder ’16’	Statusbyte Nachrichten-ID
2-3	2	’XX XX’	unverändert
4	1	'XX'	unverändert
5-7	3	'nn..nn'	geprüfter Ladebetrag aus Kommandonachricht
8-10	3	'nn..nn'	neuer aktueller Betrag
11-13	3	'nn..nn'	AS-ID aus Kommandonachricht
14-21	8	'nn..nn'	unverändert
22-24	3	'nn..nn'	unverändert
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

4. Die übrigen Daten werden geschrieben:

   - Wenn die Nachrichten-ID den Wert '14' oder '16' hat, wird der gesamte Record '01' des EF_BETRAG überschrieben:

     - Byte 1-3 werden mit dem neuen aktuellen Betrag überschrieben,

     - Byte 4-9 werden mit den neuen Maximalbeträgen aus der Kommandonachricht überschrieben.

   - Wenn die Nachrichten-ID den Wert '10' oder '12' hat, werden nur Byte 1-3 des Record '01' des EF_BETRAG mit dem neuen aktuellen Betrag überschrieben. Die Maximalbeträge bleiben unverändert.

   - Die Sequenznummer LSEQ in Record '01' des EF_LSEQ wird mit dem um 1 inkrementierten Wert überschrieben.

5. Das niedrigstwertige Bit des Statusbyte im Record '01' des EF_LLOG wird gesetzt.

6. Falls CLA = 'E0' war, werden Byte 1-10 des Record '01' des EF_LLOG (DATA) als Antwortdaten mit Returncode '90 00' ausgegeben.

Falls CLA = 'E4' war, wird über DATA mit K_{LT} und ICV aus Kommandonachricht der CFB-MAC berechnet. Als Antwortdaten werden dann die 18 Byte DATA|CFB-MAC mit Returncode '90 00' ausgegeben.

2.4. ENTLADEN

2.4.1. Übersicht

Das Entladen einer elektronischen Geldbörse mittels der Kommandovarianten des **ENTLADEN** ist nur für Börsenkarten möglich, die auf ein Kartenkonto bezogen sind, da bei einem Entladen der entsprechende Betrag vom Börsenverrechnungskonto auf das Kartenkonto gebucht wird.

Das Leeren von Wertkarten ohne Bezug auf ein Kartenkonto und das Entladen von kontobezogenen Börsenkarten gegen Bargeld erfolgt mit dem Kommando **ABBUCHEN**. Das Kommando **ENTLADEN** wird für Wertkarten mittels der AC NEV gesperrt.

Je nach Belegung des Parameters P1 in der Kommandonachricht kann mit dem Kommando **ENTLADEN**

- ein Entladen eingeleitet werden mit
  - Prüfung der Sequenznummer LSEQ oder des Wiederholungszählers WZ auf Überlauf,
  - Überprüfung der ACs für die Kommandovariante,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Führen eines Wiederholungszählers,
  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat,
- der gesamte aktuelle Betrag aus der Börse entladen werden mit
  - Überprüfung der ACs für die Kommandovariante,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Prüfung des Zertifikats in den Kommandodaten,
  - Prüfung der Ladesequenznummer LSEQ und des Wiederholungszählers WZ in den Kommandodaten,
  - Rücksetzen des aktuellen Betrags auf den Wert 0,
  - Inkrementieren der Ladesequenznummer,
  - Protokollierung,
  - optionaler Veränderung des Maximalbetrags für den aktuellen Betrag und/oder des maximalen Transaktionsbetrags der elektronischen Geldbörse,
  - Ausgabe von Antwortdaten mit Zertifikat.

Durch die AC, die für EF_BETRAG in DF_BÖRSE und **ENTLADEN** festgelegt ist, wird der Zugriff aller Varianten des Kommandos auf Daten der elektronischen Geldbörse geregelt.

Das Kommando **ENTLADEN** kann mit der AC NEV gesperrt werden.

Wenn das Entladen nicht gesperrt ist (AC ≠ NEV), soll es nur möglich sein, wenn der Entladevorgang von der zuständigen Ladezentrale autorisiert wurde. Hierzu müssen die Daten der Kommandonachricht des **Entladen** mit einem Zertifikat der Ladezentrale versehen werden. Die Antwortnachricht an die Ladezentrale muß durch die elektronische Geldbörse mit einem Zertifikat versehen werden, damit keine ungerechtfertigte Umbuchung vom Börsenverrechnungskonto auf das Kartenkonto erfolgt.

Bevor die Ladezentrale ein Zertifikat für **Entladen** erzeugt, muß sie sich von der Echtheit der Börsenkarte und der für das Entladen relevanten Daten überzeugen können. Hierzu muß die Börsenkarte die Antwortdaten des **Entladen einleiten** (**wiederholen**) mit einem Zertifikat versehen.

Durch eine AC vom Typ ZERT für EF_BETRAG in DF_BÖRSE und das Kommando **ENTLADEN** muß festgelegt werden, welcher Schlüssel zur Zertifikatsbildung verwendet werden soll. Durch die AC darf keine Schlüsselgruppe (Schlüsselnummer > '03') angegeben werden.

Der durch diese AC vom Typ ZERT referenzierte kartenindividuelle Schlüssel wird im folgenden als K_{LD} bezeichnet. Die Schlüssel-Version (Generationsnummer) wird KV genannt.

Insgesamt können für **ENTLADEN** die folgenden ACs gesetzt werden:

- NEV,
- Typ ZERT mit Einzelschlüssel.

Die AC für EF_BETRAG und das Kommando **ENTLADEN** wird in den FCP des EF_BETRAG durch CLA, INS = 'E0 32' identifiziert.

2.4.2. Kommando- und Antwortnachrichten

2.4.2.1. Entladen einleiten (wiederholen)

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'00' '20'	P1 für **Entladen einleiten** P1 für **Entladen einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1C’	L_c
6	1	'20' '24'	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
7-9	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
10-12	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
13-15	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-30	4	JJJJ MM TT	Datum des Ladeterminals
31-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	‘3A’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’21’ ’25’	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
2-3	2	‘XX XX’	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Entladebetrag
8-10	3	'nn..nn'	aktueller Betrag (identisch mit Byte 5-7)
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Unter K_{LD} verschlüsselte Konto- und Kartendaten
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'
59-60	2	'XX XX'	Statusbyte SW1-SW2

2.4.2.2. Entladen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'80' 'A0'	P1 für **Entladen** ohne Änderung der Maximalbeträge P1 für **Entladen** mit Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'30' '34'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
10-12	3	'XX..XX'	Filler (wird durch das Kommando nicht ausgewertet)
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-30	4	JJJJ MM TT	Datum des Ladeterminals
31-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’31’ ’35	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
5-7	3	'nn..nn'	Entladebetrag
8-10	3	'00..00'	neuer aktueller Betrag
11-12	2	‘XX XX’	Statusbyte SW1-SW2

2.4.3. Ablauf in der Börsenkarte

Durch alle Kommandovarianten sind die Grundsätze des Kommandoablaufs in Kapitel 2.2. einzuhalten.

2.4.3.1. Entladen einleiten (P1 = ’00’)

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
   - Falls die Sequenznummer LSEQ in Record '01' des EF_LSEQ den Wert '00 00' hat: Abbruch mit Returncode '96 C1'.
   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.
   - Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '11', '13', '15', '17' (vorher **Laden** beendet) oder '31', '35' (vorher **Entladen** beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Das **Entladen einleiten** wird in Record '01' des EF_LLOG durch ein internes Append protokolliert. Der Record '01' wird folgendermaßen belegt:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’21’	Statusbyte Nachrichten-ID + 1
2-3	2	’XX XX’	Sequenznummer LSEQ aus EF_LSEQ
4	1	’01’	Wiederholunszähler WZ für **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
8-10	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
11-13	3	'XX..XX'	Filler aus Kommandonachricht
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals aus Kommandonachricht
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals aus Kommandonachricht
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{LD} über die folgenden 50 Byte DATA mit 6 Byte Filler '00' gebildet:

   - Byte 1-10 aus Record '01' des EF_LLOG,
   - Byte 2-27 aus Record '01' des EF_BÖRSE,
   - Byte 1-7 aus Record '02' des EF_LLOG,
   - Byte 4-9 aus Record '01' des EF_BETRAG,
   - 1 Byte Schlüssel-Version (Generationsnummer) des K_{LD}.

4. DATA|ZERT werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.4.3.2. Entladen einleiten wiederholen (P1 = ’20’)

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
   - Falls der Wiederholungszähler WZ in Byte 4 des Record '01' des EF_LLOG den Wert 'FF' hat: Abbruch mit Returncode '96 C0'.
   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.
   - Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '01', '03', '05', '07' (vorher **Laden einleiten** (**wiederholen**)) oder '21', '25' (vorher **Entladen einleiten** (**wiederholen**)) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Das **Entladen einleiten wiederholen** wird in Record '01' des EF_LLOG durch ein internes Update protokolliert. Hierbei

   - wird die Sequenznummer LSEQ unverändert gelassen,
   - wird der Wiederholungszähler WZ um 1 inkrementiert,
   - werden die übrigen Felder mit den aktuellen Werten überschrieben.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'25'	Statusbyte Nachrichten-ID + 1
2-3	2	'XX XX'	unverändert
4	1	'XX'	um 1 inkrementiert
5-7	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
8-10	3	'nn..nn'	aktueller Betrag aus EF_BETRAG
11-13	3	'XX..XX'	Filler aus Kommandonachricht
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals aus Kommandonachricht
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals aus Kommandonachricht
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{LD} über die folgenden 50 Byte DATA mit 6 Byte Filler '00' gebildet:

   - Byte 1-10 aus Record '01' des EF_LLOG,
   - Byte 2-27 aus Record '01' des EF_BÖRSE,
   - Byte 1-7 aus Record '02' des EF_LLOG,
   - Byte 4-9 aus Record '01' des EF_BETRAG,
   - 1 Byte Schlüssel-Version (Generationsnummer) des K_{LD}.

4. DATA|ZERT werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.4.3.3. Entladen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.
   - Falls das Statusbyte in Record '01' des EF_LLOG keinen der Werte '21' oder '25' (vorher **Entladen einleiten** (**wiederholen**)) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.
   - Falls bei **Entladen** mit Änderung der Maximalbeträge die Maximalbeträge in der

Kommandonachricht nicht BCD-kodiert sind: Abbruch mit Returncode ’6A 80’.

- Wenn die Sequenznummer LSEQ aus der Kommandonachricht nicht mit der Sequenznummer LSEQ in Byte 2-3 des Record '01' des EF_LLOG übereinstimmt: Abbruch mit Returncode '6A 80'.

- Wenn der Wiederholungszähler WZ aus der Kommandonachricht nicht mit dem Wiederholungszähler in Byte 4 des Record '01' des EF_LLOG übereinstimmt: Abbruch mit Returncode '6A 80'.

- Wenn das Zertifikat aus der Kommandonachricht falsch ist: Abbruch mit Returncode '66 88'.

2. Das **Entladen** wird in Record '01' des EF_LLOG durch ein internes Update protokolliert. Hierbei

   - bleiben die Sequenznummer LSEQ, der Wiederholungszähler WZ, Terminal-ID und Sequenznummer des Terminals unverändert,

   - werden die übrigen Felder mit den aktuellen Werten überschrieben.

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’30’ oder ’34’	Statusbyte Nachrichten-ID
2-3	2	’XX XX’	unverändert
4	1	'XX'	unverändert
5-7	3	'nn..nn'	aktueller Betrag (entspricht Entladebetrag)
8-10	3	'00..00'	neuer aktueller Betrag
11-13	3	'nn..nn'	AS-ID aus Kommandonachricht
14-21	8	'nn..nn'	unverändert
22-24	3	'nn..nn'	unverändert
25-28	4	JJJJ MM TT	Datum aus Kommandonachricht
29-31	3	HH MM SS	Uhrzeit aus Kommandonachricht
32-33	2	'XX XX'	Sequenznummer BSEQ aus Byte 2-3 des Record '01' des EF_BLOG

3. Die übrigen Daten werden geschrieben:

   - Wenn die Nachrichten-ID den Wert '34' hat, wird der gesamte Record '01' des EF_BETRAG überschrieben:

     - Byte 1-3 werden mit '00 00 00' überschrieben,

     - Byte 4-9 werden mit den neuen Maximalbeträgen aus der Kommandonachricht überschrieben.

   - Wenn die Nachrichten-ID den Wert '30' hat, werden nur Byte 1-3 des Record '01' des

EF_BETRAG mit '00 00 00' überschrieben. Die Maximalbeträge bleiben unverändert.

- Die Sequenznummer LSEQ in Record '01' des EF_LSEQ wird mit dem um 1 inkrementierten Wert überschrieben.

4. Das niedrigstwertige Bit des Statusbyte im Record '01' des EF_LLOG wird gesetzt.

5. Byte 1-10 des Record '01' in EF_LLOG werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.5. **ABBUCHEN** und **RÜCKBUCHEN**

2.5.1. Übersicht

Je nach Belegung des Parameters P1 in der Kommandonachricht kann mit dem Kommando **ABBUCHEN**

- das Abbuchen eines Zahlungsbetrages eingeleitet werden mit
  - Prüfung der Zahlungssequenznummer auf Überlauf,
  - Ausgabe der Zahlungssequenznummer mit Zertifikat,
- ein Zahlungsbetrag aus der elektronischen Geldbörse abgebucht werden mit,
  - Prüfung des Zertifikats in den Kommandodaten,
  - Prüfung der Zahlungssequenznummer BSEQ auf Überlauf,
  - Prüfung der Zahlungssequenznummer BSEQ in den Kommandodaten,
  - Prüfung, ob der Zahlungsbetrag nicht 0 und nicht zu groß ist,
  - Verminderung des aktuellen Betrags um den Zahlungsbetrag,
  - Inkrementieren der Zahlungssequenznummer,
  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat.

Durch **Rückbuchen** kann der aktuelle Betrag wieder um den bei der letzten Zahlung abgebuchten Betrag erhöht werden mit

- Prüfung, ob die letzte Transaktion ein erfolgreiches Abbuchen zugunsten der Händlerkarte war, die die Rückbuchung durchführt,
- Prüfung, ob das Abbuchen mit der Sequenznummer HSEQ aus den Kommandodaten durchgeführt wurde,
- Prüfung, ob seit der Abbuchung keine Lade- oder Entladetransaktion begonnen wurde,

- Erhöhen des aktuellen Betrages um den Rückbuchungsbetrag.

Durch die AC, die jeweils für EF_BETRAG in DF_BÖRSE und die Kommandos **ABBUCHEN** und **RÜCKBUCHEN** festgelegt ist (identifiziert durch CLA, INS = 'E0 34' bzw. CLA, INS = 'E0 36'), wird der Zugriff der Kommandos auf Daten der elektronischen Geldbörse geregelt.

Jedes der beiden Kommandos kann durch die AC NEV gesperrt werden.

Wenn das **ABBUCHEN** nicht durch NEV gesperrt ist, muß die Antwortnachricht des **Abbuchen einleiten** mit einem Zertifikat versehen werden, da die Börsenkarte sich und die ausgegebene Zahlungssequenznummer durch diese Antwortnachricht authentisieren soll.

Die Authentizität von Daten einer Kommandonachricht zum Abbuchen muß durch die elektronische Geldbörse überprüfbar sein, damit unberechtigtes Abbuchen aus der elektronischen Geldbörse verhindert wird. Daher müssen diese Daten der Kommandonachricht für das **Abbuchen** mit einem Zertifikat versehen werden.

Die Antwortnachricht der Börsenkarte des **Abbuchen** muß mit einem Zertifikat versehen sein, damit die Authentizität der Nachricht nachprüfbar ist, so daß keine ungerechtfertigten Zahlungen zugunsten des Händlers verursacht werden können. Insbesondere kann anhand der Sequenznummern in der Antwortnachricht geprüft werden, ob versucht wird, alte Abbuchungsdaten einzuspielen.

Wenn das **RÜCKBUCHEN** nicht durch NEV gesperrt ist, muß die Authentizität von Daten einer Kommandonachricht zum **Rückbuchen** durch die elektronische Geldbörse überprüfbar sein, damit unberechtigtes Rückbuchen verhindert wird. Daher müssen diese Daten der Kommandonachricht für das Rückbuchen mit einem Zertifikat versehen werden.

Durch eine AC vom Typ ZERT für EF_BETRAG in DF_BÖRSE und die Kommandos **ABBUCHEN** und **RÜCKBUCHEN** muß festgelegt werden, welcher Schlüssel zur Zertifikatsbildung verwendet werden soll. Durch die AC darf eine Schlüsselgruppe (Schlüsselnummer > '03') angegeben werden. Der zur Zertifikatsbildung verwendete kartenindividuelle Schlüssel wird im folgenden als K_{RD} bezeichnet. Die logische Schlüsselnummer des K_{RD} wird KID genannt. Die zu verwendende Schlüsselnummer KID wird in den Kommandonachrichten von **Abbuchen** (**einleiten**) an die Börsenkarte übergeben. Die durch **Rückbuchen** zu verwendende Schlüsselnummer ist in Byte 37 des Record '01' des EF_BLOG gespeichert. Dies gilt für alle drei Kommandovarianten auch dann, wenn durch die AC ein Einzelschlüssel referenziert wird.

Insgesamt können für **ABBUCHEN** und **RÜCKBUCHEN** jeweils die folgenden ACs gesetzt werden:

- NEV,
- Typ ZERT.

Für die Varianten des **ABBUCHEN** und für **Rückbuchen** wird immer CLA = 'E0' verwendet, da zwar eine Zertifikatsbildung erfolgt, aber kein Secure Messaging im Sinne von Kapitel 3. aus [LIT 1].

2.5.2. Kommando- und Antwortnachrichten

2.5.2.1. Abbuchen einleiten

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'E0'	CLA
2	1	'34'	INS
3	1	'00'	P1 für **Abbuchen einleiten**
4	1	'00'	P2, fester Wert
5	1	'0A'	L_c
6	1	'40'	Nachrichten-ID für **Abbuchen einleiten**
7-14	8	'XX..XX'	Zufallszahl RND der Händlerkarte
15	1	'XX'	logische Schlüsselnummer KID des K_{RD}
16	1	'13'	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'41'	Nachrichten-ID für **Abbuchen einleiten**
2-3	2	'XX XX'	Sequenznummer BSEQ aus EF_BSEQ
4-11	8	'XX..XX'	RND aus der Kommandonachricht
12-19	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 1-11|'00 00 00 00 00'
20-21	2	'XX XX'	Statusbyte SW1-SW2

2.5.2.2. Abbuchen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘34’	INS
3	1	'80'	P1 für **Abbuchen**
4	1	‘00’	P2, fester Wert
5	1	‘28’	L_c
6	1	'50'	Nachrichten-ID für **Abbuchen**
7-8	2	'XX XX'	Sequenznummer BSEQ der Transaktion
9-18	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer)
19-22	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
23-26	4	'XX..XX'	Sequenznummer SSEQ der Händlerkarte
27-34	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 6-26|'00 00 00'
35-37	3	'nn..nn'	Zahlungsbetrag des Händlerterminals
38-41	4	JJJJ MM TT	Datum des Händlerterminals
42-44	3	HH MM SS	Uhrzeit des Händlerterminals
45	1	'XX'	logische Schlüsselnummer KID des K_{RD}
46	1	‘2B’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’51’	Nachrichten-ID für **Abbuchen**
2-3	2	'XX XX'	Sequenznummer BSEQ der Transaktion
4-5	2	'XX XX'	Sequenznummer LSEQ des letzten erfolgreichen Ladens
6-8	3	'nn..nn'	geprüfter Zahlungsbetrag
9-18	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer) aus der Kommandonachricht
19-22	4	'XX..XX'	Sequenznummer HSEQ aus der Kommandonachricht
23-32	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
33-40	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über Byte 1-32
41-43	3	'nn..nn'	neuer aktueller Betrag
44-45	2	‘XX XX’	Statusbyte SW1-SW2

2.5.2.3. Rückbuchen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘36’	INS
3	1	'80'	P1, Control Byte
4	1	‘00’	P2, fester Wert
5	1	‘1E’	L_c
6	1	'70'	Nachrichten-ID für **Rückbuchen**
7-16	10	’nn..nD’	Händlerkartennummer ('D' Hexziffer)
17-20	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
21-28	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 6-20|'00'
29-32	4	JJJJ MM TT	Datum des Händlerterminals
33-35	3	HH MM SS	Uhrzeit des Händlerterminals
36	1	'04'	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’71’	Nachrichten-ID für **Rückbuchen**
2-4	3	’nn..nn’	neuer aktueller Betrag
5-6	2	‘XX XX’	Statusbyte SW1-SW2

2.5.3. Ablauf in der Börsenkarte

2.5.3.1. Abbuchen einleiten

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.

   - Falls die Sequenznummer BSEQ in Record '01' des EF_BSEQ den Wert '00 00' hat: Abbruch mit Returncode '96 C2'.

   - Wenn die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.

- Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{RD} nicht mit der durch die AC vom Typ ZERT referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

2. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{RD} über die folgenden 11 Byte DATA mit 5 Byte Filler '00' gebildet:

   - 1 Byte '41' (Nachrichten-ID + 1),
   - Byte 1-2 aus Record '01' des EF_BSEQ,
   - 8 Byte Zufallszahl RND aus der Kommandonachricht.

3. DATA|ZERT werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.5.3.2. Abbuchen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
   - Falls die Sequenznummer BSEQ in Record '01' des EF_BSEQ den Wert '00 00' hat: Abbruch mit Returncode '96 C2'.
   - Falls die Nachrichten-ID nicht korrekt ist oder falls der Zahlungsbetrag in der Kommandonachricht nicht BCD-kodiert ist: Abbruch mit Returncode '6A 80'.
   - Falls der Zahlungsbetrag aus der Kommandonachricht den Wert '00 00 00' hat: Abbruch mit Returncode '97 01'.
   - Wenn die Sequenznummer BSEQ aus der Kommandonachricht nicht mit der Sequenznummer BSEQ in Byte 1-2 des Record '01' des EF_BSEQ übereinstimmt: Abbruch mit Returncode '6A 80'.
   - Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{RD} nicht mit der durch die AC vom Typ ZERT referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.
   - Wenn das Zertifikat aus der Kommandonachricht falsch ist: Abbruch mit Returncode '66 88'.
   - Falls aktueller Betrag aus Byte 1-3 oder maximaler Transaktionsbetrag aus Byte 7-9 des Record '01' des EF_BETRAG kleiner als der Zahlungsbetrag aus der Kommandonachricht ist: Abbruch mit Returncode '97 02'.

2. Der neue aktuelle Betrag wird berechnet:

   neuer aktueller Betrag = aktueller Betrag aus Record '01' des EF_BETRAG -

Zahlungsbetrag aus der Kommandonachricht.

3. Das Abbuchen wird in Record ’01’ des EF_BLOG durch ein internes Append protokolliert.

   Hierbei wird die zu protokollierende Sequenznummer LSEQ des letzten erfolgreichen Ladens wie folgt ermittelt:

   Wenn das linke Halbbyte des Statusbyte in Record ’01’ des EF_LLOG den Wert 1 hat, enthält der Record '01' des EF_LLOG die Protokolldaten eines erfolgreichen **Laden**. In diesem Fall wird LSEQ aus Byte 2-3 des Record '01' des EF_LLOG entnommen. Andernfalls wird es Byte 2-3 des Record '02' des EF_LLOG entnommen.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’50’	Statusbyte, Nachrichten-ID
2-3	2	’XX XX’	Sequenznummer BSEQ aus Record ’01’ des EF_BSEQ
4-5	2	’XX XX’	Sequenznummer LSEQ des letzten erfolgreichen Ladens
6-8	3	’nn..nn’	Zahlungsbetrag aus der Kommandonachricht
9-18	10	’nn..nD’	Händlerkartennummer aus der Kommandonachricht
19-22	4	'XX..XX'	Sequenznummer HSEQ aus der Kommandonachricht
23-26	4	'XX..XX'	Sequenznummer SSEQ aus der Kommandonachricht
27-29	3	'nn..nn'	neuer aktueller Betrag
30-33	4	JJJJ MM TT	Datum aus der Kommandonachricht
34-36	3	HH MM SS	Uhrzeit aus der Kommandonachricht
37	1	'XX'	Schlüsselnummer KID des verwendeten K_{RD} aus der Kommandonachricht

4. Die übrigen Daten werden geschrieben:

   - Byte 1-3 des Record '01' des EF_BETRAG werden mit dem neuen aktuellen Betrag überschrieben.

   - Die Sequenznummer BSEQ in Record '01' des EF_BSEQ wird mit dem um 1 inkrementierten Wert überschrieben.

5. Das niedrigstwertige Bit des Statusbyte in Record '01' des EF_BLOG wird gesetzt.

6. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit dem Schlüssel K_{RD} über die folgenden 32 Byte DATA gebildet:

   - Byte 1-22 aus Record '01' des EF_BLOG,
   - Byte 2-11 aus Record '01' des EF_BÖRSE,

7. DATA|ZERT|Byte 27-29 des Record '01' des EF_BLOG werden als Antwortdaten mit dem Returncode '90 00' ausgegeben.

2.5.3.3. Rückbuchen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
   - Falls die Nachrichten-ID nicht korrekt ist: Abbruch mit Returncode '6A 80'.
   - Falls das Statusbyte in Record '01' des EF_BLOG einen anderen Wert als '51' hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.
   - Falls die Händlerkartennummer aus der Kommandonachricht nicht mit dem in Record '01' des EF_BLOG gespeicherten Wert übereinstimmt: Abbruch mit Returncode '6A 80'.
   - Falls die Sequenznummer HSEQ aus der Kommandonachricht nicht mit dem in Record '01' des EF_BLOG gespeicherten Wert übereinstimmt: Abbruch mit Returncode '6A 80'.
   - Wenn das Zertifikat aus der Kommandonachricht falsch ist (die Schlüsselnummer KID wird Record '01' des EF_BLOG entnommen): Abbruch mit Returncode '66 88'.
   - Falls der in Byte 32-33 des Record '01' des EF_LLOG gespeicherte Wert von BSEQ mit dem in Byte 2-3 des Record '01' des EF_BLOG gespeicherten Wert übereinstimmt, ist seit der in Record '01' des EF_BLOG gespeicherten Abbuchung eine Lade- oder Entladetransaktion begonnen worden: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte aus Record '01' des EF_LLOG enthält.

2. Der neue aktuelle Betrag wird berechnet:

   neuer aktueller Betrag = aktueller Betrag aus Record '01' des EF_BETRAG + Zahlungsbetrag aus Byte 6-8 des Record '01' des EF_BLOG.

3. Das Rückbuchen wird in Record '01' des EF_BLOG durch ein internes Update protokolliert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'70'	Statusbyte, Nachrichten-ID
2-3	2	'XX XX'	unverändert
4-5	2	'XX XX'	unverändert
6-8	3	'nn..nn'	unverändert
9-18	10	'nn..nD'	unverändert
19-22	4	'XX..XX'	unverändert
23-26	4	'XX..XX'	unverändert
27-29	3	'nn..nn'	neuer aktueller Betrag
30-33	4	JJJJ MM TT	Datum aus der Kommandonachricht
34-36	3	HH MM SS	Uhrzeit aus der Kommandonachricht
37	1	'XX'	unverändert

4. Byte 1-3 des Record ’01’ des EF_BETRAG werden mit dem neuen aktuellen Betrag überschrieben.

5. Das niedrigstwertige Bit des Statusbyte in Record '01' des EF_BLOG wird gesetzt.

6. Die 4 Byte bestehend aus

   - Byte 1 des Record '01' im EF_BLOG,
   - Byte 27-29 des Record '01' im EF_BLOG

   werden als Antwortdatum zusammen mit Returncode '90 00' ausgegeben.

2.6. ANTWORT WIEDERHOLEN

2.6.1. Übersicht

Je nach Belegung des Parameters P1 in der Kommandonachricht können mit dem Kommando **ANTWORT WIEDERHOLEN**

- die Antwortdaten des letzten erfolgreichen **Laden** oder **Entladen** (mit und ohne Änderung der Maximalbeträge), das in Record '01' des EF_LLOG gespeichert ist, oder
- die Antwortdaten des letzten erfolgreichen **Abbuchen**, das in Record '01' des EF_BLOG gespeichert ist, erneut ausgegeben werden.

Hierzu wird CLA, INS der jeweils gespeicherten letzten erfolgreichen Kommandovariante aus dem Statusbyte in Byte 1 des Record '01' des EF_LLOG (P1 = '00') bzw. des EF_BLOG (P1 = '20') rekonstruiert.

ANTWORT WIEDERHOLEN kann sowohl mit CLA = 'E0' als auch mit CLA = 'E4' aufgerufen werden. CLA der gespeicherten Kommandovariante muß mit dem bei Aufruf von **ANTWORT WIEDERHOLEN** verwendeten CLA-Byte übereinstimmen.

Zur erneuten Ausgabe des CFB-MAC in den Antwortdaten eines mit Secure Messaging durchgeführten **Laden** benötigt die Börsenkarte einen ICV und eine Schlüsselnummer KID des verwendeten Schlüssels. ICV und KID sind in den Kommandodaten eines **Ladedaten wiederholen** mit CLA = 'E4' enthalten.

Durch die AC vom Typ PRO, die für **LADEN** gesetzt ist, wird festgelegt, welcher Schlüssel bzw. welche Schlüsselgruppe bei dem Secure Messaging für **Ladedaten wiederholen** zu verwenden ist. Die Schlüsselnummer KID in den Kommandodaten des **Ladedaten wiederholen** muß mit der durch die AC referenzierten Schlüsselnummer übereinstimmen oder in der durch die AC referenzierten Schlüsselgruppe enthalten sein.

Da eventuell ein neuer ICV in **Ladedaten wiederholen** verwendet wird, ist der CFB-MAC in der Antwortnachricht des **Ladedaten wiederholen** i. a. verschieden von dem MAC in der

ursprünglichen Antwortnachricht des **Laden**.

Für das Kommando **ANTWORT WIEDERHOLEN** selbst werden keine ACs gesetzt.

2.6.2. Kommandonachrichten

Command APDU ohne Secure Messaging

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘38’	INS
3	1	'00' '20'	P1 für **Ladedaten wiederholen** P1 für **Zahlungsdaten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘XX’	L_e, abhängig von der Länge der jeweils auszugebenden Antwortdaten

Command APDU mit Secure Messaging (nur für Ladedaten wiederholen)

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘38’	INS
3	1	'00'	P1 für **Ladedaten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘09’	L_c
6-13	8	'XX..XX'	ICV für die MAC-Bildung über die Antwortdaten
14	1	'XX'	Logische Schlüsselnummer KID des K_{LT}
15	1	‘XX’	L_e, abhängig von der Länge der jeweils auszugebenden Antwortdaten

2.6.3. Ablauf in der Börsenkarte

2.6.3.1. Ladedaten wiederholen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das Statusbyte in Record '01' des EF_LLOG keinen der in der folgenden Tabelle angegebenen Werte hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des

Statusbyte enthält.

Statusbyte	Kommando
’11’	**Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge
'13'	**Laden** mit Secure Messaging ohne Änderung der Maximalbeträge
'15'	**Laden** ohne Secure Messaging mit Änderung der Maximalbeträge
'17'	**Laden** mit Secure Messaging mit Änderung der Maximalbeträge
'31'	**Entladen** ohne Änderung der Maximalbeträge
'35'	**Entladen** mit Änderung der Maximalbeträge

- Falls das CLA-Byte den Wert ’E4’ hat:

  - Falls durch das Statusbyte in Record ’01’ des EF_LLOG eine Kommandovariante ohne Secure Messaging kodiert wird: Abbruch mit Returncode ’66 05’.

  - Falls die in Kommandodaten angegebene logische Schlüsselnummer KID des K_{LT} nicht mit der durch die AC vom Typ PRO für **LADEN** referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

- Falls das CLA-Byte den Wert 'E0' hat:

  - Falls durch das Statusbyte in Record '01' des EF_LLOG eine Kommandovariante mit Secure Messaging kodiert wird: Abbruch mit Returncode '66 05'.

2. Die Antwortdaten der durch das Statusbyte in Record '01' des EF_LLOG kodierten Kommandovariante werden wie für die Kommandovariante spezifiziert ausgegeben:

   Laden

   - Falls CLA = 'E0' war, werden Byte 1-10 des Record '01' des EF_LLOG (DATA) als Antwortdaten mit Returncode '90 00' ausgegeben.

     Falls CLA = 'E4' war, wird über DATA mit K_{LT} und ICV aus Kommandonachricht der CFB-MAC berechnet. Als Antwortdaten werden dann die 18 Byte DATA|CFB-MAC mit Returncode '90 00' ausgegeben.

   Entladen

   Byte 1-10 des Record '01' des EF_LLOG werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.6.3.2. Zahlungsdaten wiederholen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls das Statusbyte in Record '01' des EF_BLOG keinen der Werte '51' (**Abbuchen**) oder '71' (**Rückbuchen**) hat: Abbruch mit Returncode ’9F XX’, wobei ’XX’ den Wert des Statusbyte enthält.

   - Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.

2. Die Antwortdaten der durch das Statusbyte in Record '01' des EF_BLOG kodierten Kommandovariante werden wie für die Kommandovariante spezifiziert ausgegeben:

   Abbuchen

   - Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit dem Schlüssel K_{RD} (die Schlüsselnummer KID wird Record '01' des EF_BLOG entnommen) über die folgenden 32 Byte DATA gebildet:

     - Byte 1-22 aus Record '01' des EF_BLOG,
     - Byte 2-11 aus Record '01' des EF_BÖRSE,

   - DATA|ZERT|Byte 27-29 des Record '01' des EF_BLOG werden als Antwortdaten mit dem Returncode '90 00' ausgegeben.

   Rückbuchen

   Die 4 Byte bestehend aus

   - Byte 1 des Record ’01’ im EF_BLOG,
   - Byte 27-29 des Record ’01’ im EF_BLOG

   werden als Antwortdatum zusammen mit Returncode ’90 00’ ausgegeben.

Die elektronische Geldbörse – Händlerkarte –

Version 2.2
22.01.1997, Ausgabedatum: 31.07.1995

Inhalt

1. Daten der Händlerkarte

Mit ***Händlerkarte*** wird eine Chipkarte bezeichnet, deren Struktur der Daten, Sicherheitsarchitektur sowie Standard- und Administrationskommandos der Spezifikation in [LIT 1] entsprechen und die die Applikation Geldbörsen-SAM bestehend aus dem Applikationsverzeichnis (ADF) und den zugeordneten AEFs sowie die in Kapitel 2. spezifizierten Ergänzungskommandos enthält. Das ADF der Applikation Geldbörsen-SAM wird mit DF_BSAM bezeichnet.

Die Ergänzungskommandos der Applikation Geldbörsen-SAM greifen explizit auf die folgenden EFs zu:

- EF_SUMME,
- EF_KONTO,
- EF_HSEQ,
- EF_HLOG.

Diese müssen sich im DF_BSAM befinden. Da die Ergänzungskommandos der Applikation Geldbörsen-SAM nur ausgeführt werden können, wenn das DF_BSAM aktuell ist, ist für die Ergänzungskommandos das Auffinden dieser EFs unabhängig von der Position des DF_BSAM in der Händlerkarte und von für das DF_BSAM vergebenen SFIs möglich.

In Abhängigkeit davon, welche ACs für die Ergänzungskommandos der Applikation Geldbörsen-SAM gesetzt sind, müssen in der Händlerkarte globale und/oder DF-spezifische EF_KEYs und EF_PWDx mit den jeweiligen Deskriptor-EFs vorhanden sein, damit die für die Ergänzungskommandos gesetzten ACs erfüllt werden können.

Das MF der Händlerkarte muß die folgenden EFs enthalten:

- ein EF_KEY und zugehöriges EF_KEYD,
- das EF_ID und
- das EF_RAND, wenn der Zufallszahlengenerator wie in Kapitel 2.6 in [LIT 1] beschrieben realisiert wird.

Das MF der Händlerkarte kann ein EF LOG und ein EF_VERSION enthalten.

Das MF der Händlerkarte und die darin enthaltenen EF_LOG, EF_RAND und EF_VERSION entsprechen den Spezifikationen in Kapitel 6. von [LIT 1].

Das EF_ID entspricht der Spezifikation in Kapitel 6.9 von [LIT 1] bis auf die Belegung von Byte 18-20 des Records. In diese drei Byte wird anstelle des Währungskennzeichens der BCD-kodierte Entgeltcode eingetragen. Standardwert für den Entgeltcode ist '00 00 00'.

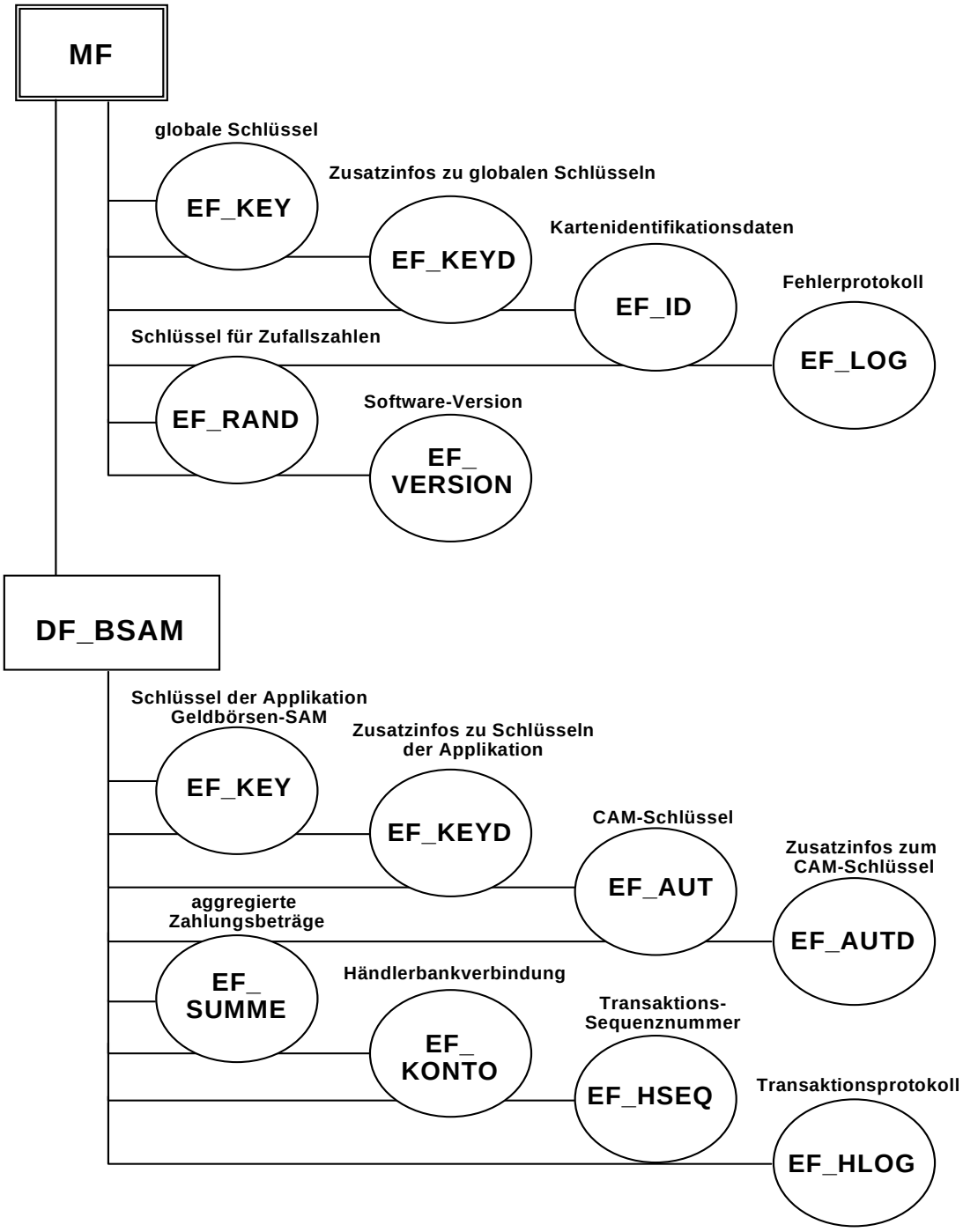
Im folgenden werden Aufbau, Datei-Kontrollinformation und Inhalt des EF_KEY und EF_KEYD im MF der Händlerkarte sowie des DF_BSAM und der darin enthaltenen EFs beschrieben.

Hierbei ist das DF_BSAM direkt im MF enthalten. Die für die Applikation Geldbörsen-SAM relevanten DF-spezifischen Schlüssel sind im EF_KEY abgelegt, das direkt im DF_BSAM enthalten ist.

Die Spezifikation der Ergänzungskommandos der Applikation Geldbörsen-SAM in Kapitel 2. setzt diese spezielle Kartenkonfiguration mit den hier gewählten ACs nicht voraus. Die Ergänzungskommandos müssen auch dann korrekt funktionieren, wenn

- das DF_BSAM nicht direkt im MF enthalten ist,
- für die Ergänzungskommandos andere, im Rahmen der Spezifikation zulässige ACs gesetzt sind,
- sich durch ACs referenzierte DF-spezifische Schlüssel nicht im EF_KEY des DF_BSAM, sondern eines übergeordneten DF befinden.

Die folgende Grafik gibt eine Übersicht über die Dateien der Händlerkarte.



1.1. EF_KEY im MF

Das EF_KEY im MF enthält den 16 Byte langen kartenindividuellen Schlüssel K_{Card} zur Absicherung der Kartenadministration.

1.1.1. FCP

Für das EF_KEY des MF sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 44'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 10'	Datei-ID des EF_KEY
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_KEY

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Für das globale EF_KEY ist Speicherplatz für vier Records allokiert, so daß drei weitere Schlüssel nachträglich mittels APPEND RECORD eingebracht werden können.

Tag ’82’

Das EF_KEY ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEY muß '0010' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das globale EF_KEY die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_KEY die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF_KEY zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.1.2. Daten

Das globale EF_KEY enthält im Record '01' den globalen kartenindividuellen Schlüssel K_{Card}, der die Schlüsselnummer '00' hat.

Logische Schlüsselnummer	Schlüssel
'00'	16 Byte langer K_{Card}

1.2. EF_KEYD im MF

Das EF_KEYD im MF enthält die Zusatzinformationen zu den globalen Schlüsseln der Händlerkarte, also zur Zeit zum K_{Card}.

1.2.1. FCP

Für das EF_KEYD des MF sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 14'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 05'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 13'	Datei-ID des EF_KEYD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_KEYD

Tag ’81’

Für das globale EF_KEYD ist Speicherplatz für vier Records allokiert, so daß Zusatzinformationen zu drei weiteren Schlüsseln nachträglich mittels APPEND RECORD eingebracht werden können.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_KEYD die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das globale EF_KEYD die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf auf dieses EF also nur zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.2.2. Daten

Das EF_KEYD enthält einen Record, der die Zusatzinformation zu dem K_{Card} enthält.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Fehlbedienungs-zähler	Schlüssel-Version
'00'	'10'	'07'	'FF'	'00'

1.3. ADF der Applikation Geldbörsen-SAM

Für das ADF der Applikation Geldbörsen-SAM (DF_BSAM) sind beim Anlegen die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’16’		Tag und Länge für FCP
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A3 00'	Datei-ID des DF_BSAM
'84'	'09'	'D2 76 00 00 25 42 53 01 00'	DF-Name (AID) des DF_BSAM
'86'	'02'	'00 40'	AC für das DF_BSAM

Wenn das DF_BSAM mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FCP ausgegeben:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’1A’		Tag und Länge für FCP
'81'	'02'	'XX XX'	freier Speicherplatz in der Händlerkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A3 00'	Datei-ID des DF_BSAM
'84'	'09'	'D2 76 00 00 25 42 53 01 00'	DF-Name (AID) des DF_BSAM
'86'	'02'	'00 40'	AC für das DF_BSAM

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

In zwei Byte wird der in der Händlerkarte für das Anlegen weiterer Dateien zur Verfügung stehende

freie Speicherplatz in Byte angegeben.

Tag ’86’

Für die Kommandogruppe ADMIN wird für das DF_BSAM die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Wenn das DF_BSAM selektiert ist, darf also ein CREATE FILE, DELETE FILE, INCLUDE oder EXCLUDE nur ausgeführt werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

Der Applikation Geldbörsen-SAM sind 6 Dateien als AEFs zugeordnet:

SFI '17': EF_ID im MF,

SFI '18': EF_KEYD im DF_BSAM,

SFI '19': EF_SUMME im DF_BSAM,

SFI '1A': EF_KONTO im DF_BSAM,

SFI '1B': EF_HSEQ im DF_BSAM,

SFI '1C': EF_HLOG im DF_BSAM.

Wenn das DF_BSAM mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FMD mit den Pfaden der AEFs ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_BSAM direkt im MF befindet):

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’64’	’28’		Tag und Länge für FMD
'85'	'03'	'17 00 03'	Pfad für das AEF mit SFI '17' (EF_ID im MF)
'85'	'05'	18 A3 00 00 13	Pfad für das ADF mit SFI '18' (EF_KEYD im DF_BSAM)
'85'	'05'	'19 A3 00 02 04'	Pfad für das AEF mit SFI '19' (EF_SUMME im DF_BSAM)
'85'	'05'	'1A A3 00 02 05'	Pfad für das AEF mit SFI '1A' (EF_KONTO im DF_BSAM)
'85'	'05'	'1B A3 00 02 06'	Pfad für das AEF mit SFI '1B' (EF_HSEQ im DF_BSAM)
'85'	'05'	'1C A3 00 02 07'	Pfad für das AEF mit SFI '1C' (EF_HLOG im DF_BSAM)

Wenn das DF_BSAM mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, wird die folgende FCI mit den ACs der AEFs im zusammengesetzten Datenobjekt mit Tag ’A5’ ausgegeben (hierbei wird vorausgesetzt, daß sich

das DF_BSAM direkt im MF befindet):

Tag	Länge (in Byte)	Wert	Erläuterung
’6F’	’5A’		Tag und Länge für FCI
'81'	'02'	'XX XX'	freier Speicherplatz in der Händlerkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A3 00'	Datei-ID des DF_BSAM
'84'	'09'	'D2 76 00 00 25 42 53 01 00'	DF-Name (AID) des DF_BSAM
'86'	'02'	'00 40'	AC für das DF_BSAM
'A5'	'3E'		Tag und Länge für ACs der AEFs
'86'	'07'	'17 00 40 00 00 00 F0'	AC für das AEF mit SFI '17' (EF_ID im MF)
'86'	'07'	18 00 40 00 00 00 40'	AC für das AEF mit SFI '18' (EF_KEYD im DF_BSAM)
'86'	'0F'	'19 00 40 00 53 00 40 E0 40 00 B4 E0 42 00 B1'	AC für das AEF mit SFI '19' (EF_SUMME im DF_BSAM)
'86'	'07'	'1A 00 40 00 53 00 40'	AC für das AEF mit SFI '1A' (EF_KONTO im DF_BSAM)
'86'	'07'	'1B 00 40 00 53 00 40'	AC für das AEF mit SFI '1B' (EF_HSEQ im DF_BSAM)
'86'	'07'	'1C 00 40 00 53 00 40'	AC für das AEF mit SFI '1C' (EF_HLOG im DF_BSAM)

1.4. EF_KEY

Zur Erzeugung von Zertifikaten für Einzeltransaktionen und Summensätze, zum MAC-gesicherten Lesen von Applikationsdaten und zur Absicherung der Kommunikation mit Börsenkarten benötigt die Händlerkarte die folgenden DF-spezifischen Schlüssel, die im EF_KEY des DF_BSAM abgelegt sind:

- ein 16 Byte langer händlerkartenindividueller Schlüssel K_{ZD} mit Schlüsselnummer '01' zur Erzeugung von Einzeltransaktions- und Summenzertifikaten,
- ein 16 Byte langer kartenindividueller Schlüssel K_{CD} mit der Schlüsselnummer '03' zum MAC-gesicherten Lesen von Applikationsdaten,
- ein 16 Byte langer Masterkey KGK_{RD} mit einer logischen Schlüsselnummer zwischen '05' und '0E' zur Ableitung der 8 Byte langen börsenkartenindividuellen Schlüssel K_{RD}, die der Absicherung der Abbuchungs- bzw. Rückbuchungstransaktionen mit einer Börsenkarte dienen.

Der händlerkartenindividuelle Schlüssel K_{ZD} ist nur der Händlerkarte und den Evidenz-Zentralen (EZ) bekannt. Die EZen besitzen den zugehörigen KGK_{ZD}, aus dem sie mittels der

Kartenidentifikationsdaten aus dem EF_ID der Händlerkarte den K_{ZD} ableitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Die EZen können über mehrere KGK_{ZD} verfügen. Diese und alle daraus abgeleiteten K_{ZD} werden durch ihre Schlüssel-Version (Generationsnummer) identifiziert. Die Schlüssel-Version zur logischen Schlüsselnummer des K_{ZD} im zugehörigen EF_KEYD der Händlerkarte zeigt an, aus welchem KGK_{ZD} der K_{ZD} einer Händlerkarte abgeleitet ist.

Der K_{CD} ist aus einem KGK_{CD} unter Verwendung der Kartenidentifikationsdaten im EF_ID der Händlerkarte abgeleitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Ein Bankensonderfunktionsterminal oder ein Ladeterminal mit Sicherheitsmodul besitzt den KGK_{CD}, aus dem das Terminal mittels der Kartenidentifikationsdaten aus dem EF_ID der Händlerkarte bei Bedarf den K_{CD} ableitet.

Jede Händlerkarte besitzt genau einen KGK_{RD} mit einer Schlüsselnummer KID zwischen '05' und '0E', aus dem sie bei Bedarf mittels der Kartenidentifikationsdaten aus dem EF_ID einer Börsenkarte den börsenkartenindividuellen K_{RD} ableitet (vgl. hierzu Kapitel 2.5 von [LIT 1]). Durch Angabe der Schlüsselnummer KID wird der Schlüssel an der Schnittstelle zur Börsenkarte identifiziert.

1.4.1. FCP

Für das EF_KEY des DF_BSAM sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 33'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 10'	Datei-ID des EF_KEY
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_KEY

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_KEY enthält maximal 3 Records der Länge 17 Byte, so daß 51 Byte für die Nutzdaten des EF_KEY zu allokieren sind.

Tag ’82’

Das EF_KEY ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEY muß '0010' sein.

Tag ’86’

Für das Kommando READ RECORD wird die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.4.2. Daten

Das EF_KEY im DF_BSAM enthält 3 Records mit den DF-spezifischen Schlüsseln des DF_BSAM.

Logische Schlüsselnummer	Schlüssel
'01'	16 Byte langer K_{ZD}
'03'	16 Byte langer K_{CD}
'XX'	16 Byte langer KGK_{RD}

Die logische Schlüsselnummer 'XX' muß einen Wert zwischen '05' und '0E' haben.

1.5. EF_KEYD

Das EF_KEYD im DF_BSAM enthält die Zusatzinformationen zu den DF-spezifischen Schlüsseln des DF_BSAM.

1.5.1. FCP

Für das EF_KEYD sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 0F'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 05'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 13'	Datei-ID des EF_KEYD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_KEYD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_KEYD enthält maximal 3 Records von 5 Byte Länge, so daß 15 Byte für die Nutzdaten des EF_KEYD zu allokieren sind.

Tag ’82’

Das EF_KEYD ist ein lineares EF, dessen Recordlänge zu 5 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KEYD muß '0013' sein.

TAG ’86’

Für das Kommando READ RECORD wird die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.5.2. Daten

Das EF_KEYD enthält 3 Records, die die Zusatzinformation zu den DF-spezifischen Schlüsseln des DF_BSAM enthalten.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Fehlbedienungs-zähler	Schlüssel-Version
'01'	'10'	'07'	'FF'	'YY'
'03'	'10'	'07'	'FF'	'00'
'XX'	'10'	'02'	'FF'	'00'

1.6. EF_AUT

Das EF_AUT im DF_BSAM enthält den 16 Byte langen kryptographischen Schlüssel K_{CAM} für die Authentikation der Händlerkarte gegenüber der externen Welt mit dem Kommando INTERNAL AUTHENTICATE. Auf diese Weise kann sich ein Bankensonderfunktionsterminal von der Echtheit der Händlerkarte überzeugen.

Der kartenindividuelle Schlüssel K_{CAM} ist nur der Händlerkarte bekannt. Ein Bankensonderfunktionsterminal kann den KGK_{CAM} besitzen, aus dem es mittels der Kartenidentifikationsdaten aus dem EF_ID im MF der Karte den entsprechenden K_{CAM} ableiten kann.

1.6.1. FCP

Für das EF_AUT des DF_BSAM sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 11'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 11'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 11'	Datei-ID des EF_AUT
'86'	'06'	'00 60 00 F0 00 60'	ACs für das EF_AUT

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_AUT enthält maximal einen Record von 17 Byte Länge, so daß 17 Byte für die Nutzdaten des EF_AUT zu allokieren sind.

Tag ’82’

Das EF_AUT ist ein lineares EF, dessen Recordlänge auf 17 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUT muß '0011' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_AUT die AC '00 F0' (NEV) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_AUT die AC '00 60' (ENC_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen und verschlüsselt ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.6.2. Daten

Das EF_AUT im DF_BSAM enthält einen 17 Byte langen Record mit dem K_{CAM} der Applikation Geldbörsen-SAM. Dieser hat die logische Schlüsselnummer '00'.

Logische Schlüsselnummer	Schlüssel
'00'	16 Byte langer K_{CAM}

1.7. EF_AUTD

Das EF_AUTD im DF_BSAM enthält die Zusatzinformationen zu dem K_{CAM} der Applikation Geldbörsen-SAM.

1.7.1. FCP

Für das EF_AUTD sind die folgenden FCP festzulegen:

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 04'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 04'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 14'	Datei-ID des EF_AUTD
'86'	'06'	'00 40 00 00 00 40'	ACs für das EF_AUTD

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_AUTD enthält maximal einen Record von 4 Byte Länge, so daß 4 Byte für die Nutzdaten des EF_AUTD zu allokieren sind.

Tag ’82’

Das EF_AUTD ist ein lineares EF, dessen Recordlänge zu 4 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_AUTD muß '0014' sein.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_AUTD die AC '00 00' (ALW) festgelegt.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_AUTD die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, der unter Verwendung des Schlüssels K_{Card} aus dem EF_KEY des MF gebildet ist.

1.7.2. Daten

Das EF_AUTD enthält einen 4 Byte langen Record, der die Zusatzinformationen zu dem K_{CAM} enthält.

Logische Schlüsselnummer	Schlüssellänge	Algorithmus-ID	Schlüssel-Version
'00'	'10'	'07'	'00'

1.8. EF_SUMME

EF_SUMME bezeichnet ein zyklisches EF, in dessen Records die Daten geführt werden, die zur Erzeugung eines Summensatzes dienen, der zusammen mit den Einzeltransaktionen vom Händler bei seiner Evidenz-Zentrale eingereicht wird. Diese Daten werden im folgenden als ***Summendaten*** bezeichnet.

In den Records von EF_SUMME ist insbesondere die Sequenznummer SSEQ gespeichert, die der eindeutigen Identifikation eines Summensatzes dient, um das wiederholte Einreichen von Summensätzen erkennen zu können.

Durch die Kommandovariante **Kassenschnitt** des Kommandos **ZERTIFIKAT** wird aus den Summendaten des Records '01' des EF_SUMME ein Summensatz mit Zertifikat erzeugt. Durch **Kassenschnitt** wird außerdem mit einem internen Append der Record '01' des EF_SUMME zum Summieren neuer Zahlungsbeträge vorbereitet. Hierbei werden SSEQ um 1 inkrementiert sowie Anzahl und Summe der Zahlungsbeträge auf 0 zurückgesetzt.

Ein Summensatz mit Zertifikat kann zu den Summendaten in einem beliebigen Record des EF_SUMME mit der Kommandovariante **Summensatz** von **ZERTIFIKAT** so lange ausgegeben werden, wie die entsprechenden Summendaten nicht durch die Protokolldaten weiterer Transaktionen überschrieben sind.

Durch die Kommandovarianten **Zahlung** und **Fehlzahlung** des Kommandos **ZERTIFIKAT** wird der jeweilige Record '01' durch ein internes Update aktualisiert, wobei die Anzahl der summierten Zahlungsbeträge (Transaktionszähler TZ) um 1 inkrementiert wird und die Summe der Zahlungsbeträge bei **Zahlung** um den Zahlungsbetrag erhöht wird und bei **Fehlzahlung** unverändert bleibt.

Die Kommandovariante **Zahlung prüfen** des Ergänzungskommandos **ZAHLUNG** prüft, ob der gewünschte Zahlungsbetrag nicht zu einem Überlauf der Summe von Zahlungsbeträgen führt.

Durch die ACs, die für das EF_SUMME und die Ergänzungskommandos **ZAHLUNG** und **ZERTIFIKAT** der Applikation Geldbörsen-SAM gesetzt werden, wird der Zugriff der Kommandovarianten auf die Daten der Applikation geregelt. Die hierzu zur Verfügung stehenden ACs und deren Bedeutung sind in den Kapiteln 2.3. und 2.4. und in Kapitel 1.1 in [LIT 4A] erläutert.

1.8.1. FCP

Für das EF_SUMME sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’1D’		Tag und Länge für FCP
'81'	'02'	'00 27'	allokierter Speicherplatz in Byte
'82'	'03'	'06 41 0D'	Datei-Deskriptor für zyklisches EF
'83'	'02'	'02 04'	Datei-ID des EF_SUMME
'86'	'0E'	'00 40 00 53 00 40 E0 40 00 B4 E0 42 00 B1'	ACs für das EF_SUMME

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_SUMME enthält maximal drei Records von 13 Byte Länge, so daß 39 Byte für die Nutzdaten des EF_SUMME zu allokieren sind.

Tag ’82’

Das EF_SUMME ist ein zyklisches EF, dessen Recordlänge auf 13 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_SUMME ist ’02 04’.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_SUMME die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Die Records aus EF_SUMME können also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, wobei der Schlüssel K_{Card} zu verwenden ist.

Für das Ergänzungskommando **ZAHLUNG** (CLA, INS = 'E0 40') wird die AC '00 B4' (ZERT_D mit Schlüsselnummer '04') festgelegt. Die Bedeutung dieser AC ist in Kapitel 1.1 in [LIT 4A] und in der Spezifikation des Kommandos in Kapitel 2.3. beschrieben.

Für das Ergänzungskommando **ZERTIFIKAT** (CLA, INS = 'E0 42') wird die AC '00 B1' (ZERT_D mit Schlüsselnummer '01') festgelegt. Die Bedeutung dieser AC ist in Kapitel 1.1 in [LIT 4A] und in der Spezifikation des Kommandos in Kapitel 2.4. beschrieben.

1.8.2. Daten

Das EF_SUMME enthält drei 17 Byte lange Records, die den folgenden Aufbau haben:

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	’XX..XX’	Sequenznummer SSEQ des Summensatzes
5-8	4	’XX..XX’	Transaktionszähler TZ, Anzahl Transaktionen seit letztem **Kassenschnitt**
9-13	5	'nn..nn'	Summe der Zahlungsbeträge seit letztem **Kassenschnitt**

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1-4

Byte 1-4 enthalten die binär kodierte Sequenznummer SSEQ des Summensatzes.

Die Sequenznummer SSEQ in Record '01' des EF_SUMME wird mit dem Wert 1 personalisiert. Durch die Variante **Kassenschnitt** des Kommandos **ZERTIFIKAT** wird der Wert von SSEQ um 1 inkrementiert und bei einem internen Append in den neuen Record '01' des EF_SUMME geschrieben. Der Überlauf der Sequenznummer wird dadurch angezeigt, daß sie den Wert 0

annimmt. Ist dies der Fall, kann keine weitere Transaktion durch die Händlerkarte durchgeführt werden. Ein weiterer **Kassenschnitt** ist ebenfalls nicht mehr möglich.

Die Sequenznummer in Record '01' muß mit dem Wert '00 00 00 01' personalisiert werden.

Byte 5-8

Byte 5-8 enthalten die binär kodierte Anzahl TZ der seit dem letzten **Kassenschnitt** erfolgten Zahlungen und Fehlzahlungen. TZ wird im Record '01' mit 0 personalisiert. Durch **Kassenschnitt** wird TZ auf den Wert 0 zurückgesetzt und bei einem internen Append in den neuen Record '01' geschrieben. TZ wird jedesmal bei einem internen Update des Record '01' um 1 inkrementiert, wenn **Zahlung** oder **Fehlzahlung** erfolgreich beendet werden. Wenn TZ in Record '01' den Wert 'FF FF FF FF' hat, kann vor einem **Kassenschnitt** keine weitere Transaktion durchgeführt werden.

Byte 5-8 in Record '01' müssen mit dem Wert '00 00 00 00' personalisiert werden.

Byte 9-13

Byte 9-13 enthalten die 10-stellige BCD-kodierte Summe aller seit dem letzten **Kassenschnitt** mit **Zahlung** zertifizierten Zahlungsbeträge. Die Summe wird in Record '01 mit 0 personalisiert. Durch **Kassenschnitt** wird die Summe auf den Wert 0 zurückgesetzt und bei einem internen Append in den neuen Record '01' geschrieben. Die Summe wird jedesmal bei einem internen Update des Record '01' um den Zahlungsbetrag inkrementiert, wenn **Zahlung** erfolgreich beendet wird. Wenn die Summe in Record '01' + Zahlungsbetrag größer als 99 99 99 99 99 würde, kann die entsprechende Zahlung nicht erfolgen.

Durch den Multiplikator in Byte 21 des EF_ID im MF der Händlerkarte wird bestimmt, welchen Wert eine Einheit der BCD-kodierten Summe bezogen auf die Währung DEM hat. Zur Zeit ist der Multiplikator 10^{-2}, so daß die Summe in Pfennigen angegeben ist.

Byte 9-13 in Record '01' müssen mit dem Wert '00 00 00 00 00' personalisiert werden.

1.9. EF_KONTO

EF_KONTO bezeichnet das lineare EF, in dessen Record '01' die Händlerbankverbindung gespeichert ist. Auf das hierdurch identifizierte Konto werden die Umsätze der Einzeltransaktionen gebucht, die durch die Händlerkarte erzeugt werden. Die Händlerbankverbindung wird im entsprechenden Summensatz zusammen mit den Einzeltransaktionen eingereicht.

1.9.1. FCP

Für das EF_KONTO sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 0A'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 0A'	Datei-Deskriptor für lineares EF
'83'	'02'	'02 05'	Datei-ID des EF_KONTO
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_KONTO

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_KONTO enthält maximal einen Record von 10 Byte Länge, so daß 10 Byte für die Nutzdaten des EF_KONTO zu allokieren sind.

Tag ’82’

Das EF_KONTO ist ein lineares EF, dessen Recordlänge auf 10 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_KONTO ist '02 05'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_KONTO die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Die Records aus EF_KONTO können also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.9.2. Daten

Das EF_KONTO enthält einen 10 Byte langen Record '01', der den folgenden Aufbau hat:

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	’nn..nn’	Bankleitzahl kontoführendes Institut für Händlerkonto
5-9	5	'nn..nn'	Kontonummer Händlerkonto
10	1	'nD'	Prüfziffer über Byte 1-9

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1-4

Byte 1-4 enthalten die 8-stellige BCD-kodierte Bankleitzahl des kontoführenden Instituts für das Händlerkonto.

Byte 5-9

Byte 5-9 enthalten die 10-stellige BCD-kodierte Kontonummer des Händlerkontos.

Byte 10

Byte 10 enthält die Prüfziffer nach dem Verfahren 'Luhn formula for computing modulus 10' über die Ziffern aus Byte 1-9 sowie einen Feldseparator 'D'.

1.10. EF_HSEQ

EF_HSEQ bezeichnet ein lineares EF, in dessen Record '01' die Sequenznummer HSEQ der Händlerkarte für Zahlungen und Fehlzahlungen gespeichert ist. Die Sequenznummer dient dazu, die Transaktionen einer Händlerkarte eindeutig zu identifizieren, um Doppeleinreichungen von Zahlungen und Fehlzahlungen erkennen zu können.

Die Sequenznummer wird mit dem Wert 1 personalisiert und jedesmal durch die Karte um 1 inkrementiert, wenn eine Transaktion durch **Zahlung** oder **Fehlzahlung** mit Erzeugung eines Transaktionszertifikats erfolgreich beendet wird.

Wenn die Sequenznummer den Wert 0 erreicht hat, kann die Kommandovariante **Zahlung einleiten** nicht mehr ausgeführt werden, so daß dann keine weiteren Transaktionen möglich sind.

1.10.1. FCP

Für das EF_HSEQ sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 04'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 04'	Datei-Deskriptor für lineares EF
'83'	'02'	'02 06'	Datei-ID des EF_HSEQ
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_HSEQ

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_HSEQ enthält maximal einen Record von 4 Byte Länge, so daß 4 Byte für die Nutzdaten des EF_HSEQ zu allokieren sind.

Tag ’82’

Das EF_HSEQ ist ein lineares EF, dessen Recordlänge auf 4 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_HSEQ ist '02 06'.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_HSEQ die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Ein Record aus EF_HSEQ kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.10.2. Daten

Das EF_HSEQ enthält in dem 4 Byte langen Record '01' die binär kodierte Sequenznummer.

Die Sequenznummer muß mit dem Wert '00 00 00 01' personalisiert werden.

1.11. EF_HLOG

EF_HLOG bezeichnet ein zyklisches EF, in dem die letzten 10 Transaktionen einer Händlerkarte protokolliert sind.

Wenn die Prüfungen von **Zahlung einleiten** positiv verlaufen sind, schreibt die Händlerkarte durch ein internes Append die Protokolldaten der Transaktion in Record '01' dieses EF. Wenn die Kommandovarianten **Zahlung prüfen, Zahlung oder Fehlzahlung** erfolgreich ausgeführt wurden, wird der Record '01' durch ein internes Update entsprechend aktualisiert.

Der Record mit der Nummer '01' enthält also immer die Protokolldaten der letzten Transaktion. Das Statusbyte des Records zeigt an, welche Kommandovariante bisher ausgeführt wurde und ob alle Schreibvorgänge durchgeführt werden konnten.

Durch **Zahlung** und **Fehlzahlung** wird außerdem ein Transaktionszertifikat mit entsprechender Kennung berechnet, in das Daten des Record '01' des EF_HLOG eingehen. Ein Transaktionszertifikat zu den Daten in einem beliebigen Record des EF_HLOG kann mit der Variante **Antwort wiederholen** von **ZERTIFIKAT** so lange erneut ausgegeben werden, wie die entsprechenden Daten der Zahlung oder Fehlzahlung nicht durch die Protokolldaten weiterer Transaktionen überschrieben sind.

1.11.1. FCP

Für das EF_HLOG sind die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'02 30'	allokierter Speicherplatz in Byte
'82'	'03'	'06 41 38'	Datei-Deskriptor für zyklisches EF
'83'	'02'	'02 07'	Datei-ID des EF_HLOG
'86'	'06'	'00 40 00 53 00 40'	ACs für das EF_HLOG

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_HLOG enthält maximal 10 Records von 56 Byte Länge, so daß 560 Byte für die Nutzdaten des EF_HLOG zu allokieren sind.

Tag ’82’

Das EF_HLOG ist ein zyklisches EF, dessen Recordlänge auf 56 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_HLOG ist ’02 07’.

Tag ’86’

Für das Kommando READ RECORD wird für das EF_HLOG die AC '00 53' (PRO_D mit Schlüsselnummer '03') festgelegt. Ein Record aus EF_HLOG kann also bei Bedarf MAC-gesichert mit dem Schlüssel K_{CD} ausgelesen werden.

Für die Kommandos APPEND RECORD und UPDATE RECORD wird die AC '00 40' (PRO_G mit Schlüsselnummer '00') festgelegt. Mit den Kommandos darf also nur auf das EF zugegriffen werden, wenn die Kommandonachricht mit einem korrekten MAC versehen ist, wobei der Schlüssel K_{Card} zu verwenden ist.

1.11.2. Daten

Das EF_HLOG enthält mindestens 10 Records, die jeweils den folgenden Aufbau haben:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-5	4	’XX..XX’	Sequenznummer SSEQ des Summensatzes der Transaktion
6-9	4	’XX..XX’	Sequenznummer HSEQ der Transaktion
10-31	22	’XX..XX’	EF_ID der Börsenkarte
32-33	2	'XX XX'	Sequenznummer BSEQ der Börsenkarte
34-35	2	'XX XX'	Sequenznummer LSEQ der Börsenkarte
36-38	3	'nn..nn'	Transaktionsbetrag
39-48	10	'nn..nD'	Daten des Börsenverrechnungskontos der Börsenkarte
49-52	4	JJJJ MM TT	Datum der Transaktion
53-55	3	HH MM SS	Uhrzeit der Transaktion
56	1	'XX'	Schlüsselnummer KID des zur Zertifikatsberechnung verwendeten KGK_{RD}

Im folgenden werden die Komponenten eines Records näher erläutert:

Byte 1

Das Statusbyte zeigt an,

- welche Kommandovariante der Kommandos **ZAHLUNG** oder **ZERTIFIKAT** in diesem Record protokolliert ist und

- in welchem Zustand die jeweilige Kommandovariante beendet wurde.

Das Statusbyte wird wie folgt kodiert:

b8	b7	b6	b5	b4	b3	b2	b1	Bedeutung
x	x	x						Kommando: b4 b3 b2 von INS
0	0	0						**ZAHLUNG**
0	0	1						**ZERTIFIKAT**
								alle anderen Werte RFU
			x	x	x			Kommandovariante: b8 b7 b6 von P1
						x		immer 0, sonst RFU
							x	bei Kommandoende beschriebene Dateien
							0	erste Datei beschrieben
							1	letzte Datei beschrieben

Wenn ein Ergänzungskommando nicht nur protokolliert wird, sondern auch weitere Applikationsdaten verändert, wird zunächst die Protokollierung in Record '01' des EF_HLOG durchgeführt, wobei das Bit b1 des Statusbyte zu 0 gesetzt wird. Anschließend werden die übrigen Schreibvorgänge des Kommandos durchgeführt. Danach wird das Bit b1 des Statusbyte zu 1 gesetzt. Werden außer der Protokollierung keine Schreibvorgänge durch das Kommando durchgeführt, wird das Bit b1 des Statusbyte bei der Protokollierung direkt zu 1 gesetzt.

Wenn das Bit b1 des Statusbyte in Record '01' des EF_HLOG bei Aufruf eines Ergänzungskommandos der Händlerkarte den Wert 0 hat, wird das Kommando mit einer Fehlermeldung abgebrochen.

Durch Bit b8..b3 des Statusbyte im jeweiligen Record '01' des EF_HLOG wird angezeigt, welche Kommandovariante zuletzt protokolliert wurde. Die Ergänzungskommandos werten diese Bit aus, um festzustellen, welche Kommandovariante ausgeführt werden darf. Wird die erforderliche Reihenfolge von Kommandovarianten nicht eingehalten, wird die entsprechende Variante abgebrochen.

Die folgende Tabelle gibt einen Überblick über die möglichen Werte des Statusbyte in Byte 1 der Records in EF_HLOG, wenn das EF_HLOG als erste Datei beschrieben wurde (ST0) und wenn es als letzte Datei beschrieben (ST1) wurde.

Kommando	ST0	ST1
Zahlung einleiten	-	'01'
Zahlung prüfen	-	'05'
Zahlung	'30'	'31'
Fehlzahlung	'34'	'35'

Das Statusbyte muß im Record '01' mit dem Wert '31' personalisiert werden.

Byte 2-5

In Byte 2-5 wird die binär kodierte Sequenznummer SSEQ des Summensatzes gespeichert, die bei der Durchführung der in diesem Record gespeicherten Transaktion aktuell war.

Byte 2-5 des Record '01' müssen mit dem Wert '00 00 00 00' personalisiert werden.

Byte 6-9

In Byte 6-9 wird die binär kodierte Sequenznummer aus EF_HSEQ gespeichert, die bei der Durchführung der in diesem Record gespeicherten Transaktion durch die Kommandos **ZAHLUNG** und **ZERTIFIKAT** verwendet wurde.

Byte 6-9 des Record '01' müssen mit dem Wert '00 00 00 00' personalisiert werden.

Byte 10-31

In Byte 10-31 wird das EF_ID der an der protokollierten Transaktion beteiligten Börsenkarte gespeichert. Das EF_ID wird der Händlerkarte mit der Kommandovariante **Zahlung einleiten** übergeben.

Byte 32-33

In Byte 32-33 wird die binär kodierte Sequenznummer BSEQ für Abbuchungs- und Rückbuchungstransaktionen der an der protokollierten Transaktion beteiligten Börsenkarte gespeichert. Diese Sequenznummer wird der Händlerkarte mit der Kommandovariante **Zahlung einleiten** übergeben.

Byte 34-35

In Byte 34-35 wird die binär kodierte Sequenznummer LSEQ für Lade- und Entladetransaktionen der an der protokollierten Transaktion beteiligten Börsenkarte gespeichert. Diese Sequenznummer wird der Händlerkarte mit der Kommandovariante **Zahlung prüfen** übergeben.

Byte 36-38

Byte 36-38 enthalten den 6-stelligen BCD-kodierten Zahlungsbetrag, der mit der Kommandovariante **Zahlung prüfen** an die Händlerkarte übergeben wird. Die Wertigkeit des Betrages wird durch Byte 21 des EF_ID im MF bestimmt (zur Zeit Pfennige).

Handelt es sich bei der gespeicherten Transaktion um eine Fehlbuchung, wird dieses Feld in das Zertifikat für **Fehlbuchung** nicht mit einbezogen.

Byte 39-48

Byte 45-54 enthalten die Daten des Börsenverrechnungskontos der an einer erfolgreichen Transaktion beteiligten Börsenkarte. Diese Daten werden der Händlerkarte mit der Kommandovariante **Zahlung prüfen** übergeben.

Handelt es sich bei der gespeicherten Transaktion um eine Fehlbuchung, wird dieses Feld in das Zertifikat für **Fehlbuchung** nicht mit einbezogen.

Byte 49-52

Byte 49-52 enthalten das BCD-kodierte Datum JJJJ MM TT, das in den Kommandodaten von **Zahlung einleiten** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Terminal keine Uhr besitzt.

Byte 52-55

Byte 53-55 enthalten die BCD-kodierte Uhrzeit HH MM SS, die in den Kommandodaten von **Zahlung einleiten** übergeben wurde. Das Datum kann mit 0 belegt sein, wenn das entsprechende Terminal keine Uhr besitzt.

Byte 56

In Byte 56 wird die binär kodierte Schlüsselnummer KID des Schlüssels KGK_{RD} gespeichert, der bei der Durchführung der gespeicherten Transaktion zur Zertifikatsberechnung über Kommando- und Antwortdaten einer Börsenkarte verwendet wurde.

Byte 10-56 des Record ’01’ werden mit ’00’ personalisiert.

2. Ergänzungskommandos der Applikation Geldbörsen-SAM

Für die Applikation Geldbörsen-SAM werden die Ergänzungskommandos **ZAHLUNG** und **ZERTIFIKAT** verwendet. Die Kommandos besitzen jeweils verschiedene Varianten.

Als Kommandoklasse CLA wird für alle Ergänzungskommandos 'E0' verwendet. Die folgende Tabelle zeigt die Kodierung von INS und P1 für die verschiedenen Kommandovarianten:

Bedeutung	INS	P1							
		b8	b7	b6	b5	b4	b3	b2	b1
immer 0 0 0 0 0, sonst RFU					x	x	x	x	x
ZAHLUNG	'40'	x	x	x					
- **Zahlung einleiten**		0	0	0					
- **Zahlung prüfen**		0	0	1					
- **Daten für Rückbuchung**		0	1	0					
- **Antwort wiederholen**		0	1	1					
- alle anderen Werte RFU									
ZERTIFIKAT	'42'	x	x	x					
- **Zahlung**		1	0	0					
- **Fehlzahlung**		1	0	1					
- **Antwort wiederholen**		0	1	1					
- **Kassenschnitt**		0	0	0					
- **Summensatz**		0	0	1					
- alle anderen Werte RFU									

Für die Kommandovarianten **Summensatz** und **Antwort wiederholen** von **ZERTIFIKAT** enthält der Parameter P2 die Recordnummer des Records aus EF_SUMME bzw. EF_HLOG, aus dessen Daten der Summensatz bzw. die Antwortdaten der gespeicherten Kommandovariante **(Fehl-)Zahlung** generiert werden sollen. Hierbei dürfen die Recordnummern '00' und 'FF' nicht verwendet werden.

Für die übrigen Kommandovarianten wird P2 immer zu '00' kodiert.

Für die von verschiedenen Komponenten verwendeten Sequenznummern werden die folgenden Abkürzungen verwendet:

LSEQ: Sequenznummer für **LADEN** und **ENTLADEN** (2 Byte binär) der elektronischen Geldbörse,

BSEQ: Sequenznummer für **ABBUCHEN** und **RÜCKBUCHEN** (2 Byte binär) der elektronischen Geldbörse,

HSEQ: Sequenznummer für Transaktionen (4 Byte binär) der Händlerkarte,

SSEQ: Summensequenznummer (4 Byte binär) der Händlerkarte,

TZ: Transaktionszähler (4 Byte binär) der Händlerkarte.

2.1. Returncodes

Die folgenden Tabellen enthalten die Returncodes, die von den Ergänzungskommandos mit ihren Varianten bei erfolgreicher Kommandoausführung oder bei Kommandoabbruch verwendet werden.

Die folgenden Returncodes zeigen an, daß das Kommando erfolgreich beendet wurde:

Kommando erfolgreich beendet

SW1	SW2	Bedeutung
'90'	'00'	OK
'61'	L_a	Wrong length in L_e, L_a indicates the length of data sent ($L_a > L_e$ possible)

Der folgende Returncode zeigt an, daß das Kommando im Prinzip erfolgreich beendet wurde, daß jedoch die angegebene Warnung zu beachten ist:

Warnung (Kommando beendet)

SW1	SW2	Bedeutung
'63'	'CX'	Use of internal retry routine (Counter 'X', valued from 0 to 15)

Der Returncode '63 CX' "Use of internal retry routine" gibt an, daß die Karte einen Schreibvorgang mehrfach durchführen mußte, um das Kommando erfolgreich abzuschließen. Die Warnung weist auf interne Speicherprobleme der Karte hin und dient zu Diagnosezwecken.

Die folgenden Returncodes zeigen an, daß das Kommando aufgrund des angegebenen Fehlers abgebrochen wurde:

Fehlercodes (Kommando abgebrochen)

SW1	SW2	Bedeutung
‘64’	‘00’	No precise diagnosis wird ausgegeben, wenn bei der Ausführung des Kommandos ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
‘65’	‘81’	Memory failure wird ausgegeben, wenn die Ausführung fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'03'	keine AC gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC
'66'	'05'	Secure Messaging in CLA falsch
'66'	'11'	durch AC (und KID in Kommandodaten) referenzierter Schlüssel nicht gefunden
'66'	'12'	Paritätsfehler des durch AC (und KID in Kommandodaten) referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu dem durch AC (und KID in Kommandodaten) referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ des durch AC (und KID in Kommandodaten) referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID unzulässig oder fehlerhaft
'66'	'16'	KID in Kommandodaten stimmt nicht mit der durch eine AC referenzierten Schlüsselnummer überein oder ist nicht in der referenzierten Schlüsselgruppe enthalten
'66'	'88'	Zertifikat falsch

SW1	SW2	Bedeutung
‘67’	‘00’	Wrong length (L_c incorrect or L_e not present)
'69'	'85'	DF_BSAM bei Kommandoaufruf nicht selektiert
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn das Format der Kommandodaten nicht der Spezifikation entspricht oder Kommandodaten außerhalb des spezifizierten Wertebereichs liegen
'6A'	'82'	File not found wird ausgegeben, wenn ein EF, auf das das Kommando zugreifen will, nicht vorhanden ist oder Struktur oder Recordlänge des EF nicht der Spezifikation entsprechen
'6A'	'83'	Record not found wird ausgegeben, wenn auf einen Record mit Recordnummer größer als die des LAST Record zugegriffen werden soll
'6A'	'84'	Speicherplatz reicht nicht aus wird ausgegeben, wenn der für EF_HLOG allokierte Speicherplatz für die Protokollierung mittels eines internen Appends nicht ausreicht
‘6A’	‘86’	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported
'96'	'01'	Schreibvorgänge eines Ergänzungskommandos unvollständig
'96'	'02'	Applikationsdaten falsch kodiert
'96'	'CX'	Überlauf der Sequenznummer (SSEQ: X = 3, HSEQ: X = 4, TZ: X = 5)
'97'	'02'	Transaktionsbetrag zu groß
'9F'	'XX'	Reihenfolge der Kommandos nicht eingehalten, letzte Kommandovariante war XX

2.2. Grundsätze des Kommandoablaufs

Für alle Ergänzungskommandos der Applikation Geldbörsen-SAM gilt

- Wenn das DF_BSAM (eindeutig identifiziert durch den DF-Namen) bei Kommandoaufruf nicht selektiert ist: Abbruch mit Returncode '69 85'.
- Die EFs, auf die durch die Kommandos explizit zugegriffen wird, befinden sich im aktuellen DF_BSAM. Alle Kommandovarianten können nur ausgeführt werden, wenn das DF_BSAM bei ihrem Aufruf aktuell ist. Das Auffinden eines EF, auf das zugegriffen werden soll, muß daher unabhängig von der Position des DF_BSAM in der Händlerkarte und unabhängig von für das DF_BSAM vergebenen SFI möglich sein.

  Wenn ein EF, aus dem im Verlauf der Kommandoausführung Daten gelesen werden sollen oder in das Daten geschrieben werden sollen, nicht gefunden werden kann oder Struktur (linear oder zyklisch) oder Recordlänge des EF nicht der Spezifikation entsprechen: Abbruch mit Returncode '6A 82'.

  Durch die internen Lese- und Schreibroutinen können die Fehlercodes '64 00' und '65 81' verursacht werden.

  Bei den internen Schreibroutinen wird unterschieden zwischen internem Update und internem Append.

  - Das interne Update überschreibt Daten in den angegebenen Record und bricht mit dem Returncode '6A 83' ab, wenn der entsprechende Record nicht angelegt ist.
  - Das interne Append muß wie APPEND RECORD die Struktur (linear oder zyklisch) des EF respektieren, dem ein Record hinzugefügt werden soll. Es schreibt immer einen ganzen Record. Bei fehlendem Speicherplatz führt es zu einem Abbruch mit '6A 84'.

  Beide Arten von internen Schreibroutinen können zur Warnung '63 CX' führen.
- Falls das CLA-Byte den Wert 'E4' hat: Abbruch mit Returncode '66 05'.
- Falls die Längenangabe L_c nicht der Spezifikation entspricht oder nicht der tatsächlichen Länge der Kommandodaten entspricht oder falls L_e fehlt, obwohl es gemäß Spezifikation vorhanden sein muß, oder wenn L_e vorhanden ist, obwohl es gemäß Spezifikation fehlen muß: Abbruch mit Returncode '67 00'.
- Wenn ein EF, aus dem im Verlauf der Kommandoausführung Daten gelesen werden sollen, den benötigten Record nicht enthält: Abbruch mit Returncode '6A 83'.
- Falls ein Speicherfehler von der ec-Karte erkannt wird: Abbruch mit Returncode '65 81'.
- Wenn für ein Ergänzungskommando und das EF_SUMME keine AC gesetzt ist: Abbruch mit Returncode '66 03'
- Wenn für ein Ergänzungskommando und das EF_SUMME eine AC gesetzt ist, die nicht der Kommandospezifikation entspricht: Abbruch mit Returncode '66 04'
- Falls ein Schlüssel nicht gefunden wird: Abbruch mit Returncode '66 11'.
- Falls ein Paritätsfehler eines Schlüssels von der ec-Karte erkannt wird: Abbruch mit

Returncode ’66 12’.

- Falls die Zusatzinformationen zu einem Schlüssel nicht gefunden werden: Abbruch mit Returncode '66 13'.
- Falls der Fehlbedienungszähler eines Schlüssels abgelaufen ist: Abbruch mit Returncode '66 14'.
- Falls Schlüssellänge und Algorithmus-ID zu einem Schlüssel fehlerhaft oder unzulässig sind: Abbruch mit Returncode '66 15'.
- Jeder bei der Ausführung eines Ergänzungskommandos festgestellte MAC-Fehler führt zur Dekrementierung des Fehlbedienungszählers des betroffenen Schlüssels.
- Die Ausführung jedes Ergänzungskommandos hat zur Folge, daß eine vorher erzeugte Zufallszahl oder ein vorher übergebener ICV ungültig werden.
- Durch das Bit b1 des Statusbyte im Record '01' des EF_HLOG im Verzeichnis DF_BSAM wird angezeigt, ob alle Schreibvorgänge der protokollierten Kommandovariante durchgeführt wurden. Ist dies der Fall, hat dieses Bit den Wert 1.
  Wenn das Bit b1 des Statusbyte in Record '01' des EF_HLOG den Wert 0 hat, gilt für **jedes** Ergänzungskommando der Applikation Geldbörsen-SAM: Abbruch mit Returncode '96 01'.

- Durch Bit b8..b3 des Statusbyte im Record '01' des EF_HLOG wird angezeigt, welche Kommandovariante zuletzt protokolliert wurde. Die Ergänzungskommandos werten diese Bit aus, um festzustellen, welche Kommandovariante ausgeführt werden darf. Wird die erforderliche Reihenfolge von Kommandovarianten nicht eingehalten: Abbruch mit Returncode '9F XX', wobei in 'XX' das Statusbyte aus Record '01' des EF_HLOG ist.
- Immer wenn ein in einem EF der Händlerkarte gespeicherter Betrag durch ein Ergänzungskommando in Rechen- oder Vergleichsoperationen verwendet wird, wird geprüft, ob der jeweilige Betrag BCD-kodiert ist. Ist das nicht der Fall, wird das Kommando mit dem Returncode '96 02' abgebrochen.
- Alle Kommandovarianten geben bei Abbruch aufgrund eines festgestellten Fehlers (Returncode $\neq$ '90 00', '61 L_a') nur den Fehlercode und keine Antwortdaten aus.
- In der folgenden Beschreibung der Abläufe wird als Returncode bei erfolgreicher Beendung des Kommandos '90 00' ausgegeben.

  Wenn das jeweilige Kommando mit einem nicht spezifizierten Wert in L_e aufgerufen wird, kann das Kommando ebenfalls mit Returncode '61 L_a' beendet werden.

2.3. ZAHLUNG

2.3.1. Übersicht

Alle Kommandovarianten, die der Kommunikation mit einer Börsenkarte dienen, sind zum Kommando **ZAHLUNG** zusammengefaßt. Je nach Belegung des Parameters P1 in der Kommandonachricht können mit dem Kommando **ZAHLUNG**

- eine Zahlung eingeleitet werden mit
  - Prüfung der Sequenznummern SSEQ und HSEQ und des Transaktionszählers TZ auf Überlauf,
  - Auswertung der ACs für **ZAHLUNG**,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Prüfung, ob zuvor ein GET CHALLENGE stattgefunden hat,
  - Prüfung des Zertifikats in den Kommandodaten,
  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat,

- Zahlungsdaten einer Börsenkarte geprüft werden mit
  - Auswertung der ACs für **ZAHLUNG**,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Prüfung des Zertifikats in den Kommandodaten,
  - Prüfung der Händlerkartennummer und der Sequenznummern HSEQ und BSEQ in den Kommandodaten,
  - Prüfung, ob die Erhöhung der Summe in EF_SUMME um den gewünschten Zahlungsbetrag den Maximalwert nicht übersteigt,
  - Protokollierung,

- Daten für eine Rückbuchung erzeugt werden mit
  - Auswertung der ACs für **ZAHLUNG**,
  - Prüfung, ob in Record '01' ein erfolgreiches **Fehlbuchen** gespeichert ist,
  - Ausgabe von Antwortdaten,

- die Antwortdaten des letzten erfolgreichen **Zahlung einleiten** erneut ausgegeben werden mit
  - Auswertung der ACs für **ZAHLUNG**,
  - Prüfung, ob in Record '01' ein erfolgreiches **Zahlung einleiten** gespeichert ist,
  - Ausgabe der Antwortdaten des gespeicherten **Zahlung einleiten**.

Durch die AC, die für EF_SUMME in DF_BSAM und das Kommando **ZAHLUNG** festgelegt ist, wird der Zugriff aller Varianten des Kommandos **ZAHLUNG** auf Daten der Händlerkarte geregelt.

Die Händlerkarte soll nur dann eine Transaktion beginnen, wenn sie sich zuvor von der Authentizität der beteiligten Börsenkarte und deren Sequenznummer überzeugt hat. Hierzu erzeugt die Börsenkarte ein Zertifikat über Antwortdaten des **Abbuchen einleiten**. Diese Daten werden zusammen mit dem Zertifikat in den Kommandodaten der Kommandovariante **Zahlung einleiten** an die Händlerkarte übergeben. Antwortdaten des **Zahlung einleiten** werden mit einem Zertifikat versehen, damit die Börsenkarte die Authentizität der Händlerkarte prüfen kann.

Die Variante **Zahlung prüfen** der Händlerkarte wird nur dann mit einem positiven Antwortcode beendet, wenn der Nachweis erbracht ist, daß zuvor eine Abbuchung in der Börsenkarte erfolgt ist. Hierzu erzeugt die Börsenkarte ein Zertifikat über Antwortdaten des **Abbuchen**. Diese Daten werden zusammen mit dem Zertifikat in den Kommandodaten der Kommandovariante **Zahlung prüfen** an die Händlerkarte übergeben.

Die Börsenkarte darf nur dann eine Rückbuchung vornehmen, wenn sicher ist, daß der entsprechende Betrag dem Händler nicht gutgeschrieben wurde. Dies weist die Händlerkarte durch die Ausgabe entsprechender Daten mit Zertifikat bei der Kommandovariante **Daten für Rückbuchung** nach.

Durch eine AC vom Typ ZERT für EF_SUMME und das Kommando **ZAHLUNG** muß festgelegt werden, welcher Schlüssel zur Zertifikatsbildung verwendet werden soll. Durch die AC darf eine Schlüsselgruppe (Schlüsselnummer > '03') referenziert werden. Es darf durch die AC auch ein Schlüssel mit Algorithmus-ID '01' oder '02' (Masterkey) referenziert werden. Ein durch die AC referenzierter Masterkey wird im folgenden als KGK_{RD} bezeichnet. Ein daraus abzuleitender börsenkartenindividueller Schlüssel bzw. ein referenzierter Einzelschlüssel wird K_{RD} genannt.

Die zu verwendende Schlüsselnummer KID und das EF_ID einer Börsenkarte zum Ableiten eines börsenkartenindividuellen Schlüssels werden in der Kommandonachricht von **Zahlung einleiten** an die Händlerkarte übergeben. Die übrigen Varianten des Kommandos **ZAHLUNG** entnehmen KID und bei Bedarf EF_ID dem Record '01' des EF_HLOG.

Wenn der Schlüssel ein Masterkey KGK_{RD} mit Algorithmus-ID '01' ist, handelt es sich bei dem abgeleiteten K_{RD} um einen 16 Byte langen Schlüssel. Dem K_{RD} ist in diesem Fall der Triple-DES (CBC) als kryptographischer Algorithmus zugeordnet.

Wenn der Schlüssel ein Masterkey KGK_{RD} mit Algorithmus-ID '02' ist, handelt es sich bei dem abgeleiteten K_{RD} um einen 8 Byte langen Schlüssel, dem der DES (CBC) als kryptographischer Algorithmus zugeordnet ist.

Für einen abgeleiteten K_{RD} werden logische Schlüsselnummer, Fehlbedienungszähler und Schlüssel-Version des zugehörigen KGK_{RD} verwendet. Insbesondere wird der Fehlbedienungszähler des KGK_{RD} dekrementiert, wenn durch eine Variante von **ZAHLUNG** ein fehlerhafter K_{RD}-MAC festgestellt wird.

Eine andere AC als vom Typ ZERT darf für das Kommando **ZAHLUNG** und das EF_SUMME in DF_BSAM nicht gesetzt werden.

2.3.2. Kommando- und Antwortnachrichten

2.3.2.1. Zahlung einleiten

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'00'	P1 für **Zahlung einleiten**
4	1	‘00’	P2, fester Wert
5	1	'2A'	L_c
6	1	'41'	Nachrichten-ID für **Abbuchen einleiten** der Börsenkarte
7-8	2	‘XX XX’	Sequenznummer BSEQ der Börsenkarte
9-16	8	'XX..XX'	RND: Zufallszahl (GET CHALLENGE)
17-24	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 6-16|'00 00 00 00 00'
25-46	22	'XX..XX'	EF_ID der Börsenkarte
47	1	'XX'	logische Schlüsselnummer KID des KGK_{RD} oder K_{RD}
48	1	‘1D’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’50’	Nachrichten-ID für **Abbuchen** der Börsenkarte
2-3	2	'XX XX'	Sequenznummer BSEQ aus der Kommandonachricht
4-13	10	'nn..nD'	Händlerkartennummer aus Byte 1-10 des EF_ID ('D' Hexziffer)
14-17	4	'XX..XX'	Sequenznummer HSEQ aus EF_HSEQ
18-21	4	'XX..XX'	Sequenznummer SSEQ aus Record '01' des EF_SUMME
22-29	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 1-21|'00 00 00'
30-31	2	'XX XX'	Statusbyte SW1-SW2

2.3.2.2. Zahlung prüfen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'20'	P1 für **Zahlung prüfen**
4	1	‘00’	P2, fester Wert
5	1	'28'	L_c
6	1	'51'	Nachrichten-ID für **Abbuchen** der Börsenkarte
7-8	2	'XX XX'	Sequenznummer BSEQ der Börsenkarte
9-10	2	'XX XX'	Sequenznummer LSEQ der Börsenkarte
11-13	3	'nn..nn'	durch die Börsenkarte geprüfter Zahlungsbetrag
14-23	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer)
24-27	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
28-37	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos der Börsenkarte ('D' Hexziffer)
38-45	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über Byte 6-37

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1-2	2	‘XX XX’	Statusbyte SW1-SW2

2.3.2.3. Daten für Rückbuchung

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'40'	P1 für **Daten für Rückbuchung**
4	1	‘00’	P2, fester Wert
5	1	‘17’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	”70’	Nachrichten-ID für **Rückbuchen** der Börsenkarte
2-11	10	'nn..nD'	Händlerkartennummer aus Byte 1-10 des EF_ID ('D' Hexziffer)
12-15	4	'XX..XX'	Sequenznummer HSEQ aus Byte 6-9 des Record '01' im EF_HLOG
16-23	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 1-15|'00'
24-25	2	‘XX XX’	Statusbyte SW1-SW2

2.3.2.4. Antwort wiederholen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'60'	P1 für **Antwort wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1D’	L_e

Response APDU

Antwortdaten identisch mit denen von **Zahlung einleiten**

2.3.3. Ablauf in der Händlerkarte

2.3.3.1. Zahlung einleiten

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls die Sequenznummer SSEQ in Byte 1-4 des Record '01' des EF_SUMME den Wert '00..00' hat: Abbruch mit Returncode '96 C3'.

   - Falls die Sequenznummer HSEQ aus Record '01' des EF_HSEQ den Wert '00..00' hat:

Abbruch mit Returncode ’96 C4’.

- Falls der Transaktionszähler TZ in Byte 5-8 des Record '01' in EF_SUMME den Wert 'FF FF FF FF' hat: Abbruch mit Returncode '96 C5'.

- Falls das Statusbyte in Record '01' des EF_HLOG keinen der Werte '31' (**Zahlung** beendet) oder '35' (**Fehlzahlung** beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

- Falls nicht unmittelbar vorher GET CHALLENGE ausgeführt wurde: Abbruch mit Returncode '66 01'.

- Wenn die Nachrichten-ID in der Kommandonachricht nicht den Wert '41' hat oder wenn die Zufallszahl in der Kommandonachricht verschieden von der mit GET CHALLENGE erzeugten Zufallszahl ist: Abbruch mit Returncode '6A 80'.

- Falls die in den Kommandodaten enthaltene Schlüsselnummer KID nicht mit der durch die AC vom Typ ZERT referenzierten Schlüsselnummer übereinstimmt oder nicht in der durch die AC referenzierten Schlüsselgruppe enthalten ist: Abbruch mit Returncode '66 16'.

- Falls das Zertifikat in den Kommandodaten falsch ist: Abbruch mit Returncode '66 88'.

  Wenn durch die AC vom Typ ZERT (und KID in den Kommandodaten) ein Masterkey KGK_{RD} referenziert wird (Algorithmus-ID '01' oder '02'), muß zur Prüfung des Zertifikats zunächst der K_{RD} aus dem KGK_{RD} und den Daten des EF_ID aus der Kommandonachricht mit dem in Kapitel 2.5 von [LIT 1] beschriebenen Verfahren abgeleitet werden.

2. **Zahlung einleiten** wird in Record '01' des EF_HLOG durch ein internes Append protokolliert:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’01’	Statusbyte
2-5	4	’XX..XX’	Sequenznummer SSEQ aus Byte 1-4 des Record ’01’ des EF_SUMME
6-9	4	’XX..XX’	Sequenznummer HSEQ aus EF_HSEQ
10-31	22	’XX..XX’	EF_ID aus der Kommandonachricht
32-33	2	’XX XX’	Sequenznummer BSEQ aus der Kommandonachricht
34-35	2	’00 00’	Filler
36-38	3	’00..00’	Filler
39-48	10	’00..00’	Filler
49-52	4	’00..00’	Filler
53-55	3	’00..00’	Filler
56	1	’XX’	Schlüsselnummer KID aus der Kommandonachricht

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der

(Retail-)CBC-MAC mit K_{RD} über die folgenden 21 Byte DATA mit 3 Byte Filler '00' gebildet:

- '50',
- Byte 32-33 aus Record '01' des EF_HLOG,
- Byte 1-10 des Record '01' des EF_ID,
- Byte 6-9 aus Record '01' des EF_HLOG,
- Byte 2-5 aus Record '01' des EF_HLOG.

4. DATA|ZERT werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.3.3.2. Zahlung prüfen

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

- Falls das Statusbyte in Record '01' des EF_HLOG nicht den Wert '01' (**Zahlung einleiten** erfolgreich beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

- Wenn die Nachrichten-ID in der Kommandonachricht nicht den Wert '51' hat oder wenn der Zahlungsbetrag in der Kommandonachricht nicht BCD-kodiert ist: Abbruch mit Returncode '6A 80'.

- Falls die Sequenznummern BSEQ und HSEQ in der Kommandonachricht nicht mit den entsprechenden in Record '01' des EF_HLOG gespeicherten Werten übereinstimmen: Abbruch mit Returncode '6A 80'.

- Falls die Händlerkartennummer in der Kommandonachricht nicht mit der Händlerkartennummer in Byte 1-10 des EF_ID übereinstimmt: Abbruch mit Returncode '6A 80'.

- Falls das Zertifikat in den Kommandodaten falsch ist: Abbruch mit Returncode '66 88'.

  Die Schlüsselnummer KID zum Auffinden des benötigten Schlüssels K_{RD} bzw. des KGK_{RD} zum Ableiten des K_{RD} wird Record '01' des EF_HLOG entnommen. Das EF_ID zum Ableiten eines K_{RD} wird ebenfalls Record '01' des EF_HLOG entnommen.

  Bemerkung:

  Der K_{RD} wurde bereits durch die Kommandovariante **Zahlung einleiten** abgeleitet. Er muß daher an dieser Stelle nicht notwendig neu erzeugt werden, sondern kann durch die Händlerkarte zwischengespeichert werden.

- Falls Summe der Zahlungsbeträge in Byte 9-13 des Record '01' des EF_SUMME + Zahlungsbetrag aus der Kommandonachricht größer als '99 99 99 99 99': Abbruch mit

Returncode ’97 02’.

2. **Zahlung prüfen** wird durch ein internes Update in Record ’01’ des EF_HLOG protokolliert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’05’	Statusbyte
2-5	4	’XX..XX’	unverändert
6-9	4	'XX..XX'	unverändert
10-31	22	'XX..XX'	unverändert
32-33	2	'XX XX'	unverändert
34-35	2	'XX XX'	Sequenznummer LSEQ aus der Kommandonachricht
36-38	3	'nn..nn'	Zahlungsbetrag aus der Kommandonachricht
39-48	10	'nn..nD'	Daten des Börsenverrechnungskontos aus der Kommandonachricht
49-52	4	'00..00'	unverändert
53-55	3	'00..00'	unverändert
56	1	'XX'	unverändert

3. Der Returncode ’90 00’ wird ausgegeben.

2.3.3.3. Daten für Rückbuchung

1. Falls das Statusbyte in Record ’01’ des EF_HLOG nicht den Wert ’35’ (**Fehlzahlung**) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Die Schlüsselnummer KID zum Auffinden des benötigten Schlüssels K_{RD} bzw. des KGK_{RD} zum Ableiten des K_{RD} wird Record '01' des EF_HLOG entnommen. Das EF_ID zum Ableiten eines K_{RD} wird ebenfalls Record '01' des EF_HLOG entnommen.

   Bemerkung:

   Der K_{RD} wurde bereits durch die Kommandovariante **Zahlung einleiten** abgeleitet. Er muß daher an dieser Stelle nicht notwendig neu erzeugt werden, sondern kann durch die Händlerkarte zwischengespeichert werden.

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{RD} über die folgenden 15 Byte DATA mit 1 Byte Filler '00' gebildet:

   - '70',
   - Byte 1-10 des Record '01' des EF_ID,
   - Byte 6-9 aus Record '01' des EF_HLOG,

4. DATA|ZERT werden als Antwortdaten mit Returncode ’90 00’ ausgegeben.

2.3.3.4. Antwort wiederholen

1. Falls das Statusbyte in Record ’01’ des EF_HLOG nicht den Wert ’01’ (**Zahlung einleiten**) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Die Schlüsselnummer KID zum Auffinden des benötigten Schlüssels K_{RD} bzw. des KGK_{RD} zum Ableiten des K_{RD} wird Record '01' des EF_HLOG entnommen. Das EF_ID zum Ableiten eines K_{RD} wird ebenfalls Record '01' des EF_HLOG entnommen.

   Bemerkung:

   Der K_{RD} wurde bereits durch die Kommandovariante **Zahlung einleiten** abgeleitet. Er muß daher an dieser Stelle nicht notwendig neu erzeugt werden, sondern kann durch die Händlerkarte zwischengespeichert werden.

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{RD} über die folgenden 21 Byte DATA mit 3 Byte Filler '00' gebildet:

   - '50',
   - Byte 32-33 aus Record '01' des EF_HLOG,
   - Byte 1-10 des Record '01' des EF_ID,
   - Byte 6-9 aus Record '01' des EF_HLOG,
   - Byte 2-5 aus Record '01' des EF_HLOG.

4. DATA|ZERT werden als Antwortdaten mit Returncode '90 00' ausgegeben.

2.4. ZERTIFIKAT

2.4.1. Übersicht

Alle Kommandovarianten, die der Berechnung von Zertifikaten zur Einreichung bei der zuständigen Evidenz-Zentrale (EZ) dienen, sind zum Kommando **ZERTIFIKAT** zusammengefaßt. Je nach Belegung des Parameters P1 in der Kommandonachricht können mit dem Kommando **ZERTIFIKAT**

- eine Zahlung mit Erzeugen eines Zertifikats erfolgreich beendet werden mit
  - Auswertung der ACs für **ZERTIFIKAT**,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,

  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat,

- eine Transaktion mit Erzeugen eines Zertifikats für eine Fehlzahlung beendet werden mit
  - Auswertung der ACs für **ZERTIFIKAT**,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Protokollierung,
  - Ausgabe von Antwortdaten mit Zertifikat,

- die Antwortdaten mit Zertifikat einer Zahlung oder Fehlzahlung erneut ausgegeben werden mit
  - Auswertung der ACs für **ZERTIFIKAT**,
  - Prüfung, ob im Record des EF_HLOG, dessen Recordnummer in P2 angegeben wird, eine der Varianten **Zahlung** oder **Fehlzahlung** gespeichert ist,
  - Ausgabe der Antwortdaten mit Zertifikat der gespeicherten Kommandovariante.

- ein Kassenschnitt durchgeführt werden mit
  - Prüfung der Sequenznummer SSEQ auf Überlauf,
  - Auswertung der ACs für **ZERTIFIKAT**,
  - Prüfung, ob der Status der Applikation die Kommandoausführung erlaubt,
  - Ausgabe eines Summensatzes mit Zertifikat zu den Summendaten in Record '01' des EF_SUMME,
  - Anlegen eines neuen Record '01' im EF_SUMME mit initialisierten Summendaten,

- ein Summensatz mit Zertifikat ausgegeben werden mit
  - Auswertung der ACs für **ZERTIFIKAT**,
  - Ausgabe eines Summensatzes mit Zertifikat zu den Summendaten im Record des EF_SUMME, dessen Recordnummer in P2 angegeben wird.

Durch die AC, die für EF_SUMME in DF_BSAM und das Kommando **ZERTIFIKAT** festgelegt ist,

wird festgelegt, mit welchem Schlüssel die Varianten des Kommandos das jeweilige Zertifikat berechnen sollen.

Hierzu muß eine AC vom Typ ZERT für EF_SUMME in DF_BSAM und das Kommando **ZERTIFIKAT** gesetzt werden. Durch die AC darf keine Schlüsselgruppe (Schlüsselnummer > '03') referenziert werden. Der zur Zertifikatsberechnung zu verwendende händlerkartenindividuelle Schlüssel wird im folgenden als K_{ZD} bezeichnet. Die Schlüssel-Version (Generationsnummer) wird KV genannt.

Eine andere AC als vom Typ ZERT mit Einzelschlüssel darf für EF_SUMME und **ZERTIFIKAT** nicht gesetzt werden.

2.4.2. Kommando- und Antwortnachrichten

2.4.2.1. Zahlung

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'80'	P1 für **Zahlung**
4	1	‘00’	P2, fester Wert
5	1	'07'	L_c
6-9	4	JJJJ MM TT	Datum des Händlerterminals
10-12	3	HH MM SS	Uhrzeit des Händlerterminals
13	1	‘37’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E9’	EBCDIC-Kodierung des Buchstaben Z, Kennung für Zahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	‘nn..nD’	Börsenkartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der Börsenkarte
32-33	2	'XX XX'	Sequenznummer LSEQ der Börsenkarte
34-36	3	'nn..nn'	Zahlungsbetrag
37-46	10	‘nn..nD’	Daten des Börsenverrechnungskontos ('D': Hexziffer)
47-54	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 48 Byte Byte 1-46|'00 00'
55	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
56-57	2	'XX XX'	Statusbyte SW1-SW2

2.4.2.2. Fehlzahlung

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'A0'	P1 für **Fehlzahlung**
4	1	‘00’	P2, fester Wert
5	1	'07'	L_c
6-9	4	JJJJ MM TT	Datum des Händlerterminals
10-12	3	HH MM SS	Uhrzeit des Händlerterminals
13	1	‘28’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘C6’	EBCDIC-Kodierung des Buchstaben F, Kennung für Fehlzahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	‘nn..nD’	Börsenkartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der Börsenkarte
32-39	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 32 Byte Byte 1-31|'00'
40	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
41-42	2	'XX XX'	Statusbyte SW1-SW2

2.4.2.3. Antwort wiederholen

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'60'	P1 für **Antwort wiederholen**
4	1	‘XX’	P2, Recordnummer (zwischen '01' und 'FE')
5	1	‘XX’	L_e, abhängig von der Länge der Antwortdaten der gespeicherten Kommandovariante

Response APDU

Aufbau der Antwortdaten identisch mit dem der Antwortdaten von **Zahlung** oder **Fehlzahlung**

2.4.2.4. Kassenschnitt

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'00'	P1 für **Kassenschnitt**
4	1	‘00’	P2, fester Wert
5	1	‘20’	L_e

Response APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1-10	10	’nn..nD’	Händlerbankverbindung ('D' Hexziffer)
11-14	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
15-18	4	'XX..XX'	Anzahl aller zugehörigen Zahlungen und Fehlzahlungen
19-23	5	'nn..nn'	Summe der Zahlungsbeträge
24-31	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 24 Byte Byte 1-23|'00'
32	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
33-34	2	'XX XX'	Statusbyte SW1-SW2

2.4.2.5. Summensatz

Command APDU

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'20'	P1 für **Summensatz**
4	1	‘XX’	P2, Recordnummer (zwischen '01' und 'FE')
5	1	‘20’	L_e

Response APDU

Aufbau der Antwortdaten identisch mit dem der Antwortdaten von **Kassenschnitt**

2.4.3. Ablauf in der Händlerkarte

2.4.3.1. Zahlung

1. Falls das Statusbyte in Record ’01’ des EF_HLOG nicht den Wert ’05’ (**Zahlung prüfen** beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. **Zahlung** wird in Record '01' des EF_HLOG durch ein internes Update protokolliert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’30’	Statusbyte
2-5	4	’XX..XX’	unverändert
6-9	4	'XX..XX'	unverändert
10-31	22	'XX..XX'	unverändert
32-33	2	'XX XX'	unverändert
34-35	2	'XX XX'	unverändert
36-38	3	'nn..nn'	unverändert
39-48	10	'nn..nD'	unverändert
49-52	4	JJJJ MM TT	Datum aus der Kommandonachricht
53-55	3	HH MM SS	Uhrzeit aus der Kommandonachricht
56	1	'XX'	unverändert

3. Die übrigen Daten werden geschrieben:

   - Die Summendaten in Record '01' des EF_SUMME werden mit einem internen Update aktualisiert:

     - Byte 1-4 bleiben unverändert,

     - Byte 5-8 werden um 1 inkrementiert,

     - Byte 9-13 werden um den Transaktionsbetrag in Byte 36-38 in Record '01' des EF_HLOG inkrementiert.

   - Die Sequenznummer HSEQ im Record '01' des EF_HSEQ wird mit dem um 1 inkrementierten Wert überschrieben.

4. Das niedrigstwertige Bit des Statusbyte in Record '01' des EF_HLOG wird gesetzt.

5. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 46 Byte DATA mit 2 Byte Filler '00' gebildet:

   - 'E9',

- Byte 1-10 des Record ’01’ des EF_ID,
- Byte 2-19 aus Record ’01’ des EF_HLOG,
- Byte 32-48 aus Record ’01’ des EF_HLOG.

6. DATA|ZERT|KV werden als Antwortdaten mit Returncode ’90 00’ ausgegeben. Hierbei ist KV die Schlüsselversion des K_{ZD} aus dem zugehörigen EF_KEYD.

2.4.3.2. Fehlzahlung

1. Falls das Statusbyte in Record '01' des EF_HLOG nicht einen der Werte '01' (**Zahlung einleiten** beendet) oder '05' (**Zahlung prüfen** beendet) hat: Abbruch mit Returncode ’9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. **Fehlzahlung** wird in Record '01' des EF_HLOG durch ein internes Update protokolliert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’34’	Statusbyte
2-5	4	’XX..XX’	unverändert
6-9	4	'XX..XX'	unverändert
10-31	22	'XX..XX'	unverändert
32-33	2	'XX XX'	unverändert
34-35	2	'XX XX'	unverändert
36-38	3	'XX..XX'	unverändert
39-48	10	'XX..XX'	unverändert
49-52	4	JJJJ MM TT	Datum aus der Kommandonachricht
53-55	3	HH MM SS	Uhrzeit aus der Kommandonachricht
56	1	'XX'	unverändert

3. Die übrigen Daten werden geschrieben:

   - Die Summendaten in Record '01' des EF_SUMME werden mit einem internen Update aktualisiert:
     - Byte 1-4 bleiben unverändert,
     - Byte 5-8 werden um 1 inkrementiert,
     - Byte 9-13 bleiben unverändert.
   - Die Sequenznummer HSEQ im Record '01' des EF_HSEQ wird mit dem um 1 inkrementierten Wert überschrieben.

4. Das niedrigstwertige Bit des Statusbyte in Record ’01’ des EF_HLOG wird gesetzt.

5. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 31 Byte DATA mit 1 Byte Filler '00' gebildet:

   - 'C6',
   - Byte 1-10 des Record '01' des EF_ID,
   - Byte 2-19 aus Record '01' des EF_HLOG,
   - Byte 32-33 aus Record '01' des EF_HLOG.

6. DATA|ZERT|KV werden als Antwortdaten mit Returncode '90 00' ausgegeben. Hierbei ist KV die Schlüsselversion des K_{ZD} aus dem zugehörigen EF_KEYD.

2.4.3.3. Antwort wiederholen

1. Falls das Statusbyte in Record 'XX' des EF_HLOG keinen der Werte '31' (**Zahlung**) oder '35' (**Fehlzahlung**) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Das Zertifikat ZERT für die Antwortnachricht wird berechnet.

   - Wenn in Record 'XX' **Zahlung** gespeichert ist, wird hierzu der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 46 Byte DATA mit 2 Byte Filler '00' gebildet:
     - 'E9',
     - Byte 1-10 des Record '01' des EF_ID,
     - Byte 2-19 aus Record 'XX' des EF_HLOG,
     - Byte 32-48 aus Record 'XX' des EF_HLOG.
   - Wenn in Record 'XX' **Fehlzahlung** gespeichert ist, wird hierzu der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 31 Byte DATA mit 1 Byte Filler '00' gebildet:
     - 'C6',
     - Byte 1-10 des Record '01' des EF_ID,
     - Byte 2-19 aus Record 'XX' des EF_HLOG,
     - Byte 32-33 aus Record 'XX' des EF_HLOG.

3. DATA|ZERT|KV werden als Antwortdaten mit Returncode '90 00' ausgegeben. Hierbei ist KV die Schlüsselversion des K_{ZD} aus dem zugehörigen EF_KEYD.

2.4.3.4. Kassenschnitt

1. Es wird geprüft, ob einer der folgenden Fehlerfälle vorliegt:

   - Falls die Sequenznummer SSEQ in Byte 1-4 des Record '01' des EF_SUMME den Wert '00..00' hat: Abbruch mit Returncode '96 C3'.

   - Falls das Statusbyte in Record '01' des EF_HLOG keinen der Werte '31' (**Zahlung** beendet) oder '35' (**Fehlzahlung** beendet) hat: Abbruch mit Returncode '9F XX', wobei 'XX' den Wert des Statusbyte enthält.

2. Durch ein internes Append wird ein neuer Record '01' in EF_SUMME initialisiert:

   - Byte 1-4 werden mit dem um 1 inkrementierten Wert aus Byte 1-4 des bisherigen Record '01' beschrieben,

   - Byte 5-13 werden auf '00..00' gesetzt.

3. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 23 Byte DATA mit 1 Byte Filler '00' gebildet:

   - Byte 1-10 des Record '01' des EF_KONTO,
   - Byte 1-13 aus Record '02' des EF_SUMME.

4. DATA|ZERT|KV werden als Antwortdaten mit Returncode '90 00' ausgegeben. Hierbei ist KV die Schlüsselversion des K_{ZD} aus dem zugehörigen EF_KEYD.

2.4.3.5. Summensatz

1. Das Zertifikat ZERT für die Antwortnachricht wird berechnet. Hierzu wird der (Retail-)CBC-MAC mit K_{ZD} über die folgenden 23 Byte DATA mit 1 Byte Filler '00' gebildet:

   - Byte 1-10 des Record '01' des EF_KONTO,
   - Byte 1-13 aus Record 'XX' des EF_SUMME.

4. DATA|ZERT|KV werden als Antwortdaten mit Returncode '90 00' ausgegeben. Hierbei ist KV die Schlüsselversion des K_{ZD} aus dem zugehörigen EF_KEYD.

GeldKarte
– Ladeterminals –

Version 2.2
22.01.1997

Inhalt

4.5.5. Bestätigungsdialog

1. Einleitung

Mit **GeldKarte** wird eine Chipkarte bezeichnet, deren Struktur der Daten, Sicherheitsarchitektur sowie Standard- und Administrationskommandos der Spezifikation in [LIT 1] entsprechen, und die die Applikation elektronische Geldbörse bestehend aus dem Applikationsverzeichnis (ADF) und den zugeordneten AEFs sowie die Ergänzungskommandos der elektronischen Geldbörse enthält. Die Daten und Ergänzungskommandos der elektronischen Geldbörse sind in [LIT 4A] spezifiziert.

Zwei Typen von GeldKarten (**Kartentyp**) sind zu unterscheiden

- **Wertkarten** ohne Kontobezug und ohne PIN und
- **kontobezogene GeldKarten** mit PIN.

Das Laden einer GeldKarte erfolgt nur nach einer Online-Autorisierung, bei der geprüft wird, ob der gewünschte Ladebetrag dem Börsenverrechnungskonto der Karte gutgeschrieben werden kann. Das für das Laden einer GeldKarte zuständige Autorisierungssystem wird als **Ladezentrale** bezeichnet. Die zuständige Ladezentrale ist anhand der Kartennummer der GeldKarte eindeutig zu identifizieren.

Die Lade-Schnittstelle zu den Ladezentralen ist in Kapitel 3. spezifiziert.

Bei dem Laden einer GeldKarte übernimmt das Ladeterminal die Vermittlung zwischen der GeldKarte und der Ladezentrale und bildet die Schnittstelle zum Karteninhaber.

Zwei Ladeverfahren sind zu unterscheiden:

- Laden vom Karteninhaberkonto und
- Laden gegen andere Zahlungsmittel.

Das **Laden vom Karteninhaberkonto** (im folgenden auch als Kartenkonto bezeichnet) ist nur für kontobezogene GeldKarten (z. B. ec-Karte mit Chip oder Bankkundenkarten) nach Eingabe der PIN durch den Karteninhaber möglich. Bei einer Online-Autorisierung zum Laden vom Kartenkonto prüft die Ladezentrale die Verfügungsmöglichkeit des Karteninhabers direkt am Kartenkonto oder anhand vorgegebener Limite und veranlaßt die Abbuchung des Ladebetrags vom Kartenkonto des Karteninhabers zugunsten des Börsenverrechnungskontos.

Bankleitzahl und Kontonummer des Kartenkontos sind in der kontobezogenen GeldKarte verschlüsselt gespeichert und werden der Ladezentrale bei jeder Online-Anfrage zum Laden der GeldKarte mitgeteilt. Eine unberechtigte Veränderung dieser Kontodaten bei der Übermittlung an die Ladezentrale führt zur Abweisung einer Ladeanfrage. Bei dem Laden vom Kartenkonto wird

somit immer dasselbe Kartenkonto belastet.

Die Online-Nachrichten zum Laden vom Kartenkonto werden zwar zwischen einem Ladeterminal und der zuständigen Ladezentrale ausgetauscht, dienen aber der Kommunikation zwischen der GeldKarte und der Ladezentrale. Das Ladeterminal hat hierbei nur eine Vermittlerfunktion.

Die Ende-zu-Ende-Absicherung der Kommunikation wird dadurch erreicht, daß GeldKarte und Ladezentrale die Daten, die sie in den Nachrichten miteinander austauschen mit einem MAC versehen, der mit einem gemeinsamen, dynamisierten Schlüssel gebildet wird. Eine von GeldKarte und Ladezentrale gemeinsam geführte Sequenznummer und ein Wiederholungszähler gehen in die MAC-Bildung ein, so daß durch beide Kommunikationspartner eine Sequenzkontrolle für die ausgetauschten Daten durchgeführt werden kann.

Das **Laden gegen andere Zahlungsmittel** ist sowohl für kontobezogene GeldKarten als auch für Wertkarten ohne Kontobezug und ohne PIN möglich. Auch bei diesem Verfahren kommunizieren GeldKarte und Ladezentrale MAC-gesichert miteinander und führen Sequenznummer und Wiederholungszähler. Zusätzlich übernimmt aber das Ladeterminal eine aktive Funktion im Ladeprozeß.

Der Ladeterminalbetreiber überzeugt sich bei diesem Verfahren zunächst davon, daß der gewünschte Ladebetrag mit anderen Zahlungsmitteln an ihn bezahlt wird. Dies kann beispielsweise mittels einer Kartenautorisierung (ec-Karte, Kreditkarte und ausländische ec-Karte) oder durch eine Bargeldzahlung geschehen. Karten die zur Autorisierung eines Ladens verwendet werden, werden im folgenden als Zahlungskarten bezeichnet. Die Autorisierung eines Ladebetrags mittels ec-Karte erfolgt durch eine GA-Transaktion mit der entsprechenden Karte.

Wenn das Ladeterminal die Bestätigung dafür erhält, daß der Ladebetrag bezahlt wird, stellt es Bankleitzahl und Kontonummer eines Kontos des Ladeterminalbetreibers in die Autorisierungsanfrage ein und versieht die Anfragenachricht mit einem MAC. Durch diese "Signatur" berechtigt der Ladeterminalbetreiber die Ladezentrale, den gewünschten Ladebetrag von dem angegebenen Konto auf das Börsenverrechnungskonto der Karte zu buchen. Bei dem Laden gegen andere Zahlungsmittel wird der Ladebetrag also immer von einem Konto des Ladeterminalbetreibers abgebucht. Daher wird dieses Verfahren auch als **Laden vom Terminalkonto** bezeichnet.

Da nicht am Kartenkonto verfügt wird, ist beim Laden gegen andere Zahlungsmittel auch für kontobezogene GeldKarten keine Eingabe der Karten-PIN erforderlich.

Kontobezogene GeldKarten können mittels zweier Online-Dialoge mit der zuständigen Ladezentrale entladen werden. Hierbei veranlaßt die Ladezentrale, daß der aktuelle Betrag der GeldKarte auf 0 gesetzt wird und der gesamte Restbetrag in der GeldKarte vom Börsenverrechnungskonto auf das Kartenkonto zurückgebucht wird. Die Eingabe der PIN ist hierzu nicht erforderlich, da der entladene Restbetrag auf kein anderes Konto als das Kartenkonto gebucht werden kann.

Wie bei dem Laden vom Kartenkonto übernimmt das Terminal bei dem **Online-Entladen** auf das Kartenkonto nur eine Vermittlerrolle zwischen kontobezogener GeldKarte und Ladezentrale. Die Kommunikation wird bei dem Online-Entladen in der gleichen Weise wie bei dem Laden vom Kartenkonto Ende-zu-Ende durch MAC-Bildung und Sequenzkontrolle der GeldKarte und der Ladezentrale abgesichert.

Terminals, an denen GeldKarten entladen werden können, werden im folgenden ebenfalls als Ladeterminals bezeichnet.

Die Entlade-Schnittstelle zu den Ladezentralen ist ebenfalls in Kapitel 3. spezifiziert.

Eine GeldKarte kann nur geladen werden, wenn sie nicht verfallen und nicht gesperrt ist. Das Online-Entladen einer kontobezogenen GeldKarte ist bis 3 Monate nach Verfalldatum der Karte möglich.

Beide Typen von GeldKarten können institutsübergreifend geladen werden. Das Online-Entladen einer kontobezogenen GeldKarte kann nur an institutseigenen Ladeterminals durchgeführt werden.

2. Anforderungen an Ladeterminals

Ein Ladeterminal kann eines oder beide der oben genannten Ladeverfahren unterstützen:

- Laden von kontobezogenen GeldKarten vom Kartenkonto,
- Laden von kontobezogenen GeldKarten und von Wertkarten gegen andere Zahlungsmittel.

Zusätzlich kann ein Ladeterminal das Entladen von institutseigenen kontobezogenen GeldKarten auf das Kartenkonto unterstützen.

Ladeterminals werden ausschließlich von Kreditinstituten betrieben. Die 8 Byte lange BCD-kodierte Terminal-ID eines Ladeterminals setzt sich wie folgt zusammen:

8-stellige durch das betreibende Institut frei wählbare ID|8-stellige Betreiber-BLZ

Ein Ladeterminal kann als Zusatzfunktion die Anzeige der letzten Zahlungen oder erfolgreichen Ladungen und Entladungen anbieten.

Die Funktion eines Ladeterminals kann als zusätzliche Anwendung in andere Geräte integriert sein.

Es kann bediente und unbediente Ladeterminals geben.

2.1. Generelle Anforderungen

Die folgenden Anforderungen müssen durch jedes Ladeterminal erfüllt werden, unabhängig davon,

ob es Laden vom Kartenkonto und/oder Laden gegen andere Zahlungsmittel und/oder Online-Entladen auf das Kartenkonto unterstützt.

2.1.1. Elektrische Spezifikation und Übertragungsprotokoll

Die elektrische Spezifikation der Schnittstelle zwischen einem Ladeterminal und der GeldKarte muß den Vorgaben aus [ISO 4], [ISO 4'], [ISO 4''] und [EMV 1] genügen.

Das Übertragungsprotokoll zwischen Ladeterminal und GeldKarte ist T=1 nach [ISO 4'], [ISO 4''] und [EMV 1].

Zur Behandlung der Unterschiede der Spezifikation von T=1 in [EMV 1] einerseits und [ISO 4'] andererseits werden die folgenden Festlegungen gemacht. Die hier formulierten Anforderungen muß ein Ladeterminal erfüllen, um die ZKA-Zulassung zu erhalten.

2.1.1.1. NAD

Gemäß [EMV 1] müssen Terminals NAD='00' verwenden. Andere Werte sind nicht zulässig. Die GeldKarte lehnt daher NAD≠'00' als Protokollfehler ab.

Daher müssen alle vom Terminal gesendeten Blöcke stets NAD='00' enthalten.

2.1.1.2. Länge (LEN)

Minimale Blocklänge

Gemäß [EMV 1] wird das Senden und Empfangen von I-Blöcken mit LEN=0 nicht unterstützt. Eine GeldKarte kann somit den Empfang eines I-Blocks mit LEN=0 als einen Protokollfehler abweisen.

Daher darf das Terminal keinen I-Block mit LEN=0 senden.

IFSC und IFSD

Die GeldKarte teilt ihr IFSC im ATR mit. Zur Zeit hat IFSC mindestens den Wert 60. Größere Werte von IFSC sind zulässig.

Gemäß [ISO 4'] muß ein Terminal nur den Standardwert IFSD=32 unterstützen und kann mit einem S(IFS request) IFSD<IFSC erreichen.

Gemäß [EMV 1] muß ein Terminal IFSD=254 für das Empfangen von Blöcken unterstützen. Ein S(IFS request) des Terminals mit IFSD<IFSC ist nicht zulässig. Die GeldKarte kann immer Blöcke mit LEN≤IFSC senden.

Daher muß das Terminal für Sitzungen mit GeldKarten IFSD≥IFSC für alle Typen von GeldKarten unterstützen, auch für solche, die im ATR den Wert IFSC=254 angeben.

Dies kann "stillschweigend" geschehen, indem das Terminal Blöcke mit 32<LEN≤IFSC ohne Fehlermeldung akzeptiert, oder indem es als ersten Block nach dem Empfang des ATR ein S(IFS request) sendet mit IFSC≤IFSD≤254.

Chaining

Gemäß [EMV 1] gilt für das Chaining vom Terminal zur Chipkarte, daß für alle Blöcke außer für den abschließenden LEN=IFSC gilt. In [ISO 4'] ist diese Forderung nicht explizit enthalten. Das Terminal darf während des Chaining grundsätzlich mit LEN≤IFSC senden. GeldKarten lehnen den Empfang von I-Blöcken außer des letzten Blockes mit LEN<IFSC während des Chainings als einen Protokollfehler ab.

Das Terminal muß daher bei dem Chaining zur GeldKarte alle I-Blöcke außer dem letzten mit LEN=IFSC senden.

2.1.1.3. Behandlung von S-Blöcken

Das Senden und Empfangen von S-Blöcken durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Für das Senden eines S(IFS request) gilt allerdings die oben gemachte Einschränkung, daß IFSD<IFSC hierbei nicht zulässig ist.

2.1.1.4. Fehlerbehandlung

Die Fehlerbehandlung durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Hierbei ist die folgende Anforderung zu beachten:

Empfängt das Terminal während der Protokollabwicklung einen R-Block, so ist nur der Sequenzzähler für den weiteren Protokollablauf ausschlaggebend. Die gemäß [ISO 4'] optionale Auswertung der Bit b1 bis b4 des PCB darf durch das Terminal nur zu informativen Zwecken erfolgen.

2.1.2. Komponenten eines Ladeterminals

Ein Ladeterminal muß zur Abwicklung einer Ladetransaktion die folgenden Komponenten besitzen:

- Steuereinheit zur Abwicklung der Transaktion,

- Schnittstelle (inkl. Leser) zur Kommunikation mit einer GeldKarte,
- Schnittstelle zur Online-Kommunikation mit den Ladezentralen,
- Display zum Anzeigen der Texte für den Karteninhaber,
- Bedieneinheit (Tastatur) für den Karteninhaber.

Dem Karteninhaber muß es möglich sein, Anzeige und Kartenleser gleichzeitig im Blickfeld zu behalten. Insbesondere muß der Chipkartenleser eines unbedienten Ladeterminals für den Karteninhaber gut erreichbar sein.

Für den Karteninhaber muß deutlich erkennbar sein, wann er die GeldKarte aus dem Leser entnehmen kann. Auch bei Betriebsstörungen des Ladeterminals muß die GeldKarte aus dem Leser entnehmbar sein.

Die Eingabetastatur muß Tasten zur Betragseingabe umfassen. Zusätzlich werden Abbruch-, Korrektur- und Bestätigungstaste benötigt.

Wenn der Karteninhaber im Verlauf einer Ladetransaktion eine Auswahl treffen soll (z. B. Laden vom Kartenkonto, Laden gegen andere Zahlungsmittel), muß die Bedieneinheit die Möglichkeit zur Auswahl bieten.

2.2. Vom Ladeverfahren abhängige Anforderungen

Es ergeben sich zusätzliche Anforderungen an ein Ladeterminal in Abhängigkeit davon, ob es das Laden vom Kartenkonto, das Laden gegen andere Zahlungsmittel oder das Online-Entladen auf das Kartenkonto anbietet. Ein Ladeterminal, das zwei oder drei Funktionen bereitstellt, muß allen jeweils genannten Anforderungen genügen.

2.2.1. Laden vom Kartenkonto

Sicherheitsanforderungen an die Bauart eines Ladeterminals ergeben sich beim Laden vom Kartenkonto daraus, daß die Karteninhaber-PIN bei der Übertragung von der Eingabetastatur an die GeldKarte vor dem Ausforschen geschützt werden muß.

Hierzu sind die Kriterien III, IV.1., IV.3., VII, VIII, IX, X und XIII, bei verschlüsselter PIN-Übertragung auch IV.2., VI und XI aus den "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" ([LIT K]) zu beachten.

Die Eingabetastatur muß zur sicheren Eingabe der PIN geeignet sein.

Kapitel 4.3.3. enthält Sicherheitsanforderungen an die Ablaufkontrolle für das Laden vom Kartenkonto.

2.2.2. Laden gegen andere Zahlungsmittel

Bei dem Laden gegen andere Zahlungsmittel übernimmt das Ladeterminal eine sicherheitsrelevante Funktion im Ladeprozeß. Das Ladeterminal muß daher die Kriterien II, IV.2., V bis XIII aus den "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" in [LIT K] erfüllen.

Kapitel 4.2.3. enthält Sicherheitsanforderungen an die Ablaufkontrolle für das Laden gegen andere Zahlungsmittel.

Beim Laden gegen andere Zahlungsmittel muß das Ladeterminal eine Bestätigung dafür erhalten, daß der gewünschte Ladebetrag mit anderen Zahlungsmitteln bezahlt wird. Dies kann beispielsweise dadurch geschehen, daß

- das Ladeterminal zunächst eine Kreditkarten-Autorisierung oder eine GA-Autorisierung mit einer ec-Karte über den gewünschten Ladebetrag durchführt, oder
- das Ladeterminal Bargeld annehmen kann, oder
- ein autorisierter Bediener den Betrag an einem bedienten Ladeterminal eingibt.

Das Ladeterminal muß dem jeweiligen Zahlungsmittel entsprechend ausgerüstet sein. Beispielsweise muß für eine vorherige Autorisierung des Betrages mittels anderer Zahlungskarten

- der Kartenleser des Ladeterminals auch zur Kommunikation mit den entsprechenden Zahlungskarten geeignet sein oder ein weiterer Leser vorhanden sein und
- das Ladeterminal zur Abwicklung von Kreditkarten-Autorisierungen bzw. GA-Transaktionen mit ec-Karten ausgerüstet sein.

Es muß sichergestellt sein, daß die Bestätigung darüber, daß der Ladebetrag bezahlt wird, nicht unberechtigt erfolgt, da mit der Bestätigung die Berechtigung zur Abbuchung des Ladebetrags vom Konto des Ladeterminalbetreibers verbunden ist.

An einem bedienten Ladeterminal kann diese Bestätigung beispielsweise durch Zugriffskontrollmechanismen wie Identifikation und Authentikation des Bedieners abgesichert sein.

Wenn das Laden gegen andere Zahlungsmittel nachweisbar fehlschlägt, muß durch das

Ladeterminal sichergestellt werden, daß autorisierte Beträge storniert bzw. nicht gebucht werden. Eingenommenes Bargeld muß bei dem Fehlschlagen eines Ladens gegen andere Zahlungsmittel zurückgegeben werden können.

Für den Fall, daß über den Ausgang einer Ladetransaktion gegen andere Zahlungsmittel für das Ladeterminal keine Sicherheit besteht, muß das Journal des Ladeterminals als Nachweis für die erfolgte Zahlung mit anderen Zahlungsmitteln dienen. Daher muß die Protokollierung der Ladetransaktionen durch das Ladeterminal gegen Manipulationen geschützt sein.

2.2.3. Online-Entladen auf das Kartenkonto

Ladeterminals, die die Funktion Online-Entladen von kontobezogenen GeldKarten bereitstellen, müssen keine Anforderungen erfüllen, die über die an Ladeterminals zum Laden vom Kartenkonto hinausgehen.

Kapitel 4.5.3. enthält Sicherheitsanforderungen an die Ablaufkontrolle für das Online-Entladen auf das Kartenkonto.

2.3. Zulassung von Ladeterminals

Es dürfen nur vom ZKA zugelassene Ladeterminals eingesetzt werden. Bedingung für die Zulassung ist die funktionale Abnahme der Lade- und/oder Entladetransaktionen gemäß den Vorgaben in Kapitel 4. mit der in Kapitel 3. spezifizierten ISO 8583-Schnittstelle zu den Ladezentralen.

Gegenstand der Abnahme sind

- elektromechanische Eigenschaften der Schnittstelle zwischen Ladeterminal und Chipkarte,
- Abwicklung der Ladetransaktion und/oder Entladetransaktion durch Vermittlung der Kommunikation zwischen GeldKarte und Ladezentrale,
- Bedienung der Kundenschnittstelle.

Der Umfang der Abnahme hängt davon ab, welche(s) Lade- oder Entladeverfahren das Terminal anbieten soll.

Die Abnahme wird durch eine Stelle des Kreditgewerbes

- anhand von Testfällen,
- mit einer Chipkartensimulationen der GeldKarte und
- mittels Checklisten, die durch den Hersteller auszufüllen sind,

durchgeführt.

Weiterhin ist für die Abnahme das Sicherheitsgutachten eines Sicherheitsgutachters aus der Liste der vom ZKA benannten Gutachter beizubringen. Die in Abhängigkeit von der Funktionalität des Ladeterminals einzuhaltenden Sicherheitskriterien aus "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" in [LIT K] wurden bereits genannt. Die einzuhaltenden Sicherheitsanforderungen an die Ablaufkontrolle sind in den Kapiteln 4.2.3., 4.3.3. und 4.5.3. enthalten.

Ergebnis der Abnahme und eines positiven Sicherheitsgutachtens ist eine Typzulassung, sofern die spezifizierte ISO 8583-Schnittstelle durch das Ladeterminal eingehalten wird.

3. Schnittstelle zur Ladezentrale

Zur Abwicklung der Lade- und Entladetransaktionen werden an der Schnittstelle zur Ladezentrale Online-Nachrichten gemäß ISO 8583 (1987) verwendet.

3.1. Verwendete ISO-Nachrichten

Es werden die folgenden Abwicklungskennzeichen verwendet:

- 180000 Laden gegen Bargeld,
- 482000 Laden gegen ec-Karte,
- 483000 Laden gegen Kreditkarte,
- 484000 Laden gegen ausländische ec-Karte,
- 190000 Laden vom Kartenkonto,
- 290000 Entladen auf das Kartenkonto.

Bei den ersten vier Abwicklungskennzeichen handelt es sich um solche, die bei einem Laden gegen andere Zahlungsmittel verwendet werden. Sie dienen der Differenzierung zwischen den möglichen Zahlungsmitteln, mit denen geladen wird.

Es werden die folgenden Nachrichtentypen verwendet:

- 0200 Anfrage (Laden und Entladen),
- 0210 Antwort (Laden und Entladen),
- 0202 Quittung (nur Entladen),
- 0203 Quittungswiederholung (nur Entladen),
- 0212 Bestätigung (nur Entladen),
- 0400 Stornoanfrage (nur Laden),
- 0401 Stornowiederholung (nur Laden),

- 0410 Stornoantwort (nur Laden).

Jede Anfragenachricht und jede positive Antwortnachricht bildet einen Umschlag für die Daten, die zwischen Börsenkarte und Ladezentrale ausgetauscht werden. Diese Daten sind in BMP 62 der Nachricht enthalten. Sie werden im folgenden auch als **Ladedaten** bezeichnet.

Positive Antwortnachrichten sind in diesem Sinne

- alle Antwortnachrichten mit Antwortcode 00,
- Ladeantworten mit Antwortcode 13,
- Stornoantworten mit Antwortcode 21.

Bei den Daten in BMP 62 von Anfragenachrichten handelt es sich um Antwortdaten der Börsenkarte zu einem Kommando, das das Ladeterminal an die Börsenkarte abgesetzt hat.

BMP 62 in positiven Antwortnachrichten der Ladezentrale enthält Kommandodaten für Kommandos, die das Ladeterminal an die Börsenkarte weitergeben soll.

Anhand von Nachrichtentyp (Typ) und Abwicklungskennzeichen (Abwz.) lassen sich die verwendeten Nachrichten eindeutig identifizieren. Die folgenden Tabellen geben einen Überblick über die 9 Arten von Anfragenachrichten und 6 Arten von Antwortnachrichten zur elektronischen Geldbörse mit den im folgenden verwendeten Abkürzungen für die Nachrichtenart (ART).

ART	Typ	Abwz.	Anfragenachricht
LAK	0200	190000	Ladeanfrage zum Laden vom Kartenkonto
LAT	0200	180000 482000 483000 484000	Ladeanfrage zum Laden vom Terminalkonto
SAK	0400	190000	Stornoanfrage zum Rückbuchen eines Ladebetrags auf das Kartenkonto und Laden des Betrags 0 in die Börsenkarte
SAT	0400	180000 482000 483000 484000	Stornoanfrage zum Rückbuchen eines Ladebetrags auf das Terminalkonto und Laden des Betrags 0 in die Börsenkarte
SWK	0401	190000	Stornowiederholung zum Rückbuchen eines Ladebetrags auf das Kartenkonto und Laden des Betrags 0 in die Börsenkarte
SWT	0401	180000 482000 483000 484000	Stornowiederholung zum Rückbuchen eines Ladebetrags auf das Terminalkonto und Laden des Betrags 0 in die Börsenkarte
EA	0200	290000	Entladeanfrage zur Autorisierung des Entladens auf das Kartenkonto
QA	0202	290000	Entladequittung mit Entladenachweis der Börsenkarte zur Buchung des Entladebetrags auf das Kartenkonto und Entladen des Betrags 0 aus der Börsenkarte
QW	0203	290000	Entladequittungswiederholung mit Entladenachweis der Börsenkarte zur Buchung des Entladebetrags auf das Kartenkonto und Entladen des Betrags 0 aus der Börsenkarte

ART	Typ	Abwz.	Antwortnachricht
LK	0210	190000	Ladeantwort mit Ladedaten für die Börsenkarte zum Laden vom Kartenkonto
LT	0210	180000 482000 483000 484000	Ladeantwort mit Ladedaten für die Börsenkarte zum Laden vom Terminalkonto
SK	0410	190000	Stornoantwort mit Ladedaten zum Laden des Betrags 0 und Bestätigung der Rückbuchung auf das Kartenkonto
ST	0410	180000 482000 483000 484000	Stornoantwort mit Ladedaten zum Laden des Betrags 0 und Bestätigung der Rückbuchung auf das Terminalkonto
E	0210	290000	Entladeantwort mit Entladedaten für die Börsenkarte
Q	0212	290000	Entladebestätigung mit Entladedaten zum Entladen des Betrags 0

Alle Nachrichten haben den folgenden Aufbau:

Nachrichtentyp	Primary Bitmap	Datenfelder

Die Primary Bitmap besteht aus 8 Byte (64 Bit). Eine Nachricht hat immer dieselbe Primary Bitmap und damit denselben Aufbau, der sich aus den folgenden Tabellen ergibt. Die in den Tabellen markierten Datenfelder sind in den entsprechenden Nachrichten immer vorhanden, und das zugehörige Bit der Bitmap ist immer gesetzt. Wenn ein Datenfeld nicht verwendet wird, ist ein zugehöriges Längenfeld mit der korrekten Länge und das Wertfeld mit '00' zu belegen.

Die folgenden Tabellen geben eine Übersicht über den Aufbau der an der Schnittstelle der Ladezentralen verwendeten Nachrichten. Hierbei bedeutet

Typ: Nachrichtentyp
Lg: Länge in Byte

POV = BCD-kodiert, rechtsbündig mit führenden 0
P = BCD-kodiert, linksbündig bei ungerader Ziffernzahl rechts mit Füllzeichen 'F'
CLx = x EBCDIC-Zeichen, Ziffern ohne Vorzeichen
BIN = Binär

1): Wird nur bei bereichsübergreifenden Transaktionen ≠ 0 belegt
2): Wird nur für Ladeterminals im Kreditgewerbe ≠ 0 belegt
3): Wird nur bei Antwortcode 13 ≠ 0 belegt
4): Wird nur bei Antwortcodes 00 und 13 ≠ 0 belegt
5): Wird nur bei Antwortcodes 00 und 21 ≠ 0 belegt,
6): Wird nur bei Antwortcode 00 ≠ 0 belegt
7): Wird nur in Nachrichten mit Abwicklungskennzeichen 190000 eingestellt
x: Absender bestimmt den Wert
=: Wert aus der Anfrage(-wiederholung)
a: Wert aus der Anfrage vom Typ 0200

Die folgende Tabelle gibt eine Übersicht über die Nachrichten vom Typ 0200, 0210, 0400, 0401 und 0410, die für das Laden und das Stornieren des Ladens einer elektronischen Geldbörse verwendet werden. Der Inhalt der Nachrichtenfelder der Stornowiederholung vom Typ 0401 ist bis auf den Nachrichtentyp und dem daraus resultierenden MAC identisch mit dem Inhalt der Nachrichtenfelder der zugehörigen Stornoanfrage vom Typ 0400. Anhand des Abwicklungskennzeichens wird unterschieden, ob es sich um ein Laden vom Kartenkonto oder um ein Laden vom Terminalkonto handelt.

BMP	Typ	Typ	Typ	Typ	Bezeichnung	Inhalt	Lg	Format
	0200	0210	040x	0410				
	x	x	x	x	Nachrichtentyp		2	POV
	x	x	x	x	Primary Bitmap		8	BIN
2	x	=	a	=	Längenfeld	'F1F0'	2	CL2
	x	=	a	=	Kartennummer		10	P
3	x	=	a	=	Abwicklungskennzeichen für Laden gegen Bargeld gegen ec-Karte gegen Kreditkarte gegen ausländische ec-Karte vom Kartenkonto	 '180000' '482000' '483000' '484000' ‘190000’	3	POV
4	x	=	a	=	Transaktionsbetrag		6	POV
11	x	=	a	=	Trace-Nummer		3	POV
12	x	=	a	=	Uhrzeit	HHMMSS	3	POV
13	x	=	a	=	Datum	MMTT	2	POV
14	x	=	a	=	Verfalldatum der Börsenkarte	JJMM	2	POV
25	x	x	a	x	Konditionscode		1	POV
33	x	x	x	x	Längenfeld	'F0F3'	2	CL2
					ID zwischengeschalteter		3	POV
	x[1)]		x[1)]		Rechner			
		x		x	AS-ID			
39		x		x	Antwortcode		1	POV
41	x	=	a	=	Terminal-ID		4	POV
42	x	=	a	=	Ladeentgelt[7)] / Betreiber-BLZ		8	POV
43	x		a		Standort Ladeterminal		40	CL40
57	x	x	x	x	Längenfeld	'F0F0F9'	3	CL3
	x	x	x	x	Verschlüsselungsparameter		9	BIN
58		x			Längenfeld	'F0F0F6'	3	CL3
		x[3)]			Restbetrag		6	POV
60	x	x	x	x	Längenfeld	'F0F0F9'	3	CL3
	x	=	a	=	Kontodaten Ladeterminal		9	POV
62	x	x	x	x	Längenfeld	‘F0FxFx’	3	CL3
	x	x[4)]	x	x[5)]	Parameterwerte		xx	BIN
64	x	x	x	x	MAC		8	BIN

Die folgende Tabelle gibt eine Übersicht über die Nachrichten vom Typ 0200, 0210, 0202, 0203 und 0212, die für das Entladen einer elektronischen Geldbörse verwendet werden.

Eine Entladequittung wird automatisch durch dasselbe Ladeterminal an die Ladezentrale gesendet, das auch die Entladeanfrage geschickt hat. Entladeanfrage und Entladequittung können nicht von unterschiedlichen Ladeterminals kommen.

Der Inhalt der Nachrichtenfelder der Quittungswiederholung vom Typ 0203 ist bis auf den Nachrichtentyp und dem daraus resultierenden MAC identisch mit dem Inhalt der Nachrichtenfelder der zugehörigen Quittung vom Typ 0202.

BMP	Typ	Typ	Typ	Typ	Bezeichnung	Inhalt	Lg	Format
	0200	0210	020x	0212				
	x	x	x	x	Nachrichtentyp		2	POV
	x	x	x	x	Primary Bitmap		8	BIN
2	x x	= =	a a	= =	Längenfeld Kartennummer	'F1F0'	2 10	CL2 P
3	x	=	a	=	Abwicklungskennzeichen für Entladen	‘290000’	3	POV
4	x	=	a	=	Transaktionsbetrag		6	POV
11	x	=	a	=	Trace-Nummer		3	POV
12	x	=	a	=	Uhrzeit	HHMMSS	3	POV
13	x	=	a	=	Datum	MMTT	2	POV
14	x	=	a	=	Verfalldatum der Börsenkarte	JJMM	2	POV
25	x	x	a	x	Konditionscode		1	POV
33	x x[1)] x	x x	x x[1)]	x x	Längenfeld ID zwischengeschalteter Rechner AS-ID	'F0F3'	2 3	CL2 POV
39		x		x	Antwortcode		1	POV
41	x	=	a	=	Terminal-ID		4	POV
42	x[2)]	=	a	=	Betreiber-BLZ		8	POV
43	x		a		Standort Ladeterminal		40	CL40
57	x x	x x	x x	x x	Längenfeld Verschlüsselungsparameter	'F0F0F9'	3 9	CL3 BIN
62	x x	x x[6)]	x x	x x[6)]	Längenfeld Parameterwerte	‘F0FxFx’	3 xx	CL3 BIN
64	x	x	x	x	MAC		8	BIN

3.2. Interne Nachrichtenformate

In der Regel werden die an der Schnittstelle der Ladezentrale verwendeten ISO-Nachrichten vom Ladeterminal aufgebaut bzw. ausgewertet. Es ist aber zulässig, daß Nachrichtenfelder in Anfragenachrichten durch eine nachgelagerte Komponente im System des Ladeterminalbetreibers hinzugefügt und die entsprechenden Nachrichtenfelder in den Antwortnachrichten durch die nachgelagerte Komponente überprüft werden. Solche Nachrichtenfelder sind beispielsweise das Ladeentgelt in BMP 42 und die Kontodaten in BMP 60.

Die folgenden Daten müssen immer in die Anfrage- bzw. Antwortnachrichten eingestellt werden:

- Nachrichtentyp und Abwicklungskennzeichen

  oder eine äquivalente Kennung, durch die die Nachricht eindeutig durch die nachgelagerte Komponente des Ladeterminalbetreibers identifiziert werden kann,

- Transaktionsbetrag,

- Trace-Nummer,
- Terminal-ID und Betreiber-BLZ

  oder eine Terminalkennung, durch die das Terminal im System des Ladeterminalbetreibers eindeutig identifiziert wird,

- Datum und Uhrzeit,
- Konditions- und Antwortcode,
- die Daten der BMP 62.

Wenn durch das Sicherheitsmodul des Ladeterminals MAC-Sicherung und -Prüfung vorgenommen wird, insbesondere immer bei dem Laden gegen andere Zahlungsmittel, auch

- Verschlüsselungsparameter,
- MAC.

Anfragenachrichten des Ladeterminals, in denen Datenfelder fehlen, müssen durch eine nachgelagerte Komponente ergänzt und/oder in das ISO-Format gebracht werden. Antwortnachrichten müssen durch diese Komponente wieder in das vom Ladeterminal erwartete Format gebracht werden.

Transaktionsbetrag, Trace-Nummer, Datum und Uhrzeit, Konditions- und Antwortcode sowie die Daten der BMP 62 müssen auf der Strecke zwischen Ladeterminal und Ladezentrale und zurück unverändert bleiben.

Falls im System des Ladeterminalbetreibers individuelle Nachrichtenkennungen verwendet werden, müssen diese Kennungen durch die nachgelagerte Komponente des Ladeterminalbetreibers in das entsprechende Abwicklungskennzeichen und den entsprechenden Nachrichtentyp umgesetzt werden. Hierbei muß sichergestellt werden, daß für alle Arten von Online-Nachrichten zum Laden gegen andere Zahlungsmittel entsprechende individuelle Kennungen existieren und daß die Umsetzung immer in derselben Weise erfolgt.

Falls im System des Ladeterminalbetreibers eine individuelle Terminalkennung verwendet wird, muß diese durch die nachgelagerte Komponente des Ladeterminalbetreibers in eine im System GeldKarte eindeutige Terminal-ID umgesetzt werden, wie sie in Kapitel 2. spezifiziert ist. Hierbei muß sichergestellt werden, daß die Umsetzung immer in derselben Weise erfolgt.

Falls das Ladeterminal seine Kontodaten nicht in die Nachrichten einstellt kann dies durch die nachgelagerte Komponente erfolgen. Hierbei muß sichergestellt werden, daß die korrekten, dem Ladeterminal eindeutig zugeordneten Kontodaten eingestellt werden.

Die MAC-Bildung über die internen Nachrichten muß mit einem Sessionkey K_T' erfolgen, der aus

dem Schlüsselindex SI' und einem Schlüssel K_{MAC}' wie folgt abgeleitet wird:

$K_T' = dK_{MAC}'(SI')$

Der Schlüsselindex SI' wird durch das Ladeterminal-HSM geführt. Hierbei können K_{MAC}' und K_{MAC} verschieden sein. Dies muß aber nicht der Fall sein. Insbesondere können im System des Ladeterminalbetreibers terminalindividuelle K_{MAC}' verwendet werden. Bei Schlüsselableitung und MAC-Bildung kann auch der Triple-DES verwendet werden.

Der verwendete Schlüsselindex SI' und die Parameter zur Identifikation des K_{MAC}' im System des Ladeterminalbetreibers müssen in die interne Nachricht als Verschlüsselungsparameter eingestellt werden.

Durch die nachgelagerte Komponente des Ladeterminalbetreibers wird für interne Anfragenachrichten zunächst der K_T'-MAC der Nachricht überprüft. Anschließend wird die Nachricht nach den für das interne Format des Ladeterminalbetreibers definierten Regeln in die korrespondierende ISO-Anfragenachricht umgesetzt. Schließlich wird der K_T-MAC über die generierte ISO-Nachricht berechnet.

Auf ISO-Antwortnachrichten wird das umgekehrte Verfahren angewandt. Daten der ISO-Antwort, die nicht in die interne Nachricht eingestellt werden, müssen durch die nachgelagerte Komponente des Ladeterminalbetreibers geprüft werden. Werden hierbei Fehler festgestellt, muß dafür gesorgt werden, daß das Ladeterminal ein Storno durchführt. Dies kann dadurch erreicht werden, daß dem Ladeterminal keine Antwort geschickt wird, so daß es auf ein Timeout läuft.

3.3. Nachrichtenfolgen

3.3.1. Laden

Eine Ladetransaktion besteht aus einem Dialog aus

- Ladeanfrage (Typ 0200) und
- Ladeantwort (Typ 0210),

in denen die folgenden Felder identisch sind:

- Abwicklungskennzeichen,
- Terminal-ID und Betreiber-BLZ,
- Ladeentgelt bei einem Laden vom Kartenkonto,
- Trace-Nummer,
- Kartennummer,

- Transaktionsbetrag in BMP 4,
- Datum und Uhrzeit.

Auf eine korrekte Ladeanfrage hin, sofern der Ladebetrag autorisiert werden kann, veranlaßt die Ladezentrale die Buchung des Ladebetrags auf das Börsenverrechnungskonto und schickt in der Ladeantwort Kommandodaten für das **Laden** der Börsenkarte.

Ein Bestätigungsdialog, in dem der Ladezentrale ein erfolgreiches **Laden** der Börsenkarte explizit mitgeteilt wird, ist nicht vorgesehen. Da aber in der BMP 62 jeder Anfragenachricht an die Ladezentrale die Antwortdaten des Kommandos (**Ent-**)**Laden einleiten** (**wiederholen**) mit den aktuellen Sequenzzählern der Börsenkarte und Daten des letzten erfolgreichen Ladens oder Entladens der Börsenkarte enthalten sind, dient jeder weitere Dialog, durch den die Börsenkarte mit der Ladezentrale kommuniziert der Bestätigung eines erfolgreichen Ladens (implizite Ladebestätigung).

Andererseits kann die Ladezentrale an den Antwortdaten des (**Ent-**)**Laden einleiten** (**wiederholen**) des letzten Ladens oder Entladens der Börsenkarte in BMP 62 jeder korrekten Anfragenachricht auch erkennen, daß die letzte Ladeantwort der Ladezentrale durch die Börsenkarte nicht erfolgreich verarbeitet wurde. Auf diese Weise verursacht auch jeder weitere Dialog der Börsenkarte mit der Ladezentrale der Stornierung der Buchung eines Ladebetrags (implizites Storno).

Da es aber im Interesse des Kontoinhabers ist, von dessen Konto ein Ladebetrag abgebucht wurde, daß eine Rückbuchung bei Fehlschlagen des Ladens möglichst umgehend ausgelöst wird, ohne daß auf einen weiteren Lade- oder Entladeversuch des Karteninhabers gewartet werden muß, kann durch ein Ladeterminal ein automatisches Storno angestoßen werden.

Das (explizite) automatische Storno hat dieselbe Funktionsweise wie das implizite Storno. Es enthält in BMP 62 die Antwortdaten eines **Laden einleiten wiederholen** mit Ladebetrag 0, die die aktuellen Sequenzzähler der Börsenkarte und Daten des letzten erfolgreichen Ladens oder Entladens enthalten. Um die Daten für die Stornoanfrage zu erhalten, muß das Terminal ein **Laden einleiten wiederholen** mit Ladebetrag 0 durch die Börsenkarte erfolgreich ausführen lassen.

Kann das Kommando **Laden einleiten wiederholen** mit Ladebetrag 0 nicht durchgeführt werden, ist kein Storno möglich.

Durch ein korrektes automatisches Storno wird veranlaßt, daß die bei einer Ladetransaktion durch die Ladezentrale veranlaßte Buchung des Ladebetrags rückgängig gemacht bzw. zurückgebucht wird.

Ein Stornodialog besteht aus

- Stornoanfrage (Typ 0400) und
- Stornoantwort (Typ 0410).

Abwicklungskennzeichen, Terminal-ID und Betreiber-BLZ, Ladeentgelt bei einem Laden vom Kartenkonto, Trace-Nummer, Transaktionsbetrag in BMP 4, Kartennummer, Datum und Uhrzeit des Stornodialogs müssen identisch mit den entsprechenden Werten der zugehörigen

Ladeanfrage und -antwort sein.

Ein automatisches Storno kann durch das Ladeterminal in den folgenden Fällen ausgelöst werden:

- Die Ladeantwort der Ladezentrale trifft nicht innerhalb eines definierten Zeitraums ein.
- Die Ladeantwort ist formal falsch oder der MAC in BMP 64 der Ladeantwort ist falsch.
- Das Laden der Börsenkarte ist fehlgeschlagen.

Wenn eine Stornoanfrage innerhalb eines festgelegten Zeitfensters durch die Ladezentrale nicht beantwortet wird, oder wenn die Antwort formal falsch ist oder einen fehlerhaften MAC in BMP 64 enthält, oder wenn die Antwort korrekt ist und einen der Antwortcodes 91 oder 96 enthält, wird durch das Ladeterminal ein Stornowiederholungsdialog bestehend aus

- Stornowiederholung (Typ 0401) und
- Stornoantwort (Typ 0410)

angestoßen. Hierzu wird das Kommando **Laden einleiten wiederholen** mit Ladebetrag 0 nicht erneut an die Börsenkarte gesandt. Die Stornowiederholung wird aus der Stornoanfrage durch Änderung des Nachrichtentyps und Bildung eines neuen MAC in BMP 64 generiert.

Es dürfen höchstens zwei Wiederholungen erfolgen. Danach wird die Stornowiederholung gespeichert und zu einem späteren Zeitpunkt erneut abgesetzt.

3.3.2. Entladen

Eine Entladetransaktion besteht aus zwei Dialogen:

Entladedialog:

- Entladeanfrage (Typ 0200) und
- Entladeantwort (Typ 0210),

Bestätigungsdialog:

- Entladequittung (Typ 0202) und
- Entladebestätigung (Typ 0212).

In Anfrage- und Antwortnachricht eines Dialogs müssen die folgenden Felder der verwendeten ISO-Nachrichten identisch sein:

- Abwicklungskennzeichen (290000),
- Terminal-ID und Betreiber-BLZ,
- Trace-Nummer,

- Transaktionsbetrag in BMP 4,
- Kartennummer,
- Datum und Uhrzeit.

Abwicklungskennzeichen, Terminal-ID und Betreiber-BLZ, Trace-Nummer, Transaktionsbetrag in BMP 4, Kartennummer, Datum und Uhrzeit des Bestätigungsdialoges müssen identisch mit den entsprechenden Werten der zugehörigen Entladeanfrage und -antwort sein.

Durch den Entladedialog wird von der Ladezentrale die Autorisierung zum Entladen in Form von zertifizierten Kommandodaten für das Kommando **Entladen** gegeben. Mit diesen Daten wird der Betrag in der Börsenkarte dann auf 0 gesetzt. In dem anschließenden Bestätigungsdialog wird der Ladezentrale das Entladen mittels der Antwortdaten des **Entladen einleiten** (mit Entladebetrag 0) nachgewiesen.

Die Ladezentrale bucht erst bei Erhalt dieses Nachweises den entladenen Betrag vom Börsenverrechnungskonto auf das Kartenkonto. Durch die Entladebestätigung mit Daten zum Entladen des Betrags 0 bestätigt sie die Buchung. Wenn in der Börsenkarte nach dem Entladen ein erfolgreiches Entladen des Betrags 0 stattgefunden hat, ist dies der Nachweis in der Börsenkarte dafür, daß die Ladezentrale die zu der Entladetransaktion gehörende Buchung auch durchgeführt hat.

Dieser Nachweis in der Börsenkarte kann aber auch durch das **Laden einleiten (wiederholen)** zum Laden eines Betrages in die Börsenkarte erbracht werden. Neben der expliziten Entladequittung ist daher auch jede andere korrekte Anfragenachricht mit Daten des (**Ent-**)**Laden einleiten (wiederholen)** eine implizite Entladequittung, die zur Buchung des Entladebetrags auf das Kartenkonto führt.

Wenn eine Entladequittung innerhalb eines festgelegten Zeitfensters durch die Ladezentrale nicht beantwortet wird, oder wenn die Antwort formal falsch ist oder einen fehlerhaften MAC in BMP 64 enthält, oder wenn die Antwort korrekt ist und einen der Antwortcodes 91 oder 96 enthält, wird durch das Ladeterminal ein Bestätigungswiederholungsdialog bestehend

- aus Entladequittungswiederholung (Typ 0203) und
- Entladebestätigung (Typ 0212)

angestoßen. Die jeweilige Wiederholungsnachricht wird aus der entsprechenden Anfragenachricht durch Änderung des Nachrichtentyps und Bildung eines neuen MAC generiert.

Es dürfen höchstens zwei Wiederholungen erfolgen. Danach wird die Quittungswiederholung gespeichert und zu einem späteren Zeitpunkt erneut abgesetzt.

3.4. Erläuterung der Nachrichtenfelder

Im folgenden werden zunächst die Nachrichtenfelder der verwendeten Nachrichten beschrieben, in denen das Ladeterminal bzw. eine zwischengeschaltete Komponente und die Ladezentrale Daten austauschen.

Anschließend in Kapitel 3.4.2. werden die Daten zwischen Börsenkarte und Ladezentrale in BMP 62 erläutert.

Die Daten, die Ladeterminal bzw. zwischengeschaltete Komponente und Ladezentrale miteinander in den anderen BMP austauschen, müssen konsistent mit den Daten zwischen Börsenkarte und Ladezentrale in BMP 62 sein. Die Konsistenzanforderungen werden ebenfalls in Kapitel 3.4.2. formuliert.

3.4.1. Daten zwischen Terminal und Ladezentrale

BMP 02: Kartennummer

In dieses Feld der Anfragenachricht wird die Kartennummer aus Byte 1-9 und die Prüfziffer aus dem linken Halbbyte von Byte 10 des EF_ID einer Börsenkarte eingestellt:

Branchenhauptschlüssel	'67'	Byte 1 aus EF_ID
Kurz-Bankleitzahl	'2n nn nn'	Byte 2-4 aus EF_ID
individuelle Kartennummer	'nn nn nn nn nn'	Byte 5-9 aus EF_ID
Prüfziffer	'n'	linkes Halbbyte von Byte 10 aus EF_ID
Filler	'F'	bei Aufbau der BMP einzustellen.

BMP 03: Abwicklungskennzeichen

Es werden die folgenden Abwicklungskennzeichen verwendet:

180000: Laden gegen Bargeld,

482000: Laden gegen ec-Karte,

483000: Laden gegen Kreditkarte,

484000: Laden gegen ausländische ec-Karte,

190000: Laden vom Kartenkonto,

290000: Entladen auf das Kartenkonto.

Das Abwicklungskennzeichen muß in allen Nachrichten, die zu einer Transaktion gehören identisch sein.

BMP 04: Transaktionsbetrag

Ladeanfragen und -antworten enthalten als Transaktionsbetrag den Ladebetrag, der vom Kartenkonto bzw. dem Terminalkonto auf das Börsenverrechnungskonto umgebucht und in die elektronischen Geldbörse geladen werden soll. Stornoanfragen, -wiederholungen und antworten müssen denselben Transaktionsbetrag wie die zugehörige Ladeanfrage enthalten.

Entladeanfragen, -antworten, -quittungen, -quittungswiederholungen und -bestätigungen enthalten als Transaktionsbetrag den Entladebetrag, der vom Börsenverrechnungskonto auf das Kartenkonto umgebucht und aus der elektronischen Geldbörse entladen werden soll. Alle Entladenachrichten müssen denselben Transaktionsbetrag enthalten.

Die Wertigkeit des BCD-kodierten Betrages wird durch Byte 21 des EF_ID einer Börsenkarte bzw. Byte 21 der Parameterwerte in BMP 62 einer Anfrage oder Quittung festgelegt.

BMP 11: Trace-Nummer

Die Trace-Nummer kennzeichnet alle zu einer Ladetransaktion bzw. Entladetransaktion gehörenden Nachrichten eines Terminals. Sie wird im Ladeterminal generiert und vor einer Transaktion um 1 inkrementiert.

Ladeanfrage und -antwort einer Ladetransaktion enthalten dieselbe Trace-Nummer. Eventuell durch das Terminal ausgelöste Stornoanfragen, -wiederholungen und -antworten müssen dieselbe Trace-Nummer enthalten wie die zugehörige Ladeanfrage.

Entladeanfrage, -antwort, -quittung, -quittungswiederholung und -bestätigung einer Ladetransaktion enthalten dieselbe Trace-Nummer.

BMP 12: Uhrzeit

Dieses Nachrichtenfeld enthält die aktuelle Uhrzeit des Ladeterminals zu Beginn der Transaktion, d. h. bei Erstellung der Ladeanfrage oder der Entladeanfrage

BMP 13: Datum

Dieses Nachrichtenfeld enthält das aktuelle Datum des Ladeterminals zu Beginn der Transaktion, d. h. bei Erstellung der Ladeanfrage oder Entladeanfrage

BMP 14: Verfalldatum der Börsenkarte

Dieses Nachrichtenfeld enthält das Verfalldatum aus Byte 11-12 des EF_ID einer Börsenkarte.

BMP 25: Konditionscode

Durch den Konditionscode 16 kann in allen Anfragenachrichten angezeigt werden, daß es sich bei dem Ladeterminal um ein Bankensonderfunktionsterminal handelt. Für andere Terminals wird der Konditionscode 00 verwendet.

In Antwortnachrichten vom Typ 0210 mit Antwortcode 0 und Abwicklungskennzeichen 180000, 482000, 483000, 484000 oder 190000 zeigt eine Belegung des Konditionscode mit 98 an, daß die Maximalbeträge der elektronischen Geldbörse geändert werden sollen. Der Konditionscode 00 bedeutet, daß die Maximalbeträge unverändert bleiben sollen.

BMP 33: ID zwischengeschalteter Rechner oder AS-ID

Bei bereichsübergreifenden Transaktionen enthält dieses Nachrichtenfeld in Anfragenachrichten die ID des zwischengeschalteten Rechners.

In Antwortnachrichten wird hier durch die Ladezentrale immer die AS-ID eingestellt.

Die Belegung erfolgt analog zu der im nationalen Online-Verbund.

BMP 39: Antwortcodes

In der folgenden Tabelle wird die Bedeutung der verwendeten Antwortcodes in Abhängigkeit von der jeweiligen Nachrichtenart (ART) erläutert.

Code	ART	Bedeutung
0	LK, LT	Ladebetrag ist autorisiert und wird gebucht
	E	Entladen ist genehmigt
	SK, ST	Storno ist genehmigt und Ladebetrag wird rückgebucht
	Q	Entladebetrag wird gebucht
4[1)2)]	alle	Die Karte ist nicht zugelassen, Karte einziehen
5[1)2)]	alle	Karte aus Sicherheitsgründen abweisen
13	LK	Restlimit (AS) nicht ausreichend (Restbetrag in BMP 58, Ladebetrag 0 in BMP 62)
21	SK, ST	Buchung des zugehörigen Ladebetrags nicht erfolgt
30	alle	Formatfehler
54[1)2)]	alle	Gültigkeitsdauer überschritten oder Karte nicht aktiviert
56[1)2)]	alle	Entschlüsselte Kartendaten unbekannt oder fehlerhaft
57	SK, ST, Q	Abweichende Kartennummer in Q nur, wenn QA bzw. QW in derselben Transaktion wie EA
58[1)2)]	LK, LT, SK, ST	Terminalkontodaten unbekannt oder fehlerhaft, Terminal für Ladeverfahren nicht zugelassen
	E, Q	Entladen an institutsfremdem Terminal
62[1)2)]	alle	Die Karte ist gesperrt
64	SK, ST	Abweichender Transaktionsbetrag
76	alle	Sequenznummer in BMP 57 falsch
77	alle	Fehler in BMP 62, insbesondere MAC-Fehler
78	alle	Sequenzfehler in BMP 62
79	SK, ST, Q	Nachricht formal korrekt aber zuständiges AS nicht erreichbar
91	alle	Zuständiges AS nicht erreichbar
92[1)2)]	alle	Falsches Routing
96	alle	Zuständiges AS nicht verfügbar
97	alle	MAC falsch (BMP 64)
98	LK, LT, E	Datum/Uhrzeit nicht plausibel

1) Nicht in Stornoantwort, wenn zugehörige Ladeanfrage positiv beantwortet wurde

2) Nicht in Entladebestätigung, wenn zugehörige Entladeanfrage positiv beantwortet wurde

Im folgenden werden die Antwortcodes 13 (nur für LK), 21 (nur für SK und ST) und 79 (nur für SK, ST und Q) näher erläutert:

13: Wenn der gewünschte Ladebetrag bei einem Laden vom Kartenkonto das Restlimit übersteigt, die Ladeanfrage aber sonst fehlerfrei ist, wird dies durch den Antwortcode 13 angezeigt. Die Ladeantwort enthält dann

- in BMP 58 den zum Laden noch zur Verfügung stehenden Restbetrag und
- in BMP 62 Ladedaten zum Laden der Börsenkarte mit dem Betrag 0, um die in der Börsenkarte begonnene Ladetransaktion mit dem Laden des Betrags 0 zu beenden.

21: Mit einer Stornoanfrage bzw. Stornowiederholung fordert das Ladeterminal Daten zum Laden des Betrags 0 in die Börsenkarte, um so eine abgebrochene Ladetransaktion für die

Börsenkarte zu beenden. Gleichzeitig wird durch die in BMP 62 enthaltenen Daten nachgewiesen, daß mit der Börsenkarte ein **Laden einleiten** (**wiederholen**) stattgefunden hat. Hiermit ist gezeigt, daß vorher durch die Ladezentrale erzeugte Ladedaten nicht zu einem Laden geführt haben. Die Ladezentrale kann daher die eventuell vorher veranlaßte Buchung eines Ladebetrags rückgängig machen.

Wenn der Ladebetrag zuvor nicht gebucht wurde, die Stornoanfrage aber sonst fehlerfrei ist, wird dies durch den Antwortcode 21 angezeigt. Da der Betrag weder gebucht noch vorgemerkt ist, kann in diesem Fall keine Stornierung oder Rückbuchung des Ladebetrags vorgenommen werden. Der Ladevorgang kann aber für die Börsenkarte dadurch beendet werden, daß der Ladebetrag 0 geladen wird. Entsprechende Ladedaten werden daher in BMP 62 der Stornoantwort eingestellt.

79: Mit dem Antwortcode 79 kann dem Terminal in Stornoantwort oder Entladebestätigung angezeigt werden,

- daß die empfangende Stelle bei der formalen Prüfung der jeweiligen Anfragenachricht keinen Fehler festgestellt hat, die Anfragenachricht aber zur Zeit nicht an die zuständige Ladezentrale weiterleiten kann und

- daß die empfangende Stelle die spätere Weiterleitung bzw. Wiederholung der jeweiligen Anfragenachricht zur zuständigen Ladezentrale übernimmt.

Hierdurch wird das Terminal aufgefordert, keine (weitere) Wiederholungsnachricht zu der jeweiligen Anfragenachricht abzusetzen.

BMP 41: Terminal-ID

Ladeterminals werden ausschließlich von Kreditinstituten betrieben. Die 8 Byte lange BCD-kodierte Terminal-ID eines Ladeterminals setzt sich daher wie folgt zusammen:

8-stellige durch das betreibende Institut frei wählbare ID|8-stellige Betreiber-BLZ

In BMP 41 werden die linken 4 Byte der Terminal-ID eingestellt.

Zusammen mit der Betreiber-BLZ in den rechten vier Byte der BMP 42 wird hierdurch das Ladeterminal in der Ladezentrale eindeutig identifiziert.

Nachrichten, die zu derselben Lade- bzw. Entladetransaktion gehören, müssen dieselbe Terminal-ID enthalten.

BMP 42: Ladeentgelt/Betreiber-BLZ

In die linken vier Byte wird in Nachrichten zum institutsübergreifenden Laden vom Kartenkonto (Abwicklungskennzeichen 190000, Nachrichtentyp 0200, 0210, 040x, 0410) rechtsbündig das

Ladeentgelt in Pfennigen BCD-kodiert eingestellt.

Für eine im ZKA abzustimmende Übergangsfrist bedeutet hierbei die Belegung dieser vier Byte mit '00 00 00 00', daß in der zuständigen Ladezentrale die Standardgebühren zu ermitteln sind. Soll das Ladeentgelt entfallen (0,00 DM), wird in die vier Byte '99 99 99 99' eingestellt. Maximales Ladeentgelt ist damit 999999,98 DM.

Nach Ablauf der Übergangsfrist werden die Werte '00 00 00 00' und '99 99 99 99' wie eingestellt interpretiert.

In allen übrigen Nachrichten können die linken vier Byte intern durch das betreibende Institut belegt werden.

Ladeterminals werden ausschließlich von Kreditinstituten betrieben. Die 8 Byte lange BCD-kodierte Terminal-ID eines Ladeterminals setzt sich daher wie folgt zusammen:

8-stellige durch das betreibende Institut frei wählbare ID|8-stellige Betreiber-BLZ

In BMP 42 werden die rechten 4 Byte der Terminal-ID eingestellt.

Zusammen mit der Terminal-ID in BMP 41 wird hierdurch das Ladeterminal in der Ladezentrale eindeutig identifiziert.

Ladeanfrage und -antwort einer Ladetransaktion enthalten dieselbe BMP 42. Eventuell durch das Terminal ausgelöste Stornoanfragen, -wiederholungen und -antworten müssen dieselbe BMP 42 enthalten wie die zugehörige Ladeanfrage.

Entladeanfrage, -antwort, -quittung, -quittungswiederholung und -bestätigung einer Entladetransaktion enthalten dieselbe BMP 42.

BMP 43: Standort des Ladeterminals

Anfrage/-wiederholung und Quittung/-wiederholung enthalten in diesem Nachrichtenfeld eine Kurzbeschreibung des Standorts des Ladeterminals. Die ersten 10 Byte (von links gelesen) der 40 Byte müssen bereits zur Identifizierung des Standorts geeignet sein.

BMP 57: Verschlüsselungsparameter

Die Verschlüsselungsparameter bestehen aus 9 Byte: 8 Byte Schlüsselindex zur Bildung des Sessionkeys und 1 Byte Schlüsselgeneration.

Der Schlüsselindex wird durch das Ladeterminal vor Beginn einer neuen Transaktion inkrementiert. Alle Nachrichten einer Transaktion, die vom Ladeterminal generiert werden, müssen denselben

Schlüsselindex enthalten.

Ladeanfrage und eventuell durch das Terminal ausgelöste Stornoanfragen und Stornowiederholungen müssen demnach denselben Schlüsselindex verwenden.

Entladequittung und -quittungswiederholung müssen denselben Schlüsselindex enthalten wie die zugehörige Entladeanfrage und -antwort.

Antwortnachrichten der Ladezentrale können einen anderen Schlüsselindex enthalten als die zugehörige Anfrage des Terminals.

Die Zusammengehörigkeit von Antwort und Anfrage prüft das Terminal anhand der Trace-Nummer.

BMP 58: Restbetrag

Eine Ladeantwort mit Antwortcode 13 enthält in diesem Feld den zum Laden noch zur Verfügung stehenden Restbetrag.

BMP 60: Kontodaten des Ladeterminals

Eine Ladeanfrage zum institutsübergreifenden Laden am Kartenkonto enthält in diesem Feld die Daten des Kontos, dem die Gebühren für das Laden an diesem Terminal gutgeschrieben werden sollen.

Eine Ladeanfrage zum Laden vom Terminalkonto enthält in diesem Feld die Kontodaten des Kontos, von dem der Ladebetrag abgebucht werden soll.

Die Kontodaten sind in beiden Fällen folgendermaßen kodiert:

- 4 Byte BCD-kodierte Bankleitzahl,
- 5 Byte BCD-kodierte Kontonummer.

In eine Stornoanfrage bzw. Stornowiederholung muß das Ladeterminal dieselben Kontodaten einstellen wie in die zugehörige Ladeanfrage. Die Ladezentrale übernimmt die Kontodaten unverändert in die Lade- bzw. Stornoantwort.

BMP 62: Daten zwischen Börsenkarte und Ladezentrale

In BMP 62 sind die Daten enthalten, die Börsenkarte und Ladezentrale miteinander austauschen. Vergleiche hierzu Kapitel 3.4.2.

BMP 64: Terminal-MAC

Der Terminal-MAC wird wie bei electronic cash berechnet und geprüft. Zur MAC-Berechnung und -Prüfung wird ein Schlüssel K_{MAC} verwendet, der durch seine Generationsnummer (Byte 9 in BMP 57) eindeutig identifiziert wird.

Aus K_{MAC} und dem aktuellen Schlüsselindex SI (Byte 1-8 in BMP 57) wird der Transaktionsschlüssel K_T berechnet:

$$K_T = dK_{MAC}(SI).$$

Der MAC in BMP 64 ist der CBC-MAC mit K_T zu der Nachricht ohne BMP 64. Bezeichnet man die Nachricht ohne BMP 64 mit N, dann berechnet sich der MAC wie folgt:

$$MAC = mK_T(N).$$

Zur Zeit sind K_{MAC}, SI und K_T jeweils 8 Byte lang. Es ist daran gedacht, in Zukunft 16 Byte lange Werte, Triple-DES-Entschlüsselung im ECB-Mode bei der Schlüsselableitung und Retail-MAC-Bildung zu verwenden.

3.4.2. Daten zwischen Börsenkarte und Ladezentrale

Börsenkarte und Ladezentrale tauschen in BMP 62 Daten (Ladedaten) MAC-gesichert miteinander aus.

Das Ladeterminal stellt hier die Daten des EF_ID der Börsenkarte und die Antwortdaten des entsprechenden Börsenkommandos ein.

Die Ladezentrale prüft den MAC der eingestellten Daten und die Konsistenz der übrigen Nachrichtendaten mit diesen Daten. Sie prüft die Daten anhand der von ihr gespeicherten Daten zu dem Ladeterminal und zu der Börsenkarte.

Die Ladezentrale baut eine Antwortnachricht auf, in deren BMP 62 sie im positiven Fall wiederum Daten einstellt. Diese Daten entnimmt das Ladeterminal der Antwortnachricht, und prüft die Konsistenz der übrigen Nachrichtendaten mit diesen Daten. Es prüft die Nachricht anhand der von ihm gespeicherten Daten.

Nach erfolgreicher Prüfung erzeugt das Ladeterminal gemäß den Vorgaben, die sich aus Abwicklungskennzeichen, Nachrichtentyp, Antwortcode und Konditionscode der Nachricht ergeben eine Kommandonachricht an die Börsenkarte.

Das Längenfeld in BMP 62 einer Anfragenachricht ist immer mit F0F8F0 zu belegen.

Die Daten in BMP 62 einer Anfragenachricht haben immer den folgenden Aufbau:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der ec-Karte
13-15	3	'JJ MM TT'	Aktivierungsdatum der ec-Karte
16-17	2	'02 80'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp
23	1	'XX'	Nachrichten-ID
24-25	2	‘XX XX’	Sequenznummer LSEQ
26	1	'XX'	Wiederholungszähler WZ
27-29	3	'nn..nn'	Transaktionsbetrag
30-32	3	'nn..nn'	aktueller Betrag der Börsenkarte
33-42	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
43-58	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} CBC-verschlüsselte Karten- und Kontodaten Bei Wertkarten: '00..00'
59	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
60-61	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
62	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
63-65	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
66-68	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
69-71	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
72	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
73-80	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 23-72|'00 00 00 00 00 00'

Das Längenfeld in BMP 62 einer Antwortnachricht ist immer mit 'F0F6F5' zu belegen.

Die Daten in BMP 62 einer Antwortnachricht haben den folgenden Aufbau

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'67'	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der ec-Karte
13-15	3	'JJ MM TT'	Aktivierungsdatum der ec-Karte
16-17	2	'02 80'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp
23	1	'XX'	Nachrichten-ID
24-25	2	'XX XX'	Sequenznummer LSEQ
26	1	'XX'	Wiederholungszähler WZ
27-29	3	'nn..nn'	Ladebetrag
30-32	3	'nn..nn'	AS-ID der Ladezentrale
33-40	8	'nn..nn'	Terminal-ID des Ladeterminals
41-43	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
44-47	4	JJJJ MM TT	Datum des Ladeterminals
48-50	3	HH MM SS	Uhrzeit des Ladeterminals
51-53	3	'nn..nn'	neuer Maximalbetrag
54-56	3	'nn..nn'	neuer maximaler Transaktionsbetrag
57	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD}
58-65	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 23-57|'00 00 00 00 00'

3.4.2.1. Einzelfelder der Anfragenachricht

Im folgenden werden die einzelnen Felder der BMP 62 einer Anfragenachricht näher erläutert.

Byte 1-22

Byte 1-22 enthalten die Kartenidentifikationsdaten aus dem Record '01' des EF_ID der Börsenkarte.

Das Terminal stellt diese 22 Byte in die Anfragenachricht ein. BMP 2 und BMP 14 müssen konsistent mit diesen Daten sein:

- Die Kartennummer in BMP 2 ohne Füllzeichen 'F' muß mit der Kartennummer in Byte 1-10

der BMP 62 ohne Trennzeichen 'D' übereinstimmen.

- Das Verfalldatum in BMP 14 muß mit dem Verfalldatum in Byte 11-12 der BMP 62 übereinstimmen.

Byte 23-80

In Byte 23-80 stellt das Terminal die Antwortdaten eines Kommandos **Entladen einleiten (wiederholen)** oder **Laden einleiten (wiederholen)** ohne K_{LT}-MAC und ohne Returncode ein.

An diese Daten und Daten in anderen BMP der Anfragenachricht werden Konsistenzanforderungen gestellt. Die folgende Tabelle faßt die Konsistenzanforderungen an

- Art der Nachricht (ART, festgelegt durch Abwicklungskennzeichen in BMP 3 und Nachrichtentyp, vgl. Kapitel 3.1. zur Kodierung von ART),

- Transaktionsbetrag in BMP 4 (BT),

- verwendetes GeldKartenkommando (identifiziert durch NID, die Nachrichten-ID in Byte 23 der BMP 62),

- Transaktionsbetrag in Byte 27-29 der BMP 62 (BK),

für Entladequittungen und Entladequittungswiederholungen auch an

- Statusbyte in Byte 59 der BMP 62 (STAT),

- Transaktionsbetrag der letzten erfolgreichen Transaktion in Byte 63-65 der BMP 62 (BKALT)

zusammen.

ART	BT	Kommando	NID	BK	STAT
LAK	= BK	**Laden einleiten (wiederholen)** ohne Secure Messaging	'01' oder '05'		
LAT	= BK	**Laden einleiten (wiederholen)** mit Secure Messaging	'03' oder '07'		
SAK	= BT in zugeh. LAK	**Laden einleiten wiederholen** ohne Secure Messaging	'05'	0	
SAT	= BT in zugeh. LAT	**Laden einleiten wiederholen** mit Secure Messaging	'07'	0	
SWK	= BT in zugeh. LAK	**Laden einleiten wiederholen** ohne Secure Messaging	'05'	0	
SWT	= BT in zugeh. LAT	**Laden einleiten wiederholen** mit Secure Messaging	'07'	0	
EA	= BK	**Entladen einleiten (wiederholen)**	'21' oder '25'		
QA	= BKALT	**Entladen einleiten**	'21'	0	'31'
QW	= BKALT	**Entladen einleiten**	'21'	0	'31'

Außerdem müssen Anforderungen an die Konsistenz zwischen Sequenznummer in Byte 24-25 der BMP 62 und Wiederholungszähler in Byte 26 der BMP 62 einer Stornoanfrage und zugehöriger Ladeanfrage durch das jeweilige Terminal beachtet werden:

Bezeichnet man

- die Sequenznummer in Byte 24-25 der BMP 62 der Stornoanfrage mit LSEQN,
- die Sequenznummer in Byte 24-25 der BMP 62 der zugehörigen Ladeanfrage mit LSEQ,
- den Wiederholungszähler in Byte 26 der BMP 62 der Stornoanfrage mit WZN und
- den Wiederholungszähler in Byte 26 der BMP 62 der zugehörigen Ladeanfrage mit WZ,

muß gelten LSEQN = LSEQ und WZN > WZ.

3.4.2.2. Einzelfelder der Antwortnachricht

Im folgenden werden die einzelnen Felder der BMP 62 einer Antwortnachricht näher erläutert.

Byte 1-22

Die Ladezentrale übernimmt die Kartenidentifikationsdaten aus Byte 1-22 der BMP der zugehörigen Anfragenachricht unverändert in die Antwortnachricht. BMP 2 und BMP 14 müssen konsistent mit diesen Daten sein:

- Die Kartennummer in BMP 2 ohne Füllzeichen 'F' muß mit der Kartennummer in Byte 1-10 der BMP 62 ohne Trennzeichen 'D' übereinstimmen.

- Das Verfalldatum in BMP 14 muß mit dem Verfalldatum in Byte 11-12 der BMP 62 übereinstimmen.

Byte 23-65

Byte 23-65 enthalten Byte 6-48 der Kommandonachricht eines **Laden**, eventuell mit Ladebetrag 0, oder **Entladen**, eventuell mit Entladebetrag 0.

Byte 23

Durch die Nachrichten-ID in Byte 23 in BMP 62 jeder positiven Antwortnachricht wird durch die Ladezentrale festgelegt, welche Kommandonachricht das Ladeterminal mit den Daten in BMP 62 an die Börsenkarte schicken soll. Die Nachrichten-ID muß mit Abwicklungskennzeichen (Abwz.) in BMP 3, Nachrichtentyp (Typ) und Konditionscode (KC) in BMP 25 der Nachricht konsistent sein. Die folgende Tabelle zeigt die zu verwendenden Werte für die Nachrichten-ID:

ART	Typ	Abwz.	KC	NID
LK	0210	190000	0 98	’10’ ’14’
LT		180000 482000 483000 484000	0 98	’12’ ’16’
SK	0410	190000	0	’10’
ST		180000 482000 483000 484000	0	’12’
E	0210	290000	0	’30’
Q	0212	290000	0	’30’

Byte 24-26

Die Sequenznummer LSEQ und der Wiederholungszähler WZ müssen unverändert aus Byte 24-26 der BMP 62 in der zugehörigen Anfragenachricht übernommen werden.

Byte 27-29

Der Transaktionsbetrag muß in Abhängigkeit von der Art der Nachricht und des verwendeten Antwortcodes (AC) die folgenden Werte haben:

ART	Typ	Abwz.	AC	Betrag
LK	0210	190000	0	Ladebetrag aus BMP 4, aus Anfrage übernommen
			13	0
LT		180000 482000 483000 484000	0	Ladebetrag aus BMP 4, aus Anfrage übernommen
SK	0410	190000	0, 21	0
ST		180000 482000 483000 484000		
E	0210	290000	0	beliebig, wird von der Börsenkarte nicht ausgewertet, Empfehlung: aus Anfrage übernommen
Q	0212	290000	0	beliebig, wird von der Börsenkarte nicht ausgewertet Empfehlung: aus Anfrage übernommen

Byte 30-32

Die AS-ID in Byte 30-32 muß identisch mit der in BMP 33 eingestellten AS-ID sein.

Byte 33-40

Die in Byte 33-40 eingestellte Terminal-ID ergibt sich aus BMP 41 und BMP 42:

- Byte 33-36: BMP 41
- Byte 37-40: rechte 4 Byte der BMP 42.

Byte 41-43

Die Trace-Nummer in Byte 41-43 ist identisch mit der Trace-Nummer in BMP 11.

Byte 44-47

Das Datum in Byte 44-47 ergibt sich wie folgt:

- Byte 44-45: Jahr der Ladezentrale,
- Byte 46-47: Datum aus BMP 13.

Byte 48-50

Die Uhrzeit in Byte 48-50 ist identisch mit der Uhrzeit in BMP 12.

Byte 51-56

Enthält die Ladeantwort den Konditionscode 98, sind die Maximalbeträge in Byte 51-56 mit den neuen Werten zu belegen. Bei Konditionscode 0 wertet die Börsenkarte Byte 51-56 nicht aus. Diese Felder können dann beliebig belegt werden (Empfehlung: '00').

Byte 57

Die Generationsnummer (Schlüssel-Version) KV wird der BMP 62 der zugehörigen Anfrage entnommen.

4. Abläufe an einem Ladeterminal

Der Ablauf des Ladens an einem Ladeterminal untergliedert sich in zwei Teilabläufe:

- Vorbereitung des Ladens und
- Durchführung einer Ladetransaktion.

Falls notwendig, wird ein dritter Teilablauf

- automatisches Storno einer Ladetransaktion

durchgeführt.

Der Ablauf des Online-Entladens auf das Kartenkonto an einem Ladeterminal untergliedert sich in drei Teilabläufe:

- Vorbereiten des Entladens,
- Entladedialog und
- Bestätigungsdialog.

Die Teilabläufe des Ladens und Entladens werden im folgenden in Form von Diagrammen mit erläuternden Texten dargestellt. Hierbei werden die beiden verschiedenen Ladeverfahren, jeweils unterteilt in Laden und automatisches Storno, und das Online-Entladen, unterteilt in Entladen und Bestätigungsdialog, in separaten Kapiteln behandelt.

In den Diagrammen sind die einzelnen Schritte numeriert und in der nachfolgenden Erläuterung erklärt. In dem Diagramm bedeuten

Cn: Durch das Ladeterminal abzusetzendes Kommando bzw. abzusetzende Anfrage in Schritt

n,

An: Durch das Ladeterminal durchzuführende Aktion in Schritt n,

Rn: Antwort der jeweiligen Komponente in Schritt n.

Immer wenn der Karteninhaber im Verlauf des Ladens oder Entladens zu einer Eingabe aufgefordert wird, hat er bis zur Bestätigung der Eingabe die Möglichkeit, den Vorgang abzubrechen oder eine Korrektur der Eingabe vorzunehmen.

Wenn während einer Ladetransaktion ein Abbruch erfolgt ist, zeigt das Ladeterminal an

Laden abgebrochen, bitte Karte entnehmen

Wenn während einer Entladetransaktion ein Abbruch erfolgt ist, zeigt das Ladeterminal an

Entladen abgebrochen, bitte Karte entnehmen

Bei Wartezeiten, die Eingaben zeitweise unmöglich machen wird angezeigt

Bitte warten

Die übrigen Anzeigetexte sowie Reaktionen auf Fehlerfälle sind der jeweiligen Erläuterung zu entnehmen.

Alle beschriebenen Anzeigetexte sind sinngemäß zu verwenden.

Mit der GeldKarte kommuniziert das Ladeterminal mittels Standardkommandos der Karte, die in Kapitel 8 von [LIT 1] spezifiziert sind und mittels Ergänzungskommandos, die in Kapitel 2 von [LIT 4A] spezifiziert sind.

Die Returncodes '90 00', '63 CX' und '61 L_a' mit L_a = tatsächliche Länge der Antwortdaten eines Chipkarten-Kommandos der GeldKarte zeigen an, daß ein Kommando erfolgreich ausgeführt wurde.

Alle übrigen Returncodes der Chipkarte werden als negative Returncodes bezeichnet. Zu den möglichen Returncodes der einzelnen Kommandos vgl. Kapitel 8 in [LIT 1] sowie Kapitel 2 in [LIT 4A].

Als Fehlerfälle bei der Ausführung von Kommandos durch die GeldKarte werden im folgenden

bezeichnet

- Kommunikationsfehler, die das Kommunikationsprotokoll T = 1 meldet (beispielsweise Timeout),
- fehlerhafte Länge von Antwortdaten oder Returncodes,
- negative oder nicht definierte Returncodes,
- fehlerhaft kodierte Antwortdaten, sofern die Kodierung durch das Ladeterminal überprüft wird, und
- fehlerhafter MAC in den Antwortdaten, sofern der MAC durch das Ladeterminal überprüft wird.

Zu den nicht definierten Returncodes gehört auch der Returncode '61 XX', wenn der Wert von 'XX' nicht mit der tatsächlichen Länge der Antwortdaten übereinstimmt, oder wenn 'XX' = L_e aus der Kommandonachricht ist.

Mit der Ladezentrale kommuniziert das Ladeterminal mittels Online-Nachrichten. Die Ladezentrale erwartet hierbei die in Kapitel 3. spezifizierten Nachrichten im ISO 8583-Format.

In die Anfragenachrichten an die Ladezentrale stellt das Ladeterminal Antwortdaten von GeldKartenkommandos ein. Die Antwortnachrichten der Ladezentrale enthalten Kommandodaten für Kommandos, die das Ladeterminal an die GeldKarte absetzt. Auf diese Weise vermittelt das Ladeterminal die Kommunikation zwischen GeldKarte und Ladezentrale.

4.1. Vorbereiten des Ladens

Das Diagramm im folgenden Kapitel 4.1.2. zeigt die ersten Schritte eines Ladens, die stattfinden nachdem feststeht, daß eine GeldKarte geladen werden soll. Nachdem der Karteninhaber seine Karte eingesteckt hat, wird geprüft,

- ob die Applikation elektronische Geldbörse auf der Karte vorhanden ist,
- ob die GeldKarte gültig ist,
- ob das Laden mit dem Kommando **Laden einleiten** oder **Laden einleiten wiederholen** an die GeldKarte begonnen werden muß,
- ob es sich um eine Wertkarte oder eine kontobezogene GeldKarte handelt.

In Abhängigkeit vom Kartentyp und von den unterstützten Ladeverfahren bietet das Ladeterminal dem Karteninhaber die möglichen Ladeverfahren an.

Nach Bestätigung eines Ladeverfahrens durch den Karteninhaber wird der Ladebetrag ermittelt.

Anschließend wird in Abhängigkeit vom gewählten Ladeverfahren mit dem Laden gegen andere Zahlungsmittel (Ablauf in Kapitel 4.2.) oder mit dem Laden vom Kartenkonto fortgefahren (Ablauf in Kapitel 4.3.).

4.1.1. Varianten des Ablaufs

Der im folgenden Kapitel 4.1.2. detailliert beschriebene Ablauf der Terminalvorbereitung kann je nach Ladeverfahren variiert werden.

4.1.1.1. Laden gegen andere Zahlungsmittel

Wenn das Ladeterminal das Laden gegen Zahlungskarten unterstützt, die bei einer Autorisierung nur gelesen aber nicht beschrieben werden, ist auch die folgende Vorgehensweise möglich:

- Das Ladeverfahren wird **vor** dem Lesen der Daten der GeldKarte ausgewählt.
- Die gewählte Zahlungskarte wird in den Leser des Ladeterminals gesteckt.
- Die zu einer Autorisierung benötigten Daten der Zahlungskarte werden gelesen und gespeichert. Die für die Autorisierung erforderlichen Schritte zur PIN-Eingabe und sicheren Speicherung der PIN werden durchgeführt.
- Die Zahlungskarte wird aus dem Leser entnommen.
- Schritt 1. bis Schritt 5. der Terminalvorbereitung werden wie beschrieben durchgeführt.
- In Schritt 6. der Terminalvorbereitung wird geprüft, ob das gewählte Ladeverfahren für den Kartentyp möglich ist.

  Wenn das Ladeverfahren für den Kartentyp nicht möglich ist, wird in Abhängigkeit von dem gewählten Ladeverfahren mit der folgenden Anzeige abgebrochen

  Laden vom Konto nicht möglich, bitte Karte entnehmen

  bzw.

  Laden gegen <Zahlungsmittel> nicht möglich, bitte Karte entnehmen

Dieses Verfahren bietet für diese speziellen Zahlungskarten den Vorteil, daß in Schritt 4. des Ladens gegen andere Zahlungsmittel zur Durchführung der Kartenautorosierung kein Kartenaustausch erforderlich ist.

Nachteilig ist, daß dem Karteninhaber kein kartentypspezifisches Auswahlmenü angeboten werden kann und daß es eventuell nicht einheitlich für alle durch das Terminal akzeptierte Zahlungskarten

anwendbar ist.

4.1.1.2. Laden vom Kartenkonto

Bei einem Laden vom Kartenkonto kann die PIN-Eingabe vor der Betragsauswahl erfolgen. Hierzu können die Abläufe in Kapitel 4.1.2. und 4.3. wie folgt modifiziert werden:

Wenn in Schritt 6. der Terminalvorbereitung das Laden vom Kartenkonto ausgewählt wurde, kann die Anzeige der Beträge und die Auswahl des Ladebetrags in Schritt 7. der Terminalvorbereitung übersprungen und mit dem Laden vom Kartenkonto gemäß der Ablaufbeschreibung in Kapitel 4.3. begonnen werden.

Die Anzeige der Beträge und die Auswahl des Ladebetrags erfolgt dann in Schritt 6. des Ladens vom Kartenkonto im Anschluß an eine korrekte PIN-Eingabe.

4.1.2. Detaillierte Ablaufbeschreibung

GeldKarte			Ladeterminal		Ladezentrale	
			A1	Anzeige: **Bitte Karte einstecken**		
R2	ATR der GeldKarte	<--- --->	C2	Reset GeldKarte		
R3	OK	<--- --->	C3	SELECT FILE DF_BÖRSE		
R4	Daten aus EF_ID	<--- --->	C4 A4	READ RECORD EF_ID Daten prüfen und speichern		
R5	Record '01' aus EF_LLOG	<--- --->	C5 A5	READ RECORD '01' EF_LLOG Statusbyte auswerten und speichern		
R6	Daten aus EF_BÖRSE	<--- --->	C6 A6	READ RECORD EF_BÖRSE Kartentyp auswerten Anzeige: Lademöglichkeiten		
R7	Beträge aus EF_BETRAG	<--- --->	C7 A7	READ RECORD EF_BETRAG Anzeige: aktueller Betrag maximaler Ladebetrag Betragseingabe anfordern Beträge speichern		

Erläuterung

1. Am Kundendisplay wird angezeigt:

   Bitte Karte einstecken

2. Nachdem die GeldKarte eingesteckt ist, wird durch das Ladeterminal ein Reset der Karte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer GeldKarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Karte nicht lesbar, bitte Karte entnehmen

3. Die Applikation elektronische Geldbörse wird geöffnet, indem das ADF der Applikation DF_BÖRSE durch das Terminal mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 45 50 01 00'	AID der elektronischen Geldbörse

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Karte nicht lesbar, bitte Karte entnehmen

   Nachdem der Applikationskontext geöffnet ist, können die AEFs der Applikation mittels SFI referenziert werden.

4. Das Terminal liest mittels READ RECORD die Kartenidentifikationsdaten im Record '01' des EF_ID im MF der GeldKarte (SFI '17'). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’BC’	P2, Reference Control Byte
5	1	’16’	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der GeldKarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der GeldKarte
16-17	2	'0280'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Die empfangenen Daten werden geprüft.

Wenn die Prüfziffer nicht korrekt ist oder wenn Branchenhauptschlüssel, erste Ziffer der Kurz-BLZ, Verfalldatum, Aktivierungsdatum, Ländercode, Währungskennzeichen oder Wertigkeit der Währung nicht korrekt kodiert sind, bricht das Ladeterminal mit der folgenden Meldung ab:

Kartendaten falsch, bitte Karte entnehmen

Das Ladeterminal überprüft Verfalldatum und Aktivierungsdatum. Wenn aktuelles Datum < Aktivierungsdatum oder aktuelles Datum > Verfalldatum, bricht das Ladeterminal mit der Meldung ab

Karte ungültig, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Die Daten werden gespeichert.

Währungskennzeichen und Wertigkeit der Währung werden verwendet, um den Gegenwert der in der GeldKarte gespeicherten Betragswerte errechnen und anzeigen zu können. Die möglichen Belegungen für die Wertigkeit der Währung und deren Bedeutung sind in Kapitel 6.9 von [LIT 1] beschrieben.

Anhand der Kurz-BLZ kann die für die GeldKarte zuständige Ladezentrale identifiziert werden.

5. Das Terminal liest mittels READ RECORD die Protokolldaten des letzten Lade-/Entladeschritts der GeldKarte im Record '01' des EF_LLOG der GeldKarte (SFI '1C'). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'E4'	P2, Reference Control Byte
5	1	'21'	L_e, Recordlänge des EF_LLOG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-3	2	’XX XX’	Sequenznummer LSEQ der Lade-/Entladetransaktion
4	1	’XX’	Wiederholungszähler WZ für das **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geladener bzw. entladener Betrag
8-10	3	'nn..nn'	(neuer) aktueller Betrag
11-13	3	'nn..nn'	AS-ID der Ladezentrale
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
25-27	4	JJJJ MM TT	Datum der Lade-/Entladetransaktion
28-31	3	HH MM SS	Uhrzeit der Lade-/Entladetransaktion
32-33	2	'XX XX'	Sequenznummer BSEQ des letzten Abbuchens bzw. Rückbuchens

Wenn das Statusbyte in Byte 1 einen der Werte '11', '13', '15', '17', '31' oder '35' hat, muß das erste Kommando der Ladetransaktion ein **Laden einleiten** sein.

Wenn das Statusbyte in Byte 1 einen der Werte '01', '03', '05', '07', '21' oder '25' hat, muß das erste Kommando der Ladetransaktion ein **Laden einleiten wiederholen** sein.

Das Ladeterminal hält fest, mit welchem Kommando die Ladetransaktion zu beginnen ist.

Wenn das Statusbyte einen anderen Wert hat wird mit

Kartendaten falsch, bitte Karte entnehmen

abgebrochen.

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

6. In diesem Schritt stellt das Ladeterminal fest, welche Lademöglichkeiten dem Karteninhaber angezeigt werden können. Wenn das Ladeterminal nur das Laden gegen andere Zahlungsmittel unterstützt, bietet es bei mehreren möglichen Zahlungsmitteln eine Auswahl der unterstützten Zahlungsmitteln an.

   In welcher Form dem Karteninhaber eine Auswahl am Kundendisplay angeboten wird und wie eine Auswahl getroffen werden kann, wird hier nicht festgelegt. Lediglich der Text

   Laden gegen <Zahlungsmittel>

zur Beschreibung der verschiedenen angebotenen Ladeverfahren ist sinngemäß zu verwenden.

Nach erfolgter Auswahl und bei nur einem möglichen Zahlungsmittel zeigt das Terminal an

Laden gegen <Zahlungsmittel>, bitte bestätigen

Wenn das Terminal nur das Laden vom Kartenkonto oder sowohl das Laden vom Konto als auch das Laden gegen andere Zahlungsmittel unterstützt, liest es mittels READ RECORD den Record '01' des EF_BÖRSE (SFI '19'). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’CC’	P2, Reference Control Byte
5	1	’1B’	L_e, Recordlänge des EF_BÖRSE

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Kennung für den Kartentyp '00': kontobezogene GeldKarte 'FF': Wertkarte ohne Kontobezug
2-5	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Börsenverrechnungskonto
6-10	5	'nn..nn'	Kontonummer Börsenverrechnungskonto
11	1	'nD'	Prüfziffer über Byte 1-9
12-27	16	'XX..XX'	(Triple-)DES-Verschlüsselung im CBC-Mode mit K_{LD} und ICV = '00..00' der 16 Byte 4 Byte Bankleitzahl|5 Byte Kontonummer des Kartenkontos| 2 Byte Kartenfolgenummer|1 Byte Freizügigkeitsschlüssel| '00 00 00 00'

Wenn Byte 1 des Records keinen der Werte ’00’ oder ’FF’ hat, bricht das Ladeterminal mit der Meldung ab

Kartendaten falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Das Ladeterminal wertet den Kartentyp aus.

Wenn es sich um eine Wertkarte handelt, und das Terminal nur das Laden vom Kartenkonto unterstützt, bricht es mit der Meldung ab

Laden nicht möglich, bitte Karte entnehmen

Wenn es sich um eine kontobezogene GeldKarte handelt, und das Terminal nur das Laden vom Kartenkonto unterstützt, zeigt es an

Laden vom Konto, bitte bestätigen

Wenn es sich um eine kontobezogene GeldKarte handelt, und das Terminal sowohl das Laden vom Kartenkonto als auch das Laden gegen ein anderes oder mehrere andere Zahlungsmittel unterstützt, bietet es die Möglichkeiten zur Auswahl an.

In welcher Form dem Karteninhaber eine Auswahl am Kundendisplay angeboten wird und wie eine Auswahl getroffen werden kann, wird hier nicht festgelegt. Lediglich die Texte

Laden vom Konto

Laden gegen <Zahlungsmittel>

zur Beschreibung der verschiedenen angebotenen Ladeverfahren sind sinngemäß zu verwenden.

Je nach erfolgter Auswahl zeigt das Terminal entweder an

Laden vom Konto, bitte bestätigen

oder

Laden gegen <Zahlungsmittel>, bitte bestätigen

7. Nachdem das Ladeverfahren feststeht, wird der Ladebetrag ermittelt. Hierzu liest das

Terminal den aktuellen Betrag und den zulässigen Maximalbetrag der GeldKarte aus Record '01' des EF_BETRAG mit dem Kommando READ RECORD (SFI '18'). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'09'	L_e, Recordlänge des EF_BETRAG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Wenn einer der drei Beträge nicht BCD-kodiert ist, gibt das Ladeterminal die Meldung aus:

Kartendaten falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Das Ladeterminal berechnet den maximal möglichen Ladebetrag <maximaler Ladebetrag> als Differenz zwischen Maximalbetrag und aktuellem Betrag aus EF_BETRAG. Ist <maximaler Ladebetrag> nicht größer als 0, bricht das Ladeterminal mit der Meldung ab

Laden nicht möglich, bitte Karte entnehmen

Optional kann an dieser Stelle zusätzlich das Guthaben angezeigt werden:

Guthaben DM: <aktueller Betrag>

Laden nicht möglich, bitte Karte entnehmen

Wenn <maximaler Ladebetrag> größer als 0 ist, fordert das Terminal mit der folgenden Anzeige zur Eingabe des Ladebetrags auf

Guthaben DM: <aktueller Betrag>

Maximaler Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Hierbei müssen mindestens die beiden letzten Zeilen zusammen angezeigt werden.

Aus Transparenzgründen können nur volle DM-Beträge eingegeben werden. Es sind maximal 4-stellige volle DM-Beträge eingebbar.

Die Eingabe des Ladebetrags erfolgt mittels Zifferntasten, optional zusätzlich mittels Tasten für vollständige Beträge.

Als zusätzliche Option kann angeboten werden, den maximalen Ladebetrag ohne Betragseingabe zu laden.

Wenn der eingegebene Ladebetrag größer als <maximaler Ladebetrag> ist, zeigt das Ladeterminal an:

Ladebetrag zu hoch

Maximaler Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Wenn der Ladebetrag 0 eingegeben wird, zeigt das Ladeterminal an:

Ladebetrag 0 nicht möglich

Maximaler Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Hierbei müssen jeweils mindestens die beiden letzten Zeilen zusammen angezeigt werden.

Nach maximal drei Fehlversuchen der Betragseingabe wird das Laden abgebrochen:

Laden abgebrochen, bitte Karte entnehmen

Bei erfolgreicher Betragseingabe werden die Beträge aus EF_BETRAG und der Ladebetrag gespeichert.

4.2. Laden gegen andere Zahlungsmittel

4.2.1. Rückerstattung des Ladebetrags

4.2.1.1. Rückerstattung des Ladebetrags an den Karteninhaber

Einbehalten und Rückerstattung des gezahlten bzw. autorisierten Ladebetrags durch den Ladeterminalbetreiber wird in Abhängigkeit vom Status einer Ladetransaktion (bestehend aus Laden und eventuell automatischem Storno) bei Abbruch oder erfolgreichem Abschluß folgendermaßen gehandhabt:

- Wenn das Ladeterminal den Nachweis darüber besitzt, daß der Ladebetrag nicht vom Terminalkonto abgebucht wird und durch die Ladezentrale keine Daten zum Laden der GeldKarte erzeugt wurden, wird der Ladebetrag bei Abbruch des Ladens rückerstattet.

  Das ist solange der Fall, bis in Schritt 5 des Ladens die MAC-gesicherte Ladeanfrage erzeugt wird, und in dem Fall, daß in Schritt 5 des Ladens eine formal korrekte Ladeantwort mit fehlerfreiem MAC und negativem Antwortcode von der Ladezentrale eintrifft.

- Wenn das Ladeterminal den Nachweis darüber besitzt, daß der Ladebetrag nicht in die GeldKarte geladen wurde und auch nicht mehr geladen werden kann, wird der Ladebetrag bei Abbruch der Transaktion rückerstattet.

  Das ist dann der Fall, wenn das Ladeterminal in Schritt 2 eines automatischen Storno eine formal korrekte Antwortnachricht des Kommandos **Laden einleiten wiederholen** mit Ladebetrag 0 mit fehlerfreiem MAC und positivem Antwortcode von der GeldKarte erhält.

  Der Ladebetrag wird in diesem Fall auch dann zurückerstattet, wenn Schritt 3 und 5 des Storno noch nicht erfolgreich beendet wurden.

- Wenn das Ladeterminal den Nachweis darüber besitzt, daß der Ladebetrag in die GeldKarte geladen wurde, wird der Ladebetrag einbehalten.

  Dies ist dann der Fall, wenn das Ladeterminal in Schritt 7 eines Ladens eine formal korrekte Antwortnachricht des Kommandos **Laden**, eventuell in Schritt 2 eines automatischen Storno eine formal korrekte Antwortnachricht des Kommandos **Ladedaten wiederholen** mit fehlerfreiem MAC und positivem Antwortcode erhält.

- In allen übrigen Fällen besitzt das Ladeterminal keinen Nachweis darüber, ob der Ladebetrag in die GeldKarte geladen wurde oder nicht oder ob durch die Ladezentrale Daten zum Laden der GeldKarte erzeugt wurden. In diesen Fällen wird der Ladebetrag ebenfalls einbehalten und der Karteninhaber gebeten, sich zur Klärung an das kartenausgebende Institut zu wenden.

4.2.1.2. Rückbuchung des Ladebetrags auf das Terminalkonto

Die Ladezentrale veranlaßt die Abbuchung des Ladebetrags vom Terminalkonto, wenn sie eine korrekte Ladeanfrage des Ladeterminals erhält, bevor sie die Ladeantwort an das Ladeterminal absendet.

Folgende Fälle sind bzgl. der Rückbuchung des Ladebetrags auf das Terminalkonto bzw. der Warnung für den Ladeterminalbetreiber zu unterscheiden:

1. Wenn am Ladeterminal feststeht, daß eine Abbuchung des Ladebetrags vom Terminalkonto nicht stattgefunden hat, muß durch das Ladeterminal keine Rückbuchung veranlaßt werden.

   Das ist solange der Fall, bis in Schritt 5 des Ladens die MAC-gesicherte Ladeanfrage an die Ladezentrale geschickt wird, und in dem Fall, daß in Schritt 5 des Ladens eine formal korrekte Ladeantwort mit fehlerfreiem MAC und negativem Antwortcode von der Ladezentrale eintrifft.

2. Wenn am Ladeterminal feststeht, daß der Ladebetrag in die GeldKarte geladen wurde, darf durch das Ladeterminal keine Rückbuchung veranlaßt werden.

   Dies ist dann der Fall, wenn die GeldKarte in Schritt 7 des Ladens bei Ausführung des Kommandos **Laden** oder in Schritt 2 des automatischen Stornos bei Ausführung des Kommandos **Ladedaten wiederholen** eine korrekte Antwortnachricht mit fehlerfreiem MAC und positivem Returncode zurückgibt.

3. In allen anderen Fällen muß das Ladeterminal die Rückbuchung des Ladebetrags veranlassen, indem es ein automatisches Storno einleitet, fortsetzt oder wiederholt.

   Der Ladeterminalbetreiber ist hierbei wie folgt zu informieren:

   a) Wenn am Ladeterminal feststeht, daß das automatische Storno zu einer Rückbuchung geführt hat bzw. daß keine Abbuchung stattgefunden hat, wird der Vorgang ohne Warnung für den Ladeterminalbetreiber beendet.

      Das ist dann der Fall, wenn in Schritt 3 des automatischen Stornos

      - eine formal korrekte Stornoantwort mit fehlerfreiem MAC und positivem Antwortcode von der Ladezentrale eintrifft oder
      - eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 4, 5, 54, 56, 58, 62, 92 von der Ladezentrale eintrifft, wodurch anzeigt wird, daß die zugehörige Ladeanfrage des Ladeterminals durch die Ladezentrale entweder mit demselben Antwortcode abgewiesen oder nicht empfangen wurde, so daß eine Abbuchung des Ladebetrags vom Terminalkonto in keinem Fall stattgefunden hat.

   b) Wenn am Ladeterminal feststeht, daß das automatische Storno wiederholt wird, wird ebenfalls keine Warnung für den Ladeterminalbetreiber erzeugt.

      Dies ist dann der Fall, wenn in Schritt 2 des automatischen Stornos die erforderlichen Daten von der GeldKarte erzeugt wurden, aber in Schritt 3 des

automatischen Stornos einer der folgenden Fälle eintritt:

- die Stornoanfrage kann nicht an die Ladezentrale gesendet werden (Stornowiederholung durch das Ladeterminal),
- die Stornoantwort trifft innerhalb eines definierten Zeitraums nicht ein, ist formal nicht korrekt oder enthält einen fehlerhaften MAC (Stornowiederholung durch das Ladeterminal),
- die Stornoantwort ist fehlerfrei, enthält aber einen der Antwortcodes 79 (Stornowiederholung durch eine andere Komponente), 91 oder 96 (Stornowiederholung durch das Ladeterminal).

c) Wenn am Ladeterminal feststeht, daß das automatische Storno nicht zu einer Rückbuchung führt, obwohl eventuell eine Abbuchung des Ladebetrags vom Terminalkonto stattgefunden hat, und daß keine Stornowiederholung durchgeführt wird, wird der Vorgang mit einer Warnung für den Ladeterminalbetreiber beendet. Hierdurch wird der Ladeterminalbetreiber aufgefordert, eine Klärung auf anderem Wege herbeizuführen.

Dies ist dann der Fall, wenn das automatische Storno nicht durchführbar ist, weil die hierzu von der GeldKarte benötigten Daten nicht erhalten werden können:

- In Schritt 0 des automatischen Stornos ist ein Reset der GeldKarte oder die Selektion des DF_BÖRSE nicht möglich.
- In Schritt 1 des automatischen Stornos ist das GET CHALLENGE nicht durchführbar.
- In Schritt 2 des automatischen Stornos bricht das **Laden einleiten wiederholen** mit einem anderen Fehlerfall als Returncode '9F 13' bzw. '9F 17' ab oder das **Ladedaten wiederholen** führt zu einem Fehlerfall.

Dies ist auch der Fall, wenn das automatische Storno in Schritt 3 durch die Ladezentrale mit einer formal korrekten Stornoantwort und einem der Antwortcodes 30, 57, 64, 76, 77, 78, 97 oder einem für diesen Schritt nicht definierten Antwortcode abgelehnt wird, oder wenn in Schritt 3 des automatischen Storno dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Stornoantwort der Ladezentrale eintrifft.

4.2.2. Protokollierung

Um im Reklamationsfall den Ausgang einer Transaktion zweifelsfrei feststellen zu können, müssen mindestens die folgenden Daten einer Ladetransaktion zum Laden gegen andere Zahlungsmittel manipulationssicher protokolliert werden

- Ladeverfahren,
- Betreiber-BLZ und Terminal-ID,

- Trace-Nummer,
- Datum und Uhrzeit,
- Kartenidentifikationsdaten (Daten des EF_ID) der GeldKarte,
- Ladebetrag,
- Sequenznummer LSEQ und Wiederholungszähler WZ bzw. WZN der GeldKarte, (ab erfolgreicher Beendung von Schritt 3 des Ladens bzw. Schritt 2. des Stornos),
- aktueller Betrag der elektronischen Geldbörse zu Beginn der Transaktion (ab erfolgreicher Beendung von Schritt 3 des Ladens),
- Ergebnis der Transaktion mit Information darüber
  - ob der Ladebetrag rückerstattet wurde oder nicht,
  - in welchem Status die Transaktion abgebrochen oder erfolgreich beendet wurde.

4.2.3. Sicherheit

Die durch ein Ladeterminal, das das Laden gegen andere Zahlungsmittel unterstützt, zu erfüllenden Kriterien aus [LIT K] sind Kapitel 2.2.2. zu entnehmen.

Das Ladeterminal kommuniziert bei dem Laden gegen andere Zahlungsmittel MAC-gesichert sowohl mit der GeldKarte als auch mit der Ladezentrale.

Ein Ladeterminal zum Laden gegen andere Zahlungsmittel muß daher ein Hardware-Sicherheitsmodul (HSM) besitzen. Durch Bauart der Hardware und Sicherheitsmechanismen der Software des Sicherheitsmoduls muß gewährleistet werden, daß die verwendeten Schlüssel gegen Auslesen geschützt sind und daß kryptographische Operationen nur in gegen Auslesen und unberechtigte Veränderung geschützten Bereichen durchgeführt werden.

Beim Laden gegen andere Zahlungsmittel muß die MAC-Sicherung der Online-Nachrichten auf der gesamten Strecke zwischen Ladeterminal-Sicherheitsmodul und Ladezentrale erfolgen. Das Verfahren zur MAC-Sicherung der Online-Nachrichten ist in Kapitel 3.4. beschrieben. Im folgenden wird zunächst davon ausgegangen, daß das Ladeterminal die von der Ladezentrale erwarteten ISO-Nachrichten erzeugt und mit dem K_T-MAC sichert. Später wird darauf eingegangen, daß im System eines Ladeterminalbetreibers auch interne Nachrichtenformate verwendet werden können.

Die Kommunikation mit der GeldKarte wird mittels Secure Messaging gemäß Kapitel 3.1 in [LIT 1] abgesichert. Hierzu benötigt das Ladeterminal einen 16 Byte langen Masterkey KGK_{LT}, dem eine

Schlüsselnummer KID zwischen '0F' und '18' zugeordnet ist. Aus diesem Masterkey und den Daten des EF_ID der GeldKarte wird zur Absicherung der Kommunikation mit einer GeldKarte der jeweils kartenindividuelle Schlüssel K_{LT} abgeleitet. Das Verfahren zur Schlüsselableitung ist in Kapitel 2.5 von [LIT 1] beschrieben.

Das Verfahren zur MAC-Berechnung des Secure Messaging (CFB-MAC) ist in Kapitel 2.4 von [LIT 1] beschrieben.

Es ist daran gedacht, in Zukunft auch mehrere Generationen von KGK_{LT} in einem Ladeterminal-Sicherheitsmodul und doppelt lange K_{LT} zur Berechnung von Retail-CFB-MACs zu verwenden.

4.2.3.1. Anforderungen an die Ablaufsicherung

Am Ladeterminal muß nicht nur die Integrität der Nachrichten zwischen Terminal und Ladezentrale bzw. zwischen Terminal und GeldKarte durch die MAC-Bildung sichergestellt werden, sondern es muß auch die Ablaufsicherheit des Ladevorgangs gewährleistet werden.

Die Ablaufsicherung an einem Terminal zum Laden gegen andere Zahlungsmittel ist wesentlich, da das Ladeterminal, anders als bei einem Laden vom Kartenkonto, eine aktive Rolle im Ladeprozeß spielt:

- Am Ladeterminal wird der Gegenwert des Ladebetrags eingenommen.
- Das Ladeterminal weist mit dem K_T-MAC über die Ladeanfrage nach, daß der Ladebetrag eingenommen wurde.
- Durch den K_T-MAC des Ladeterminals wird der Ladezentrale garantiert, daß der Ladebetrag von dem in der Nachricht angegebenen Terminalkonto abgebucht werden kann.
- Das Ladeterminal muß bei Auftreten einer Störung im Ladevorgang entscheiden können, ob der Gegenwert des Ladebetrags zurückerstattet werden soll.
- Das Ladeterminal muß gesicherte Protokollinformation liefern, um lückenlos für alle Transaktionen zweifelsfrei feststellen zu können, ob das Saldo des Terminalkontos korrekt ist und alle Zahlungsbeträge korrekt einbehalten bzw. rückerstattet wurden.
- Das Ladeterminal muß manipulationssichere Protokollinformation liefern, um bei Reklamationen durch den Karteninhaber oder die Ladezentrale Daten und Ergebnis der reklamierten Transaktion unzweifelhaft feststellen zu können.

Im folgenden werden die Anforderungen an die Ablaufsicherheit eines Ladeterminals zum Laden gegen andere Zahlungsmittel erläutert. Die Anforderungen zielen nicht nur darauf, einen Terminalbetreiber vor den Risiken zu schützen, die für ihn eventuell durch eigene Ladeterminals ohne Ablaufsicherung erzeugt werden, sondern sie haben das Ziel, alle Beteiligten des Zahlungssystems GeldKarte vor Schäden, unnötigen und nicht nachvollziehbaren Reklamationen

zu schützen, die durch solche Ladeterminals verursacht werden können.

Absicherung der MAC-Berechnung am Ladeterminal

1. K_T-MAC

Der K_T-MAC über eine Ladeanfrage zum Laden gegen andere Zahlungsmittel darf durch das HSM eines Ladeterminals nur dann berechnet und ausgegeben werden, wenn

- sich die GeldKarte vorher authentisiert hat,
- die Zahlungsbestätigung für den Ladebetrag vorliegt,
- die Terminal-Identifikation in der Anfrage korrekt ist,
- die Kontodaten in der Ladeanfrage korrekt sind,
- die Trace-Nummer in der Anfrage diese Ladetransaktion eindeutig identifiziert,

Auf diese Weise wird sichergestellt daß

- die Chipkarte am Ladeterminal eine funktionierende, echte GeldKarte ist, so daß die Wahrscheinlichkeit eines Fehlers bei der Kommunikation mit der GeldKarte im weiteren Verlauf der Transaktion gering ist und Reklamationen vermieden werden,
- Ladeanfragen, die zu einer Belastung des Terminalbetreibers führen, nicht möglich sind, ohne daß er den Gegenwert des Ladebetrags erhält,
- durch die Ladezentrale immer das tatsächlich am Ladevorgang beteiligte Terminal und dessen Betreiber festgestellt werden können, und eine Täuschung der Ladezentrale durch eine gefälschte Terminal-ID oder die Terminal-ID eines nicht am Ladevorgang beteiligten Terminals nicht möglich ist, so daß ein Ladeterminalbetreiber nicht bestreiten kann, daß sein Terminal an der Transaktion beteiligt war,
- durch die Ladezentrale immer das korrekte Konto belastet wird und eine Täuschung der Ladezentrale durch gefälschte Kontodaten oder die Kontodaten eines nicht am Ladevorgang beteiligten Terminals nicht möglich ist,
- am Ladeterminal das Einspielen von Ladeantworten, die anderen Terminals oder anderen Transaktionen des Terminals zuzuordnen sind, erkannt werden kann, da die Ladezentrale Terminal-ID und Trace-Nummer aus der Ladeanfrage unverändert in die Ladeantwort übernimmt.

2. K_{LT}-MAC

Der K_{LT}-MAC über das Kommando zum Laden gegen andere Zahlungsmittel an eine GeldKarte darf durch das HSM eines Ladeterminals nur dann berechnet und ausgegeben werden, wenn

- die Ladezentrale durch den K_T-MAC über die Ladeantwort bestätigt, daß dieses Laden gegen andere Zahlungsmittel mit diesem Betrag an diesem Ladeterminal ausgeführt werden darf.

Hierdurch wird sichergestellt, daß

- eine GeldKarte nur an dem Ladeterminal geladen werden kann, das durch die Ladezentrale hierzu autorisiert wurde und dessen Daten für diese Ladetransaktion in der Ladezentrale gespeichert wurden, so daß Ladeterminalbetreiber und Ladezentrale sicher sein können, daß die in der beteiligten GeldKarte und der Ladezentrale gespeicherten Terminaldaten nicht gefälscht sind.

Der K_{LT}-MAC über das Kommando zum Einleiten des Ladens gegen andere Zahlungsmittel an eine GeldKarte darf durch das HSM eines Ladeterminals nur dann berechnet und ausgegeben werden, wenn

- die eingestellte Terminal-ID und Trace-Nummer korrekt sind.

Hierdurch wird sichergestellt, daß

- im Reklamationsfall durch einen Karteninhaber das beteiligte Terminal und die Transaktion an diesem Terminal eindeutig und zweifelsfrei aus dem Lade-Log der GeldKarte entnommen werden können, so daß Transaktionen immer rückverfolgbar sind und ein Terminalbetreiber nicht ungerechtfertigt beschuldigt werden kann.

3. Risiken einer MAC-Bildung ohne Absicherung

Das HSM eines Ladeterminals darf aus den folgenden Gründen nicht dazu verwendet werden können, ohne vorherige Prüfung zumindest des Inhalts der zu sichernden Daten einen K_T-MAC bzw. K_{LT}-MAC zu berechnen und auszugeben:

- Wenn das HSM eines Ladeterminals einen K_T-MAC über beliebige Daten berechnet, kann es mißbraucht werden, um K_T-MACs über Ladeanfragen zum Laden gegen andere Zahlungsmittel mit teilweise oder ganz gefälschten Terminaldaten zu berechnen.

- Wenn das HSM eines Ladeterminals K_T-MACs über beliebige Daten berechnet, kann es mißbraucht werden, um K_T-MACs über Antwortnachrichten zu berechnen, mit denen dem Ladeterminal selbst und allen anderen Ladeterminals mit demselben K_{MAC} gefälschte Antworten der Ladezentrale untergeschoben werden können.

Insbesondere wäre es möglich anstelle von positiven Antwortnachrichten negative Antwortnachrichten an ein Ladeterminal zu schicken und so unberechtigt die Rückerstattung des Ladebetrags an den Karteninhaber zu bewirken.

- Wenn das HSM eines Ladeterminals K_{LT}-MACs über beliebige Daten berechnet, kann es mißbraucht werden, um Kommandos zum Laden gegen andere Zahlungsmittel an eine GeldKarte zu senden, obwohl die Ladezentrale diese Kommandodaten an ein anderes Terminal geschickt hat. Im Zusammenspiel mit einem HSM, das ungeprüft K_T-MACs berechnet ist es dann möglich, die GeldKarte auf Kosten des Terminalbetreibers, an den die Kommandodaten zum Laden gegen andere Zahlungsmittel durch die Ladezentrale geschickt wurden, zu laden.

  Ferner kann ein solches HSM dazu mißbraucht werden, Kommandos zum Einleiten des Ladens gegen andere Zahlungsmittel mit gefälschten Terminaldaten an eine GeldKarte zu schicken.

Absicherung der Zahlungsbestätigung

Bei einem Laden gegen andere Karten kann die Prüfung der Zahlungsbestätigung durch die Ablaufsicherung des Ladeterminals erfolgen, wenn zur Autorisierung der Transaktion mit der Zahlungskarte kryptographisch gesicherte Nachrichten gesendet und empfangen werden, die durch die Ablaufsicherung kontrollierbar sind.

Bei dem Laden gegen Bargeld muß auf andere Weise sichergestellt werden, daß die Zahlungsbestätigung für den Ladebetrag nicht unberechtigt erfolgen kann. Erfolgt die Zahlungsbestätigung an einem bedienten Terminal durch einen Bediener, dann muß der Bediener durch die Ablaufsicherung vorher identifiziert und authentisiert werden. Auf diese Weise ist nicht nur sichergestellt, daß eine Ladetransaktion immer einem bestimmten Bediener zuzuordnen ist sondern auch, daß das Ladeterminal nicht unberechtigt auf Kosten eines Ladeterminalbetreibers benutzt werden kann.

Erfolgt die Zahlungsbestätigung an einem unbedienten Terminal durch einen automatischen Bargeldprüfer, so muß sichergestellt werden, daß die Kommunikation zwischen Bargeldprüfer und Ablaufsicherung des Ladeterminals nicht gefälscht werden kann. Dies kann dadurch erreicht werden, daß Bargeldprüfer und Ablaufsicherung kryptographisch gesichert miteinander kommunizieren. Falls dies nicht der Fall ist, muß die Kommunikationsstrecke zwischen Bargeldprüfer und Ablaufsicherung des Ladeterminals durch Hardware geschützt sein. Insbesondere darf es nicht möglich sein, das Ladeterminal aus dem gemeinsamen Hardwareschutz zu entfernen und die Schnittstelle des Ladeterminals zum Bargeldprüfer unberechtigt anzusteuern.

Absicherung der Protokollierung

In bestimmten Fehlerfällen muß der zunächst erhaltene Zahlungsbetrag an den Karteninhaber

rückerstattet werden. Diese Fälle können immer unzweifelhaft identifiziert werden, da sie mit kryptographischen Operationen verknüpft sind. Voraussetzung ist aber, daß das Ladeterminal eine korrekte Ablaufsicherung besitzt, die einen Zustandsautomaten realisiert, der durch die kryptographischen Operationen gesteuert wird:

- Wenn der Nachweis erbracht ist, daß der Ladebetrag nicht vom Terminalkonto abgebucht wird und durch die Ladezentrale keine Daten zum Laden der GeldKarte erzeugt wurden, wird der Ladebetrag bei Abbruch des Ladens rückerstattet.

  Dies ist bei einem Abbruch nach "Transaktion beginnen" (in Schritt 1. des Ladens) und vor "K_T-MAC für Ladeanfrage berechnen" sowie nach "K_T-MAC für negative Ladeantwort erfolgreich geprüft" (in Schritt 5. des Ladens) der Fall. K_T-MAC-Berechnung und -Prüfung verstehen sich hierbei immer inklusive der Prüfung von Nachrichteninhalt und Zustand der Ablaufsicherung.

- Wenn der Nachweis erbracht ist, daß der Ladebetrag nicht in die GeldKarte geladen wurde und auch nicht mehr geladen werden kann, wird der Ladebetrag bei Abbruch des Ladens rückerstattet.

  Dies ist bei einem Abbruch nach "K_{LT}-MAC für **Laden einleiten wiederholen** mit Ladebetrag 0 erfolgreich geprüft" (in Schritt 2. des Storno). K_{LT}-MAC-Prüfung versteht sich hierbei inklusive der Prüfung von Nachrichteninhalt und Zustand der Ablaufsicherung.

Erfolgt der Abbruch an einer anderen Stelle der Ladetransaktion, wird der Zahlungsbetrag einbehalten. Hierbei ist zu unterscheiden, ob der Karteninhaber den Zahlungsbetrag später eventuell reklamieren darf oder nicht. Auch diese Unterscheidung kann durch eine Ablaufsicherung festgestellt werden:

- Wenn der Nachweis erbracht ist, daß der Ladebetrag in die GeldKarte geladen wurde, wird der Ladebetrag einbehalten, ohne daß spätere Reklamationen möglich oder zulässig sind.

  Dies ist bei einem Abbruch nach "K_{LT}-MAC für **Laden** erfolgreich geprüft" (in Schritt 7. des Ladens) oder nach "K_{LT}-MAC für **Ladedaten wiederholen** erfolgreich geprüft" (in Schritt 2. des Storno) der Fall. K_{LT}-MAC-Prüfung versteht sich hierbei inklusive der Prüfung von Nachrichteninhalt und Zustand der Ablaufsicherung.

- In allen übrigen Fällen besitzt die Ablaufkontrolle bei Abbruch der Ladetransaktion keinen Nachweis darüber, ob der Ladebetrag in die GeldKarte geladen wurde oder nicht oder ob durch die Ladezentrale Daten zum Laden der GeldKarte erzeugt wurden. In diesen Fällen wird der Ladebetrag ebenfalls einbehalten und der Karteninhaber gebeten, sich zur Klärung an das kartenausgebende Institut zu wenden. Wurde die GeldKarte nicht geladen, wird der einbehaltene Ladebetrag später im allgemeinen durch den Kartenausgeber vom Ladeterminalbetreiber reklamiert.

  Durch die Ablaufkontrolle können auch diese Fälle anhand der ausgeführten kryptographischen Operationen eindeutig identifiziert werden.

Die Rückerstattung oder Einbehaltung des Zahlungsbetrages kann im allgemeinen nicht durch die Ablaufkontrolle des Ladeterminals kontrolliert werden. Dies ist nur dann der Fall, wenn auch die Rückerstattung mittels kryptographischer Operationen gesichert wird, beispielsweise bei der Stornierung von Transaktionen mittels Zahlungskarten oder bei kryptographisch gesicherter Kommunikation mit einem automatischen Bargeldprüfer. In allen anderen Fällen kann durch die Ablaufsicherung nur sichergestellt werden, daß

- der Zustand, in dem die Transaktion abgebrochen wurde, zusammen mit den Daten der Transaktion und dem Ergebnis "Zahlungsbetrag rückzuerstatten" oder "Zahlungsbetrag einzubehalten, keine Reklamation möglich" "Zahlungsbetrag einzubehalten, Reklamation möglich" manipulationssicher und lückenlos protokolliert wird, um für den Abgleich des Terminalkontos und den Reklamationsfall unzweifelhaft die Transaktionsergebnisse zur Verfügung zu haben,

- der Karteninhaber über den Ausgang der Transaktion nicht getäuscht werden kann, indem die Anzeige des Ladeterminals und die spezifizierten Anzeigetexte durch die Ablaufsicherung kontrolliert werden.

Die Manipulationssicherheit der Protokollierung kann dadurch erreicht werden, daß die Protokolldaten, durch die Ablaufsicherung gesteuert, kryptographisch gesichert werden. Der Ladeterminalbetreiber muß dann die Möglichkeit haben, die Protokolldaten zu prüfen und, für den Reklamationsfall, deren Echtheit Dritten gegenüber nachzuweisen.

Wenn an einem bedienten Ladeterminal gegen Bargeld geladen wird, muß durch die Protokollierung der Ablaufkontrolle auch festgehalten werden, welcher Bediener für diese Transaktion authentisiert war.

4.2.3.2. Maßnahmen zur Ablaufsicherung

Die zuvor erläuterten Anforderungen an die Ablaufsicherung sind nicht zu erfüllen, wenn das Ladeterminal über ein HSM verfügt, das die Schlüssel K_{MAC} und KGK_{LT} enthält, aber über ungeprüfte Daten K_T- oder K_{LT}-MACs berechnet und durch manipulierbare Software im Steuerrechner des Ladeterminals betrieben wird oder werden kann.

Die Ablaufsicherheit kann dadurch erreicht werden, daß das HSM einen Zustandsautomaten implementiert und nur dann eine MAC-Bildung vornimmt, wenn dies in dem jeweiligen Zustand des HSM zulässig ist und die Daten, die mit einem MAC versehen werden sollen, dem Zustand entsprechend korrekt sind. Der entsprechende Zustandsautomat mit den zugehörigen Prüfroutinen kann auch in dem Steuerrechner des Ladeterminals implementiert werden. Dann ist aber sicherzustellen, daß

- die Software des Steuerrechners fehlerfrei arbeitet und nicht verändert werden kann,

- die Kommunikation zwischen Steuerrechner und HSM nicht manipuliert werden kann und

- das HSM ausschließlich an Steuerrechnern betrieben werden kann, die die korrekte Software enthalten.

Im folgenden wir erläutert, mit welchen Daten und Prüfschritten im Rahmen der MAC-Berechnung und MAC-Bildung die Ablaufsicherung und Protokollierung für den in den Kapiteln 4.2.4. und 4.2.5. spezifizierten Ablauf einer Transaktion zum Laden gegen andere Zahlungsmittel zu realisieren ist. Hierbei wird davon ausgegangen, daß die Ablaufsicherung zusammen mit MAC-Prüfung und MAC-Bildung im HSM realisiert ist.

Trace-Nummer und Schlüsselindex

Trace-Nummer und Schlüsselindex müssen im HSM geführt werden. Sie werden im HSM bei Beginn einer neuen Transaktion (Schritt 1. des Ladens) inkrementiert. Vor der Berechnung eines MAC über eine Kommandonachricht an die GeldKarte oder eine Online-Nachricht an die Ladezentrale und bei der Prüfung des MAC einer Online-Nachricht prüft das HSM, ob in die Nachricht die aktuelle Trace-Nummer eingestellt ist. Auf diese Weise ist sichergestellt, daß die Transaktionen zum Laden gegen andere Zahlungsmittel am Ladeterminal und in der GeldKarte eindeutig identifizierbar sind und die Zusammengehörigkeit von Online-Anfragen und -Antworten eindeutig feststellbar ist.

Terminal-ID und Terminalkontodaten

Die Terminal-ID und die Kontodaten des Ladeterminals sind im HSM gespeichert. Bevor das HSM einen MAC über ein Kommando an die GeldKarte oder eine Online-Nachricht an die Ladezentrale berechnet, prüft es, ob Terminal-ID und/oder Kontodaten mit den gespeicherten Werten übereinstimmen. Auf diese Weise ist eine Fälschung dieser Daten ausgeschlossen. Insbesondere akzeptiert ein Ladeterminal keine Nachricht, die die Ladezentrale nicht an dieses Terminal geschickt hat.

Nachrichtenkennungen

Im HSM ist hinterlegt, welche Kommandonachrichten an die GeldKarte, Kommandoantworten der GeldKarte, Online-Nachrichten an die Ladezentrale und Online-Nachrichten von der Ladezentrale in welchem Zustand zulässig sind. Die zulässigen Nachrichten sind der Spezifikation in Kapitel 4.2.4 zu entnehmen. Kommandonachrichten und Antwortnachrichten der GeldKarte sind immer anhand der Nachrichten-ID zu identifizieren. Online-Nachrichten an die Ladezentrale und von der Ladezentrale sind anhand von Abwicklungskennzeichen und Nachrichtentyp zu identifizieren. Vor der MAC-Berechnung über eine Nachricht und bei der MAC-Prüfung einer Nachricht wird durch das HSM geprüft, ob die Nachricht an dieser Stelle der Transaktion zulässig ist. Auf diese Weise wird sichergestellt, daß das HSM ausschließlich MACs über die hierfür zulässigen Daten berechnet.

EF_ID der GeldKarte

Bei Beginn der Transaktion (Schritt 1. des Ladens) wird das EF_ID der beteiligten GeldKarte an

das HSM übergeben und dort für die Dauer der Transaktion sicher gespeichert.

Während der Transaktion leitet das HSM den K_{LT} nur aus dem gespeicherten EF_ID ab. Das HSM prüft vor der MAC-Sicherung einer Online-Anfrage, ob in BMP 62 das authentisierte EF_ID eingestellt ist. Auf diese Weise wird sichergestellt, daß nur die in Schritt 3. eines Ladens authentisierte GeldKarte im Rahmen der Transaktion geladen werden kann.

Ladebetrag

Bei Beginn der Transaktion (Schritt 1. des Ladens) wird der durch den Karteninhaber bestätigte Ladebetrag an das HSM übergeben und dort für die Transaktion sicher gespeichert.

Vor der Berechnung eines MAC über eine Kommandonachricht an die GeldKarte oder eine Online-Nachricht an die Ladezentrale und bei der Prüfung des MAC einer Antwortnachricht der GeldKarte oder einer Online-Nachricht prüft das HSM, ob in die Nachricht der korrekte Ladebetrag eingestellt ist. Auf diese Weise wird sichergestellt, daß nur der in Schritt 4. eines Ladens bestätigte Ladebetrag im Rahmen der Transaktion geladen werden kann.

Zähler der GeldKarte

Im Rahmen einer erfolgreichen MAC-Prüfung des **Laden einleiten (wiederholen)** in Schritt 3. des Ladens speichert das HSM die authentisierten Zähler LSEQ, WZ der GeldKarte. Falls ein Storno durchgeführt wird, speichert das HSM in Schritt 2. des Storno im Rahmen einer erfolgreichen MAC-Prüfung des **Laden einleiten (wiederholen)** mit Ladebetrag 0 den neuen Wiederholungszähler WZN.

Vor der Berechnung eines MAC über eine Kommandonachricht zum **Laden** an die GeldKarte oder eine Online-Nachricht an die Ladezentrale und bei der Prüfung des MAC einer Antwortnachricht des **Laden** oder **Laden einleiten wiederholen** im Rahmen eines Storno der GeldKarte oder des MAC einer Online-Nachricht prüft das HSM, ob LSEQ und WZ bzw. WZN in der Nachricht korrekt sind.

Die jeweils durchzuführenden Prüfungen sind in den Kapiteln 4.2.4. und 4.2.5. beschrieben.

Auf diese Weise kann unzweifelhaft festgestellt werden, ob das Laden einer GeldKarte erfolgreich war bzw. ob das Laden der GeldKarte mit Ladedaten der Transaktion in Zukunft unmöglich ist. Hierdurch wird sichergestellt, daß der Ladebetrag korrekt einbehalten bzw. rückerstattet wird.

Aktueller Betrag der GeldKarte

Im Rahmen einer erfolgreichen MAC-Prüfung des **Laden einleiten (wiederholen)** in Schritt 3. des Ladens speichert das HSM den aktuellen Betrag der GeldKarte. Der aktuelle Betrag wird für die Protokollierung benötigt, um im Reklamationsfall zweifelsfrei feststellen zu können, wie hoch der Betrag vor dem Laden war.

Challenges

Die Zufallszahlen RND2, RND4, RND6 und RND8 in den Schritten 3. bzw. 4. des Ladens, 2. bzw. 5. des Storno werden durch das HSM erzeugt, gespeichert und bei der jeweiligen Prüfung des K_{LT}-MAC als ICV verwendet. Auf diese Weise wird sichergestellt, daß das HSM die Nachrichten

der GeldKarte authentisieren kann.

4.2.3.3. Verwendung eines internen Nachrichtenformats

Es ist zulässig, daß am Ladeterminal nicht die kompletten, von der Ladezentrale erwarteten ISO-Nachrichten erzeugt werden. Nachrichtenfelder in den ISO-Anfragenachrichten können durch eine nachgelagerte Komponente im System des Ladeterminalbetreibers hinzugefügt und die entsprechenden Nachrichtenfelder in den Antwortnachrichten durch die nachgelagerte Komponente überprüft werden. In Kapitel 3.2. ist festgelegt, welche Daten immer in die Anfragenachrichten im internen Format eingestellt werden müssen und wie interne Nachrichten durch die nachgelagerte Komponente des Ladeterminalbetreibers zu handhaben sind..

Bei den einzustellenden Daten handelt es sich, mit Ausnahme der Kontodaten des Ladeterminals, um die zuvor als sicherheitsrelevant identifizierten Daten der Online-Nachrichten sowie um die Daten, die der Kommunikation zwischen GeldKarte und Ladezentrale dienen.

Die nachgelagerte Komponente muß zur MAC-Prüfung und -Bildung ein Sicherheitsmodul besitzen. Durch die Ablaufsicherung der nachgelagerten Komponente muß manipulationssicher gewährleistet werden, daß MACs nur im Rahmen der Umsetzung berechnet werden, so daß das Sicherheitsmodul nicht mißbraucht werden kann, um MACs über beliebige Nachrichten zu berechnen. Die andernfalls entstehenden Risiken wurden bereits erläutert.

Wenn die nachgelagerte Komponente wie beschrieben arbeitet, kann die Ablaufsicherung im Ladeterminal in analoger Weise wie bei der Verwendung von ISO-Nachrichten am Ladeterminal realisiert werden. Eventuell entfällt am Ladeterminal das Einstellen und Prüfen der Terminalkontodaten. Dies wird durch die nachgelagerte Komponente übernommen.

4.2.4. Laden

In dem folgenden Diagramm werden die Schritte des Ladens gegen andere Zahlungsmittel dargestellt, die nachfolgend näher erläutert werden.

GeldKarte			Ladeterminal			Ladezentrale	
			A1	Transaktion beginnen			
		<---	C2	GET CHALLENGE			
R2	RND1	--->					
			A3	RND2 erzeugen und speichern, K_{LT} ableiten und speichern			
			C3	**Laden einleiten** oder **Laden einleiten wiederholen** mit K_{LT}-MAC ICV = RND1			
		<---					
R3	Antwortdaten des **Laden einleiten (wiederholen)** mit K_{LT}-MAC ICV = RND2	--->					
			A3	Antwortdaten und K_{LT}-MAC prüfen			
			A4	Zahlungsbestätigung für den Ladebetrag			
			A5	K_T ableiten			
			C5	Ladeanfrage 0200 Abwkz. 180000 482000 483000 484000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R5	Ladeantwort 0210 Abwkz. 180000 482000 483000 484000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A5	Ladeantwort und K_T-MAC prüfen			

GeldKarte			Ladeterminal		Ladezentrale	
		<---	C6	GET CHALLENGE		
R6	RND3	--->				
			A7	RND4 erzeugen und speichern		
			C7	**Laden** mit K_{LT}-MAC ICV = RND3		
		<---				
R7	Antwortdaten des **Laden** mit K_{LT}-MAC ICV = RND4					
		--->				
			A7	Antwortdaten und K_{LT}-MAC prüfen Anzeige: geladener Betrag, neuer aktueller Betrag, eventuell: neue Maximalbeträge,		

Erläuterung

1. Nachdem Ladeverfahren und Ladebetrag feststehen, beginnt das Ladeterminal eine neue Transaktion. Hierbei muß sichergestellt sein, daß

   - Trace-Nummer TSEQ und
   - Schlüsselindex SI

   gegenüber der letzten Transaktion inkrementiert sind.

   Hierbei wird die Trace-Nummer nach Erreichen des maximal möglichen Wertes auf 0 zurückgesetzt.

   Hat der Schlüsselindex den Maximalwert erreicht, wird das Terminal für den weiteren Betrieb als Ladeterminal gesperrt.

   In allen Schritten der Transaktion müssen durch das Ladeterminal dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet werden.

2. Mit dem Kommando GET CHALLENGE läßt sich das Ladeterminal eine Zufallszahl RND1

von der GeldKarte geben. Die folgende Tabelle zeigt den Aufbau der Command-APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'84'	INS
3	1	'00'	P1, fester Wert
4	1	'00'	P2, fester Wert
5	1	'08'	L_e

Wenn das Kommando erfolgreich ausgeführt wurde, gibt die GeldKarte eine 8 Byte lange Zufallszahl RND1 als Antwortdatum aus.

Im Fehlerfall wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

3. Das Ladeterminal leitet aus dem gespeicherten KGK_{LT} und den Daten des EF_ID der GeldKarte den 8 Byte langen kartenindividuellen Schlüssel K_{LT} ab. Es erzeugt und speichert außerdem eine 8 Byte lange Zufallszahl RND2.

   In Abhängigkeit vom Wert des Statusbyte der GeldKarte wird die Kommandonachricht für das **Laden einleiten** oder das **Laden einleiten wiederholen** mit Secure Messaging aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'00' '20'	P1 für **Laden einleiten** P1 für **Laden einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘2D’	L_c
6	1	'02' '06'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
7-9	3	'00..00'	Filler
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-41	8	'XX..XX'	RND2 ICV für die MAC-Bildung über die Antwortdaten
42	1	'XX'	Logische Schlüsselnummer KID des KGK_{LT}
43-50	8	'XX..XX'	CFB-MAC mit K_{LT} über die 48 Byte Byte 1-42|Byte 51|'00 00 00 00 00' mit ICV = RND1
51	1	‘42’	L_e

Der Ladebetrag muß der vom Karteninhaber bestätigte Betrag sein.

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

RND2 ist die vom Ladeterminal erzeugte Zufallszahl.

KID ist die Schlüsselnummer des KGK_{LT}

Das Ladeterminal berechnet den CFB-MAC mit K_{LT} für das Kommando, wobei es RND1 der GeldKarte als ICV verwendet.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück.

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'03' '07'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'
59-66	8	'XX..XX'	CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-58|'00 00 00 00 00 00' mit ICV = RND2

Das Ladeterminal prüft, ob Nachrichten-ID und Ladebetrag die korrekten Werte haben. Es prüft den CFB-MAC mit K_{LT}, wobei es die von ihm erzeugte Zufallszahl RND2 als ICV verwendet.

In allen Fehlerfällen wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

Bei erfolgreicher Kommandoausführung werden der aktuelle Betrag der GeldKarte, die Sequenznummer LSEQ und der Wiederholungszähler WZ für die Dauer der Transaktion gespeichert.

Die komplette Antwortnachricht wird noch in Schritt 5. des Ladens benötigt.

4. Das Ladeterminal erhält die Bestätigung, daß der Ladebetrag an den Ladeterminalbetreiber mittels des gewählten Zahlungsmittels bezahlt ist oder wird.

Wenn zur Bestätigung des Ladebetrags die Autorisierung mittels einer anderen Chipkarte oder Magnetstreifenkarte verwendet werden soll und hierzu derselbe Leser verwendet wird wie für die GeldKarte, wird die Meldung angezeigt

Bitte GeldKarte entnehmen

Die Transaktionsdaten der Ladetransaktion werden gespeichert.

Anschließend erfolgen die Schritte und Anzeigen der jeweiligen Kartenautorisierung, wie sie für das entsprechende Kartensystem festgelegt sind (GA-Autorisierung für ec-Karten).

Nach Abschluß der Autorisierung wird dann angezeigt:

Bitte GeldKarte wieder einstecken

Nach erneutem Einstecken der GeldKarte müssen die Schritte 2. bis 4. des Einleitens erneut durchgeführt werden. In Schritt 4. muß nach dem Lesen des EF_ID geprüft werden, ob die erneut eingesteckte Karte identisch mit der zunächst verwendeten Karte ist, indem die Kartennummer in Byte 1-10 des gespeicherten EF_ID mit der Kartennummer des neu gelesenen EF_ID verglichen wird.

Bei Fehlerfällen in diesen Schritten wird mit der folgenden Meldung abgebrochen

Chipfehler, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

Anschließend muß die Autorisierung des Ladebetrags rückgängig gemacht werden. In welcher Form dies geschieht, hängt von der jeweiligen Zahlungskarte ab.

Steht ein zweiter Leser zur Verfügung, oder kann die Autorisierung mittels aufgezeichneter Kartendaten der Zahlungskarte durchgeführt werden, kann das Entnehmen und erneute Einstecken der GeldKarte entfallen.

Nach erfolgter Zahlungsbestätigung und eventuell notwendigem erneuten Einstecken der GeldKarte zeigt das Ladeterminal an:

Ladebetrag DM: . . . . , Laden wird bearbeitet

Wenn die Zahlung des Ladebetrags nicht bestätigt wird, wird die Transaktion mit der Meldung abgebrochen

Laden abgebrochen, bitte Karte entnehmen

falls die GeldKarte sich im Leser befindet, ansonsten mit der Meldung

Laden abgebrochen

5. Eine Ladeanfrage mit Nachrichtentyp 0200 und Abwicklungskennzeichen 180000 für das Laden gegen Bargeld, 482000 für das Laden mittels einer GA-Transaktion mit ec-Karte, 483000 für das Laden mittels einer Kreditkarte oder 484000 für das Laden mittels einer GA-Transaktion mit einer ausländischen ec-Karte an die Ladezentrale wird aufgebaut. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

   Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

   Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

   In BMP 62 der Ladeanfrage stellt das Ladeterminal das EF_ID der GeldKarte und die geprüften Antwortdaten der GeldKarte zu **Laden einleiten (wiederholen)** ein.

   Der Transaktionsschlüssel K_T wird wie in Kapitel 3. spezifiziert aus dem K_{MAC} unter Verwendung des Schlüsselindex SI abgeleitet. Die Ladeanfrage wird mit dem K_T MAC-gesichert.

   Mit dem MAC über die Anfragenachricht bestätigt das Ladeterminal, daß der Ladebetrag für dieses Laden von dem Konto, dessen Daten in BMP 60 eingestellt sind, abgebucht werden kann.

   Es muß durch das Ladeterminal oder einer nachgelagerten Sicherheitskomponente, die die Nachricht ergänzt und umschlüsselt, sichergestellt werden, daß in BMP 60 korrekte

Kontodaten eingestellt werden.

Das Ladeterminal schickt die Ladeanfrage an die Ladezentrale.

Wenn es nicht möglich ist, die Ladeanfrage an die Ladezentrale zu senden, oder wenn die Ladeantwort der Ladezentrale nicht innerhalb eines definierten Zeitraums eintrifft, beginnt das Ladeterminal ein automatisches Storno, dessen Ablauf in Kapitel 4.2.5. beschrieben ist.

Wenn das Ladeterminal eine Antwortnachricht erhält, prüft es, mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

Stellt das Terminal hierbei Fehler fest, beginnt es ein automatisches Storno.

Wenn die Antwortnachricht formal korrekt ist und einen fehlerfreien MAC aber einen negativen Antwortcode (≠ 0) enthält, bricht das Ladeterminal mit der folgenden Meldung ab:

<Text>, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 58, 76, 91, 92, 96, 97, 98 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler

Anschließend muß der Ladebetrag rückerstattet werden. In welcher Form dies geschieht,

hängt von dem jeweiligen Zahlungsmittel ab.

6. Wenn die Antwortnachricht formal korrekt ist, einen fehlerfreien MAC und den Antwortcode 0 enthält, läßt sich das Ladeterminal mit dem Kommando GET CHALLENGE eine Zufallszahl RND3 von der GeldKarte geben.

   Läßt sich das Kommando GET CHALLENGE auch nach Wiederholung (eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE) nicht erfolgreich durchführen, zeigt das Ladeterminal an

   Chipfehler, Rückerstattung des Ladebetrags nicht möglich

   Wenden Sie sich an ihr kartenausgebendes Institut

   Bitte Karte entnehmen

   Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

7. Das Ladeterminal baut aus den Daten in BMP 62 der Antwortnachricht die Kommandonachricht eines **Laden** mit Secure Messaging auf. Hierzu erzeugt und speichert es eine Zufallszahl RND4.

   Es wertet den Konditionscode in BMP 25 aus, um festzustellen, ob das Laden mit oder ohne Änderung der Maximalbeträge erfolgen soll. Hat der Konditionscode den Wert 98, ist P1 = 'A0' zu setzen, hat der Konditionscode den Wert 00, ist P1 = '80' zu setzen.

   Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'80' 'A0'	P1 für **Laden** ohne Änderung der Maximalbeträge P1 für **Laden** mit Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘3C’	L_c
6	1	'12' '16'	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49-56	8	'XX..XX'	RND4
57	1	'XX'	Logische Schlüsselnummer KID des KGK_{LT}
58-65	8	'XX..XX'	CFB-MAC mit K_{LT} über die 64 Byte Byte 1-57|Byte 66|'00 00 00 00 00 00 00' mit ICV = RND3
66	1	‘12’	L_e

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht.

Es stellt Byte 1-5, Byte 49-57 und Byte 66 wie spezifiziert ein und berechnet den CFB-MAC mit K_{LT} über Byte 1-57, den es anschließend in Byte 58-65 einstellt. Als ICV zur MAC-Berechnung verwendet das Terminal RND3 der GeldKarte.

Das Terminal speichert die verwendete Nachrichten-ID.

Bei Konditionscode 98 prüft das Terminal, ob beide Maximalbeträge oder nur ein Maximalbetrag geändert wurden. Dazu vergleicht es den neuen Maximalbetrag aus Byte 34-36 der Command-APDU mit dem gespeicherten Maximalbetrag der GeldKarte und den maximalen Transaktionsbetrag aus Byte 37-39 der Command-APDU mit dem gespeicherten maximalen Transaktionsbetrag.

Das Terminal speichert den geänderten Betrag bzw. die geänderten Beträge.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’13’ ’17’	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'nn..nn'	geladener Betrag
8-10	3	'nn..nn'	neuer aktueller Betrag
11-18	8	'XX..XX'	CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-10|'00 00 00 00 00 00' mit ICV = RND4

Das Ladeterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Ladeterminal gespeicherten Werten überein?
- Ist der Ladebetrag korrekt?
- Ist der MAC korrekt?

Zur MAC-Prüfung verwendet das Terminal als ICV die gespeicherte Zufallszahl RND4. Stellt

das Terminal bei einer Prüfung einen Fehler fest, leitet es ein automatisches Storno ein.

Wenn die GeldKarte bei Ausführung des **Laden** einen negativen Returncode zurückgibt oder wenn die Prüfung der Antwortdaten des **Laden** einen Fehler ergibt, beginnt das Ladeterminal ein automatisches Storno.

In allen anderen Fehlerfällen bei Ausführung des **Laden**, versucht das Ladeterminal, das **Laden** erneut auszuführen.

Hierzu läßt sich das Ladeterminal, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE, mittels GET CHALLENGE eine neue Zufallszahl RND3' von der Karte ausgeben. Es berechnet und speichert selbst eine neue Zufallszahl RND4' und berechnet den MAC für die Kommandonachricht neu.

Wenn die GeldKarte das Kommando dann erfolgreich bearbeitet, wird mit den Antwortdaten wie oben beschrieben verfahren. Andernfalls leitet das Ladeterminal ein automatisches Storno ein.

Bei erfolgreicher Ausführung des **Laden** mit positivem Ausgang der Prüfungen und wenn die Maximalbeträge nicht geändert wurden, entnimmt das Ladeterminal den neuen aktuellen Betrag der Antwortnachricht und zeigt an

Ladebetrag DM: <Ladebetrag> geladen

Neues Guthaben DM: <neuer aktueller Betrag>

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

Wenn die Prüfungen positiv ausfallen und beide Maximalbeträge geändert wurden, zeigt das Ladeterminal an

Ladebetrag DM: <Ladebetrag> geladen

Neues Guthaben DM: <neuer aktueller Betrag>

Neuer Maximalbetrag DM: <neuer Maximalbetrag>

Maximaler Zahlungsbetrag DM: <neuer maximaler Transaktionsbetrag>

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

Wenn einer der beiden Maximalbeträge nicht geändert wurde, wird dieser nicht angezeigt. Der geänderte Betrag kann zusammen mit der Meldung "Bitte Karte entnehmen" angezeigt werden.

Für die GeldKarte und das Ladeterminal ist die Transaktion damit beendet.

4.2.5. Automatisches Storno

Ein automatisches Storno wird dann durchgeführt, wenn in Schritt 5 des Ladens eine Ladeanfrage mit einem Terminal-MAC an die Ladezentrale geschickt wurde, aber keine oder eine fehlerhafte Ladeantwort eintrifft oder eine korrekte Ladeantwort mit Antwortcode 0 eintrifft, aber in Schritt 7 der Ladetransaktion kein Nachweis über ein erfolgreiches **Laden** in die GeldKarte erzeugt werden kann.

Sollte in Schritt 2 eines automatischen Storno der Nachweis über das erfolgreiche Laden nachträglich erzeugt werden, wird das automatische Storno abgebrochen.

Bei der erfolgreichen Durchführung eines automatischen Storno wird in die GeldKarte der Betrag 0 geladen, wodurch sichergestellt ist, daß

- der Ladebetrag der Ladetransaktion nicht mehr in die GeldKarte geladen werden kann (nach erfolgreicher Beendung des Schritt 2),

- der Ladebetrag nicht vom Terminalkonto abgebucht wird bzw. daß der Ladebetrag auf das Terminalkonto rückgebucht wird (nach erfolgreicher Beendung von Schritt 3)

- die Transaktion für die GeldKarte mit Heraufsetzen der Sequenznummer LSEQ abgeschlossen wurde (nach erfolgreicher Beendung von Schritt 5).

Die Schritte 4 und 5 werden nur durchgeführt, wenn das Ladeterminal in Schritt 3 eine positive Stornoantwort (Antwortcode 0 oder 21) auf eine Stornoanfrage vom Typ 0400 erhält.

Das automatische Storno gehört zu derselben Transaktion wie das zu stornierende Laden. Es werden daher für das automatische Storno dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet wie für das zugehörige Laden.

In dem folgenden Diagramm werden die Schritte des Storno eines Ladens gegen andere Zahlungsmittel dargestellt, die nachfolgend näher erläutert werden.

GeldKarte			Ladeterminal			Ladezentrale	
			A0	Storno beginnen			
		<---	C1	GET CHALLENGE			
R1	RND5	--->					
			A2	RND6 erzeugen und speichern, K_{LT} ableiten und speichern			
			C2	**Laden einleiten wiederholen** mit Betrag 0 K_{LT}-MAC ICV = RND5			
		<---					
R2	Antwortdaten des **Laden einleiten wiederholen** mit Betrag 0 K_{LT}-MAC ICV = RND6	--->					
			A2	Antwortdaten und K_{LT}-MAC prüfen			
			C3	Stornoanfrage 0400 eventuell Storno-wiederholung 0401 Abwkz. 180000 482000 483000 484000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R3	Stornoantwort 0410 Abwkz. 180000 482000 483000 484000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A3	Stornoantwort und K_T-MAC prüfen eventuell Anzeige: **Laden abgebrochen**			

GeldKarte			Ladeterminal		Ladezentrale	
		<---	C4	GET CHALLENGE		
R4	RND7	--->				
			A5	RND8 erzeugen und speichern		
			C5	**Laden** mit Betrag 0 K_{LT}-MAC ICV = RND7		
		<---				
R5	Antwortdaten des **Laden** mit Betrag 0 K_{LT}-MAC ICV = RND8					
		--->				
			A5	Antwortdaten und K_{LT}-MAC prüfen Anzeige: **Laden abgebrochen**		

Erläuterung

0. Zu Beginn eines automatischen Storno führt das Ladeterminal ein Reset der GeldKarte durch und selektiert das DF_BÖRSE erneut (vgl. Schritt 2. und 3. in Kapitel 4.1.)

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Chipfehler, Rückerstattung des Ladebetrags nicht möglich

   Wenden Sie sich an Ihr kartenausgebendes Institut

   Bitte Karte entnehmen

   Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

1. Mit dem Kommando GET CHALLENGE läßt sich das Ladeterminal eine Zufallszahl RND5 von der GeldKarte ausgeben.

   Läßt sich das Kommando GET CHALLENGE auch nach zweifacher Wiederholung und eventuell erneutem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE nicht erfolgreich durchführen, zeigt das Ladeterminal an

   Chipfehler, Rückerstattung des Ladebetrags nicht möglich

Wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

2. Das Ladeterminal erzeugt und speichert eine 8 Byte lange Zufallszahl RND6.

Die Kommandonachricht für das **Laden einleiten wiederholen** mit Ladebetrag 0 und mit Secure Messaging wird aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'20'	P1 für **Laden einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘2D’	L_c
6	1	'06'	Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
7-9	3	'00..00'	Filler
10-12	3	'00..00'	Ladebetrag 0
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-41	8	'XX..XX'	RND6 ICV für die MAC-Bildung über die Antwortdaten
42	1	'XX'	Logische Schlüsselnummer KID des KGK_{LT}
43-50	8	'XX..XX'	CFB-MAC mit K_{LT} über die 48 Byte Byte 1-42|Byte 51|'00 00 00 00 00' mit ICV = RND5
51	1	‘42’	L_e

Der Ladebetrag muß 0 sein.

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

RND6 ist die vom Ladeterminal erzeugte Zufallszahl.

KID ist die Schlüsselnummer des KGK_{LT}

Das Ladeterminal berechnet den CFB-MAC mit K_{LT} für das Kommando, wobei es RND5 der GeldKarte als ICV verwendet.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück.

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'07'	Nachrichten-ID für **Laden einleiten wiederholen** mit Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'00..00'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'
59-66	8	'XX..XX'	CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-58|'00 00 00 00 00 00' mit ICV = RND6

Das Ladeterminal prüft, ob die Nachrichten-ID den korrekten Wert hat, und ob der Ladebetrag 0 ist.

Es prüft den CFB-MAC mit K_{LT}, wobei es die von ihm erzeugte Zufallszahl RND6 als ICV verwendet.

Seien

LSEQN die Sequenznummer aus Byte 2-3,

WZN der Wiederholungszähler aus Byte 4

der Antwortdaten und

LSEQ die vom Ladeterminal gespeicherte Sequenznummer,

WZ der vom Ladeterminal gespeicherte Wiederholungszähler.

Wenn gilt

LSEQN = LSEQ und

WZN > WZ

sind die Antwortdaten des **Laden einleiten wiederholen** der für den Aufbau einer Stornoanfrage benötigte Nachweis dafür, daß das Laden nicht erfolgt ist und nicht mehr erfolgen kann.

Der neue Wiederholungszähler WZN wird für die Dauer der Transaktion gespeichert. Die komplette Antwortnachricht wird noch in Schritt 3. des Storno benötigt. Es wird mit Schritt 3. des Storno fortgefahren.

Gibt die GeldKarte als Antwort zu **Laden einleiten wiederholen** den Returncode '9F 13' bzw. '9F 17' zurück, wurde das **Laden** eventuell zuvor korrekt bearbeitet.

In diesem Fall veranlaßt das Ladeterminal die erneute Ausgabe der Antwortdaten des **Laden** mit dem Kommando **Ladedaten wiederholen**. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘38’	INS
3	1	'00'	P1 für **Ladedaten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘09’	L_c
6-13	8	'XX..XX'	RND4"
14	1	'XX'	Logische Schlüsselnummer KID des KGK_{LT}
15	1	‘12’	L_e

RND4” ist eine von dem Ladeterminal neu erzeugte und gespeicherte Zufallszahl.

Wenn die GeldKarte anschließend die Antwortdaten des **Laden** ausgibt, werden diese wie in Schritt 7. des Ladens beschrieben geprüft. Im positiven Fall, wird das erfolgreiche Laden dem Karteninhaber wie in Schritt 7. des Ladens angezeigt.

Tritt bei der Ausführung des **Ladedaten wiederholen** ein Fehler auf, wird mit der Meldung abgebrochen

Chipfehler, Rückerstattung des Ladebetrags nicht möglich

Wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

Wenn bei der Ausführung des **Laden einleiten wiederholen**

- weder korrekte Antwortdaten, die nachweisen, daß das Laden nicht erfolgt ist,
- noch einer der Returncodes '9F 13' oder '9F 17'

ausgegeben werden, versucht das Ladeterminal, das **Laden einleiten wiederholen** erneut auszuführen, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE.

Gelingt es hierbei, von der GeldKarte korrekte Antwortdaten zu erhalten, die nachweisen, daß nicht geladen wurde, wird der in den Antwortdaten enthaltene Wiederholungszähler

WZN gespeichert und mit Schritt 3. des Storno fortgefahren.

Bei jedem anderen Ausgang des zweiten **Laden einleiten wiederholen** wird mit der Meldung abgebrochen

Chipfehler, Rückerstattung des Ladebetrags nicht möglich

Wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

3. Wenn das automatische Storno eingeleitet wurde, weil die Ladeanfrage in Schritt 5. des Ladens nicht an die Ladezentrale geschickt werden konnte, wird die Transaktion an dieser Stelle mit der Anzeige

Laden abgebrochen, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

für die GeldKarte und das Ladeterminal beendet.

Andernfalls kann die Transaktion optional an dieser Stelle für die GeldKarte mit der Anzeige

Laden abgebrochen, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

beendet werden, wonach der Ladebetrag rückerstattet werden muß. In welcher Form dies geschieht, hängt von dem jeweiligen Zahlungsmittel ab. Für das Ladeterminal ist die Transaktion hiermit noch nicht beendet. Es muß vielmehr durch das Ladeterminal sichergestellt sein, daß die Stornoanfrage und eventuell die Stornowiederholungen wie in diesem Schritt 3. beschrieben durchgeführt werden, um die Rückbuchung des Ladebetrags auf das Terminalkonto zu veranlassen.

Das Ladeterminal baut mit den in Schritt 2 erhaltenen Antwortdaten des **Laden einleiten wiederholen** mit LSEQN = LSEQ und WZN > WZ als nötigem Nachweis der GeldKarte eine Stornoanfrage mit Nachrichtentyp 0400 und dem Abwicklungskennzeichen aus der zugehörigen Ladeanfrage (180000, 482000, 483000 bzw. 484000) an die Ladezentrale auf. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

In BMP 62 der Ladeanfrage stellt das Ladeterminal das EF_ID der GeldKarte und die geprüften Antwortdaten der GeldKarte zu **Laden einleiten wiederholen** ein.

Die Stornoanfrage wird mit dem K_T MAC-gesichert.

Das Ladeterminal schickt die Stornoanfrage an die Ladezentrale. Wenn es nicht möglich ist, die Stornoanfrage an die Ladezentrale zu senden, wird so verfahren als wäre innerhalb des definierten Zeitraums keine Stornoantwort eingetroffen.

Wenn das Ladeterminal eine Stornoantwort erhält, prüft es, mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

In den folgenden Fällen beendet das Ladeterminal in diesem Schritt die Transaktion für die GeldKarte:

- Innerhalb eines definierten Zeitraums trifft keine Stornoantwort ein,

  Optional kann der Karteninhaber das Warten auf die Stornoantwort vorzeitig abbrechen.
- Es trifft eine Stornoantwort ein, die formal falsch ist oder deren MAC fehlerhaft ist,
- Es trifft eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21 ein.

Hierzu wird dem Karteninhaber angezeigt

<Text>, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode/Ereignis	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 57, 58, 64, 76, 91, 92, 96, 97 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler
79, keine Antwort, formal falsche Antwort Abbruch durch den Karteninhaber	Laden abgebrochen

Anschließend muß der Ladebetrag rückerstattet werden. In welcher Form dies geschieht, hängt von dem jeweiligen Zahlungsmittel ab.

Die Schritte 4. und 5. des automatischen Storno werden nur dann durchgeführt, wenn das Ladeterminal innerhalb eines definierten Zeitraums eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 0 oder 21 erhält und der Karteninhaber die Transaktion nicht abgebrochen hat.

Falls das Ladeterminal eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21, 91, 96 erhalten hat, ist die Transaktion auch für das Ladeterminal beendet. Wenn der Antwortcode ≠ 4, 5, 54, 56, 58, 62, 79, 92 ist, generiert es eine Warnung für den Ladeterminalbetreiber, da das Storno nicht erfolgreich beendet werden konnte.

In allen anderen Fällen muß das Ladeterminal die im folgenden beschriebenen Schritte durchführen, um die Rückbuchung des Ladebetrags auf das Terminalkonto doch noch zu veranlassen.

Stornowiederholung

Falls keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, führt das Ladeterminal eine Stornowiederholung durch.

Das Ladeterminal generiert aus der Stornoanfrage eine Stornowiederholung, indem es den Nachrichtentyp 0400 durch den Typ 0401 ersetzt und über die Nachricht mit K_T einen neuen MAC rechnet.

Diese Nachricht sendet es an die Ladezentrale.

Falls das Ladeterminal zu der Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 0, 21, 4, 5, 54, 56, 58, 62, 79, 92 erhält, ist das Storno für das Ladeterminal erfolgreich beendet.

Falls das Ladeterminal zu der Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21, 4, 5, 54, 56, 58, 62, 79, 91, 92, 96 erhalten hat, ist die Transaktion für das Ladeterminal ebenfalls beendet. Es generiert in diesem Fall eine Warnung für den Ladeterminalbetreiber, da das Storno nicht erfolgreich beendet werden konnte.

Zweite Stornowiederholung

Falls keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wiederholt das Ladeterminal die Stornowiederholung.

Falls das Ladeterminal zu der zweiten Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 0, 21 4, 5, 54, 56, 58, 62, 79, 92 erhält, ist das Storno für das Ladeterminal erfolgreich beendet.

Falls das Ladeterminal zu der zweiten Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21, 4, 5, 54, 56, 58, 79, 62, 91, 92, 96 erhalten hat, ist die Transaktion für das Ladeterminal ebenfalls beendet. Es generiert eine Warnung für den Ladeterminalbetreiber, da das Storno nicht erfolgreich beendet werden konnte.

Weitere Stornowiederholungen

Falls zu der zweiten Stornowiederholung keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, beendet das Ladeterminal das Storno.

Falls bereits dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Stornoantwort der Ladezentrale eingetroffen ist, beendet das Ladeterminal das Storno und generiert eine Warnung für den Ladeterminalbetreiber. Andernfalls bestehen zwei Möglichkeiten:

Das Ladeterminal generiert es eine Warnung für den Ladeterminalbetreiber und ist bereit für die nächste Transaktion oder es speichert die Stornowiederholung und versucht zu einem späteren Zeitpunkt, die gespeicherte Stornowiederholung abzusetzen.

Hierbei wird die folgende Vorgehensweise empfohlen:

Bevor eine neue Transaktion begonnen wird, versucht das Terminal, die gespeicherte Stornowiederholung abzusetzen. Falls wiederum keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wird keine

neue Transaktion begonnen. Die Stornowiederholung wird weiterhin gespeichert, um sie später erneut abzusetzen. Eine Ausnahme bildet nur der Fall, daß das Ladeterminal dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Stornoantwort der Ladezentrale erhält. Dann beendet das Ladeterminal das Storno, generiert eine Warnung für den Ladeterminalbetreiber und ist bereit für eine neue Transaktion.

Auf diese Weise wird sichergestellt, daß maximal eine Stornowiederholung in dem Ladeterminal gespeichert und verwaltet werden muß.

Wenn während der Bearbeitung der Stornowiederholungen keine Eingaben möglich sind, zeigt das Terminal an

Bitte warten

4. Wenn das Ladeterminal zu der Stornoanfrage eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 0 oder 21 enthält, läßt sich das Ladeterminal mit dem Kommando GET CHALLENGE eine Zufallszahl RND7 von der GeldKarte geben.

   Läßt sich das Kommando GET CHALLENGE auch nach zweifacher Wiederholung, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE nicht erfolgreich durchführen, zeigt das Ladeterminal an

   Chipfehler, Rückerstattung des Ladebetrags

   Bitte Karte entnehmen

   Anschließend muß der Ladebetrag rückerstattet werden. In welcher Form dies geschieht, hängt von dem jeweiligen Zahlungsmittel ab.

5. Das Ladeterminal baut aus den Daten in BMP 62 der Stornoantwort die Kommandonachricht eines **Laden** mit Secure Messaging zum Laden des Ladebetrags 0 auf. Hierzu erzeugt und speichert es eine Zufallszahl RND8.

   Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E4’	CLA
2	1	‘30’	INS
3	1	'80'	P1 für **Laden** ohne Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘3C’	L_c
6	1	'12'	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'00..00'	Ladebetrag
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49-56	8	'XX..XX'	RND4
57	1	'XX'	Logische Schlüsselnummer KID des KGK_{LT}
58-65	8	'XX..XX'	CFB-MAC mit K_{LT} über die 64 Byte Byte 1-57|Byte 66|'00 00 00 00 00 00 00' mit ICV = RND3
66	1	‘12’	L_e

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht.

Es stellt Byte 1-5, Byte 49-57 und Byte 66 wie spezifiziert ein und berechnet den CFB-MAC mit K_{LT} über Byte 1-57, den es anschließend in Byte 58-65 einstellt. Als ICV zur MAC-Berechnung verwendet das Terminal RND7 der GeldKarte.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’13’	Nachrichten-ID für **Laden** mit Secure Messaging ohne Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'00..00'	geladener Betrag
8-10	3	'nn..nn'	neuer aktueller Betrag
11-18	8	'XX..XX'	CFB-MAC mit K_{LT} (KID aus Kommandonachricht) über Byte 1-10|'00 00 00 00 00 00' mit ICV = RND4

Das Ladeterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Ladeterminal gespeicherten Werten LSEQ und WZN überein?
- Ist der Ladebetrag 0?
- Ist der MAC korrekt?

Zur MAC-Prüfung verwendet das Terminal als ICV die gespeicherte Zufallszahl RND8.

Bei positivem Ausgang der Prüfungen zeigt das Ladeterminal an

Laden abgebrochen, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

In allen Fehlerfällen zeigt das Ladeterminal an

Chipfehler, Rückerstattung des Ladebetrags

Bitte Karte entnehmen

Anschließend muß der Ladebetrag rückerstattet werden. In welcher Form dies geschieht,

hängt von dem jeweiligen Zahlungsmittel ab.

4.3. Laden vom Kartenkonto

4.3.1. Rückbuchung des Ladebetrags auf das Kartenkonto

Bei dem Laden vom Kartenkonto wird der Ladebetrag von einem Konto des Karteninhabers abgebucht. Die Ladezentrale veranlaßt die Abbuchung des Ladebetrags vom Kartenkonto, wenn sie eine korrekte Ladeanfrage des Ladeterminals erhält, bevor sie die Ladeantwort an das Ladeterminal absendet.

Folgende Fälle sind bzgl. der Rückbuchung des Ladebetrags bzw. der Warnung des Karteninhabers zu unterscheiden:

1. Wenn am Ladeterminal feststeht, daß eine Abbuchung des Ladebetrags vom Kartenkonto nicht stattgefunden hat, muß durch das Ladeterminal keine Rückbuchung veranlaßt werden.

   Dies ist solange der Fall, bis in Schritt 8 des Ladens die Ladeanfrage an die Ladezentrale gesendet wird oder wenn in Schritt 8 des Ladens eine formal korrekte Ladeantwort mit fehlerfreiem MAC und negativem Antwortcode von der Ladezentrale eintrifft.

2. Wenn am Ladeterminal feststeht, daß der Ladebetrag in die GeldKarte geladen wurde, muß durch das Ladeterminal ebenfalls keine Rückbuchung veranlaßt werden.

   Dies ist dann der Fall, wenn die GeldKarte in Schritt 9 des Ladens bei Ausführung des Kommandos **Laden** oder in Schritt 1 des automatischen Stornos bei Ausführung des Kommandos **Ladedaten wiederholen** eine korrekte Antwortnachricht mit positivem Returncode zurückgibt.

3. In allen anderen Fällen muß das Ladeterminal die Rückbuchung des Ladebetrags veranlassen, indem es ein automatisches Storno einleitet, fortsetzt oder wiederholt.

   Der Karteninhaber ist hierbei wie folgt zu informieren:

   a) Wenn am Ladeterminal feststeht, daß das automatische Storno zu einer Rückbuchung geführt hat bzw. daß keine Abbuchung stattgefunden hat, wird der Vorgang ohne Warnung für den Karteninhaber beendet, auch wenn das eventuell anschließende **Laden** der GeldKarte mit Ladebetrag 0 nicht erfolgen kann.

      Das ist dann der Fall, wenn in Schritt 2 des automatischen Stornos

      - eine formal korrekte Stornoantwort mit fehlerfreiem MAC und positivem Antwortcode von der Ladezentrale eintrifft oder
      - eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 4, 5, 54, 56, 58, 62, 92 von der Ladezentrale eintrifft, wodurch anzeigt wird, daß die zugehörige Ladeanfrage des Ladeterminals durch die Ladezentrale entweder mit demselben Antwortcode abgewiesen oder nicht empfangen wurde, so daß

eine Abbuchung des Ladebetrags vom Kartenkonto in keinem Fall stattgefunden hat.

b) Wenn am Ladeterminal feststeht, daß das automatische Storno wiederholt wird, wird der Vorgang ebenfalls ohne Warnung für den Karteninhaber beendet.

Dies ist dann der Fall, wenn in Schritt 1 des automatischen Stornos die erforderlichen Daten von der GeldKarte erzeugt wurden, aber in Schritt 2 des automatischen Stornos einer der folgenden Fälle eintritt:

- die Stornoanfrage kann nicht an die Ladezentrale gesendet werden (Stornowiederholung durch das Ladeterminal),
- die Stornoantwort trifft innerhalb eines definierten Zeitraums nicht ein, ist formal nicht korrekt oder enthält einen fehlerhaften MAC (Stornowiederholung durch das Ladeterminal),
- die Stornoantwort ist fehlerfrei, enthält aber einen der Antwortcodes 79 (Stornowiederholung durch eine andere Komponente), 91 oder 96 (Stornowiederholung durch das Ladeterminal).

c) Wenn am Ladeterminal feststeht, daß das automatische Storno nicht zu einer Rückbuchung führt, obwohl eventuell eine Abbuchung des Ladebetrags vom Kartenkonto stattgefunden hat, und daß keine Stornowiederholung durchgeführt wird, wird der Vorgang mit einer Warnung für den Karteninhaber beendet. Hierdurch wird der Karteninhaber aufgefordert, sich zur Klärung an sein kartenausgebendes Institut zu wenden.

Dies ist dann der Fall, wenn das automatische Storno nicht durchführbar ist, weil die hierzu von der GeldKarte benötigten Daten nicht erhalten werden können:

- In Schritt 0 des automatischen Stornos ist ein Reset der GeldKarte oder die Selektion des DF_BÖRSE nicht möglich.
- In Schritt 1 des automatischen Stornos bricht das **Laden einleiten wiederholen** mit einem anderen Fehlerfall als Returncode '9F 11' bzw. '9F 15' ab oder das **Ladedaten wiederholen** führt zu einem Fehlerfall.

Dies ist auch der Fall, wenn das automatische Storno in Schritt 2 durch die Ladezentrale mit einer formal korrekten Stornoantwort und einem der Antwortcodes 30, 57, 64, 76, 77, 78, 97 oder einem für diesen Schritt nicht definierten Antwortcode abgelehnt wird.

4.3.2. Protokollierung

Es müssen mindestens die folgenden Daten einer Ladetransaktion zum Laden vom Kartenkonto protokolliert werden

- Ladeverfahren,
- Betreiber-BLZ und Terminal-ID,
- Trace-Nummer,
- Datum und Uhrzeit,
- Kartenidentifikationsdaten (Daten des EF_ID) der GeldKarte,
- Ladebetrag (ab erfolgreicher Beendung von Schritt 7 des Ladens),
- Sequenznummer LSEQ und Wiederholungszähler WZ bzw. WZN der GeldKarte, (ab erfolgreicher Beendung von Schritt 7. des Ladens bzw. Schritt 1. des Storno),
- aktueller Betrag der elektronischen Geldbörse zu Beginn der Transaktion (ab erfolgreicher Beendung von Schritt 7 des Ladens),
- Ergebnis der Transaktion mit Information darüber, in welchem Status die Transaktion abgebrochen oder erfolgreich beendet wurde.

4.3.3. Sicherheit

Das Laden vom Kartenkonto von kontobezogenen GeldKarten erfolgt ohne Secure Messaging zwischen Terminal und GeldKarte. Das Laden vom Kartenkonto soll aber nur möglich sein, wenn sich der Karteninhaber vor Ausführung des Kommandos gegenüber der GeldKarte bzw. der Applikation elektronische Geldbörse durch korrekte Angabe seiner PIN authentisiert hat. Auf diese Weise wird sichergestellt, daß niemand unbefugt Abbuchungen vom Kartenkonto veranlassen kann.

4.3.3.1. Sicherheitsanforderungen an den Ablauf der PIN-Prüfung

Zur PIN-Prüfung ist die PIN über die Tastatur des Ladeterminals einzugeben und von dort gesichert an die GeldKarte zu übermitteln ist. Hierbei darf die PIN den gesicherten Bereich der Tastatur nicht im Klartext verlassen. Hierzu benötigt das Ladeterminal eine Sicherheitstastatur. In Kapitel 4.3.4. wird sowohl die verschlüsselte als auch die unverschlüsselte PIN-Übertragung beschrieben. Die für in Abhängigkeit vom jeweiligen PIN-Übertragungsverfahren jeweils zu erfüllenden Kriterien sind Kapitel 2.2.1. und [LIT K] zu entnehmen.

Wenn die PIN verschlüsselt an die kontobezogene GeldKarte übertragen werden soll, müssen sich das Ladeterminal und die Karte zunächst gegenseitig authentisieren. Hierzu benötigt das Ladeterminal den 16 Byte langen Masterkey KGK_{INFO}. Aus diesem Masterkey und den Daten des EF_ID der GeldKarte wird der 8 Byte lange kartenindividuelle Schlüssel K_{INFO} abgeleitet.

Zur verschlüsselten PIN-Übertragung wird das Kommando VERIFY mit Secure Messaging

(MAC-Bildung und Verschlüsselung) gemäß Kapitel 3.2 in [LIT 1] durchgeführt. Hierzu benötigt das Ladeterminal den 16 Byte langen Masterkey KGK_{PIN}. Aus diesem Masterkey und den Daten des EF_ID der GeldKarte wird der 8 Byte lange kartenindividuelle Schlüssel K_{PIN} abgeleitet, mit dem MAC-Bildung und Verschlüsselung durchgeführt werden.

Das Verfahren zur Schlüsselableitung ist in Kapitel 2.5 von [LIT 1] beschrieben.

Die Verschlüsselung ist in Kapitel 2.3 von [LIT 1] erläutert.

Das Verfahren zur MAC-Berechnung und Verschlüsselung des Secure Messaging (CFB-MAC) ist in Kapitel 2.4 von [LIT 1] beschrieben.

Es ist daran gedacht, in Zukunft auch mehrere Generationen von KGK_{INFO} und KGK_{PIN} in einem Ladeterminal-Sicherheitsmodul und doppelt lange K_{INFO} und K_{LT} zur Berechnung von Retail-CFB-MACs zu verwenden.

Der PIN-Prüfungsprozeß mit verschlüsselter Übertragung der PIN ist in den Schritten 0. bis 6. von Kapitel 4.3.4. spezifiziert. Im folgenden wird erläutert, welche Anforderungen an die Ablaufsicherung des PIN-Prüfungsprozesses gestellt werden. Es wird dargelegt, welche Risiken entstehen, wenn diese Anforderungen nicht erfüllt werden. Bei der Darstellung wird davon ausgegangen, daß die Ablaufsicherung in der Sicherheitstastatur realisiert ist.

Handhabung der Zufallszahl RND1

Es muß sichergestellt werden, daß die in Schritt 3. des Ladens vom Kartenkonto ausgegebene Zufallszahl durch die Sicherheitstastatur neu erzeugt und sicher gespeichert wird und daß diese gespeicherte Zufallszahl als ICV bei der MAC-Berechnung mit K_{INFO} in Schritt 4. des Ladens verwendet wird.

Würde RND1 außerhalb des gesicherten Bereichs erzeugt und gespeichert und nur zur MAC-Prüfung an die Sicherheitstastatur übergeben, würde keine Authentikation der Chipkarte gegenüber der Sicherheitstastatur stattfinden, da in Schritt 4. das Einspielen alter MACs und alter Zufallszahlen möglich wäre.

Es ist für die Sicherheit des Ablaufs relevant, daß eine echte Authentikation der Chipkarte gegenüber der Sicherheitstastatur stattfindet, da andernfalls PIN-Ausforschen durch Probieren ohne Besitz der Karte möglich wäre.

Handhabung des EF_ID der GeldKarte

In Schritt 4. muß das bei einer erfolgreichen MAC-Prüfung authentisierte EF_ID der beteiligten GeldKarte in der Sicherheitstastatur gespeichert werden. In Schritt 5. muß dann der zur Verschlüsselung des VERIFY-Kommandos benötigte K_{PIN} aus KGK_{PIN} und dem in Schritt 3. gespeicherten EF_ID abgeleitet werden.

Eine erneute Übergabe des EF_ID an die Sicherheitstastatur in Schritt 5. würde die Möglichkeit eröffnen, sich VERIFY-Kommandos zu beliebigen EF_IDs berechnen zu lassen.

Die Verwendung desselben EF_ID in Schritt 3. und Schritt 4. läßt sich auch dadurch erreichen, daß

bereits in Schritt 1. das EF_ID an die Sicherheitstastatur übergeben wird und sowohl K_{INFO} als auch K_{PIN} aus dem EF_ID vorberechnet und für die Dauer des Prozesses "PIN-Prüfung" gespeichert werden.

Ausgabe der PIN aus der Sicherheitstastatur

Es muß sichergestellt werden, daß die PIN nur wie spezifiziert verschlüsselt in einem VERIFY-Kommando in Schritt 6. des Ladeablaufs nach Durchlaufen der übrigen spezifizierten Schritte erfolgt.

Nur auf diese Weise bietet die korrekte Handhabung von RND1 und EF_ID den angestrebten Schutz.

4.3.3.2. MAC-Sicherung von Online-Nachrichten

Beim Laden vom Kartenkonto erfolgt die MAC-Sicherung der Online-Nachrichten in der Regel auf der Strecke zwischen Ladeterminal-Sicherheitsmodul und Ladezentrale. Auf dieser Strecke darf durch Sicherheitskomponenten umgeschlüsselt werden. Das Verfahren zur MAC-Sicherung der Online-Nachrichten ist in Kapitel 3.4. beschrieben.

MAC-Sicherung der ISO-Nachrichten am Ladeterminal

1. Hardware-Sicherheitsmodul

Wenn der K_T-MAC über die ISO-Nachrichten zum Laden vom Kartenkonto am Ladeterminal berechnet wird, muß das Ladeterminal ein Hardware-Sicherheitsmodul besitzen. Durch Bauart der Hardware und Sicherheitsmechanismen der Software des Sicherheitsmoduls muß gewährleistet werden, daß die verwendeten Schlüssel gegen Auslesen geschützt sind und daß kryptographische Operationen nur in gegen Auslesen und unberechtigte Veränderung geschützten Bereichen durchgeführt werden.

Im allgemeinen kommt als Hardware-Sicherheitsmodul die Sicherheitstastatur zum Einsatz, die bereits für die verschlüsselte PIN-Übertragung an die GeldKarte benötigt wird.

2. Absicherung der MAC-Bildung

Im Rahmen des Ladens vom Kartenkonto dürfen ausschließlich K_T-MACs über die hierfür vorgesehenen ISO-Nachrichten berechnet werden. Diese sind vor der MAC-Bildung durch die Sicherheitstastatur anhand von Nachrichtentyp und Abwicklungskennzeichen (190000) zu identifizieren. Andernfalls kann die Sicherheitstastatur mißbraucht werden, um K_T-MACs über Nachrichten zum Laden gegen andere Zahlungsmittel zu berechnen. Die hieraus entstehenden Risiken sind in Kapitel 4.2.3. erläutert.

3. Trace-Nummer, Schlüsselindex und Terminal-ID

Trace-Nummer und Schlüsselindex sollten in der Sicherheitstastatur geführt und vor jeder Transaktion inkrementiert werden. Die Terminal-ID sollte in der Sicherheitstastatur gespeichert sein. Bei MAC-Bildung und MAC-Prüfung durch die Sicherheitstastatur sollten diese Werte in der Sicherheitstastatur auf Korrektheit geprüft werden.

Diese Anforderung muß erfüllt werden, wenn das Ladeterminal gleichzeitig das Laden gegen andere Zahlungsmittel unterstützt, da diese Werte dann aufgrund der Sicherheitsanforderungen an die Ablaufkontrolle des Ladens gegen andere Zahlungsmittel durch die Sicherheitstastatur kontrolliert werden müssen (vgl. Kapitel 4.2.3.).

4. Terminalkontodaten und Ladeentgelt

Falls Kontodaten und Ladeentgelt durch das Terminal in die Nachricht eingestellt werden und falls diese Daten später durch die nachgelagerte Komponente des Ladeterminalbetreibers nicht mehr geprüft werden, müssen diese Daten durch die Sicherheitstastatur zumindest geprüft werden, damit Mißbrauch ausgeschlossen ist.

Hierzu müssen Kontodaten und zulässiges Ladeentgelt in der Sicherheitstastatur gespeichert werden.

Verwendung eines internen Nachrichtenformates

Es ist zulässig, daß am Ladeterminal nicht die kompletten, von der Ladezentrale erwarteten ISO-Nachrichten erzeugt werden. Nachrichtenfelder in den ISO-Anfragenachrichten können durch die nachgelagerte Komponente des Ladeterminalbetreibers hinzugefügt und die entsprechenden Nachrichtenfelder in den Antwortnachrichten durch die nachgelagerte Komponente überprüft werden. In Kapitel 3.2. ist festgelegt, welche Daten in internen Nachrichten enthalten sein müssen, und wie diese Nachrichten in der nachgelagerten Komponente des Ladeterminalbetreibers in ISO-Nachrichten umzusetzen sind. Die nachgelagerte Komponente muß in diesem Fall zur MAC-Bildung und eventuell zur MAC-Prüfung ein Sicherheitsmodul besitzen.

Für das Laden vom Kartenkonto ist es insbesondere nicht notwendig, daß am Terminal Nachrichten-MACs berechnet und geprüft werden, da prinzipiell nur die Kommunikation zwischen Chipkarte und Ladezentrale abzusichern ist. Verschlüsselungsparameter und MAC können daher in den internen Nachrichten zwischen Ladeterminal und nachgelagerter Komponente fehlen.

Wenn die Nachrichten zum Laden vom Kartenkonto zwischen Ladeterminal und nachgelagerte Komponente ohne MAC übertragen werden und das Hintergundsystem des Ladeterminalbetreibers die Berechnung und Prüfung der K_T-MACs über die ISO-Nachrichten an die Ladezentrale übernimmt, müssen die für die Absicherung von ISO-Nachrichten am Ladeterminal formulierten Anforderungen an die Ablaufsicherheit durch die nachgelagerte Komponente und dessen Sicherheitsmodul erfüllt werden.

Wenn die Nachrichten zum Laden vom Kartenkonto zwischen Ladeterminal und nachgelagerter Komponente MAC-gesichert übertragen werden, muß durch die Ablaufsicherung der

nachgelagerten Komponente manipulationssicher gewährleistet werden, daß MACs nur im Rahmen der gemäß Kapitel 3.2. korrekten Umsetzung berechnet werden, so daß das Sicherheitsmodul der nachgelagerten Komponente nicht mißbraucht werden kann, um MACs über beliebige Nachrichten zu berechnen.

Wenn die nachgelagerte Komponente wie beschrieben arbeitet und die Nachrichten zwischen Ladeterminal und nachgelagerter Komponente mit einem MAC versehen werden, muß die Ablaufsicherung am Ladeterminal die für die oben erläuterten Anforderungen an die Absicherung von ISO-Nachrichten in analoger Weise erfüllen. Es entfällt eventuell das Einstellen und Prüfen der Terminalkontodaten und des Ladeentgelts. Dies muß dann die nachgelagerte Komponente übernehmen.

4.3.4. Laden

Das folgenden Diagramm zeigt die Schritte des Ladens vom Kartenkonto, die nachfolgend erläutert werden.

GeldKarte			Ladeterminal		Ladezentrale	
			A0	Transaktion beginnen		
		<---	C1	GET CHALLENGE		
R1	RND0	--->				
			A2	RND0 mit K_{INFO} zu ENCRND verschlüsseln		
			C2	EXTERNAL AUTHENTICATE mit ENCRND		
		<---				
R2	OK	--->				
			A3	RND1 erzeugen und speichern		
		<---	C3	PUT DATA RND1		
R3	OK	--->				
			C4	READ RECORD EF_INFO mit Secure Messaging		
		<---				
R4	Daten aus EF_INFO mit K_{INFO}-MAC, ICV = RND1	--->				
		<---	C5	GET CHALLENGE		
R5	RND2	--->				
			A6	PIN-Eingabe anfordern, PIN formatieren		
			C6	VERIFY formatierte PIN evtl. Secure Messaging mit K_{PIN}, ICV = RND2		
		<---				
R6	OK	--->				

GeldKarte			Ladeterminal			Ladezentrale	
			C7	**Laden einleiten** oder **Laden einleiten wiederholen** ohne Secure Messaging			
		<---					
R7	Antwortdaten des **Laden einleiten (wiederholen)** ohne Secure Messaging	--->					
			A7	Antwortdaten prüfen			
			A8	K_T ableiten			
			C8	Ladeanfrage 0200 Abwkz. 190000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R8	Ladeantwort 0210 Abwkz. 190000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A8	Ladeantwort und K_T-MAC prüfen			
			C9	**Laden** ohne Secure Messaging			
		<---					
R9	Antwortdaten des **Laden** ohne Secure Messaging	--->					
			A9	Antwortdaten prüfen Anzeige: geladener Betrag, neuer aktueller Betrag, eventuell: neue Maximalbeträge,			

Erläuterung

0. Nachdem Ladeverfahren und Ladebetrag feststehen, beginnt das Ladeterminal eine neue Transaktion. Hierbei muß sichergestellt sein, daß

   - Trace-Nummer TSEQ und

- Schlüsselindex SI

gegenüber der letzten Transaktion inkrementiert sind.

Hierbei wird die Trace-Nummer nach Erreichen des maximal möglichen Wertes auf 0 zurückgesetzt.

Hat der Schlüsselindex den Maximalwert erreicht, wird das Terminal für den weiteren Betrieb als Ladeterminal gesperrt.

In allen Schritten der Transaktion müssen durch das Ladeterminal dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet werden.

1.-6. Die folgenden Schritte dienen der PIN-Eingabe und -Prüfung.

Die Schritte 1.-5. müssen nur ausgeführt werden, wenn das Terminal die ec-PIN verschlüsselt an die GeldKarte übergibt. Andernfalls kann das Terminal mit Schritt 6. fortfahren.

Für die verschlüsselte Übergabe der ec-PIN an die GeldKarte benötigt das Terminal die Kartendaten aus dem EF_INFO, um den Format 0 PIN-Block aufzubauen. Diese Daten werden in Schritt 4. MAC-gesichert gelesen. Auf diese Weise authentisiert sich die GeldKarte gegenüber dem Terminal. Als ICV für die MAC-Bildung wird der GeldKarte in Schritt 3. die Zufallszahl RND1 übergeben.

Da das EF_INFO nur gelesen werden kann, wenn sich das Terminal vorher mit dem Schlüssel K_{INFO} gegenüber der GeldKarte authentisiert hat, müssen die Schritte 1. und 2. ausgeführt werden.

1. Mit dem Kommando GET CHALLENGE läßt sich das Ladeterminal eine Zufallszahl RND0 von der GeldKarte geben. Die folgende Tabelle zeigt den Aufbau der Command-APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'84'	INS
3	1	'00'	P1
4	1	'00'	P2
5	1	'08'	L_e

Wenn das Kommando erfolgreich ausgeführt wurde, gibt die GeldKarte eine 8 Byte lange

Zufallszahl RND0 als Antwortdatum aus.

Im Fehlerfall wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

2. Mittels der Daten des EF_ID leitet das Terminal aus dem Masterkey KGK_{INFO} den 8 Byte langen kartenindividuellen Schlüssel K_{INFO} ab. Es verschlüsselt RND0 mittels des K_{INFO} zu ENCRND.

   Das Terminal ruft das Kommando EXTERNAL AUTHENTICATE mit der folgenden Kommandonachricht auf:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'00'	CLA ohne Secure Messaging
2	1	'82'	INS
3	1	'00'	P1, fester Wert
4	1	'02'	P2, Schlüsselnummer des K_{INFO} im EF_KEY des MF
5	1	'08'	L_C
6-13	8	'XX..XX'	ENCRND

   Wenn das Kommando erfolgreich ausgeführt wurde, kann das Terminal anschließend das EF_INFO lesen.

   Im Fehlerfall wird mit der Meldung abgebrochen

   Chipfehler, bitte Karte entnehmen

3. Das Terminal erzeugt eine Zufallszahl RND1 und speichert sie sicher. Es übergibt RND1 mittels PUT DATA an die GeldKarte. Die folgende Tabelle zeigt die Command-APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘00’	CLA ohne Secure Messaging
2	1	‘DA’	INS
3	1	'01'	P1
4	1	‘00’	P2
5	1	‘08’	L_c
6-13	8	'XX..XX'	RND1

Im Fehlerfall wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

4. Mittels READ RECORD mit Secure Messaging liest das Terminal die kontospezifischen Daten im Record ’01’ des EF_INFO (SFI ’16’) MAC-gesichert. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	’04’	CLA mit Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’B4’	P2, Reference Control Byte
5	1	’19’	L_e, Recordlänge des EF_INFO + 8 Byte MAC

Wenn das READ RECORD erfolgreich ausgeführt wurde, gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1-5	5	'nn..nn'	Kundenkontonummer
6	1	'nD'	Prüfziffer Spur 2
7-8	2	'nn nD'	Service-Code
9-10	2	'0n nn'	Kartenfolgenummer
11-14	4	'nn..nn'	Bankleitzahl kontoführendes Institut
15	1	'nD'	Prüfziffer Spur 3
16	1	'0n'	Freizügigkeitsschlüssel
17	1	'nn'	Kontoart und Benutzungseinschränkung
18-25	8	'XX..XX'	CFB-MAC mit K_{INFO} über die 24 Byte Byte 1-17|'00 00 00 00 00 00 00' mit ICV = RND1

Das Terminal prüft den MAC in Byte 18-25 der Antwortdaten.

Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

Chipfehler, bitte Karte entnehmen

5. Mittels des Kommandos GET CHALLENGE fordert das Terminal eine 8 Byte lange Zufallszahl RND2 von der GeldKarte an.

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Chipfehler, bitte Karte entnehmen

6. Das Terminal fordert den Karteninhaber auf

   Bitte Geheimzahl eingeben: . . . .

   Bitte bestätigen

   Die PIN wird verdeckt eingegeben.

Wenn die ec-PIN **verschlüsselt** an die GeldKarte übergeben wird, sind folgende Schritte auszuführen:

- Mittels der Daten des EF_ID leitet das Terminal aus dem Masterkey KGK_{PIN} den 8 Byte

langen kartenindividuellen Schlüssel K_{PIN} ab.

- Das Terminal formatiert die eingegebene PIN mittels der Daten des EF_INFO zum Format 0 PIN-Block FPIN0. Diesen verschlüsselt es mittels des Schlüssels K_{PS} zum Encrypted Format 0 PIN Block EPIN (vgl. Kapitel 6.6.3.2. in [LIT 1]).

- Das Terminal baut eine Kommandonachricht für das Kommando VERIFY auf:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'04'	CLA mit Secure Messaging
2	1	'20'	INS
3	1	'00'	P1, fester Wert
4	1	'00'	P2, die PIN ist im EF_PWD0 des MF zu suchen
5	1	'10'	L_c
6-13	8	'XX..XX'	EPIN
14-21	8	'XX..XX'	CFB-MAC mit K_{PIN} über die 16 Byte Byte 1-13|'00 00 00' ICV = RND2

- Das Terminal paddet Byte 5-21 der Kommandonachricht mit den 7 Byte '80 00 00 00 00 00 00' auf eine Länge von 24 Byte und verschlüsselt diese mit K_{PIN} und ICV = RND2 zu dem 24 Byte langen Wert ENCCMD.

- Das Terminal schickt die folgende Nachricht an die GeldKarte:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'04'	CLA mit Secure Messaging
2	1	'20'	INS
3	1	'00'	P1, fester Wert
4	1	'00'	P2, die PIN ist im EF_PWD0 des MF zu suchen
5-28	24	'XX..XX'	ENCCMD

Wenn die ec-PIN **unverschlüsselt** an die GeldKarte übergeben wird, sind folgende Schritte auszuführen:

- Das Terminal formatiert die eingegeben PIN zum Format 2 PIN-Block FPIN2 (vgl.

Kapitel 1.7.2 in [LIT 4A]).

- Das Terminal sendet die folgende Kommandonachricht des Kommandos VERIFY an die GeldKarte:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'00'	CLA ohne Secure Messaging
2	1	'20'	INS
3	1	'00'	P1, fester Wert
4	1	'80'	P2, die PIN ist im DF-spezifischen EF_PWD0 zu suchen
5	1	'08'	L_c
6-13	8	'XX..XX'	FPIN2

In beiden Fällen führt die GeldKarte die PIN-Prüfung durch und setzt das Flag des entsprechenden Sicherheitszustands, wenn die PIN-Prüfung erfolgreich war. Andernfalls wird der PIN-Fehlbedienungszähler dekrementiert.

Durch den Returncode des Kommandos VERIFY teilt die GeldKarte dem Terminal mit, ob die Prüfung erfolgreich war, bzw. wie viele Versuche noch möglich sind.

Erhält das Terminal den Returncode '63 CX' mit X = 1 oder 2, zeigt es an:

Geheimzahl falsch

Bitte Geheimzahl erneut eingeben: . . . .

Bitte bestätigen

wobei mindestens die beiden letzten Zeilen zusammen angezeigt werden müssen.

Erhält das Terminal den Returncode '63 C0' oder '69 83', bricht es mit der Meldung ab

Geheimzahl zu oft falsch, bitte Karte entnehmen

In allen übrigen Fehlerfällen bricht das Terminal mit der folgenden Anzeige ab:

Chipfehler, bitte Karte entnehmen

7. In Abhängigkeit vom Wert des Statusbyte der GeldKarte wird die Kommandonachricht für das **Laden einleiten** oder das **Laden einleiten wiederholen** ohne Secure Messaging aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘30’	INS
3	1	'00' '20'	P1 für **Laden einleiten** P1 für **Laden einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1C’	L_c
6	1	'00' '04'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
7-9	3	'00..00'	Filler
10-12	3	'nn..nn'	Ladebetrag
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	‘3A’	L_e

Der Ladebetrag muß der vom Karteninhaber bestätigte Betrag sein.

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück.

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'01' '05'	Nachrichten-ID für **Laden einleiten** Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'

Das Ladeterminal prüft, ob Nachrichten-ID und Ladebetrag die korrekten Werte haben.

In allen Fehlerfällen wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

Bei erfolgreicher Kommandoausführung werden die Sequenznummer LSEQ und der Wiederholungszähler WZ gespeichert.

Das Ladeterminal zeigt an:

Ladebetrag DM: . . . . , Laden wird bearbeitet

8. Eine Ladeanfrage mit Nachrichtentyp 0200 und Abwicklungskennzeichen 190000 an die Ladezentrale wird aufgebaut. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

   Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

   Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

   In BMP 62 der Ladeanfrage stellt das Ladeterminal das EF_ID der GeldKarte und die geprüften Antwortdaten der GeldKarte zu **Laden einleiten (wiederholen)** ein.

   Der Transaktionsschlüssel K_T wird wie in Kapitel 3. spezifiziert aus dem K_{MAC} unter Verwendung des Schlüsselindex SI abgeleitet. Die Ladeanfrage wird mit dem K_T MAC-gesichert.

   Das Ladeterminal schickt die Ladeanfrage an die Ladezentrale.

   Wenn es nicht möglich ist, die Ladeanfrage an die Ladezentrale zu senden, kann an dieser Stelle mit der Anzeige

   Systemfehler, bitte Karte entnehmen

   abgebrochen werden. Alternativ wird so verfahren, als ob die Ladeantwort der Ladezentrale nicht innerhalb eines definierten Zeitraums eingetroffen wäre.

   Wenn die Ladeantwort der Ladezentrale nicht innerhalb eines definierten Zeitraums eintrifft, beginnt das Ladeterminal ein automatisches Storno, dessen Ablauf in Kapitel 4.3.5. beschrieben ist.

   Wenn das Ladeterminal eine Antwortnachricht erhält, prüft es, mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

   Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

Bei Antwortcode 13 muß der Ladebetrag in Byte 27-29 der BMP 62 immer den Wert 0 haben.

Stellt das Terminal hierbei Fehler fest, beginnt es ein automatisches Storno.

Wenn die Antwortnachricht formal korrekt ist und einen fehlerfreien MAC aber einen negativen Antwortcode (≠ 0, 13) enthält, bricht das Ladeterminal mit der folgenden Meldung ab:

<Text>, bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 58, 76, 91, 92, 96, 97, 98 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler

9. Wenn die Antwortnachricht formal korrekt ist, einen fehlerfreien MAC und den Antwortcode 0 oder 13 enthält, baut das Ladeterminal aus den Daten in BMP 62 der Antwortnachricht die Kommandonachricht eines **Laden** ohne Secure Messaging auf.

   Es wertet den Konditionscode in BMP 25 aus, um festzustellen, ob das Laden mit oder ohne Änderung der Maximalbeträge erfolgen soll. Hat der Konditionscode den Wert 98, ist P1 = 'A0' zu setzen, hat der Konditionscode den Wert 00, ist P1 = '80' zu setzen.

   Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘30’	INS
3	1	'80' 'A0'	P1 für **Laden** ohne Änderung der Maximalbeträge P1 für **Laden** mit Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'10' '14'	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'nn..nn'	Ladebetrag bei Antwortcode 13: '00 00 00'
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht. Es stellt Byte 1-5 und Byte 49 wie spezifiziert ein

Das Terminal speichert die verwendete Nachrichten-ID.

Bei Konditionscode 98 prüft das Terminal, ob beide Maximalbeträge oder nur ein Maximalbetrag geändert wurden. Dazu vergleicht es den neuen Maximalbetrag aus Byte 34-36 der Command-APDU mit dem gespeicherten Maximalbetrag der GeldKarte und den maximalen Transaktionsbetrag aus Byte 37-39 der Command-APDU mit dem gespeicherten maximalen Transaktionsbetrag.

Das Terminal speichert den geänderten Betrag bzw. die geänderten Beträge.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’11’ ’15’	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge mit Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'nn..nn'	geladener Betrag '00 00 00' bei Antwortcode 13
8-10	3	'nn..nn'	neuer aktueller Betrag

Das Ladeterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Ladeterminal gespeicherten Werten überein?
- Ist der Ladebetrag korrekt?

Wenn die GeldKarte bei Ausführung des **Laden** einen negativen Returncode zurückgibt oder wenn die Prüfung der Antwortdaten des **Laden** einen Fehler ergibt, beginnt das Ladeterminal ein automatisches Storno.

In allen anderen Fehlerfällen bei Ausführung des **Laden**, versucht das Ladeterminal, das **Laden** erneut auszuführen. Wenn die GeldKarte das Kommando dann erfolgreich bearbeitet, wird mit den Antwortdaten wie oben beschrieben verfahren. Andernfalls leitet das Ladeterminal ein automatisches Storno ein.

Wenn der Fehlerfall bei Ausführung des **Laden** zu einem Reset der GeldKarte geführt hat, kann auf die erneute Ausführung des **Laden** verzichtet werden und sofort mit dem automatischen Storno begonnen werden.

Bei erfolgreicher Ausführung des **Laden** und positiver Prüfung der Antwortdaten, wird gemäß dem Antwortcode der Ladeantwort wie folgt verfahren:

Antwortcode 13

War der Antwortcode 13 in der Ladeantwort enthalten, bricht das Ladeterminal in jedem Fehlerfall bei der Ausführung des **Laden** mit der folgenden Meldung ab

Chipfehler, bitte Karte entnehmen

Verläuft die Prüfung der Antwortdaten des **Laden** positiv, entnimmt das Ladeterminal BMP 58 der Antwortnachricht den zum Laden zur Verfügung stehenden Restbetrag.

Wenn der Restbetrag 0 ist, wird das Laden mit der folgenden Anzeige abgebrochen:

Laden nicht möglich, bitte Karte entnehmen

Wenn ein von 0 verschiedener Restbetrag zur Verfügung steht, wird durch das Ladeterminal der nun mögliche <maximale Ladebetrag> bestimmt:

Wenn der Maximalbetrag bei dem vorhergehenden **Laden** nicht geändert oder erhöht wurde, ist <maximaler Ladebetrag> = <Restbetrag>.

Wenn der Maximalbetrag durch das **Laden** verringert wurde, wird zunächst die Differenz zwischen neuem Maximalbetrag und aktuellem Betrag gebildet. Ist diese Differenz 0, wird das Laden mit der folgenden Anzeige abgebrochen:

Laden nicht möglich, bitte Karte entnehmen

Ist die Differenz kleiner als der Restbetrag, ist die Differenz der <maximale Ladebetrag>. Andernfalls ist <maximaler Ladebetrag> = <Restbetrag>.

Das Ladeterminal zeigt an

Ladebetrag DM: <Ladebetrag> nicht möglich

Möglicher Ladebetrag DM: <maximaler Ladebetrag>

Zum Laden bitte bestätigen

Optional ist eine erneute Eingabe des Ladebetrags möglich:

Ladebetrag DM: <Ladebetrag> nicht möglich

Möglicher Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Hierbei müssen mindestens die beiden letzten Zeilen zusammen angezeigt werden.

Aus Transparenzgründen können nur volle DM-Beträge eingegeben werden. Es sind maximal 4-stellige volle DM-Beträge eingebbar.

Die Eingabe des Ladebetrags erfolgt mittels Zifferntasten, optional zusätzlich mittels Tasten für vollständige Beträge.

Wenn der eingegebene Ladebetrag größer als <maximaler Ladebetrag> ist, zeigt das Ladeterminal an:

Ladebetrag zu hoch

Möglicher Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Wenn der Ladebetrag 0 eingegeben wird, zeigt das Ladeterminal an:

Ladebetrag 0 nicht möglich

Möglicher Ladebetrag DM: <maximaler Ladebetrag>

Ladebetrag DM: . . . . , bitte bestätigen

Hierbei müssen jeweils mindestens die beiden letzten Zeilen zusammen angezeigt werden.

Nach maximal drei Fehlversuchen der Betragseingabe wird das Laden abgebrochen:

Laden abgebrochen, bitte Karte entnehmen

Bei erfolgreicher Betragseingabe wird der Ladebetrag gespeichert und eine neue Transaktion begonnen. Hierbei muß sichergestellt sein, daß

- Trace-Nummer TSEQ und
- Schlüsselindex SI

gegenüber der letzten Transaktion inkrementiert sind.

Die Trace-Nummer wird nach Erreichen des maximal möglichen Wertes auf 0

zurückgesetzt.

Hat der Schlüsselindex den Maximalwert erreicht, wird das Terminal für den weiteren Betrieb als Ladeterminal gesperrt.

In allen Schritten der neuen Transaktion müssen durch das Ladeterminal dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet werden.

Eine erneute PIN-Eingabe ist nicht erforderlich, so daß anschließend mit Schritt 7 des Ladens vom Kartenkonto fortgefahren wird, wobei das **Laden einleiten** zu verwenden ist.

Antwortcode 0

Die folgenden Schritte werden ausgeführt, wenn der Antwortcode 0 in der Ladeantwort enthalten war:

Wenn die Prüfungen der Antwortdaten positiv ausfallen und die Maximalbeträge nicht geändert wurden, entnimmt das Ladeterminal den neuen aktuellen Betrag der Antwortnachricht und zeigt an

Ladebetrag DM: <Ladebetrag> geladen

Neues Guthaben DM: <neuer aktueller Betrag>

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

Wenn die Prüfungen positiv ausfallen und beide Maximalbeträge geändert wurden, zeigt das Ladeterminal an

Ladebetrag DM: <Ladebetrag> geladen

Neues Guthaben DM: <neuer aktueller Betrag>

Neuer Maximalbetrag DM: <neuer Maximalbetrag>

Maximaler Zahlungsbetrag DM: <neuer maximaler Transaktionsbetrag>

Bitte Karte entnehmen

Hierbei müssen mindestens die beiden ersten Zeilen zusammen angezeigt werden.

Wenn einer der beiden Maximalbeträge nicht geändert wurde, wird dieser nicht angezeigt. Ein geänderter Betrag kann zusammen mit der Meldung "Bitte Karte entnehmen" angezeigt werden.

Für die GeldKarte und das Ladeterminal ist die Transaktion damit beendet.

4.3.5. Automatisches Storno

Ein automatisches Storno wird dann durchgeführt, wenn in Schritt 8 des Ladens eine Ladeanfrage an die Ladezentrale geschickt wurde, aber keine oder eine fehlerhafte Ladeantwort eintrifft oder eine korrekte Ladeantwort mit Antwortcode 0 eintrifft, aber in Schritt 9 der Ladebetrag nicht mittels **Laden** in die GeldKarte geladen werden kann.

Sollte in Schritt 1 eines automatischen Storno der Nachweis über das erfolgreiche Laden nachträglich erzeugt werden, wird das automatische Storno abgebrochen.

Bei der erfolgreichen Durchführung eines automatischen Storno wird in die GeldKarte der Betrag 0 geladen, wodurch sichergestellt ist, daß

- der Ladebetrag der Ladetransaktion nicht mehr in die GeldKarte geladen werden kann (nach erfolgreicher Beendung des Schritt 1),

- der Ladebetrag nicht vom Kartenkonto abgebucht wird bzw. daß der Ladebetrag auf das Kartenkonto rückgebucht wird (nach erfolgreicher Beendung von Schritt 2)

- die Transaktion für die GeldKarte mit Heraufsetzen der Sequenznummer LSEQ abgeschlossen wurde (nach erfolgreicher Beendung von Schritt 3).

Der Schritt 3 wird nur durchgeführt, wenn das Ladeterminal in Schritt 2 eine positive Stornoantwort (Antwortcode 0 oder 21) auf eine Stornoanfrage vom Typ 0400 erhält.

Das automatische Storno gehört zu derselben Transaktion wie das zu stornierende Laden. Es werden daher für das automatische Storno dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet wie für das zugehörige Laden.

In dem folgenden Diagramm werden die Schritte des Storno eines Ladens vom Kartenkonto dargestellt, die nachfolgend näher erläutert werden.

GeldKarte			Ladeterminal			Ladezentrale	
			A0	Storno beginnen			
			C1	**Laden einleiten wiederholen** mit Betrag 0 ohne Secure Messaging			
R1	Antwortdaten des **Laden einleiten wiederholen** mit Betrag 0 ohne Secure Messaging	<--- --->	A1	Antwortdaten prüfen			
			C2	Stornoanfrage 0400 eventuell Storno-wiederholung 0401 Abwkz. 190000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R2	Stornoantwort 0410 Abwkz. 190000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
			A2	Stornoantwort und K_T-MAC prüfen eventuell Anzeige: **Laden abgebrochen**	<---		
			C3	**Laden** mit Betrag 0 ohne Secure Messaging			
R3	Antwortdaten des **Laden** mit Betrag 0 ohne Secure Messaging	<--- --->	A3	Antwortdaten prüfen Anzeige: **Laden abgebrochen**			

Erläuterung

0. Zu Beginn eines automatischen Storno führt das Ladeterminal ein Reset der GeldKarte durch und selektiert das DF_BÖRSE erneut (vgl. Schritt 2. und 3. in Kapitel 4.1.). Im Fehlerfall wird hierbei mit der folgenden Anzeige abgebrochen

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

1. Die Kommandonachricht für das **Laden einleiten wiederholen** mit Ladebetrag 0 und ohne Secure Messaging wird aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'E0'	CLA
2	1	'30'	INS
3	1	'20'	P1 für **Laden einleiten wiederholen**
4	1	'00'	P2, fester Wert
5	1	'1C'	L_c
6	1	'04'	Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
7-9	3	'00..00'	Filler
10-12	3	'00..00'	Ladebetrag
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	'3A'	L_e

Der Ladebetrag muß 0 sein.

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'05'	Nachrichten-ID für **Laden einleiten wiederholen** ohne Secure Messaging
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'00..00'	Ladebetrag
8-10	3	'nn..nn'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'

Das Ladeterminal prüft, ob die Nachrichten-ID den korrekten Wert hat, und ob der Ladebetrag 0 ist.

Seien

LSEQN die Sequenznummer aus Byte 2-3,

WZN der Wiederholungszähler aus Byte 4

der Antwortdaten und

LSEQ die vom Ladeterminal gespeicherte Sequenznummer,

WZ der vom Ladeterminal gespeicherte Wiederholungszähler.

Wenn gilt

LSEQN = LSEQ und

WZN > WZ

sind die Antwortdaten des **Laden einleiten wiederholen** der für den Aufbau einer Stornoanfrage benötigte Nachweis dafür, daß das Laden nicht erfolgt ist und nicht mehr erfolgen kann.

Der neue Wiederholungszähler WZN wird gespeichert, und es wird mit Schritt 2. des Storno fortgefahren.

Gibt die GeldKarte als Antwort zu **Laden einleiten wiederholen** den Returncode '9F 11' bzw. '9F 15' zurück, wurde das **Laden** eventuell zuvor korrekt bearbeitet.

In diesem Fall veranlaßt das Ladeterminal die erneute Ausgabe der Antwortdaten des **Laden** mit dem Kommando **Ladedaten wiederholen**. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘38’	INS
3	1	'00'	P1 für **Ladedaten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘0A’	L_e

Wenn die GeldKarte anschließend die Antwortdaten des **Laden** ausgibt, werden diese wie in Schritt 9. des Ladens beschrieben geprüft. Im positiven Fall, wird das erfolgreiche Laden dem Karteninhaber wie in Schritt 9. des Ladens angezeigt.

Tritt bei der Ausführung des **Ladedaten wiederholen** ein Fehler auf, wird mit der Meldung abgebrochen

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Wenn bei der Ausführung des **Laden einleiten wiederholen**

- weder korrekte Antwortdaten, die nachweisen, daß das Laden nicht erfolgt ist,

- noch einer der Returncodes '9F 11' oder '9F 15'

ausgegeben werden, versucht das Ladeterminal, das **Laden einleiten wiederholen** erneut auszuführen, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE.

Gelingt es hierbei, von der GeldKarte korrekte Antwortdaten zu erhalten, die nachweisen, daß nicht geladen wurde, wird der in den Antwortdaten enthaltene Wiederholungszähler WZN gespeichert und mit Schritt 2. des Storno fortgefahren.

Bei jedem anderen Ausgang des zweiten **Laden einleiten wiederholen** wird mit der Meldung abgebrochen

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

2. Die Transaktion kann optional an dieser Stelle für die GeldKarte mit der Anzeige

Laden abgebrochen, bitte Karte entnehmen

beendet werden. Für das Ladeterminal ist die Transaktion hiermit noch nicht beendet. Es muß vielmehr durch das Ladeterminal sichergestellt sein, daß die Stornoanfrage und eventuell die Stornowiederholungen wie in diesem Schritt 2. beschrieben durchgeführt werden, um die Rückbuchung des Ladebetrags auf das Kartenkonto zu veranlassen.

Das Ladeterminal baut mit den in Schritt 1. erhaltenen Antwortdaten des **Laden einleiten wiederholen** mit LSEQN = LSEQ und WZN > WZ als nötigem Nachweis der GeldKarte eine Stornoanfrage mit Nachrichtentyp 0400 und Abwicklungskennzeichen 190000 an die Ladezentrale auf. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

In BMP 62 der Stornoanfrage stellt das Ladeterminal das EF_ID der GeldKarte und die geprüften Antwortdaten der GeldKarte zu **Laden einleiten wiederholen** ein.

Die Stornoanfrage wird mit dem K_T MAC-gesichert.

Das Ladeterminal schickt die Stornoanfrage an die Ladezentrale. Wenn es nicht möglich ist, die Stornoanfrage an die Ladezentrale zu senden, wird so verfahren als wäre innerhalb des definierten Zeitraums keine Stornoantwort eingetroffen.

Wenn das Ladeterminal eine Stornoantwort erhält, prüft es, mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

In den folgenden Fällen beendet das Ladeterminal in diesem Schritt die Transaktion für die GeldKarte:

- Innerhalb eines definierten Zeitraums trifft keine Stornoantwort ein,

  Optional kann der Karteninhaber das Warten auf die Stornoantwort vorzeitig abbrechen.
- Es trifft eine Stornoantwort ein, die formal falsch ist oder deren MAC fehlerhaft ist,
- Es trifft eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21 ein.

Hierzu wird dem Karteninhaber angezeigt

<Text1><, Text2>

Bitte Karte entnehmen

<Text1> ist der folgenden Tabelle zu entnehmen:

Antwortcode/Ereignis	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 57, 58, 64, 76, 91, 92, 96, 97 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler
79, keine Antwort, formal falsche Antwort	Laden abgebrochen

<Text2> ist der folgenden Tabelle zu entnehmen:

Antwortcode/Ereignis	**Text**
30, 57, 64, 76, 77, 78, 97 nicht definierter Antwortcode	wenden Sie sich an Ihr kartenausgebendes Institut
4, 5, 54, 56, 58, 62, 79, 91, 92, 96 keine Antwort, formal falsche Antwort	-

Der Schritt 3. des automatischen Storno wird nur dann durchgeführt, wenn das Ladeterminal innerhalb eines definierten Zeitraums eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 0 oder 21 erhält und der Karteninhaber die Transaktion nicht abgebrochen hat.

Falls das Ladeterminal eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 0, 21, 91, 96 erhalten hat, ist die Transaktion auch für das Ladeterminal beendet.

In allen anderen Fällen muß das Ladeterminal die im folgenden beschriebenen Schritte durchführen, um die Rückbuchung des Ladebetrags auf das Terminalkonto doch noch zu veranlassen.

Stornowiederholung

Falls keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, führt das Ladeterminal eine Stornowiederholung durch.

Das Ladeterminal generiert aus der Stornoanfrage eine Stornowiederholung, indem es den Nachrichtentyp 0400 durch den Typ 0401 ersetzt und über die Nachricht mit K_T einen neuen MAC rechnet.

Diese Nachricht sendet es an die Ladezentrale.

Falls das Ladeterminal zu der Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 91, 96 erhält, ist das Storno für das Ladeterminal beendet.

Zweite Stornowiederholung

Falls keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wiederholt das Ladeterminal die Stornowiederholung.

Falls das Ladeterminal zu der zweiten Stornowiederholung eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode ≠ 91, 96 erhält, ist das Storno für das Ladeterminal beendet.

Weitere Stornowiederholungen

Falls zu der zweiten Stornowiederholung keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, beendet das Ladeterminal das Storno.

Falls bereits dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Stornoantwort der Ladezentrale eingetroffen ist, beendet das Ladeterminal das Storno. Andernfalls speichert das Ladeterminal die Stornowiederholung und versucht, zu einem späteren Zeitpunkt, die gespeicherte Stornowiederholung abzusetzen.

Hierbei wird die folgende Vorgehensweise empfohlen:

Bevor eine neue Transaktion begonnen wird, versucht das Terminal, die gespeicherte Stornowiederholung abzusetzen. Falls wiederum keine Stornoantwort eintrifft oder falls eine Stornoantwort mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Stornoantwort mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wird keine neue Transaktion begonnen. Die Stornowiederholung wird weiterhin gespeichert, um sie später erneut abzusetzen. Eine Ausnahme bildet nur der Fall, daß das Ladeterminal dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Stornoantwort der Ladezentrale erhält. Dann beendet das Ladeterminal das Storno und ist bereit für eine neue Transaktion.

Auf diese Weise wird sichergestellt, daß maximal eine Stornowiederholung in dem Ladeterminal gespeichert und verwaltet werden muß.

Wenn während der Bearbeitung von Stornowiederholungen keine Eingaben möglich sind, zeigt das Terminal an

Bitte warten

3. Wenn das Ladeterminal zu der Stornoanfrage vom Typ 0400 eine formal korrekte Stornoantwort mit fehlerfreiem MAC und Antwortcode 0 oder 21 erhält, baut das Ladeterminal aus den Daten in BMP 62 der Stornoantwort die Kommandonachricht eines

Laden ohne Secure Messaging zum Laden des Ladebetrags 0 auf.

Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘30’	INS
3	1	'80'	P1 für **Laden** ohne Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'10'	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
10-12	3	'00..00'	Ladebetrag
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Laden** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht. Es stellt Byte 1-5 und Byte 49 wie spezifiziert ein.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden

Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’11’	Nachrichten-ID für **Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Laden einleiten**
5-7	3	'00..00'	geladener Betrag
8-10	3	'nn..nn'	neuer aktueller Betrag

Das Ladeterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Ladeterminal gespeicherten Werten LSEQ und WZN überein?
- Ist der Ladebetrag 0?

Bei positivem Ausgang der Prüfungen zeigt das Ladeterminal an

Laden abgebrochen, bitte Karte entnehmen

In allen Fehlerfällen zeigt das Ladeterminal an

Chipfehler, bitte Karte entnehmen

4.4. Vorbereiten des Entladens

Das folgende Diagramm zeigt die ersten Schritte eines Entladevorgangs, die stattfinden nachdem feststeht, daß eine GeldKarte entladen werden soll. Nachdem der Karteninhaber seine Karte eingesteckt hat, wird geprüft,

- ob die Applikation elektronische Geldbörse auf der Karte vorhanden ist,
- ob die GeldKarte für das Entladen gültig ist,
- ob es sich um eine kontobezogene GeldKarte handelt,
- ob das Entladen mit dem Kommando **Entladen einleiten** oder **Entladen einleiten wiederholen** an die GeldKarte begonnen werden muß,

- welcher Restbetrag und damit Entladebetrag in der GeldKarte vorhanden ist.

In Abhängigkeit von den Ergebnissen der Prüfung bietet das Ladeterminal dem Karteninhaber an, einen von 0 verschiedenen Restbetrag aus der Karte zu entladen.

Nach Bestätigung durch den Karteninhaber wird mit dem Entladen (Ablauf in Kapitel 4.5.4.) fortgefahren.

GeldKarte			Ladeterminal		Ladezentrale	
			A1	Anzeige: **Bitte Karte einstecken**		
		<---	C2	Reset GeldKarte		
R2	ATR der GeldKarte	--->				
		<---	C3	SELECT FILE DF_BÖRSE		
R3	OK	--->				
		<---	C4	READ RECORD EF_ID		
R4	Daten aus EF_ID	--->				
			A4	Daten prüfen und speichern		
			C5	READ RECORD EF_BÖRSE		
		<---				
R5	Daten aus EF_BÖRSE	--->				
			A5	Kartentyp auswerten		
			C6	READ RECORD '01' EF_LLOG		
		<---				
R6	Record '01' aus EF_LLOG	--->				
			A6	Statusbyte auswerten und speichern		
			C7	READ RECORD EF_BETRAG		
		<---				
R7	Beträge aus EF_BETRAG	--->				
			A7	Anzeige: Entladen des aktuellen Betrags		
				Beträge speichern		

Erläuterung

1. Am Kundendisplay wird angezeigt:

   Bitte Karte einstecken

2. Nachdem die GeldKarte eingesteckt ist, wird durch das Ladeterminal ein Reset der Karte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer GeldKarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Karte nicht lesbar, bitte Karte entnehmen

3. Die Applikation elektronische Geldbörse wird geöffnet, indem das ADF der Applikation DF_BÖRSE durch das Terminal mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 45 50 01 00'	AID der elektronischen Geldbörse

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Karte nicht lesbar, bitte Karte entnehmen

   Nachdem der Applikationskontext geöffnet ist, können die AEFs der Applikation mittels SFI referenziert werden.

4. Das Terminal liest mittels READ RECORD die Kartenidentifikationsdaten im Record '01' des EF_ID im MF der GeldKarte (SFI '17'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’BC’	P2, Reference Control Byte
5	1	’16’	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der GeldKarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der GeldKarte
16-17	2	'0280'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Die empfangenen Daten werden geprüft.

Wenn die Prüfziffer nicht korrekt ist oder wenn Branchenhauptschlüssel, erste Ziffer der Kurz-BLZ, Verfalldatum, Aktivierungsdatum, Ländercode, Währungskennzeichen oder Wertigkeit der Währung nicht korrekt kodiert sind, bricht das Ladeterminal mit der folgenden Meldung ab:

Kartendaten falsch, bitte Karte entnehmen

Das Ladeterminal überprüft Verfalldatum und Aktivierungsdatum. Wenn aktuelles Datum < Aktivierungsdatum oder aktuelles Datum > Verfalldatum + 3 Monate, bricht das Ladeterminal mit der Meldung ab

Karte ungültig, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Wenn im Ladeterminal gespeichert ist, welche Karten als institutseigene Karten zum Entladen akzeptiert werden, prüft es, ob es sich um eine institutseigene Karte handelt, indem es die Kurz-BLZ aus Byte 2-4 des gelesenen Records auswertet. Stellt es hierbei fest, daß die GeldKarte nicht zum Entladen akzeptiert wird, bricht es mit der Meldung ab

Entladen nicht möglich, bitte Karte entnehmen

Verfügt das Ladeterminal nicht über die benötigte Information, wird diese Prüfung übersprungen. Sie muß dann in einer nachgelagerten Komponente des Terminalbetreibers erfolgen.

Die Daten werden gespeichert.

Währungskennzeichen und Wertigkeit der Währung werden verwendet, um den Gegenwert der in der GeldKarte gespeicherten Betragswerte errechnen und anzeigen zu können. Die möglichen Belegungen für die Wertigkeit der Währung und deren Bedeutung sind in Kapitel 6.9 von [LIT 1] beschrieben.

Anhand der Kurz-BLZ kann die für die GeldKarte zuständige Ladezentrale identifiziert werden.

5. In diesem Schritt stellt das Ladeterminal fest, ob es sich um eine kontobezogene GeldKarte handelt. Hierzu liest es mittels READ RECORD den Record '01' des EF_BÖRSE (SFI '19'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'CC'	P2, Reference Control Byte
5	1	'1B'	L_e, Recordlänge des EF_BÖRSE

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Kennung für den Kartentyp '00': kontobezogene GeldKarte 'FF': Wertkarte ohne Kontobezug
2-5	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Börsenverrechnungskonto
6-10	5	'nn..nn'	Kontonummer Börsenverrechnungskonto
11	1	'nD'	Prüfziffer über Byte 1-9
12-27	16	'XX..XX'	(Triple-)DES-Verschlüsselung im CBC-Mode mit K_{LD} und ICV = '00..00' der 16 Byte 4 Byte Bankleitzahl|5 Byte Kontonummer des Kartenkontos|2 Byte Kartenfolgenummer| 1 Byte Freizügigkeitsschlüssel|'00 00 00 00'

Wenn Byte 1 des Records keinen der Werte ’00’ oder ’FF’ hat, bricht das Ladeterminal mit der Meldung ab

Kartendaten falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Das Ladeterminal wertet den Kartentyp aus.

Wenn es sich um eine Wertkarte handelt, bricht das Terminal mit der Meldung ab

Entladen nicht möglich, bitte Karte entnehmen

6. Das Terminal liest mittels READ RECORD die Protokolldaten des letzten Lade-/Entladeschritts der GeldKarte im Record ’01’ des EF_LLOG der GeldKarte (SFI ’1C’). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’E4’	P2, Reference Control Byte
5	1	’21’	L_e, Recordlänge des EF_LLOG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-3	2	’XX XX’	Sequenznummer LSEQ der Lade-/Entladetransaktion
4	1	’XX’	Wiederholungszähler WZ für das **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geladener bzw. entladener Betrag
8-10	3	'nn..nn'	(neuer) aktueller Betrag
11-13	3	'nn..nn'	AS-ID der Ladezentrale
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
25-27	4	JJJJ MM TT	Datum der Lade-/Entladetransaktion
28-31	3	HH MM SS	Uhrzeit der Lade-/Entladetransaktion
32-33	2	'XX XX'	Sequenznummer BSEQ des letzten Abbuchens bzw. Rückbuchens

Wenn das Statusbyte in Byte 1 einen der Werte '11', '13', '15', '17', '31' oder '35' hat, muß das erste Kommando der Entladetransaktion ein **Entladen einleiten** sein.

Wenn das Statusbyte in Byte 1 einen der Werte '01', '03', '05', '07', '21' oder '25' hat, muß das erste Kommando der Entladetransaktion ein **Entladen einleiten wiederholen** sein.

Das Ladeterminal speichert den gelesenen Record und hält fest, mit welchem Kommando zu beginnen ist.

Wenn das Statusbyte einen anderen Wert hat, wird mit

Kartendaten falsch, bitte Karte entnehmen

abgebrochen.

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

7. Das Terminal liest den aktuellen Betrag aus Record '01' des EF_BETRAG mit dem Kommando READ RECORD (SFI '18'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'09'	L_e, Recordlänge des EF_BETRAG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Wenn einer der drei Beträge nicht BCD-kodiert ist, gibt das Ladeterminal die Meldung aus:

Kartendaten falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Das Ladeterminal prüft den aktuellen Betrag. Hat <aktueller Betrag> den Wert 0, bricht es mit der Anzeige ab

Entladen nicht möglich, bitte Karte entnehmen

Andernfalls zeigt das Ladeterminal an

Guthaben DM: <aktueller Betrag>

Entladen bitte bestätigen

Wenn das Entladen vom Karteninhaber bestätigt wird, werden die Beträge aus EF_BETRAG gespeichert.

4.5. Entladen auf das Kartenkonto

4.5.1. Buchung des Entladebetrags

Erst durch das Kommando **Entladen** in Schritt 4. des Entladedialogs wird der aktuelle Betrag in der GeldKarte auf 0 gesetzt. Bis zur erfolgreichen Durchführung von Schritt 3. des Entladedialogs kann daher das Entladen abgebrochen werden, ohne daß eine Buchung des Entladebetrags vom Börsenverrechnungskonto auf das Kartenkonto erfolgen muß.

Wenn am Ladeterminal in Schritt 4. des Entladedialogs und Schritt 1. des Bestätigungsdialogs, beispielsweise aufgrund von Kommunikationsstörungen, keine Gewißheit darüber gewonnen werden kann, ob das **Entladen** erfolgreich durchgeführt wurde, wird der Karteninhaber aufgefordert, sich zur Klärung an sein kartenausgebendes Institut zu wenden.

Sobald das Kommando **Entladen** in Schritt 4. des Entladedialogs erfolgreich ausgeführt wurde, muß sichergestellt werden, daß die Buchung des entladenen Betrages vom Börsenverrechnungskonto auf das Kartenkonto durch die Ladezentrale veranlaßt wird. Die Ladezentrale veranlaßt die Buchung, sobald sie den Nachweis erhält, daß und in welcher Höhe die GeldKarte entladen wurde.

Dieser Nachweis erfolgt in der Regel explizit durch eine korrekte Entladequittung des Ladeterminals im Rahmen des Bestätigungsdialogs. Ladeterminal und Karteninhaber werden über die erfolgte Buchung des Entladebetrags durch eine Entladebestätigung mit Antwortcode 0 informiert.

Erhält das Ladeterminal keine Entladebestätigung, eine Entladebestätigung mit formalen Fehlern oder eine Entladebestätigung mit negativem Antwortcode 79, 91 oder 96 wird das Ladeterminal (im Falle keine Entladebestätigung, Entladebestätigung mit formalen Fehlern, Antwortcode 91, 96) bzw. eine andere Komponente (im Falle Antwortcode 79) weiterhin versuchen, mit Entladequittungswiederholungen den Bestätigungsdialog und damit die Buchung des Entladebetrags erfolgreich zu beenden. Dem Karteninhaber wird daher das erfolgreiche Entladen der GeldKarte angezeigt und er wird nicht aufgefordert, sich an sein kartenausgebendes Institut zu wenden.

Erhält das Ladeterminal eine korrekte Entladebestätigung mit einem anderen negativen Antwortcode oder einem nicht definierten Antwortcode, wurde zwar die GeldKarte geleert, der

Entladebetrag wurde aber nicht vom Börsenverrechnungskonto auf das Kartenkonto gebucht. Der Karteninhaber wird daher aufgefordert, sich zur Klärung an sein kartenausgebendes Institut zu wenden.

4.5.2. Protokollierung

Es müssen mindestens die folgenden Daten einer Transaktion zum Entladen auf das Kartenkonto protokolliert werden:

- "Entladen",
- Betreiber-BLZ und Terminal-ID,
- Trace-Nummer,
- Datum und Uhrzeit,
- Kartenidentifikationsdaten (Daten des EF_ID) der GeldKarte,
- Entladebetrag (ab erfolgreicher Beendung von Schritt 2 des Entladens),
- Sequenznummer LSEQ und Wiederholungszähler WZ der GeldKarte, (ab erfolgreicher Beendung von Schritt 2 des Entladens),
- aktueller Betrag der elektronischen Geldbörse zu Beginn der Transaktion (ab erfolgreicher Beendung von Schritt 2 des Entladens),
- Ergebnis der Transaktion mit Information darüber, in welchem Status die Transaktion abgebrochen oder erfolgreich beendet wurde.

4.5.3. Sicherheit

Das Entladen auf das Kartenkonto von kontobezogenen GeldKarten erfolgt ohne Secure Messaging und ohne PIN-Eingabe.

Für die MAC-Sicherung der Online-Nachrichten zum Entladen auf das Kartenkonto gelten die analogen Anforderungen wie die für das Laden vom Kartenkonto, die in Kapitel 4.3.3.2. formuliert sind.

4.5.4. Entladen

Das folgende Diagramm zeigt die Schritte des Entladedialogs, die nachfolgend erläutert werden:

GeldKarte			Ladeterminal			Ladezentrale	
			A1	Transaktion beginnen			
		<---	C2	**Entladen einleiten** oder **Entladen einleiten wiederholen** ohne Secure Messaging			
R2	Antwortdaten des **Entladen einleiten (wiederholen)** ohne Secure Messaging	--->					
			A2	Antwortdaten prüfen			
			A3	K_T ableiten			
			C3	Entladeanfrage 0200 Abwkz. 290000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R3	Entladeantwort 0210 Abwkz. 290000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A3	Entladeantwort und K_T-MAC prüfen			
		<---	C4	**Entladen**			
R4	Antwortdaten des **Entladen**	--->					
			A4	Antwortdaten prüfen eventuell Anzeige: **Entladen abgebrochen**			

Erläuterung

1. Nachdem feststeht, daß die GeldKarte entladen werden soll, beginnt das Ladeterminal eine neue Transaktion. Hierbei muß sichergestellt sein, daß

   - Trace-Nummer TSEQ und

- Schlüsselindex SI

gegenüber der letzten Transaktion inkrementiert sind.

Hierbei wird die Trace-Nummer nach Erreichen des maximal möglichen Wertes auf 0 zurückgesetzt.

Hat der Schlüsselindex den Maximalwert erreicht, wird das Terminal für den weiteren Betrieb als Ladeterminal gesperrt.

In allen Schritten der Transaktion müssen durch das Ladeterminal dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet werden.

2. In Abhängigkeit vom Wert des Statusbyte der GeldKarte wird die Kommandonachricht für das **Entladen einleiten** oder das **Entladen einleiten wiederholen** ohne Secure Messaging aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'00' '20'	P1 für **Entladen einleiten** P1 für **Entladen einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1C’	L_c
6	1	'20' '24'	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
7-9	3	'00..00'	Filler
10-12	3	'00..00'	Filler
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	‘3A’	L_e

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'21' '25'	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'nn..nn'	Entladebetrag
8-10	3	'nn..nn'	aktueller Betrag (identisch mit Byte 5-7)
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Unter K_{LD} verschlüsselte Konto- und Kartendaten
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'

Das Ladeterminal prüft, ob die Nachrichten-ID den korrekten Wert hat.

In allen Fehlerfällen wird mit der Meldung abgebrochen

Chipfehler, bitte Karte entnehmen

Bei erfolgreicher Kommandoausführung werden die Sequenznummer LSEQ, der Wiederholungszähler WZ und der Entladebetrag gespeichert.

Das Ladeterminal zeigt an:

Entladebetrag DM: <Entladebetrag>, Entladen wird bearbeitet

3. Eine Entladeanfrage mit Nachrichtentyp 0200 und Abwicklungskennzeichen 290000 an die Ladezentrale wird aufgebaut. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

   Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

   Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

   In BMP 62 der Entladeanfrage stellt das Ladeterminal den Inhalt des EF_ID der GeldKarte und die Antwortdaten der GeldKarte zu **Entladen einleiten (wiederholen)** ein.

   Der Transaktionsschlüssel K_T wird wie in Kapitel 3. spezifiziert aus dem K_{MAC} unter Verwendung des Schlüsselindex SI abgeleitet. Die Entladeanfrage wird mit dem K_T MAC-gesichert.

   Das Ladeterminal schickt die Entladeanfrage an die Ladezentrale.

   Wenn die Entladeanfrage nicht an die Ladezentrale gesendet werden kann, insbesondere wenn durch eine nachgelagerte Komponente des Terminalbetreibers festgestellt wird, daß die Karte als institutsfremd nicht zum Entladen akzeptiert wird, oder wenn die Entladeantwort der Ladezentrale nicht innerhalb eines definierten Zeitraums eintrifft, bricht das Ladeterminal mit der Meldung ab

   Entladen abgebrochen, bitte Karte entnehmen

   Wenn das Ladeterminal eine Antwortnachricht erhält, prüft es mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

   Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

   Stellt das Terminal hierbei Fehler fest, bricht es mit der Meldung ab

Entladen abgebrochen, bitte Karte entnehmen

Wenn die Antwortnachricht formal korrekt ist und einen fehlerfreien MAC aber einen negativen Antwortcode (≠ 0) enthält, bricht das Ladeterminal mit der folgenden Meldung ab:

<Text>, bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 58, 76, 91, 92, 96, 97, 98 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler

4. Wenn die Antwortnachricht formal korrekt ist, einen fehlerfreien MAC und den Antwortcode 0 enthält, baut das Ladeterminal aus den Daten in BMP 62 der Antwortnachricht die Kommandonachricht eines **Entladen** auf.

   Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'80'	P1 für **Entladen** ohne Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'30'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
10-12	3	'00..00'	Filler
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht. Es stellt Byte 1-5 und Byte 49 wie spezifiziert ein.

Das Terminal speichert die verwendete Nachrichten-ID.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte einen negativen Returncode zurückgibt, bricht das Ladeterminal mit der Meldung ab

Entladen abgebrochen, bitte Karte entnehmen

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'31'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
5-7	3	'nn..nn'	Entladebetrag
8-10	3	'00..00'	neuer aktueller Betrag

Das Ladeterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Ladeterminal gespeicherten Werten überein?
- Ist der Entladebetrag korrekt?
- Hat der neue aktuelle Betrag den Wert 0?

Wenn die Prüfungen der Antwortdaten positiv ausfallen, leitet das Ladeterminal unmittelbar einen Bestätigungsdialog ein.

Wenn das **Entladen** weder zu einem negativen Returncode noch zur Ausgabe korrekter Antwortdaten geführt hat, versucht das Ladeterminal, das **Entladen** erneut auszuführen, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE.

Wenn die GeldKarte bei der zweiten Ausführung des **Entladen** einen negativen Returncode ≠ '9F 31' ausgibt, bricht das Ladeterminal mit der Meldung ab

Entladen abgebrochen, bitte Karte entnehmen

In allen anderen Fällen leitet das Ladeterminal einen Bestätigungsdialog ein.

Immer wenn das Ladeterminal einen Bestätigungsdialog einleitet, zeigt es unverändert an

Entladebetrag DM: <Entladebetrag>, Entladen wird bearbeitet

4.5.5. Bestätigungsdialog

Ein Bestätigungsdialog wird immer durchgeführt, wenn Schritt 4. des Entladedialogs durchgeführt wurde und das **Entladen** nicht mit einem negativen Returncode (≠ '9F 31') beendet wurde.

Bei der erfolgreichen Durchführung eines Bestätigungsdialog wird aus der GeldKarte der Betrag 0 entladen, wodurch der Nachweis erzeugt wird, daß die Buchung des Entladebetrags vom Börsenverrechnungskonto auf das Kartenkonto vorgenommen wurde.

Sollte in Schritt 1. eines Bestätigungsdialogs nachträglich der Nachweis erbracht werden, daß das **Entladen** in Schritt 4. des Entladedialogs nicht durchgeführt wurde, wird der Bestätigungsdialog abgebrochen.

Der Schritt 3 wird nur durchgeführt, wenn das Ladeterminal in Schritt 2 eine positive Entladebestätigung (Antwortcode 0) auf eine Entladequittung vom Typ 0202 erhält.

Der Bestätigungsdialog gehört er zu derselben Transaktion wie das zu bestätigende Entladen. Es werden daher für den Bestätigungsdialog dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet wie für das zugehörige Entladen.

In dem folgenden Diagramm werden die Schritte des Bestätigungsdialogs dargestellt, die nachfolgend näher erläutert werden.

GeldKarte			Ladeterminal			Ladezentrale	
			A0	Bestätigung beginnen			
		<---	C1	**Entladen einleiten**			
R1	Antwortdaten des **Entladen einleiten** mit Betrag 0	--->					
			A1	Antwortdaten prüfen			
			C2	Entladequittung 0202 eventuell Entladequittungs-wiederholung 0203 Abwkz. 290000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R2	Entladebestätigung 0212 Abwkz. 290000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A2	Entladebestätigung und K_T-MAC prüfen eventuell Anzeige: **Entladen abgebrochen**			
		<---	C3	**Entladen**			
R3	Antwortdaten des **Entladen** mit Betrag 0	--->					
			A3	Antwortdaten prüfen Anzeige: entladener Betrag			

Erläuterung

0. Zu Beginn eines Bestätigungsdialogs führt das Ladeterminal ein Reset der GeldKarte durch und selektiert das DF_BÖRSE erneut (vgl. Schritt 2. und 3. in Kapitel 4.4.). Im Fehlerfall wird hierbei mit der folgenden Anzeige abgebrochen

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

1. Die Kommandonachricht für das **Entladen einleiten** wird aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'E0'	CLA
2	1	'32'	INS
3	1	'00'	P1 für **Entladen einleiten**
4	1	'00'	P2, fester Wert
5	1	'1C'	L_c
6	1	'20'	Nachrichten-ID für **Entladen einleiten**
7-9	3	'00..00'	Filler
10-12	3	'00..00'	Filler
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Ladeterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
27-29	4	JJJJ MM TT	Datum des Ladeterminals
30-33	3	HH MM SS	Uhrzeit des Ladeterminals
34	1	'3A'	L_e

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

Das Kommando wird an die GeldKarte gesandt.

Gibt die GeldKarte den als Antwort zu **Entladen einleiten** den Returncode '9F 21' bzw. '9F 25' zurück, wurde das **Entladen** zuvor nicht bearbeitet. In diesem Fall bricht das Ladeterminal mit der Anzeige ab

Entladen abgebrochen, bitte Karte entnehmen

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'21'	Nachrichten-ID für **Entladen einleiten**
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'01'	Wiederholungszähler WZ des Kommandos
5-7	3	'00..00'	Entladebetrag
8-10	3	'00..00'	aktueller Betrag
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen Karten: Unter K_{LD} verschlüsselte Konto- und Kartendaten, Bei Wertkarten: '00..00'
37	1	'XX'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'

Das Ladeterminal prüft, ob die Nachrichten-ID den korrekten Wert hat, und ob Entladebetrag und aktueller Betrag 0 sind.

Seien

LSEQN die Sequenznummer aus Byte 2-3,
WZN der Wiederholungszähler aus Byte 4,
STAT das Statusbyte aus Byte 37,
LSEQALT die Sequenznummer aus Byte 38-39,
WZALT der Wiederholungszähler aus Byte 40,
BKALT der Transaktionsbetrag aus Byte 41-43

der Antwortdaten und

NID die vom Ladeterminal gespeicherte Nachrichten-ID des **Entladen**
LSEQ die vom Ladeterminal gespeicherte Sequenznummer,
WZ der vom Ladeterminal gespeicherte Wiederholungszähler,
BK der vom Ladeterminal gespeicherte Entladebetrag, der bei dem

vorhergehenden Entladedialog entladen werden sollte.

Wenn gilt:

LSEQN = LSEQ + 1,
STAT = NID + 1,
LSEQALT = LSEQ,
WZALT = WZ,
BKALT = BK

wurde das **Entladen** in Schritt 4 des Entladens erfolgreich ausgeführt. LSEQN und WZN werden gespeichert, und es wird mit Schritt 2 des Bestätigungsdialogs fortgefahren.

Wenn sichergestellt ist, daß die Entladequittung und die Entladequittungswiederholungen wie in Schritt 2. beschrieben durchgeführt werden, kann die Transaktion optional an dieser Stelle für die GeldKarte mit der Anzeige

Entladebetrag DM: <Entladebetrag> entladen

Bitte Karte entnehmen

beendet werden.

Wenn die GeldKarte bei der Ausführung des **Entladen einleiten**

- weder einen der Returncodes '9F 21' noch '9F 25'
- noch korrekte Antwortdaten, die die erfolgreiche Ausführung des **Entladen** nachweisen,

ausgibt, versucht das Ladeterminal, das **Entladen einleiten** erneut auszuführen, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE.

Gibt die GeldKarte bei der zweiten Ausführung des **Entladen einleiten** korrekte Antwortdaten aus, die die erfolgreiche Ausführung des **Entladen** nachweisen, wird mit Schritt 2. des Bestätigungsdialogs fortgefahren.

Bei jedem anderen Ausgang des zweiten **Entladen einleiten** wird mit der Meldung abgebrochen

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

2. Wenn das Ladeterminal in Schritt 1 die Antwortdaten des **Entladen einleiten** als Nachweis

des erfolgreichen Entladens der GeldKarte erhalten hat, baut es eine Entladequittung mit Nachrichtentyp 0202 und Abwicklungskennzeichen 290000 an die Ladezentrale auf. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. zu beachten.

Die in Kapitel 3.4.2.1. formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

In BMP 62 der Entladequittung stellt das Ladeterminal das EF_ID der GeldKarte und die geprüften Antwortdaten der GeldKarte zu **Entladen einleiten** ein.

Die Entladequittung wird mit dem K_T MAC-gesichert.

Das Ladeterminal schickt die Entladequittung an die Ladezentrale. Wenn es nicht möglich ist, die Entladequittung an die Ladezentrale zu senden, wird so verfahren als wäre innerhalb des definierten Zeitraums keine Entladebestätigung eingetroffen.

Wenn das Ladeterminal eine Entladebestätigung erhält, prüft es, mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Ladeterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. erfüllt sind.

Wenn das Ladeterminal in diesem Schritt eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode 0 erhält, fährt es mit Schritt 3. fort. Andernfalls beendet es die Transaktion für die GeldKarte.

Erhält das Ladeterminal keine Entladebestätigung, eine Entladebestätigung mit formalen Fehlern oder fehlerhaftem MAC oder eine Entladebestätigung mit negativem Antwortcode 79, 91 oder 96, oder (optional) beendet der Karteninhaber das Warten auf die Entladebestätigung vorzeitig, wird das Ladeterminal (keine Entladebestätigung, Entladebestätigung mit formalen Fehlern, Antwortcode 91, 96) bzw. eine andere Komponente (Antwortcode 79) weiterhin versuchen, mit Entladequittungswiederholungen den Bestätigungsdialog und damit die Buchung des Entladebetrags erfolgreich zu beenden. Die Transaktion wird daher mit der folgenden Anzeige beendet, ohne daß der

Karteninhaber aufgefordert wird, sich an sein kartenausgebendes Institut zu wenden:

Mit dem in Schritt 4. des Entladedialogs entladenen <Entladebetrag> zeigt das Ladeterminal an

Entladebetrag DM: <Entladebetrag> entladen

Bitte Karte entnehmen

Erhält das Ladeterminal eine formal korrekte Entladebestätigung mit korrektem MAC mit einem anderen negativem Antwortcode als 79, 91 oder 96 oder mit einem nicht definierten Antwortcode wird dem Karteninhaber angezeigt

<Text>, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode	**Text**
4, 5, 56, 62	Karte nicht zugelassen
30, 57, 58, 76, 92, 97 nicht definierter Antwortcode	Systemfehler
54	Karte ungültig
77, 78	Chipfehler

Entladequittungswiederholung

Falls das Ladeterminal eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode ≠ 0, 91, 96 erhalten hat, ist die Transaktion auch für das Ladeterminal beendet.

Falls keine Entladebestätigung eintrifft oder falls eine Entladebestätigung mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, führt das Ladeterminal eine Entladequittungswiederholung durch.

Das Ladeterminal generiert aus der Entladequittung eine Entladequittungswiederholung, indem es den Nachrichtentyp 0202 durch den Typ 0203 ersetzt und über die Nachricht mit K_T einen neuen MAC rechnet.

Diese Nachricht sendet es an die Ladezentrale.

Zweite Entladequittungswiederholung

Falls das Ladeterminal zu der Entladequittungswiederholung eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode ≠ 91, 96 erhält, ist das Entladen auch für das Ladeterminal beendet.

Falls keine Entladebestätigung eintrifft oder falls eine Entladebestätigung mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wiederholt das Ladeterminal die Entladequittungswiederholung.

Weitere Entladequittungswiederholungen

Falls das Ladeterminal zu der zweiten Entladequittungswiederholung eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode ≠ 91, 96 erhält, ist das Entladen auch für das Ladeterminal beendet.

Falls zu der zweiten Entladequittungswiederholung keine Entladebestätigung eintrifft oder falls eine Entladebestätigung mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, beendet das Ladeterminal den Bestätigungsdialog.

Falls bereits dreimal in Folge eine formal falsche oder mit einem fehlerhaften MAC versehene Entladebestätigung der Ladezentrale eingetroffen ist, beendet das Ladeterminal den Bestätigungsdialog.

Andernfalls speichert das Ladeterminal die Entladequittungswiederholung und versucht, zu einem späteren Zeitpunkt die gespeicherte Entladequittungswiederholung abzusetzen.

Hierbei wird die folgende Vorgehensweise empfohlen:

Bevor eine neue Transaktion begonnen wird, versucht das Terminal, die gespeicherte Entladequittungswiederholung abzusetzen. Falls wiederum keine Entladebestätigung eintrifft oder falls eine Entladebestätigung mit formalen Fehlern oder fehlerhaftem MAC eintrifft oder falls eine formal korrekte Entladebestätigung mit korrektem MAC und Antwortcode 91 oder 96 eintrifft, wird keine neue Transaktion begonnen. Die Entladequittungswiederholung wird weiterhin gespeichert, um sie später erneut abzusetzen.

Auf diese Weise wird sichergestellt, daß maximal eine Entladequittungswiederholung in dem Ladeterminal gespeichert und verwaltet werden muß.

Wenn während der Bearbeitung der Entladequittungswiederholungen keine Eingaben möglich sind, zeigt das Terminal an

Bitte warten

3. Wenn das Ladeterminal zu der Entladequittung vom Typ 0202 eine formal korrekte

Entladebestätigung mit fehlerfreiem MAC und Antwortcode 0 enthält, ist dies der Nachweis, daß die Buchung des Entladebetrags vom Börsenverrechnungskonto auf das Kartenkonto veranlaßt wurde. Prinzipiell ist damit das Entladen der GeldKarte erfolgreich beendet. Um einen Nachweis hierüber in der GeldKarte zu erzeugen, wird wiederum das Kommando **Entladen** verwendet. Das Ladeterminal baut aus den Daten in BMP 62 der Entladebestätigung die Kommandonachricht des **Entladen** auf.

Der Aufbau der Command-APDU ist identisch mit dem der Command-APDU für das **Entladen** ohne Änderung der Maximalbeträge in Schritt 4. des Entladedialogs.

Byte 6-48 der Command APDU entnimmt das Ladeterminal BMP 62 der geprüften Antwortnachricht. Es stellt Byte 1-5 und Byte 49 wie spezifiziert ein.

Das Ladeterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'31'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'01'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
5-7	3	'00..00'	entladener Betrag
8-10	3	'00..00'	neuer aktueller Betrag

Da durch dieses **Entladen** lediglich der Nachweis über das erfolgreiche Entladen in die GeldKarte eingebracht werden soll, wird dem Karteninhaber das erfolgreiche Entladen unabhängig davon angezeigt, ob das Kommando erfolgreich oder fehlerhaft ausgeführt wurde.

Das Ladeterminal zeigt mit dem in Schritt 4. des Entladedialogs entladenen <Entladebetrag> an

Entladebetrag DM: <Entladebetrag> entladen

Bitte Karte entnehmen

Für die GeldKarte und das Ladeterminal ist die Transaktion damit beendet.

GeldKarte
– Händlersysteme –

Version 2.2
22.01.1997

Inhalt

1. Einleitung

Das Bezahlen mit einer GeldKarte wird offline ohne PIN-Prüfung an einem **Akzeptanzterminal** durchgeführt. Das Akzeptanzterminal besitzt als Sicherheitsmodul eine Chipkarte, die sogenannte **Händlerkarte**.

Struktur der Daten, Sicherheitsarchitektur sowie Standard- und Administrationskommandos der Händlerkarte entsprechen der Spezifikation in [LIT 1]. Weiterhin enthält sie die Applikation Geldbörsen-SAM (Secure Application Module), bestehend aus dem Applikationsverzeichnis (ADF) und den zugeordneten AEFs, sowie die Ergänzungskommandos des Geldbörsen-SAM. Die Daten und Ergänzungskommandos der Händlerkarte sind in [LIT 4B] spezifiziert.

Bei der Zahlung aus einer GeldKarte wird im Rahmen eines Dialoges zwischen GeldKarte und Händlerkarte ein Betrag aus der GeldKarte abgebucht. Durch die Händlerkarte wird anschließend ein zertifizierter Datensatz zu der Zahlung erzeugt und an das Akzeptanzterminal ausgegeben. Das Akzeptanzterminal ergänzt den Datensatz der Händlerkarte um terminalspezifische Daten und speichert den ergänzten Datensatz (**Zahlung mit Zertifikat**) verlustfrei.

Außerdem führt die Händlerkarte Summendaten, in denen die Beträge der Zahlungen aufsummiert werden.

Das Akzeptanzterminal übernimmt bei der Zahlung die Vermittlung der Kommunikation zwischen GeldKarte und Händlerkarte und bildet die Schnittstelle zum Karteninhaber der GeldKarte und optional zu einem Bediener.

Jede Zahlung mit Zertifikat zu einer GeldKarten-Zahlung wird durch die Kartennummer der Händlerkarte und eine Sequenznummer eindeutig identifiziert.

Kann eine GeldKarten-Zahlung nicht mit der Erzeugung einer Zahlung mit Zertifikat beendet werden, wird von der Händlerkarte zu der entsprechenden Sequenznummer ein zertifizierter Datensatz erzeugt, der anzeigt, daß die Zahlung fehlgeschlagen ist (Fehlzahlung). Dieser Datensatz wird durch das Akzeptanzterminal auch um terminalspezifische Daten ergänzt. Es speichert den ergänzten Datensatz (**Fehlzahlung mit Zertifikat**) ebenfalls verlustfrei.

Wenn eine Fehlzahlung erfolgt, nachdem der Zahlungsbetrag bereits aus der GeldKarte abgebucht wurde, wird der Betrag durch das Akzeptanzterminal zurückgebucht, sofern GeldKarte und

Händlerkarte noch funktionsfähig sind.

Sobald eine Zahlung erfolgreich beendet wurde, ist die Rückbuchung des Zahlungsbetrages in die GeldKarte nicht mehr möglich.

Die zertifizierten und vom Akzeptanzterminal ergänzten Datensätze zu Zahlung und Fehlzahlung werden im folgenden als **Einzeltransaktionen** bezeichnet. Zu jeder Sequenznummer der Händlerkarte werden lückenlos Einzeltransaktionen erzeugt.

Die Zahlungsbeträge der GeldKarten-Zahlungen werden dem Händler gutgeschrieben, nachdem er die Einzeltransaktionen bei der zuständigen **Evidenzzentrale** (EZ) eingereicht hat und dort die Echtheit der Zertifikate geprüft wurde.

Zur Einreichung werden die Einzeltransaktionen mittels eines **Summensatzes** mit Zertifikat zusammengefaßt, der an einem **Kassenschnitterminal** durch die Händlerkarte aus den gespeicherten Summendaten erzeugt wird. Jeder Summensatz wird durch die Händlerkartennummer und eine Summensequenznummer eindeutig identifiziert.

Zur Erzeugung des Summensatzes stellt die Händlerkarte die Kassenschnitt-Funktion bereit, durch die gleichzeitig neue Summendaten zum Aufsummieren der folgenden Zahlungsbeträge initialisiert werden.

Anhand der Summensequenznummern und der Sequenznummern der Einzeltransaktionen kann die EZ feststellen, ob Einzeltransaktionen doppelt eingereicht wurden.

Die EZ erwartet Einzeltransaktionen und Summensätze der GeldKarten-Zahlungen in einem bestimmten Format, zusammengefaßt in einer Händler-Einreichungsdatei mit Vor-Satz und Ende-Satz. Diese Formatierung nimmt das **Einreichungsterminal** vor, durch das die Umsatzdaten an die EZ übertragen werden.

2. Anforderungen an Händlersysteme

Ein Händlersystem besteht aus den (logisch getrennten) Funktionseinheiten, die in einem einzigen Gerät oder in getrennten Geräten untergebracht werden können,

- Akzeptanzterminal:

  Das Bezahlen aus einer elektronischen Geldbörse erfolgt offline ohne PIN-Prüfung an einem **Akzeptanzterminal**. Hierbei wird im Rahmen eines Dialoges zwischen GeldKarte und Händlerkarte ein Betrag aus der elektronischen Geldbörse abgebucht, und es werden für den Händler die durch die Händlerkarte zertifizierten Einzeltransaktionen erzeugt.

- Kassenschnitterminal:

  Zur Einreichung der Umsatzdaten von GeldKarten-Zahlungen bei der zuständigen EZ werden die Einzeltransaktionen und ein durch die Händlerkarte zertifizierter Summensatz zu den Einzeltransaktionen benötigt. Eine Komponente des Händlersystems muß die Funktion zur Erzeugung des Summensatzes und zur Reinitialisierung der Händlerkarte für die

Summierung neuer Umsätze (Funktion Kassenschnitt) durchführen. Diese Komponente wird als **Kassenschnitterminal** bezeichnet.

- Einreichungsterminal:

  Die Komponente des Händlersystems, die Summensätze und Einzeltransaktionen an die zuständige EZ überträgt, wird als **Einreichungsterminal** bezeichnet.

An diese drei logischen Funktionseinheiten werden im folgenden Anforderungen formuliert, die für eine Zulassung durch den ZKA erfüllt sein müssen.

2.1. Übergreifende Anforderungen

Die folgenden Anforderungen müssen durch alle Funktionseinheiten des Händlersystem erfüllt werden.

2.1.1. Sicherheitsmodul

Ein Händlersystem muß ein Sicherheitsmodul besitzen. In der Schnittstellenspezifikation [LIT 4B] ist dieses Sicherheitsmodul als Chipkarte (**Händlerkarte**) spezifiziert.

Als Händlerkarte sind sowohl Chipkarten im Standardformat (ID-1 Karten) als auch Plug-Ins (ID-000 Karten) zulässig. Die Händlerkarte sollte so untergebracht werden, daß sie vor unbefugter Wegnahme geschützt ist, da ohne Händlerkarte keine GeldKarten-Zahlungen mehr durchgeführt werden können. Der Austausch einer Händlerkarte durch autorisiertes Personal sollte aber ohne großen Aufwand möglich sein.

In einem Händlersystem können die folgenden Beziehungen zwischen Akzeptanzterminal(s) und Händlerkarte(n) bestehen:

- Jedem Akzeptanzterminal ist genau eine Händlerkarte zugeordnet (1:1-Beziehung).

- Mit einer Händlerkarte werden Zahlungen an mehreren Akzeptanzterminals abgewickelt (n:1-Beziehung).

- Mit mehreren Händlerkarten werden Zahlungen an mehreren Akzeptanzterminals abgewickelt (n:n-Beziehung).

Wenn das Händlersystem hierfür ausgelegt ist, kann der Händler das jeweils an einer Zahlung bzw. Fehlzahlung beteiligte Akzeptanzterminal durch die von ihm frei vergebbare Terminal-ID (4 Byte BCD-kodiert) in den Einzeltransaktionen kennzeichnen.

In jedem Fall muß durch das Händlersystem sichergestellt werden, daß durch das Händlersystem die einzelnen GeldKarten-Zahlungen in der korrekten Reihenfolge abgearbeitet werden, die Kunden und der Händler korrekt über den Verlauf der jeweiligen Zahlung informiert werden und zu

allen Zahlungen und Fehlzahlungen korrekte Einzeltransaktionen erzeugt und verlustfrei gespeichert werden.

Es ist zulässig, die Funktionalität der Händlerkarte durch Software in einem Hardware-Sicherheitsmodul des Händlersystems zu emulieren. Es ist vorgesehen, daß zukünftig auch entfernt stehende Sicherheitsmodule die Händlerkartenfunktion übernehmen können. Im folgenden werden die Anforderungen an ein Händlersystem formuliert, das die Händlerkarte durch Software in einem Hardware-Sicherheitsmodul emuliert.

- Ein Händlersystem mit Händlerkartenemulation (virtueller Händlerkarte) muß über ein Hardware-Sicherheitsmodul verfügen und den Sicherheitsanforderungen der "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" ([LIT K]) genügen. Das eingesetzte Hardware-Sicherheitsmodul muß geeignet sein, Masterkeys des deutschen Kreditgewerbes zu speichern.

- Die virtuelle Händlerkarte ist als Software-Modul im Hardware-Sicherheitsmodul des Händlersystems zu realisieren. Die virtuelle Händlerkarte muß sich zur Umsetzung der in Kapitel 3. des vorliegenden Dokuments spezifizierten Abläufe wie die in [LIT 4B] spezifizierte Händlerkarte verhalten. Die Emulation muß hierbei mindestens die in Kapitel 3. des vorliegenden Dokuments verwendeten Kommandos der Händlerkarte bereitstellen.

- Die Spezifikation der Datenstrukturen, der Sicherheitsalgorithmen, des Secure Messaging, der Sicherheitsarchitektur, der Datei-Kontrollinformation, der Funktion der speziellen Dateien und der Standardkommandos aus [LIT 1] sowie die Spezifikation der Ergänzungskommandos und der Ablaufkontrolle der Händlerkarte in [LIT 4B] ist durch die Emulation auf logischer Ebene einzuhalten. Insbesondere enthält die virtuelle Händlerkarte die Daten des EF_ID einer Händlerkarte und wird durch eine Händlerkartennummer eindeutig im System identifiziert.

- Es ist zulässig, daß eine virtuelle Händlerkarte, identifiziert durch ihre Händlerkartennummer, gleichzeitig mehrere Zahlungen mit unterschiedlichen GeldKarten abwickelt (multitaskingfähige virtuelle Händlerkarte). Verschiedene, gleichzeitig offene Zahlungen müssen in der Emulation durch voneinander separierte Prozesse abgewickelt werden.

  Hierbei sind die folgenden Anforderungen zu erfüllen:

  - Der Ablauf jeder einzelnen Zahlung entspricht dem in [LIT 4B] spezifizierten Ablauf mit der dort definierten Ablaufkontrolle durch das Sicherheitsmodul.

  - Für jede Zahlung wird eine neue Sequenznummer HSEQ und ein neuer Record im EF_HLOG verwendet.

    - HSEQ muß hierbei lückenlos aufsteigend sein, wobei das EF_HSEQ aus [LIT 4B] nicht verwendet werden kann, sondern zu Beginn jeder neuen Zahlung das um 1 inkrementierte HSEQ der letzten begonnenen Zahlung verwendet wird.

    - Das EF_HLOG darf mehr als 10 Records enthalten.

    - Als neuer Record im EF_HLOG wird der zyklisch auf den letzten verwendeten bzw.

auf den letzten in Verwendung befindlichen folgende Record verwendet.

Ein Record im EF_HLOG darf aber nur dann überschrieben werden, wenn die gespeicherte Zahlung nicht mehr offen ist.

- Alle Kommandos an die Händlerkartenemulation, die zu derselben Zahlung gehören, müssen den Record der Zahlung im EF_HLOG referenzieren, auch wenn dies nicht der Record '01' ist. Die Ergänzungskommandos müssen entsprechend angepaßt werden. Kommandos, in deren Parametern eine Recordnummer enthalten ist, müssen die korrekte Recordnummer referenzieren.

- Jede Zahlung muß mittels **Zahlung** oder **Fehlzahlung** beendet werden.

- Eine Rückbuchung ist nur nach einer Fehlzahlung möglich. Die Rückbuchung zu einer Fehlzahlung kann auch dann durchgeführt werden, wenn bereits neue Zahlungen begonnen wurden.

- Die multitaskingfähige virtuelle Händlerkarte darf zu ihrer Händlerkartennummer einen Kassenschnitt oder die Ausgabe eines Summensatzes nur dann ausführen, wenn keine Zahlung mehr offen ist.

Die folgenden Spezifikationen gehen davon aus, daß

- das Sicherheitsmodul des Händlersystems eine Chipkarte (Händlerkarte) gemäß der Spezifikation in [LIT 4B] ist und

- eine 1:1-Beziehung zwischen Akzeptanzterminal und Händlerkarte des Händlersystems besteht.

Die Spezifikationen sind auf Händlersysteme mit

- emulierter Händlerkarte,

- n:1-Beziehung zwischen Akzeptanzterminals und (emulierter) Händlerkarte,

- n:n-Beziehung zwischen Akzeptanzterminals und (emulierten) Händlerkarten

sinngemäß zu übertragen.

2.1.2. Elektrische Spezifikation und Übertragungsprotokoll

Die elektrische Spezifikation der Schnittstelle zwischen einem Terminal und der GeldKarte und/oder der Händlerkarte (in diesem Abschnitt zusammen als ZKA-Chipkarten bezeichnet) muß den Vorgaben aus [ISO 4], [ISO 4'], [ISO 4"] und [EMV 1] genügen.

Das Übertragungsprotokoll zwischen Terminal und ZKA-Chipkarten ist T=1 nach [ISO 4'], [ISO 4"]

und [EMV 1].

Zur Behandlung der Unterschiede der Spezifikation von T=1 in [EMV 1] einerseits und [ISO 4'] andererseits werden die folgenden Festlegungen gemacht. Die hier formulierten Anforderungen muß ein Terminal erfüllen, um die ZKA-Zulassung zum Betrieb in einem Händlersystem zu erhalten.

2.1.2.1. NAD

Gemäß [EMV 1] müssen Terminals NAD='00' verwenden. Andere Werte sind nicht zulässig. Die ZKA-Chipkarte lehnt daher NAD ≠ '00' als Protokollfehler ab.

Daher müssen alle vom Terminal gesendeten Blöcke stets NAD='00' enthalten.

2.1.2.2. Länge (LEN)

Minimale Blocklänge

Gemäß [EMV 1] wird das Senden und Empfangen von I-Blöcken mit LEN=0 nicht unterstützt. Eine ZKA-Chipkarte kann somit den Empfang eines I-Blocks mit LEN=0 als einen Protokollfehler abweisen.

Daher darf das Terminal keinen I-Block mit LEN=0 senden.

IFSC und IFSD

Die ZKA-Chipkarte teilt ihr IFSC im ATR mit. Zur Zeit hat IFSC mindestens den Wert 60. Größere Werte von IFSC sind zulässig.

Gemäß [ISO 4'] muß ein Terminal nur den Standardwert IFSD=32 unterstützen und kann mit einem S(IFS request) IFSD<IFSC erreichen.

Gemäß [EMV 1] muß ein Terminal IFSD=254 für das Empfangen von Blöcken unterstützen. Ein S(IFS request) des Terminals mit IFSD<IFSC ist nicht zulässig. Die Chipkarte kann immer Blöcke mit LENI≤FSC senden.

Daher muß das Terminal für Sitzungen mit ZKA-Chipkarten IFSD≥IFSC für alle Typen von ZKA-Chipkarten unterstützen, auch für solche ZKA-Chipkarten, die im ATR den Wert IFSC=254 angeben.

Dies kann "stillschweigend" geschehen, indem das Terminal Blöcke mit 32<LENI≤FSC ohne Fehlermeldung akzeptiert, oder indem es als ersten Block nach dem Empfang des ATR ein S(IFS request) sendet mit IFSC≤IFSD≤254.

Chaining

Gemäß [EMV 1] gilt für das Chaining vom Terminal zur Chipkarte, daß für alle Blöcke außer für den abschließenden LEN=IFSC gilt. In [ISO 4'] ist diese Forderung nicht explizit enthalten. Das Terminal darf während des Chaining grundsätzlich mit LEN≤IFSC senden. ZKA-Chipkarten lehnen den Empfang von I-Blöcken außer des letzten Blockes mit LEN<IFSC während des Chainings als einen Protokollfehler ab.

Das Terminal muß daher bei dem Chaining zur ZKA-Chipkarte alle I-Blöcke außer dem letzten mit LEN=IFSC senden.

2.1.2.3. Behandlung von S-Blöcken

Das Senden und Empfangen von S-Blöcken durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Für das Senden eines S(IFS request) gilt allerdings die oben gemachte Einschränkung, daß IFSD<IFSC hierbei nicht zulässig ist.

2.1.2.4. Fehlerbehandlung

Die Fehlerbehandlung durch das Terminal kann gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Hierbei ist die folgende Anforderung zu beachten:

Empfängt das Terminal während der Protokollabwicklung einen R-Block, so ist nur der Sequenzzähler für den weiteren Protokollablauf ausschlaggebend. Die gemäß [ISO 4'] optionale Auswertung der Bit b1 bis b4 des PCB darf durch das Terminal nur zu informativen Zwecken erfolgen.

2.1.3. Speicherung von Umsatzdaten

Dem Händler werden die Transaktionsbeträge getätigter GeldKarten-Zahlungen nur dann gutgeschrieben, wenn die zugehörigen Umsatzdaten mit korrekten Zertifikaten bei der zuständigen EZ eingereicht werden. Welche Umsatzdaten für die Einreichung benötigt werden, ist Kapitel 2.4. zu entnehmen.

Das Händlersystem muß so ausgelegt sein, daß die Einreichungsdaten

- verlustfrei in elektronischer Form gespeichert und
- bei Bedarf erneut bei der zuständigen EZ eingereicht

werden können, bis dem Händler die entsprechenden Transaktionsbeträge gutgeschrieben worden

sind.

2.2. Anforderungen an Akzeptanzterminals

Das Akzeptanzterminal steuert und vermittelt bei einer GeldKarten-Zahlung die Kommunikation zwischen GeldKarte und Händlerkarte und speichert die zu den Zahlungen und Fehlzahlungen erzeugten Einzeltransaktionen.

Das Akzeptanzterminal muß die zur Erzeugung der Summensätze benötigten Daten der Händlerkarte speichern (vgl. Kapitel 3.2.1.). Falls die Möglichkeit besteht, daß die Händlerkarte ausgetauscht wurde, beispielsweise nach einem Aus- und Wiedereinschalten des Akzeptanzterminals, muß das Terminal dies erkennen und die in Kapitel 3.3. beschriebenen Konsistenzprüfungen durchführen.

Es kann durch Kassenpersonal bediente und unbediente Akzeptanzterminals geben. Automaten, an denen mittels GeldKarte gezahlt werden kann, sind spezielle unbediente Akzeptanzterminals.

Ein Akzeptanzterminal besteht aus verschiedenen Komponenten, die in einer Baueinheit zusammengefaßt oder auf mehrere Baueinheiten verteilt sein können. Das Akzeptanzterminal umfaßt immer die Komponenten

- Steuereinheit mit der Software (Steueranwendung) zur Abwicklung der in Kapitel 3. beschriebenen Abläufe,
- Schnittstelle zur Händlerkarte,
- Schnittstelle zur GeldKarte des Kunden.

Die weiteren Komponenten des Akzeptanzterminals werden im folgenden erläutert.

2.2.1. Komponente zur Betragseingabe

In Abhängigkeit von der Art des Einsatzes eines Akzeptanzterminals kann die Betragseingabe unterschiedlich ausgelegt sein. Beispiele sind

- Übermittlung des Zahlungsbetrages an das Akzeptanzterminal durch ein Kassensystem,
- Feststehender Zahlungsbetrag an Automaten mit Waren zu einem einheitlichen Preis,
- Auswahl eines Zahlungsbetrages mittels Ziffern- oder Auswahltasten an Automaten mit Waren zu unterschiedlichen Preisen.

2.2.2. Kundenanzeige

Das Akzeptanzterminal bedient die Schnittstelle zum Kunden und muß ihn korrekt über die Höhe des zu zahlenden Betrages und über den Ausgang der GeldKarten-Zahlung informieren.

Der Ablauf einer GeldKarten-Zahlung mit Texten, die dem Kunden sinngemäß anzuzeigen sind, ist in den Kapiteln 3.4. und 3.5. beschrieben.

Die Anzeige des Guthabens auf der GeldKarte vor und/oder nach der Zahlung ist hierbei nicht zwingend vorgeschrieben. Falls durch Größe und/oder Position der Kundenanzeige nicht sichergestellt werden kann, daß nur der zahlende Kunde sein Guthaben sehen kann, sollte, wenn möglich, auf die Anzeige des Guthabens verzichtet werden.

An Akzeptanzterminals mit geringen möglichen Zahlungsbeträgen kann auf die Anzeigetexte für den Kunden verzichtet werden, wenn für ihn auf andere Weise deutlich erkennbar ist,

- wie hoch der zu zahlende Betrag ist,
- in welcher Währung zu zahlen ist,
- wie er den Zahlungsbetrag bestätigt (durch Einstecken der GeldKarte oder durch Bestätigungstaste),
- welche Beträge in welcher Währung und in welcher Reihenfolge angezeigt werden, wenn ein Akzeptanzterminal mit numerischer Anzeige, aber ohne Textanzeige nicht nur den Zahlungsbetrag, sondern auch das Guthaben der GeldKarte vor und/oder nach der Zahlung anzeigt,
- ob die Zahlung erfolgreich war oder ob ein Fehler aufgetreten ist,
- wann die GeldKarte entnommen werden kann, so daß Reklamationen aufgrund zu früh gezogener GeldKarten vermieden werden.

Für unbediente Akzeptanzterminals mit geringen möglichen Zahlungsbeträgen gelten insbesondere die folgenden reduzierten Anforderungen an die Kundenanzeige:

- Die Anzeigetexte für den Kunden können durch eine auf das Terminal aufgedruckte Benutzerführung ersetzt werden.
- Die erfolgreiche Prüfung einer Zahlung ("Zahlung erfolgt") und ein Mißerfolg ("Abbruch") müssen angezeigt werden können. Hierzu können eine grüne ("Zahlung erfolgt") und eine rote ("Abbruch") Leuchtdiode verwendet werden. Auf eine differenziertere Anzeige von Fehlerfällen kann verzichtet werden.

  Sofern erforderlich, kann außerdem "Karte nicht entnehmen" angezeigt werden. Hierzu kann eine gelbe Leuchtdiode verwendet werden.

2.2.3. Komponenten für Kundeneingaben

Wenn ein Akzeptanzterminal verschiedene Möglichkeiten der Kartenzahlung anbietet (z. B. elektronische Geldbörse und electronic cash) muß der Kunde die Möglichkeit zur Wahl des Zahlungsmittels haben.

Wenn sowohl das Zahlungsmittel elektronische Geldbörse als auch der zu zahlende Betrag feststeht und wenn der Kunde über den zu zahlenden Betrag informiert ist, bevor die GeldKarte in den Kartenleser des Akzeptanzterminals gesteckt wird, kann die Betragsbestätigung durch das Einstecken der GeldKarte erfolgen. Andernfalls muß das Akzeptanzterminal eine Betragsbestätigungstaste und eine Abbruchtaste für den Kunden besitzen.

Eine Möglichkeit zum Abbruch eines Bezahlvorgangs wird dem Kunden nur dann angeboten, wenn er im Verlauf des Bezahlvorgangs zu einer Eingabe aufgefordert wird.

In der Regel muß die Bestätigungstaste eines Akzeptanzterminals grün und die Abbruchtaste rot sein.

In der Regel wird der Zahlungsbetrag einer GeldKarten-Zahlung nicht über eine Tastatur des Akzeptanzterminals eingegeben, sondern durch ein Kassensystem an das Akzeptanzterminal übermittelt. Falls das Akzeptanzterminal über eine Eingabe- oder Wahlmöglichkeit für den Zahlungsbetrag verfügt, kann das Akzeptanzterminal bis zur Bestätigung der Eingabe außer der Möglichkeit zum Abbruch die Möglichkeit zur Korrektur der Eingabe bieten. In der Regel muß die entsprechende Korrekturtaste gelb sein.

Durch die Beschriftung der Tasten bzw. des Terminals muß die Funktion der Tasten klar erkennbar sein.

Für unbediente Akzeptanzterminals mit geringen möglichen Zahlungsbeträgen gelten die folgenden reduzierten Anforderungen an die Komponente zur Kundeneingabe:

- Auf eine Bestätigungstaste kann verzichtet werden, wenn es sich bei dem Terminal um einen Automaten mit Tasten zur Auswahl verschiedener Waren und/oder Zahlungsbeträge handelt. In diesem Fall wird die GeldKarte vor der Wahl der Ware und/oder des Zahlungsbetrages in den Automaten eingesteckt. Die Bestätigung des Zahlungsbetrages erfolgt durch Drücken des Auswahlknopfes. Durch eine entsprechende Beschriftung muß sichergestellt sein, daß der Kunde über den gewählten Zahlungsbetrag informiert ist.

- Die Abbruchtaste muß nicht notwendig die Farbe Rot haben. Eine Korrekturtaste, sofern vorhanden, muß nicht die Farbe Gelb haben. Beispielsweise kann der Geldrückgabeknopf eines Automaten dazu verwendet werden, eine Rückbuchung auszulösen, wenn die gewünschte Ware nicht ausgegeben werden kann.

2.2.4. Chipkartenleser

An unbedienten Akzeptanzterminals muß der Chipkartenleser für die GeldKarte für den Kunden gut erreichbar sein. An jedem Akzeptanzterminal muß es dem Kunden möglich sein, den Kartenleser während der gesamten Zahlung im Blickfeld zu behalten.

Für den Kunden muß deutlich erkennbar sein, wann er die GeldKarte aus dem Kartenleser entnehmen kann. Auch bei Betriebsstörungen des Akzeptanzterminals muß die GeldKarte entnehmbar sein.

Das Akzeptanzterminal kann mit einem Einzugleser oder Shutter ausgestattet werden.

2.2.5. Bedieneranzeige

Ein bedientes Terminal kann zusätzlich zu der Kundenanzeige eine Bedieneranzeige besitzen, an der der Zahlungsbetrag und der Ausgang der Zahlung angezeigt werden. An der Bedieneranzeige dürfen bei einer GeldKarten-Zahlung Informationen über den Inhalt der beteiligten elektronischen Geldbörse nicht angezeigt werden.

2.2.6. Komponente zur Belegausgabe

In der Regel wird kein spezieller Beleg zu einer GeldKarten-Zahlung ausgegeben. Wenn durch das Akzeptanzterminal ein separater Beleg ausgedruckt wird, muß die GeldKarten-Zahlung durch die folgenden Daten identifiziert werden:

- <Name des Zahlungsmittels>
- GeldKartennummer,
- Händlerkartennummer,
- Sequenznummer BSEQ der GeldKarte,
- Sequenznummern SSEQ und HSEQ der Händlerkarte und
- Betrag.

2.2.7. Serviceschnittstelle

Das Akzeptanzterminal muß eine Serviceschnittstelle besitzen, um das Servicepersonal oder, sofern es sich um ein bedientes Terminal handelt, den Bediener über den internen Zustand des Akzeptanzterminals zu informieren. Beispielsweise dient die Serviceschnittstelle dazu, die Ergebnisse der Terminalvorbereitung (vgl. Kapitel 3.3.) anzuzeigen.

2.2.8. Komponenten zur Speicherung und Ausgabe von Umsatzdaten

Das Akzeptanzterminal muß die für die Einreichung der Umsatzdaten bei der zuständigen EZ benötigten Daten verlustfrei speichern. In Kapitel 3.2. wird erläutert, welche Daten mindestens zu speichern sind. Das Akzeptanzterminal muß eine Schnittstelle zur Ausgabe der gespeicherten Daten besitzen.

Das Akzeptanzterminal muß die in Kapitel 2.1.3. für das Händlersystem geforderte Möglichkeit zur erneuten Einreichung von Umsatzdaten unterstützen.

Das Format, in dem die Umsatzdaten im Akzeptanzterminal gespeichert werden, kann ein für das Händlersystem internes Format sein. Das Format, in dem die Umsatzdaten durch das Akzeptanzterminal ausgegeben werden, kann ebenfalls intern für das Händlersystem festgelegt werden, wenn es sich bei der Ausgabe-Schnittstelle um eine Schnittstelle zu anderen Funktionseinheiten des Händlersystems handelt, durch die die ausgegebenen Umsatzdaten vor der Einreichung bei der EZ bearbeitet werden sollen.

Wenn das Akzeptanzterminal auch Einreichungsterminal ist, muß es die Umsatzdaten an die EZ in dem in Kapitel 2.4. festgelegten Format weiterleiten.

Wenn das Händlersystem mit Einreicherkarten (vgl. [LIT 4G]) arbeitet, muß am Akzeptanzterminal die maximale Anzahl der in der Einreicherkarte speicherbaren Datensätze über Parameter einstellbar sein, so daß auf eine Veränderung der Aufnahmekapazität der verwendeten Einreicherkarten reagiert werden kann.

In Abhängigkeit von dem eingestellten Wert muß das Akzeptanzterminal Zwangs-Kassenschnitte durchführen, so daß das Akzeptanzterminal in diesem Fall auch die Funktionalität eines Kassenschnitterminals beinhalten muß.

2.3. Anforderungen an Kassenschnitterminals

Zur Einreichung der Umsatzdaten von GeldKarten-Zahlungen werden die Einzeltransaktionen und ein durch die Händlerkarte zertifizierter Summensatz zu den Umsatzdaten benötigt. In Kapitel 4. wird beschrieben, auf welche Weise der Summensatz zu erzeugen ist und wie die Händlerkarte für die Summierung neuer Börsenumsätze zu initialisieren ist (Funktion Kassenschnitt).

Für den Fall, daß eine Händlerkarte nicht mehr funktionsfähig ist, muß mit der Funktion Kassenschnitt ein Ersatzsummensatz gebildet werden, dessen Erzeugung ebenfalls in Kapitel 4. beschrieben ist. Hierzu werden die Daten der gespeicherten Umsätze und die Daten der Händlerkarte benötigt. Das Kassenschnitterminal muß diese Daten entweder vorhalten oder über eine Schnittstelle zur Eingabe dieser Daten verfügen.

Zur Ausführung der Funktion Kassenschnitt werden die folgenden Komponenten benötigt:

- Steuereinheit zur Abwicklung der Funktion,
- Schnittstelle (inkl. Leser) zur Kommunikation mit einer Händlerkarte,
- eventuell Schnittstelle zur Eingabe von Händlerkartendaten und Einzeltransaktionen,
- Bedienelement zur Funktionsauslösung (z. B. Funktionstaste),
- Schnittstelle zum Händler (eventuell identisch mit Kundenanzeige),
- verlustfreier Speicher für die Summendaten.

Das Kassenschnitterminal muß die in Kapitel 2.1.3. für das Händlersystem geforderte Möglichkeit zur erneuten Einreichung von Umsatzdaten unterstützen.

Wenn das Kassenschnitterminal über keine Uhr verfügt, können Datum und Uhrzeit in dem Summensatz mit binär 0 belegt werden.

2.4. Anforderungen an Einreichungsterminals

Das Einreichungsterminal erzeugt aus einem oder mehreren Summensätzen und den zugehörigen Einzeltransaktionen einer oder mehrerer Händlerkarten eine Händler-Einreichungsdatei (vgl. Kapitel 5. zum Verfahren). Die Händler-Einreichungsdatei wird bei der für die Händlerkarte(n) zuständigen Evidenzzentrale (Händler-EZ) eingereicht.

Das Einreichungsterminal muß die in Kapitel 2.1.3. für das Händlersystem geforderte Möglichkeit zur erneuten Einreichung von Umsatzdaten unterstützen. Daher ist eine eingereichte Einreichungsdatei sicher zu speichern, bis die Umsätze auf dem Händlerkonto gutgeschrieben sind, so daß im Fehlerfalle dieselbe Einreichungsdatei noch einmal unverändert eingereicht werden kann.

Eine Händler-Einreichungsdatei besteht aus

- einem Vor-Satz (V-Satz),
- einem oder mehreren Summensätzen (S-Sätze bzw. M-Sätze) mit zugehörigen Zahlungen und Fehlzahlungen (Z-Sätze und F-Sätze), sofern vorhanden, und
- einem Ende-Satz (E-Satz).

Eine Einreichungsdatei kann S-Sätze ohne zugehörige Einzeltransaktionen enthalten. Ein S-Satz

ohne Einzeltransaktionen entsteht dann, wenn mit einer Händlerkarte ein Kassenschnitt durchgeführt wird, ohne daß mit dieser Händlerkarte seit dem letzten Kassenschnitt Zahlungen oder Fehlzahlungen erfolgten. Die Positionen 5 und 6 eines S-Satzes ohne Einzeltransaktionen müssen den Wert 0 haben. M-Sätze ohne Einzeltransaktionen dürfen in einer Einreichungsdatei nicht enthalten sein.

Die zu einem Summensatz gehörenden zertifizierten Zahlungen und Fehlzahlungen (Einzeltransaktionen) werden dadurch identifiziert, daß sie dieselbe Summensequenznummer SSEQ enthalten.

Wenn in einer Händler-Einreichungsdatei mehrere Summensätze mit zugehörigen Einzeltransaktionen eingereicht werden, sind die Summensätze nach Händlerkartennummer und innerhalb dieses Kriteriums nach Sequenznummer SSEQ sortiert. Alle Einzeltransaktionen mit einer Sequenznummer SSEQ folgen direkt auf den Summensatz mit der Sequenznummer SSEQ. Die Einzeltransaktionen mit derselben Sequenznummer SSEQ sind nach Sequenznummer HSEQ geordnet.

Das Einreichungsterminal muß sicherstellen, daß alle Einzeltransaktionen, die dieselbe Händlerkartennummer und Sequenznummer SSEQ enthalten, mit einem zugehörigen Summensatz in einer Einreichungsdatei enthalten sind. Die Verteilung von Einzeltransaktionen, die dieselbe Händlerkartennummer und Sequenznummer SSEQ enthalten, auf verschiedene Summensätze oder Einreichungsdateien führt bei der zuständigen EZ zur Abweisung eines Teils der Transaktionen als Doppelteinreichung.

Datum und Uhrzeit in Vor-Satz, Summensatz und Einzeltransaktionen können mit binär 0 belegt sein, wenn das erzeugende Terminal über keine Uhr verfügt.

Alle Character-Felder der Datensätze einer Händler-Einreichungsdatei sind EBCDIC-kodiert, linksbündig, nicht belegte Stellen mit '40' (Blank) auf die vorgeschriebene Länge aufgefüllt. Alle Beträge werden BCD-kodiert und in Pfennig notiert.

Bei der Beschreibung der einzelnen Sätze wird die folgende Notation verwendet:

C..C	Character, wobei C für ein EBCDIC-kodiertes Byte steht
'XX..XX'	binär, wobei X für ein Halbbyte in hexadezimaler Kodierung steht
'nn..nn'	BCD-kodiert, wobei n für ein Halbbyte mit $0 \leq n \leq 9$ steht

2.4.1. Vor-Satz

2.4.1.1. V-Satz für Einreichungsdateien mit Umsatzdaten verschiedener Händlerkarten

Wenn das Einreichungsterminal Summensätze und Einzeltransaktionen **verschiedener** Händlerkarten, in einer Händler-Einreichungsdatei zusammengefaßt bei der für **alle** Händlerkarten zuständigen EZ einreichen soll, muß es die hierzu erforderlichen Namen und Kennungen in den V-Satz der Datei einstellen. Dies ist beispielsweise dann der Fall, wenn ein Händler mehrere

Händlerkarten desselben Instituts verwendet und über ein zentrales Terminal (von ihm selbst oder einem Dienstanbieter betrieben) die Einreichungen bei der Händler-EZ vornimmt.

Das Einreichungsterminal muß in diesem Fall eine Schnittstelle zur Eingabe dieser Namen und Kennungen besitzen. Der Händler erhält die benötigten Namen und Kennungen von seiner Händlerbank.

Der V-Satz einer Händler-Einreichungsdatei enthält in diesem Fall die folgenden Felder:

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	'E5'	'V' für Vorsatz
2	2-6	5	BZAHL	Dateiname, EBCDIC-kodiert
3	7-21	15	C..C	Name Absender, max. 15 Character
4	22-26	5	'xn..nn'	Kennung Absender, BCD-kodiert, $0 \leq x \leq 8$
5	27-41	15	C..C	Name Händler-EZ, max. 15 Character
6	42-46	5	'9n..nn'	Kennung Händler-EZ
7	47-50	4	JJJJ MM TT	Datum
8	51-53	3	HH MM SS	Uhrzeit
9	54-56	3	'00 00 00'	Filler
10	57	1	'00'	Filler
11	58-62	5	'00..00'	Filler
12	63-66	4	'00..00'	Filler
13	67-80	14	'00..00'	Filler

2.4.1.2. V-Satz für Einreichungsdateien mit Umsatzdaten einer Händlerkarte

Wenn das Einreichungsterminal Summensätze und Einzeltransaktionen nur **einer** Händlerkarte in einer Händler-Einreichungsdatei zusammengefaßt bei der für die Händlerkarte zuständigen EZ, einreichen soll, kann auf Namen und Kennungen im V-Satz verzichtet werden. Dies ist beispielsweise dann der Fall, wenn ein Händler ein integriertes Händlersystem betreibt, das in einem Gerät Akzeptanz-, Kassenschnitt- und Einreichungsterminal vereint und mit einer eigenen Händlerkarte ausgerüstet ist.

Im V-Satz sind in diesem Fall in Position 4 bis 6 die folgenden Änderungen vorzunehmen:

Position	Byte	Länge (in Byte)	Wert	Erläuterung
3	7-21	15	TERMINAL|'40..40'	fester Name, EBCDIC-kodiert, mit '40' (Blank) auf 15 Byte aufgefüllt
4	22-26	5	'00..00'	Filler
5	27-41	15	'00..00'	Filler
6	42-46	5	'90..00'	fester Wert

2.4.2. Summensätze

2.4.2.1. Summensatz mit Zertifikat

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	'E2'	EBCDIC-Kodierung des Buchstaben S, Kennung für Summensatz mit Zertifikat der Händlerkarte
2	2-23	22	'XX..XX'	EF_ID der Händlerkarte bestehend aus Branchenhauptschlüssel, Kurz-BLZ, Kartennummer, Prüfziffer, Verfalldatum, Aktivierungsdatum, Ländercode, Entgeltcode, Wertigkeit der Währung und Chiptyp
3	24-33	10	'nn..nD'	Händlerbankverbindung (D: Hexziffer)
4	34-37	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
5	38-41	4	'XX..XX'	Anzahl aller zugehörigen Z-Sätze und F-Sätze
6	42-46	5	'nn..nn'	Summe der Transaktionsbeträge der zugehörigen Z-Sätze
7	47-50	4	JJJJ MM TT	Datum des Summensatzes
8	51-53	3	HH MM SS	Uhrzeit des Summensatzes
9	54	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
10	55-62	8	'XX..XX'	CBC-MAC mit K_{ZD} über Byte 24-46|'00'
11	63-80	18	'00..00'	Reserve

2.4.2.2. Ersatzsummensatz

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	'D4'	EBCDIC-Kodierung des Buchstaben M, Kennung für manuell erzeugten Summensatz ohne Zertifikat der Händlerkarte
2	2-23	22	'XX..XX'	EF_ID der Händlerkarte
3	24-33	10	'nn..nD'	Händlerbankverbindung (D: Hexziffer)
4	34-37	4	'XX..XX'	gemeinsame Sequenznummer SSEQ aller summierten Transaktionen
5	38-41	4	'XX..XX'	Anzahl aller Z- und F-Sätze
6	42-46	5	'nn..nn'	Summe der Transaktionsbeträge der Z-Sätze
7	47-50	4	JJJJ MM TT	Datum des Summensatzes
8	51-53	3	HH MM SS	Uhrzeit des Summensatzes
9	54-62	9	'00..00'	Filler
10	63-64	2	'00 00'	Filler
11	65-80	16	'00..00'	Reserve

2.4.3. Einzeltransaktionen

2.4.3.1. Zahlung mit Zertifikat

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	'E9'	EBCDIC-Kodierung des Buchstaben Z, Kennung für Zahlung
2	2-11	10	'nn..nD'	Händlerkartennummer (D: Hexziffer)
3	12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
4	16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
5	20-29	10	'nn..nD'	GeldKartennummer (D: Hexziffer)
6	30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
7	32-33	2	'XX XX'	Sequenznummer LSEQ der GeldKarte
8	34-36	3	'nn..nn'	Transaktionsbetrag
9	37-46	10	'nn..nD'	Daten des Börsenverrechnungskontos (D: Hexziffer)
10	47-50	4	'nn..nn'	Terminal-ID (frei zu vergeben)
11	51-54	4	JJJJ MM TT	Datum der Transaktion
12	55-57	3	HH MM SS	Uhrzeit der Transaktion
13	58	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
14	59-66	8	'XX..XX'	CBC-MAC mit K_{ZD} über Byte 1-46|'00 00'
15	67-68	2	'00 00'	Filler
16	69-80	12	'00..00'	Reserve

2.4.3.2. Fehlzahlung mit Zertifikat

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	‘C6’	EBCDIC-Kodierung des Buchstaben F, Kennung für Fehlzahlung
2	2-11	10	'nn..nD'	Händlerkartennummer (D: Hexziffer)
3	12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
4	16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
5	20-29	10	‘nn..nD’	GeldKartennummer (D: Hexziffer)
6	30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
7	32-33	2	'00 00'	Filler
8	34-36	3	'nn..nn'	Transaktionsbetrag (vom Terminal eingestellt)
9	37-46	10	‘00..00’	Filler
10	47-50	4	'nn..nn'	Terminal-ID (frei zu vergeben)
11	51-54	4	JJJJ MM TT	Datum der Transaktion
12	55-57	3	HH MM SS	Uhrzeit der Transaktion
13	58	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}
14	59-66	8	'XX..XX'	CBC-MAC mit K_{ZD} über Byte 1-31|'00'
15	67-68	2	'00 00'	Filler
16	69-80	12	'00..00'	Reserve

2.4.4. Ende-Satz

Position	Byte	Länge (in Byte)	Wert	Erläuterung
1	1	1	‘C5’	EBCDIC-Kodierung des Buchstabens E, Kennung für Ende
2	2-4	3	'nn..nn'	Anzahl der S-Sätze, BCD-kodiert
3	5-9	5	'nn..nn'	Summe der Sequenznummern SSEQ der S-Sätze, BCD-kodiert
4	10-12	3	‘nn..nn’	Anzahl der M-Sätze, BCD-kodiert
5	13-17	5	'nn..nn'	Summe der Sequenzzähler SSEQ der M-Sätze, BCD-kodiert
6	18-21	4	'nn..nn'	Anzahl der Z-Sätze, BCD-kodiert
7	22-24	3	'nn..nn'	Summe der Sequenznummern BSEQ der Z-Sätze, BCD-kodiert
8	25-28	4	'nn..nn'	Anzahl der F-Sätze, BCD-kodiert
9	29-31	3	'nn..nn'	Summe der Sequenznummern BSEQ der Fehlzahlungen, BCD-kodiert
10	32-39	8	'nn..nn'	Summe der Börsenumsätze aus S-, M- und Z-Sätzen, BCD-kodiert
11	40-41	2	'00 00'	Filler
12	42-80	39	'00..00'	Filler

2.5. Zulassung von Händlersystemen

Es dürfen nur vom ZKA zugelassene Händlersysteme eingesetzt werden. Bedingung für die Zulassung ist die funktionale Abnahme von

- GeldKarten-Zahlung (Akzeptanzterminal),
- Kassenschnitt (Kassenschnitterminal) und
- Erzeugung, Speicherung und Übertragung der Einreichungsdatei (Einreichungsterminal)

gemäß den Vorgaben in den Kapiteln 3.4. und 3.5., 4. sowie 2.4. und 5.

Gegenstand der Abnahme sind

- elektromechanische Eigenschaften der Schnittstelle(n) zwischen Akzeptanz- und Kassenschnitterminal und Chipkarte(n),
- Abwicklung einer GeldKarten-Zahlung durch Vermittlung der Kommunikation zwischen elektronischer Geldbörse und Händlerkarte (Akzeptanzterminal),
- Bedienung der Kundenschnittstelle (Akzeptanzterminal),

- Bedienung der Serviceschnittstelle und der Bedienerschnittstelle des Akzeptanzterminals,
- Bedienung der Händlerschnittstelle von Kassenschnitt- und Einreichungsterminal,
- Summensatzgenerierung und Rücksetzen der Summendaten (Kassenschnitterminal).

Die Abnahme wird durch eine Stelle des Kreditgewerbes

- anhand von Testfällen,
- mit Chipkartensimulationen (GeldKarte und Händlerkarte) und
- mittels Checklisten, die durch den Hersteller auszufüllen sind,

durchgeführt.

Für die Zulassung eines Händlersystems mit emulierter Händlerkarte ist zusätzlich ein Sicherheitsgutachten eines Sicherheitsgutachters aus der Liste der vom ZKA benannten Gutachter vorzulegen, in dem die Einhaltung der Sicherheitsanforderungen gemäß "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" ([LIT K]) und der in Kapitel 2.1.1. genannten Anforderungen nachgewiesen wird. Außerdem ist die vorgesehene Personalisierungsstelle gegenüber dem ZKA zu benennen, die ihrerseits die Einhaltung der Sicherheitsanforderungen nachweisen muß (vgl. [LIT 5]).

Ergebnis der Abnahme ist eine Typzulassung.

3. Abläufe an einem Akzeptanzterminal

Die Abläufe an einem Akzeptanzterminal gliedern sich in

- Vorbereitung des Akzeptanzterminals,

  die vor Ausführung der ersten Transaktion mit einer Händlerkarte stattfindet,

- Zahlung,

  bei der aus einer GeldKarte ein Betrag abgebucht wird und eine Zahlung mit Zertifikat erzeugt wird,

- Fehlzahlung,

  zur Beendung einer Transaktion im Fehlerfall,

- Rückbuchung,

  bei der nach einer Fehlzahlung ein zuvor aus der GeldKarte abgebuchter Betrag

zurückgebucht wird.

Die Abläufe an einem Akzeptanzterminal werden im folgenden in Form von Diagrammen mit erläuternden Texten dargestellt. In den Diagrammen sind die einzelnen Schritte numeriert und in der nachfolgenden Erläuterung erklärt. In dem Diagramm bedeuten

Cn: Durch das Akzeptanzterminal abzusetzendes Kommando Anfrage in Schritt n,

An: Durch das Akzeptanzterminal durchzuführende Aktion in Schritt n,

Rn: Antwort der jeweiligen Chipkarte in Schritt n.

Immer wenn der Karteninhaber einer GeldKarte im Verlauf einer Zahlung zu einer Eingabe aufgefordert wird, hat er bis zur Bestätigung der Eingabe die Möglichkeit, den Vorgang abzubrechen oder eine Korrektur der Eingabe vorzunehmen.

Wenn ein Abbruch erfolgt ist, zeigt das Akzeptanzterminal an

Zahlung abgebrochen, bitte Karte entnehmen

Bei Wartezeiten, die Eingaben für Karteninhaber einer GeldKarte zeitweise unmöglich machen, wird angezeigt

Bitte warten

Die übrigen Anzeigetexte für den Karteninhaber einer beteiligten GeldKarte sowie Reaktionen auf Fehlerfälle sind der jeweiligen Erläuterung zu entnehmen. **Alle beschriebenen Anzeigetexte sind sinngemäß zu verwenden**.

Die Anzeige des Guthabens auf der GeldKarte vor und/oder nach der Zahlung ist nicht zwingend vorgeschrieben. Falls durch Größe und/oder Position der Kundenanzeige nicht sichergestellt werden kann, daß nur der zahlende Kunde sein Guthaben sehen kann, sollte, wenn möglich, auf die Anzeige des Guthabens verzichtet werden.

An Akzeptanzterminals mit geringen möglichen Zahlungsbeträgen kann auf die Anzeige**texte** für

den Kunden verzichtet werden, wenn für ihn auf andere Weise deutlich erkennbar ist,

- wie hoch der zu zahlende Betrag ist,
- in welcher Währung zu zahlen ist,
- wie er den Zahlungsbetrag bestätigt (durch Einstecken der GeldKarte oder durch Bestätigungstaste),
- welche Beträge in welcher Währung und in welcher Reihenfolge angezeigt werden, wenn ein Akzeptanzterminal mit numerischer Anzeige, aber ohne Textanzeige nicht nur den Zahlungsbetrag, sondern auch das Guthaben der GeldKarte vor und/oder nach der Zahlung anzeigt,
- ob die Zahlung erfolgreich war oder ob ein Fehler aufgetreten ist,
- wann die GeldKarte entnommen werden kann, so daß Reklamationen aufgrund zu früh gezogener GeldKarten vermieden werden.

Um eine hohe Akzeptanz des Zahlungssystems GeldKarte zu erreichen, ist es wichtig, daß der Ablauf für den Kunden an einem Akzeptanzterminal in möglichst kurzer Zeit abgewickelt wird.

3.1. Behandlung von Fehlerfällen

Das Akzeptanzterminal kommuniziert mit GeldKarte und Händlerkarte mittels Standardkommandos, die in Kapitel 8 von [LIT 1] spezifiziert sind, und mittels Ergänzungskommandos, die für die GeldKarte in Kapitel 2 von [LIT 4A] und für die Händlerkarte in Kapitel 2 von [LIT 4B] spezifiziert sind.

Die Returncodes '90 00', '63 CX' und '61 L_a' mit L_a = tatsächliche Länge der Antwortdaten eines Chipkarten-Kommandos zeigen an, daß ein Kommando erfolgreich ausgeführt wurde.

Alle übrigen Returncodes der Chipkarten werden als negative Returncodes bezeichnet. Zu den möglichen Returncodes der einzelnen Kommandos vgl. Kapitel 8 in [LIT 1] sowie Kapitel 2 in [LIT 4A] und [LIT 4B].

Als Fehlerfälle bei dem Aufruf von Kommandos der GeldKarte oder Händlerkarte werden im folgenden bezeichnet

- Kommunikationsfehler, die das Kommunikationsprotokoll T = 1 meldet (beispielsweise Timeout),
- fehlerhafte Länge von Antwortdaten oder Returncodes,
- negative oder nicht definierte Returncodes und
- fehlerhaft kodierte Antwortdaten, sofern die Kodierung durch das Akzeptanzterminal

überprüft wird.

Zu den nicht definierten Returncodes gehört auch der Returncode '61 XX', wenn der Wert von 'XX' nicht mit der tatsächlichen Länge der Antwortdaten übereinstimmt oder wenn 'XX' = L_e aus der Kommandonachricht ist.

Die Reaktion des Terminals auf Fehlerfälle bei Ausführung eines Händlerkarten-Kommandos richtet sich danach, in welche der beiden folgenden Gruppen der Fehlerfall einzuordnen ist:

Gruppe A: Fehler, die anzeigen, daß die Händlerkarte auf Dauer nicht oder nicht korrekt funktioniert

Gruppe B: Fehler, die eventuell durch erneute Ausführung des Kommandos oder Ausführung eines anderen Kommandos behebbar sind

Gruppe A werden korrekte, negative Antworten mit negativen Returncodes der folgenden Tabelle zugeordnet:

Gruppe A:

SW1	SW2	Bedeutung
‘65’	‘81’	Memory failure wird ausgegeben, wenn die Ausführung fehlerhaft abgebrochen wurde und der Inhalt des EEPROM geändert wurde oder wenn beim Lesen von Daten Speicherfehler festgestellt werden
'66'	'03'	keine AC gesetzt
'66'	'04'	unzulässige oder fehlerhaft kodierte AC
'66'	'11'	durch AC (und KID in Kommandodaten) referenzierter Schlüssel nicht gefunden außer bei **Zahlung einleiten**
'66'	'12'	Paritätsfehler des durch eine AC (und KID in Kommandodaten) referenzierten Schlüssels
'66'	'13'	Zusatzinformationen zu dem durch eine AC (und KID in Kommandodaten) referenzierten Schlüssel nicht gefunden
'66'	'14'	FBZ des durch eine AC (und KID in Kommandodaten) referenzierten Schlüssels ist 0
'66'	'15'	Schlüssellänge/Algorithmus-ID unzulässig oder fehlerhaft
'67'	'81'	Antwortdaten zu lang bei READ RECORD
'69'	'85'	benötigte Daten nicht vorhanden bei GET CHALLENGE
'6A'	'82'	File not found **für Ergänzungskommandos** wird ausgegeben, wenn ein EF, auf das das Kommando zugreifen will, nicht vorhanden ist oder Struktur oder Recordlänge des EF nicht der Spezifikation entsprechen
'6A'	'84'	Speicherplatz reicht nicht aus wird ausgegeben, wenn der für eine Log-Datei allokierte Speicherplatz für die Protokollierung mittels eines internen Appends nicht ausreicht
'6A'	'83'	Record not found **für Ergänzungskommandos** außer Varianten **Antwort wiederholen** und **Summensatz** von **ZERTIFIKAT** wird ausgegeben, wenn das Kommando auf einen Record mit Recordnummer größer als die des LAST Record zugreifen müßte
'96'	'01'	Schreibvorgänge eines Ergänzungskommandos unvollständig
'96'	'02'	Applikationsdaten falsch kodiert
'96'	'CX'	Überlauf der Sequenznummer (SSEQ: X = 3, HSEQ: X = 4)

Gruppe B werden korrekte negative Antworten mit den übrigen negativen Returncodes (vgl. die folgende Tabelle) zugeordnet sowie all jene Fehlerfälle, in denen das Terminal keine Gewißheit über Erfolg oder Mißerfolg der Kommandoausführung hat. Dies sind Kommunikationsfehler, fehlerhafte Länge von Antwortdaten oder Returncodes, nicht definierte Returncodes und fehlerhaft kodierte Antwortdaten.

Gruppe B:

SW1	SW2	Bedeutung
‘64’	‘00’	No precise diagnosis wird ausgegeben, wenn bei der Ausführung des Kommandos ein Fehler auftritt, aber der Inhalt des EEPROM durch das Kommando nicht geändert wurde
'66'	'01'	keine gültige Zufallszahl vorhanden
'66'	'05'	Secure Messaging in CLA falsch
'66'	'11'	durch AC (und KID in Kommandodaten) referenzierter Schlüssel nicht gefunden bei **Zahlung einleiten**
'66'	'16'	KID in Kommandodaten stimmt nicht mit der durch eine AC referenzierten Schlüsselnummer überein oder ist nicht in der referenzierten Schlüsselgruppe enthalten
'66'	'88'	Zertifikat falsch
‘67’	‘00’	Wrong length (L_c incorrect or L_e (not) present)
'69'	'85'	DF_BSAM bei Kommandoaufruf nicht selektiert
'69'	'86'	Command not allowed
'6A'	'80'	Incorrect parameters in the data field wird ausgegeben, wenn das Format der Kommandodaten nicht der Spezifikation entspricht oder Kommandodaten außerhalb des spezifizierten Wertebereichs liegen
'6A'	'82'	File not found für READ RECORD wird ausgegeben, wenn ein EF, auf das das Kommando zugreifen will, nicht vorhanden ist oder Struktur oder Recordlänge des EF nicht der Spezifikation entsprechen
'6A'	'83'	Record not found für READ RECORD, **Antwort wiederholen** (**ZERTIFIKAT**) und **Summensatz** wird ausgegeben, wenn das Kommando auf einen Record mit Recordnummer größer als die des LAST Record zugreifen müßte
‘6A’	‘86’	Incorrect parameters P1-P2 wird ausgegeben, wenn P1-P2 keinen der für CLA, INS spezifizierten Werte haben
'6D'	'00'	Wrong instruction code wird ausgegeben, wenn INS keinen für CLA spezifizierten Wert hat
'6E'	'00'	CLA not supported
'96'	'CX'	Überlauf der Sequenznummer (TZ: X = 5)
'97'	'02'	Transaktionsbetrag zu groß
'9F'	'XX'	Reihenfolge der Kommandos nicht eingehalten, letzte Kommandovariante war XX, mit XX = '01', '05', '31', '35'

Im folgenden werden Fehlerfälle beschrieben, in denen die Händlerkarte außer Betrieb zu nehmen ist. Dies bedeutet, daß das Akzeptanzterminal mit dieser Händlerkarte nicht weiter betrieben werden darf, bis die Händlerkarte durch Servicepersonal ausgetauscht wurde.

Über die Serviceschnittstelle muß in geeigneter Weise über die Außerbetriebnahme der Händlerkarte informiert werden. Das Servicepersonal muß die Möglichkeit haben, den Betrieb der

Händlerkarte nach einem Austausch der Händlerkarte wieder freizugeben.

Tritt im Laufe einer Transaktion bei Ausführung eines Händlerkarten-Kommandos ein Fehler der Gruppe A auf, wird die Transaktion abgebrochen und die Händlerkarte außer Betrieb genommen.

Tritt im Laufe einer Transaktion ein Fehler der GeldKarte oder ein Fehler der Gruppe B der Händlerkarte auf, wird die Transaktion abgebrochen, wenn nachweislich noch keine Daten in der GeldKarte verändert wurden. Andernfalls wird versucht, die Transaktion durch fehlerbehebende Maßnahmen mit einer Zahlung erfolgreich zu beenden. Gelingt dies nicht, wird der Karteninhaber aufgefordert, sich an sein kartenausgebendes Institut zu wenden.

Wenn eine Transaktion nicht erfolgreich durch eine Zahlung beendet werden kann, das Kommando **Zahlung einleiten** durch die Händlerkarte aber schon erfolgreich bearbeitet worden ist, muß die Händlerkarte mittels einer Fehlzahlung in einen Zustand versetzt werden, in dem sie neue Transaktionen behandeln kann. Gelingt dies nicht, muß die Händlerkarte außer Betrieb genommen werden. Über die Serviceschnittstelle muß in geeigneter Weise über den Zustand des Akzeptanzterminals informiert werden.

Bei häufigem Auftreten von Fehlern der Gruppe B bei der Kommandoausführung durch die Händlerkarte sollte diese ausgetauscht werden.

Das Auftreten der Returncodes '96 C5' (Überlauf des Transaktionszählers bei 'FFFFFFFF') und '97 02' (Überlauf der Umsatzsumme bei 99.999.999,99) kann vermieden werden, wenn rechtzeitig ein Kassenschnitt durchgeführt wird.

3.2. Speicherung von Einreichungsdaten

Das Akzeptanzterminal speichert die folgenden Daten zur Einreichung bei der EZ verlustfrei.

3.2.1. Daten der Händlerkarte

Die Kartenidentifikationsdaten der Händlerkarte und die Händlerbankverbindung werden zur Erstellung eines Summensatzes für die von der Händlerkarte zertifizierten Einzeltransaktionen benötigt.

Für den Fall, daß die Händlerkarte nicht mehr funktionsfähig ist, so daß diese Daten zur Erstellung des Summensatzes durch das Kassenschnitterminal nicht mehr aus der Karte gelesen werden können, werden diese Daten durch das Akzeptanzterminal vor der ersten Transaktion aus der Karte gelesen. Sie werden zusammen mit den anschließend erzeugten Umsatzdaten (Zahlungsdaten und Fehlzahlungsdaten) gespeichert.

Kartenidentifikationsdaten der Händlerkarte

Die Kartenidentifikationsdaten der Händlerkarte sind in dem 22 Byte langen Record '01' des EF_ID der Händlerkarte gespeichert:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'67'	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der Händlerkarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der Händlerkarte
16-17	2	'0280'	Ländercode
18-20	3	'nn nn nn'	Entgeltcode (Standardwert '00 00 00')
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Händlerbankverbindung

Die Händlerbankverbindung ist in dem 10 Byte langen Record '01' des EF_KONTO der Händlerkarte gespeichert:

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Händlerkonto
5-9	5	'nn..nn'	Kontonummer Händlerkonto
10	1	'nD'	n: Prüfziffer über Byte 1-9, D: Hexziffer

3.2.2. Zahlungsdaten mit Zertifikat

Wenn das Kommando **Zahlung** durch die Händlerkarte erfolgreich ausgeführt wurde, gibt sie die folgenden Antwortdaten aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E9’	EBCDIC-Kodierung des Buchstaben Z, Kennung für Zahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	‘nn..nD’	GeldKartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
32-33	2	'XX XX'	Sequenznummer LSEQ der GeldKarte
34-36	3	'nn..nn'	Zahlungsbetrag
37-46	10	‘nn..nD’	Daten des Börsenverrechnungskontos ('D': Hexziffer)
47-54	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 48 Byte Byte 1-46|'00 00'
55	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

Bei diesen Daten handelt es sich um die Positionen 1-9, 14 und 13 eines Z-Satzes zur Einreichung bei der EZ (vgl. Kapitel 2.4.3.).

Um aus diesen Daten einen kompletten Z-Satz aufzubauen, sind die Positionen 10-12 und 15-16 zu belegen.

Wenn Position 10 mit einer Terminal-ID belegt werden soll, um Transaktionen, die mit derselben Händlerkarte an verschiedenen Terminals durchgeführt werden, zu unterscheiden, muß dies durch das Akzeptanzterminal erfolgen. Die Terminal-ID muß dann mit den Zahlungsdaten gespeichert werden.

Wenn Position 11 und 12 mit Datum und Uhrzeit der Transaktion belegt werden sollen, müssen Datum und Uhrzeit durch das Akzeptanzterminal hinzugefügt und mit den Zahlungsdaten gespeichert werden.

Position 10-12 können mit '00' belegt werden, so daß die Speicherung der entsprechenden Daten durch das Akzeptanzterminal nicht obligatorisch ist.

Position 15-16 sind mit '00' zu füllen und können zu einem späteren Zeitpunkt hinzugefügt werden, beispielsweise durch das Einreichungsterminal.

3.2.3. Fehlzahlungsdaten mit Zertifikat

Wenn bei Abbruch einer Transaktion das Kommando **Fehlzahlung** durch die Händlerkarte erfolgreich durchgeführt wird, gibt sie die folgenden Antwortdaten aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘C6’	EBCDIC-Kodierung des Buchstaben F, Kennung für Fehlzahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	‘nn..nD’	GeldKartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
32-39	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 32 Byte Byte 1-31|'00'
40	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

Bei diesen Daten handelt es sich um die Positionen 1-6, 14 und 13 eines F-Satzes zur Einreichung bei der EZ (vgl. Kapitel 2.4.3.).

Um aus diesen Daten einen kompletten F-Satz aufzubauen, sind die Positionen 7-12 und 15-16 zu belegen.

Für die Positionen 10-12 und 15-16 gelten dieselben Aussagen wie für die entsprechenden Positionen des Z-Satzes.

Wenn Position 8 mit dem Transaktionsbetrag der abgebrochenen Transaktion belegt werden soll, muß dies durch das Akzeptanzterminal erfolgen. Dies ist immer nur dann möglich, wenn der Transaktionsbetrag zum Zeitpunkt des Abbruchs schon feststand. Der hier eingetragene Transaktionsbetrag ist durch die Händlerkarte nicht geprüft worden, so daß dieses Datenfeld nur informativ ist.

Position 8 kann ebenfalls mit '00' belegt werden, so daß die Speicherung durch das Akzeptanzterminal nicht notwendig ist.

Position 7 und 9 sind mit '00' zu füllen und können zu einem späteren Zeitpunkt hinzugefügt werden, beispielsweise durch das Einreichungsterminal.

3.2.4. Rekonstruktion von Zahlungs- und Fehlzahlungsdaten

Solange der Record mit den Daten der Zahlung oder Fehlzahlung in der Protokolldatei EF_HLOG der Händlerkarte noch nicht überschrieben ist, können die Zahlungs- bzw. Fehlzahlungsdaten, insbesondere das Zertifikat, mittels des Kommandos **Antwort wiederholen** durch die Händlerkarte reproduziert werden.

Die folgende Tabelle zeigt die Command-APDU eines **Antwort wiederholen** zur erneuten

Ausgabe der Zahlungs- bzw. Fehlzahlungsdaten mit Zertifikat in Record ’XX’ der Protokolldatei.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'60'	P1 für **Antwort wiederholen**
4	1	‘XX’	P2, Recordnummer (zwischen '01' und 'FE')
5	1	‘37’ '28'	L_e für Zahlungsdaten L_e für Fehlzahlungsdaten

Die Protokolldatei hat 10 Records, so daß immer die Zahlungs- bzw. Fehlzahlungsdaten der letzten zehn Transaktionen erneut ausgegeben werden können. Die Daten der letzten Transaktion sind in dem Record mit der Nummer '01' gespeichert, die Daten der ältesten noch gespeicherten Transaktion sind in Record '0A' abgelegt, wenn schon mindestens 10 Transaktionen erfolgt sind.

Die Daten weiter zurückliegender Transaktionen können nicht mehr durch die Händlerkarte ausgegeben werden.

3.2.5. Hilfssummensätze

Optional kann sich das Akzeptanzterminal nach jedem Beenden einer Transaktion mit dem Kommando **Summensatz** von der Händlerkarte einen Hilfssummensatz ausgeben lassen, um für den Fall, daß die Händlerkarte nicht mehr funktionstüchtig ist, immer die aktuell gültigen Summendaten mit Zertifikat zur Erzeugung eines Summensatzes mit Zertifikat (vgl. Kapitel 2.4.2.) zur Verfügung zu haben.

Im Akzeptanzterminal gespeicherte Hilfssummensätze sollten immer als solche gekennzeichnet sein, damit sie durch das Händlersystem von Summensätzen unterschieden werden können, die bei dem Vorgang Kassenschnitt (vgl. Kapitel 4.) erzeugt werden.

Die folgende Tabelle zeigt die Command-APDU des Kommandos **Summensatz.**

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'20'	P1 für **Summensatz**
4	1	‘01’	P2, Recordnummer
5	1	‘20’	L_e

Bei erfolgreicher Ausführung gibt die Händlerkarte die folgenden Daten aus:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1-10	10	'nn..nD'	Händlerbankverbindung ('D' Hexziffer)
11-14	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
15-18	4	'XX..XX'	Anzahl aller zugehörigen Zahlungen und Fehlzahlungen
19-23	5	'nn..nn'	Summe der Zahlungsbeträge
24-31	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 1-23|'00'
32	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

Hierbei handelt es sich um die Summendaten mit Zertifikat, die die Positionen 3-6, 10 und 9 eines Summensatzes mit Zertifikat (vgl. Kapitel 2.4.2.) zur Einreichung bei der EZ bilden.

3.3. Vorbereitung des Akzeptanzterminals

Bevor das Akzeptanzterminal die erste Transaktion mit einer Händlerkarte beginnt, führt es einmalig vorbereitende Schritte durch, die dann in der Regel nicht bei jeder GeldKarten-Zahlung mit dieser Händlerkarte erneut durchzuführen sind.

Durch die Maßnahmen bei der Vorbereitung des Akzeptanzterminals wird ferner sichergestellt, daß durch das Terminal alle verfügbaren und zur Einreichung der erzeugten Zahlungs- und Fehlzahlungsdaten benötigten Daten gespeichert werden.

Diese vorbereitenden Schritte müssen erfolgen,

- wenn das Akzeptanzterminal zum ersten Mal mit einer Händlerkarte ausgestattet wird und
- wenn die Händlerkarte des Akzeptanzterminals gewechselt bzw. entnommen und wiedereingesteckt wurde.

Das Akzeptanzterminal muß so konstruiert sein, daß die Steueranwendung des Terminals immer erkennt, daß ein Austausch bzw. Entnehmen und Wiedereinstecken der Händlerkarte erfolgt ist oder möglicherweise erfolgt ist. Falls die Möglichkeit besteht, daß die Händlerkarte ausgetauscht bzw. entnommen und wiedereingesteckt wurde, beispielsweise nach einem Aus- und Wiederanschalten des Akzeptanzterminals, muß das Terminal mindestens die Konsistenzprüfungen für das EF_ID und für die aktuellen Summendaten der Händlerkarte mit den entsprechenden Reaktionen auf Inkonsistenzen aus Schritt 4. bzw. Schritt 6. des folgenden Ablaufs durchführen.

Das Akzeptanzterminal selektiert die Applikation Geldbörsen-SAM und liest und speichert die Schlüsselnummer KID des KGK_{RD} der Händlerkarte.

Außerdem liest und speichert es die Kartenidentifikationsdaten aus dem EF_ID der Händlerkarte und die Händlerbankverbindung aus dem EF_KONTO der Händlerkarte. Die Daten aus EF_ID und EF_KONTO werden zur Erzeugung von Summensätzen benötigt. Damit sie zum Erzeugen eines Ersatzsummensatzes für gespeicherte Einzeltransaktionen (vgl. Kapitel 4.) zur Verfügung stehen, auch wenn die Händlerkarte nicht mehr funktionsfähig ist, werden diese Daten zusammen mit den Zahlungs- und Fehlzahlungsdaten der Händlerkarte verlustfrei gespeichert.

Ferner überprüft das Akzeptanzterminal, ob die Händlerkartennummer im EF_ID und die aktuellen Summendaten im Record '01' des EF_SUMME der Händlerkarte konsistent mit den entsprechenden Daten der eventuell gespeicherten Zahlungs- und Fehlzahlungsdaten sind.

Im folgenden wird der Begriff **Zahlungsdatengruppe** verwendet. Eine Zahlungsdatengruppe besteht aus den folgenden Datensätzen:

- Daten einer Händlerkarte, bestehend aus
  - Kartenidentifikationsdaten der Händlerkarte,
  - Händlerbankverbindung,
- Zahlungs- oder Fehlzahlungsdatensätzen mit
  - identischer Händlerkartennummer,
  - identischer Summensequenznummer SSEQ und
  - lückenlos aufsteigenden Sequenznummern HSEQ.

Falls für die Zahlungsdatengruppe bereits ein Summensatz mit Zertifikat oder ein Ersatzsummensatz gebildet wurde, besteht die Zahlungsdatengruppe aus den folgenden Datensätzen:

- Summensatz mit Zertifikat oder Ersatzsummensatz mit
  - Kartenidentifikationsdaten der Händlerkarte,
  - Händlerbankverbindung,
  - Summensequenznummer SSEQ,
- Zahlungs- oder Fehlzahlungsdatensätzen mit
  - identischer Händlerkartennummer,
  - identischer Summensequenznummer SSEQ und
  - lückenlos aufsteigenden Sequenznummern HSEQ.

Eine Zahlungsdatengruppe enthält mindestens einen Zahlungs- oder Fehlzahlungsdatensatz. Die gemeinsame Händlerkartennummer in den Zahlungs- oder Fehlzahlungsdatensätzen stimmt mit

der Händlerkartennummer in den Kartenidentifikationsdaten der zugehörigen Händlerkartendaten bzw. des zugehörigen Summensatzes überein. Falls für die Zahlungsdatengruppe bereits ein Summensatz mit Zertifikat oder ein Ersatzsummensatz gebildet wurde, stimmt die gemeinsame Summensequenznummer SSEQ mit der Summensequenznummer aus dem Summensatz überein.

Die Datensätze einer Zahlungsdatengruppe bilden eine logische Gruppe, da sie zu demselben Summensatz gehören. Nachdem die Händlerkarte eines Akzeptanzterminals gegen eine andere ausgetauscht wurde oder nachdem mit der Händlerkarte eines Akzeptanzterminals ein Kassenschnitt durchgeführt wurde, werden durch das Akzeptanzterminal Zahlungs- und Fehlzahlungsdaten erzeugt, die zu einer neuen Gruppe gehören.

Das folgende Diagramm zeigt die Schritte bei der Vorbereitung des Akzeptanzterminals:

GeldKarte		**Händlerterminal**			**Händlerkarte**	
		C1	Reset Händlerkarte	--->		
				<---	R2	ATR der Händlerkarte
		C2	SELECT FILE DF_BSAM	--->		
				<---	R2	OK
		C3	READ RECORD 'XX' EF_KEYD in DF_BSAM	--->		
				<---	R3	Schlüssel-Zusatzinformationen
		A3	Schlüsselnummer KID des KGK_{RD} speichern			
		C4	READ RECORD EF_ID	--->		
				<---	R4	Daten aus EF_ID
		A4	Daten prüfen und speichern			
		C5	READ RECORD EF_KONTO	--->		
				<---	R5	Daten aus EF_KONTO
		A5	Händlerbankverbindung speichern			
		C6	READ RECORD '01' EF_SUMME	--->		
				<---	R6	Daten aus EF_SUMME
		A6	Konsistenz prüfen			

Erläuterung

Im folgenden wird nicht festgelegt, in welcher Weise die erfolgreiche Ausführung oder die Fehlerfälle in den Schritten an der Serviceschnittstelle angezeigt werden. Dies kann terminalspezifisch geregelt werden.

Wenn in einem der Schritte ein Fehler der Gruppe A auftritt, muß die Händlerkarte außer Betrieb genommen werden.

Bei anderen Fehlerfällen wird versucht, den Vorgang mit Schritt 1 erneut zu beginnen. Kann der Vorgang auch nach mehrfacher Wiederholung nicht erfolgreich beendet werden, muß die Händlerkarte außer Betrieb genommen werden.

1. Durch das Akzeptanzterminal wird ein Reset der Händlerkarte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer Händlerkarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Wenn die Händlerkarte einen korrekten ATR zurückgibt, wird mit Schritt 2. fortgefahren.

2. Die Applikation Geldbörsen-SAM wird geöffnet, indem das ADF der Applikation DF_BSAM durch das Terminal mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 42 53 01 00'	AID der Applikation Geldbörsen-SAM

   Wenn das Kommando erfolgreich durchgeführt wurde (Returncode '90 00'), können die AEFs der Applikation mittels SFI referenziert werden, und es wird mit Schritt 3. fortgefahren.

3. In diesem Schritt sucht das Terminal die Schlüsselnummer KID des KGK_{RD} der

Händlerkarte.

Hierzu liest das Terminal mittels READ RECORD sukzessive die Records des EF_KEYD im DF_BSAM (SFI '18'), bis es den ersten Record gefunden hat, dessen erstes Byte (Schlüsselnummer) einen Wert zwischen '05' und '0E' hat.

Die Suche und damit der Vorgang wird abgebrochen, wenn das READ RECORD nicht erfolgreich ausgeführt wird, insbesondere wenn der Returncode '6A 83' (Record not found) zurückgegeben wird.

Die folgende Tabelle zeigt den Aufbau der Command-APDU für READ RECORD.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'XX'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'05'	L_e, Recordlänge des EF_KEYD

Die Recordnummer 'XX' hat bei dem ersten Aufruf den Wert '01' und wird bei jedem weiteren Aufruf um 1 erhöht.

Bei der erfolgreichen Ausführung des READ RECORD gibt die Händlerkarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'XX'	logische Schlüsselnummer
2	1	'08' oder '10'	Schlüssellänge
3	1	'XX'	Algorithmus-ID
4	1	'XX'	Fehlbedienungszähler
5	1	'XX'	Schlüssel-Version

Das Terminal prüft Byte 1 und speichert den Wert als KID, wenn er einen Wert zwischen '05' und '0E' hat.

Anschließend wird mit Schritt 4. fortgefahren.

4. Das Terminal liest mittels READ RECORD die Kartenidentifikationsdaten im Record '01' des EF_ID im MF der Händlerkarte (SFI '17'). Die folgende Tabelle zeigt den Aufbau der Command-APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’BC’	P2, Reference Control Byte
5	1	’16’	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der Händlerkarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der Händlerkarte
16-17	2	'0280'	Ländercode
18-20	3	'nn nn nn'	Entgeltcode (Standardwert '00 00 00')
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Die Kodierung der empfangenen Daten wird geprüft.

Wenn die Prüfziffer nicht korrekt ist oder wenn Branchenhauptschlüssel, erste Ziffer der Kurz-BLZ, Verfalldatum, Aktivierungsdatum, Ländercode, Entgeltcode oder Wertigkeit der Währung nicht korrekt kodiert sind, muß die Händlerkarte ausgetauscht werden. Die möglichen Belegungen für die Wertigkeit der Währung und deren Bedeutung sind in Kapitel 6.9 von [LIT 1] beschrieben.

Wenn das Terminal über eine Uhr verfügt, prüft es anhand des aktuellen Datums, des Aktivierungsdatums und des Verfalldatums, ob die Händlerkarte aktiviert und nicht verfallen ist. Wenn hierbei festgestellt wird, daß die Händlerkarte nicht gültig ist, muß sie ausgetauscht werden.

Das Akzeptanzterminal speichert das EF_ID zwischen und fährt mit Schritt 5. fort.

5. Das Terminal liest mittels READ RECORD die Händlerbankverbindung im Record '01' des EF_KONTO (SFI '1A'). Die folgende Tabelle zeigt den Aufbau der Command-APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'D4'	P2, Reference Control Byte
5	1	'0A'	L_e, Recordlänge des EF_KONTO

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Händlerkonto
5-9	5	'nn..nn'	Kontonummer Händlerkonto
10	1	'nD'	n: Prüfziffer über Byte 1-9, D: Hexziffer

Die Daten des EF_KONTO werden zusammen mit dem in Schritt 4. gelesenen EF_ID zwischengespeichert.

6. Das Terminal liest mittels READ RECORD die Summendaten im Record '01' des EF_SUMME (SFI '19'). Die folgende Tabelle zeigt den Aufbau der Command-APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'CC'	P2, Reference Control Byte
5	1	'0D'	L_e, Recordlänge des EF_SUMME

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	'XX..XX'	Sequenznummer SSEQ' des Summensatzes
5-8	4	'XX..XX'	Transaktionszähler TZ, Anzahl Transaktionen seit letztem **Kassenschnitt**
9-13	5	'nn..nn'	Summe der Zahlungsbeträge seit letztem **Kassenschnitt**

Das Akzeptanzterminal stellt fest, wieviele Zahlungs- oder Fehlzahlungsdaten gespeichert sind, die die Kartennummer aus dem in Schritt 4. zwischengespeicherten EF_ID und die gelesene Summensequenznummer SSEQ' der Händlerkarte enthalten. Die festgestellte Anzahl wird mit ANZ bezeichnet. ANZ kann den Wert 0 haben.

Es wird geprüft, ob ANZ mit dem Wert des Transaktionszählers TZ aus dem gelesenen Record '01' des EF_SUMME übereinstimmt.

Stimmen diese Werte überein, werden die Zahlungs- oder Fehlzahlungsdaten, die die Kartennummer aus dem in Schritt 4. gelesenen EF_ID und die in diesem Schritt gelesene Summensequenznummer SSEQ' enthalten, durch das EF_ID und das in Schritt 5. gelesene EF_KONTO zu einer Zahlungsdatengruppe ergänzt, sofern dies nicht bereits früher geschehen ist. Dies wird auch dann durchgeführt, wenn ANZ und TZ den Wert 0 haben. In diesem Fall wird mit dem EF_ID, dem EF_KONTO und SSEQ' eine neue Zahlungsdatengruppe angelegt. Alle anschließend erzeugten Zahlungs- und Fehlzahlungsdaten mit derselben Kartennummer und derselben Sequenznummer SSEQ' werden dieser Zahlungsdatengruppe hinzugefügt. Damit ist die Vorbereitung des Akzeptanzterminals erfolgreich abgeschlossen.

Wenn ANZ und TZ verschieden sind, ist zu unterscheiden, ob das Akzeptanzterminal auch Kassenschnitterminal ist oder nicht.

An einem Akzeptanzterminal, das auch Kassenschnitterminal ist, wird in diesem Fall der Vorgang Kassenschnitt mit den in dem Akzeptanzterminal gespeicherten Zahlungsdatengruppen und der Händlerkarte durchgeführt, wie er in Kapitel 4. beschrieben ist. Falls dies noch nicht geschehen ist, werden vorher alle Zahlungs- und Fehlzahlungsdaten, die die Kartennummer der Händlerkarte enthalten, mit den aus der Händlerkarte gelesenen Daten des EF_ID und des EF_KONTO zu Zahlungsdatengruppen komplettiert.

Falls ANZ > 0 ist, wird bei dem Vorgang Kassenschnitt für die Zahlungsdatengruppe mit SSEQ = SSEQ' ein Ersatzsummensatz gebildet, da ANZ und TZ verschieden sind. Falls TZ > 0 ist, wird bei einem erfolgreichen Vorgang Kassenschnitt durch die Händlerkarte auch ein Summensatz mit Zertifikat zu der Sequenznummer SSEQ' generiert. Dieser zertifizierte Summensatz kann aber nicht zur Komplettierung der Zahlungsdatengruppe verwendet

werden, da ANZ verschieden von TZ ist. Dieser Summensatz kann gelöscht werden. Alle anderen beim Vorgang Kassenschnitt generierten Summensätze werden als Teil der einbezogenen Zahlungsdatengruppen gespeichert.

Anschließend wird Schritt 6. erneut durchgeführt.

An einem Akzeptanzterminal, das keinen Kassenschnitt durchführen kann, dürfen mit dieser Händlerkarte keine Transaktionen durchgeführt werden. An der Serviceschnittstelle ist anzuzeigen, daß die Händlerkarte ausgetauscht werden muß.

3.4. Zahlung

Vor Beginn einer Zahlung wird das Zahlungsmittel elektronische Geldbörse durch den Karteninhaber gewählt. In welcher Weise dies geschieht, ist nicht Gegenstand dieser Spezifikation.

Im folgenden Kapitel wird der Ablauf des Bezahlens eines vor der Warenausgabe feststehenden Zahlungsbetrages beschrieben.

In einem weiteren Kapitel wird ein modifizierter Ablauf skizziert, bei dem der Zahlungsbetrag erst später, nach der Ausgabe von Ware, festgelegt wird.

Wenn die Steueranwendung des Akzeptanzterminals erkennt, daß zwischen einzelnen Zahlungen möglicherweise ein Austausch bzw. Entnehmen und Wiedereinstecken der Händlerkarte erfolgt ist, beispielsweise nach einem Aus- und Wiederanschalten des Akzeptanzterminals, muß das Terminal mindestens die Konsistenzprüfungen für das EF_ID und für die aktuellen Summendaten der Händlerkarte mit den entsprechenden Reaktionen auf Inkonsistenzen aus Schritt 6. des Ablaufs aus Kapitel 3.3. durchführen.

3.4.1. Abbuchung vor Warenausgabe

3.4.1.1. Ablauf für den Karteninhaber

Der in Kapitel 3.4.1.2. detailliert beschriebene Ablauf einer GeldKarten-Zahlung ist so ausgelegt, daß die Zahlung durch den Karteninhaber **nicht** aufgrund des aktuellen Betrages seiner GeldKarte abgebrochen werden soll. Für den Karteninhaber stellen sich die im folgenden beschriebenen Varianten einer solchen Zahlung im Erfolgsfall an der Kundenanzeige wie folgt dar:

Bestätigung des Zahlungsbetrages durch Einstecken der GeldKarte:

An Terminals, deren Chipkartenleser nicht verriegelt werden kann:

1. **Zahlung DM: <Zahlungsbetrag>**
   Zum Bestätigen bitte Karte einstecken

2. **Bitte warten (Zahlung wird durchgeführt)**

3. **Zahlung erfolgt (Restguthaben DM: <Restguthaben>)**
   Bitte Karte entnehmen

An Terminals, deren Chipkartenleser verriegelt werden kann, auch:

1. **Zahlung DM: <Zahlungsbetrag>**
   Zum Bestätigen bitte Karte einstecken

2. **Guthaben DM: <aktueller Betrag>**
   Bitte warten (Zahlung wird durchgeführt)

3. **Zahlung erfolgt (Restguthaben DM: <Restguthaben>)**
   Bitte Karte entnehmen

Bestätigung des Zahlungsbetrages durch Bestätigungstaste:

An Terminals, deren Chipkartenleser nicht verriegelt werden kann:

1. **Bitte Karte einstecken**

2. **Zahlung DM: <Zahlungsbetrag>, bitte bestätigen**

3. **Bitte warten (Zahlung wird durchgeführt)**

4. **Zahlung erfolgt (Restguthaben DM: <Restguthaben>)**
   Bitte Karte entnehmen

An Terminals, deren Chipkartenleser verriegelt werden kann, auch:

1. **Bitte Karte einstecken**

2. **Zahlung DM: <Zahlungsbetrag>, bitte bestätigen**

3. **Guthaben DM: <aktueller Betrag>**
   Bitte warten (Zahlung wird durchgeführt)

4. **Zahlung erfolgt (Restguthaben DM: <Restguthaben>)**
   Bitte Karte entnehmen

Der in Kapitel 3.4.1.2. beschriebene Ablauf kann so modifiziert werden, daß der Karteninhaber die Möglichkeit erhält, die Zahlung aufgrund des aktuellen Betrages seiner GeldKarte abzubrechen. In diesem Fall ist die Bestätigung des Zahlungsbetrages durch Einstecken der GeldKarte nicht möglich. Dadurch würde nämlich ein regulärer Abbruch nach der Bestätigung zulässig, da die Anzeige des Guthabens erst nach dem Einstecken der Karte möglich ist.

Für den Karteninhaber würde sich eine solche Zahlung im Erfolgsfall an der Kundenanzeige sinngemäß wie folgt darstellen:

Zahlungsbetrag wird nach Anzeige des aktuellen Guthabens ausgewählt:

1. **Bitte Karte einstecken**

2. **Guthaben DM: <aktueller Betrag>**
   Bitte Wahl treffen oder abbrechen

3. **Zahlung DM: <Zahlungsbetrag>, bitte bestätigen**

4. **Bitte warten (Zahlung wird durchgeführt)**

5. **Zahlung erfolgt, (Restguthaben DM: <Restguthaben>)**
   Bitte Karte entnehmen

Zahlungsbetrag steht vor Anzeige des aktuellen Guthabens fest:

1. **Bitte Karte einstecken**

2. **Guthaben DM: <aktueller Betrag>**
   Zahlung DM: <Zahlungsbetrag>, bitte bestätigen

3. **Bitte warten (Zahlung wird durchgeführt)**

4. **Zahlung erfolgt, Restguthaben DM: <Restguthaben>**
   Bitte Karte entnehmen

Guthaben und Aufforderung zur Bestätigung des Zahlungsbetrags werden bei dieser Variante zusammen angezeigt. Auf diese Weise wird dem Karteninhaber zusammen mit der Anzeige des Guthabens eine reguläre Möglichkeit zum Abbruch geboten. Eine separate Anzeige des Guthabens vor dem Zahlungsbetrag ist zulässig, kann aber dazu führen, daß die GeldKarte ohne regulären Abbruch aus einem Chipkartenleser ohne Verriegelung gezogen wird.

3.4.1.2. Detaillierte Ablaufbeschreibung

Das folgende Diagramm zeigt die Schritte einer GeldKarten-Zahlung, die durch den Karteninhaber nicht aufgrund des aktuellen Guthabens abgebrochen werden soll.

GeldKarte			Händlerterminal		Händlerkarte	
			A1	Anzeige: Zahlungsbetrag, **Bitte Karte einstecken** Zahlungsbetrag speichern		
		<---	C2	Reset GeldKarte		
R2	ATR der GeldKarte	--->				
			C3	SELECT FILE DF_BÖRSE		
		<---				
R3	OK	--->				
		<---	C4	READ RECORD EF_ID		
R4	Daten aus EF_ID	--->				
			A4	Daten prüfen und speichern		
			C5	READ RECORD EF_BETRAG		
		<---				
R5	Beträge aus EF_BETRAG	--->				
			A5	Zahlungsbetrag prüfen (Anzeige: aktueller Betrag) aktuellen Betrag speichern		

GeldKarte			Händlerterminal			Händlerkarte	
			C6	GET CHALLENGE	--->		
					<---	R6	RND
		<---	C7	**Abbuchen einleiten** mit RND, KID des K_{RD}			
R7	BSEQ, RND, K_{RD}-Zertifikat	--->					
			C8	**Zahlung einleiten** mit BSEQ, RND, K_{RD}-Zertifikat, EF_ID, KID des K_{RD}	--->		
					<---	R8	BSEQ, Händlerkartennummer,HS EQ, SSEQ, K_{RD}-Zertifikat
		<---	C9	**Abbuchen** mit BSEQ, Händlerkartennummer, HSEQ, SSEQ, K_{RD}-Zertifikat, Zahlungsbetrag, Datum, Uhrzeit KID des K_{RD}			
R9	BSEQ, LSEQ, Zahlungsbetrag, Händlerkartennummer, HSEQ, Börsenverrechnungskonto, K_{RD}-Zertifikat, Restguthaben	--->					
			A9	Restguthaben speichern			

GeldKarte			Händlerterminal			Händlerkarte	
		C10	**Zahlung prüfen** mit BSEQ, LSEQ Zahlungsbetrag, Händlerkartennummer HSEQ, Börsenverrechnungskonto, K_{RD}-Zertifikat	--> <---	R10	OK
		A11	Ware ausgeben Anzeige: (Restguthaben) **Bitte Karte entnehmen** (Beleg drucken)			
		C12	**Zahlung** mit Datum, Uhrzeit	---> <---	R12	Zahlung mit K_{ZD}-Zertifikat, KV des K_{ZD}
		A13	Zahlung mit Zertifikat verlustfrei speichern			

Erläuterung

1. Der Zahlungsbetrag kann durch das Einstecken der GeldKarte (z. B. an einem Akzeptanzterminal ohne Bestätigungstaste) oder durch Betätigung der Bestätigungstaste bestätigt werden.

   Wenn der Zahlungsbetrag durch das Einstecken der GeldKarte bestätigt werden soll, wird am Kundendisplay angezeigt

   Zahlung DM: <Zahlungsbetrag>

   Zum Bestätigen bitte Karte einstecken

   Wenn der Zahlungsbetrag mit der Bestätigungstaste bestätigt werden soll, wird zunächst angezeigt

   Bitte Karte einstecken

   und nachdem die Karte eingesteckt wurde

   Zahlung DM: <Zahlungsbetrag>, bitte bestätigen

Erfolgt die Bestätigung des Zahlungsbetrages, wird dieser gespeichert und mit Schritt 2. der Zahlung fortgefahren.

Andernfalls wird die Zahlung abgebrochen. Optional wird hierzu angezeigt

Zahlung abgebrochen, bitte Karte entnehmen

2. Nachdem die GeldKarte eingesteckt ist, wird durch das Akzeptanzterminal ein Reset der Karte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer GeldKarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Wenn die GeldKarte einen korrekten ATR zurückgibt, wird mit Schritt 3. der Zahlung fortgesetzt.

   Andernfalls wird mit der folgenden Anzeige abgebrochen

   Karte nicht lesbar, bitte Karte entnehmen

3. Die Applikation elektronische Geldbörse wird geöffnet, indem das ADF der Applikation (DF_BÖRSE) durch das Terminal mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 45 50 01 00'	AID der elektronischen Geldbörse

Wenn das Kommando erfolgreich durchgeführt wurde, können die AEFs der Applikation mittels SFI referenziert werden. Es wird mit Schritt 4. der Zahlung fortgefahren.

Andernfalls wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

4. Das Terminal liest mittels READ RECORD die Kartenidentifikationsdaten im Record ’01’ des EF_ID im MF der GeldKarte (SFI ’17’). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’BC’	P2, Reference Control Byte
5	1	’16’	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der GeldKarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der GeldKarte
16-17	2	'0280'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Die Kodierung der empfangenen Daten wird geprüft:

Wenn die Prüfziffer nicht korrekt ist oder wenn Branchenhauptschlüssel, erste Ziffer der Kurz-BLZ, Verfalldatum, Aktivierungsdatum, Ländercode, Währungskennzeichen oder Wertigkeit der Währung nicht korrekt kodiert sind, bricht das Akzeptanzterminal mit der folgenden Meldung ab:

Kartendaten falsch, bitte Karte entnehmen

In allen anderen Fehlerfällen bei der Kommandoausführung wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Wenn das Akzeptanzterminal über eine Uhr verfügt, überprüft es das Aktivierungsdatum. Wenn aktuelles Datum < Aktivierungsdatum ist, bricht das Terminal mit der Meldung ab

Karte ungültig, bitte Karte entnehmen

Verlaufen alle Prüfungen positiv, werden die Daten des EF_ID gespeichert. Währungskennzeichen und Wertigkeit der Währung werden verwendet, um den Gegenwert der in der GeldKarte gespeicherten Betragswerte zu errechnen. Die möglichen Belegungen für die Wertigkeit der Währung und deren Bedeutung sind in Kapitel 6.9 von [LIT 1] beschrieben.

Anschließend wird mit Schritt 5. der Zahlung fortgefahren.

5. Es wird geprüft, ob der Zahlungsbetrag aus der elektronischen Geldbörse beglichen werden kann. Dazu liest das Terminal mittels READ RECORD den Record '01' des EF_BETRAG (SFI '18'). Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'09'	L_e, Recordlänge des EF_BETRAG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Wenn einer der drei Beträge nicht BCD-kodiert ist, gibt das Akzeptanzterminal die Meldung

aus:

Kartendaten falsch, bitte Karte entnehmen

In allen anderen Fehlerfällen bei der Kommandoausführung wird mit der folgenden Anzeige abgebrochen

Karte nicht lesbar, bitte Karte entnehmen

Das Terminal prüft, ob der Zahlungsbetrag weder den aktuellen Betrag noch den maximalen Transaktionsbetrag übersteigt. Ist der Zahlungsbetrag zu groß, bricht das Akzeptanzterminal die Zahlung mit der entsprechenden Meldung ab

Guthaben DM: <aktueller Betrag> reicht nicht zur Zahlung

Bitte Karte entnehmen

bzw.

Maximaler Zahlungsbetrag reicht nicht zur Zahlung

Bitte Karte entnehmen

Falls durch Position oder Größe der Anzeige nicht sichergestellt werden kann, daß sie nur für den Karteninhaber einsehbar ist, sollte das aktuelle Guthaben nicht angezeigt werden:

Guthaben reicht nicht zur Zahlung

Bitte Karte entnehmen

Andernfalls speichert das Terminal den aktuellen Betrag.

Wenn das Terminal über einen verriegelbaren Chipkartenleser verfügt, kann es den aktuellen Betrag anzeigen. Falls eine längere Zeit zur Durchführung der Schritte 6. bis 10. des Ablaufs erforderlich ist, sollte es zusätzlich eine Aufforderung zum Warten anzeigen:

Guthaben DM: <aktueller Betrag>

Bitte warten (Zahlung wird durchgeführt)

Wenn der Chipkartenleser des Terminals nicht verriegelt werden kann, wird der aktuelle Betrag an dieser Stelle nicht angezeigt. Wenn durch Position oder Größe der Anzeige nicht sichergestellt werden kann, daß sie nur für den Karteninhaber einsehbar ist, sollte der

aktuelle Betrag ebenfalls nicht angezeigt werden. Wenn eine längere Zeit zur Durchführung der Schritte 6. bis 10. des Ablaufs erforderlich ist, sollte aber angezeigt werden:

Bitte warten (Zahlung wird durchgeführt)

Anschließend wird mit Schritt 6. der Zahlung fortgefahren.

6. Das Akzeptanzterminal läßt sich mit dem Kommando GET CHALLENGE eine Zufallszahl RND von der Händlerkarte ausgeben. Die folgende Tabelle zeigt den Aufbau der Command-APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'84'	INS
3	1	'00'	P1, fester Wert
4	1	'00'	P2, fester Wert
5	1	'08'	L_e

Tritt bei der Kommandoausführung ein Fehler auf, wird mit der Meldung abgebrochen

Systemfehler, bitte Karte entnehmen

Vor Beginn einer neuen Transaktion mit der Händlerkarte sollten ein Reset der Händlerkarte und die erneute Selektion des DF_BSAM erfolgen.

Wenn das Kommando erfolgreich ausgeführt wurde, gibt die Händlerkarte eine 8 Byte lange Zufallszahl RND als Antwortdatum aus. Das Terminal fährt mit Schritt 7. der Zahlung fort.

7. Mit RND und der gespeicherten Schlüsselnummer KID des KGK_{RD} der Händlerkarte baut das Akzeptanzterminal die Kommandonachricht für **Abbuchen einleiten** auf und sendet sie an die GeldKarte. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘34’	INS
3	1	'00'	P1 für **Abbuchen einleiten**
4	1	‘00’	P2, fester Wert
5	1	'0A'	L_c
6	1	'40'	Nachrichten-ID für **Abbuchen einleiten**
7-14	8	'XX..XX'	Zufallszahl RND der Händlerkarte
15	1	'XX'	logische Schlüsselnummer KID des KGK_{RD}
16	1	‘13’	L_e

Bei erfolgreicher Kommandoausführung gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’41’	Nachrichten-ID für **Abbuchen einleiten**
2-3	2	‘XX XX’	Sequenznummer BSEQ aus EF_BSEQ
4-11	8	'XX..XX'	RND aus der Kommandonachricht
12-19	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 1-11|'00 00 00 00 00'

Das Terminal speichert in diesem Fall die Sequenznummer BSEQ und fährt mit Schritt 8. der Zahlung fort.

Tritt bei der Kommandoausführung ein Fehler auf, bricht das Terminal die Zahlung mit der folgenden Meldung ab

Chipfehler, bitte Karte entnehmen

8. Das Terminal baut mit den Antwortdaten der GeldKarte (Byte 6-24), den gespeicherten Daten des EF_ID der GeldKarte (Byte 25-46) und der gespeicherten Schlüsselnummer KID des KGK_{RD} (Byte 47) die Kommandonachricht eines **Zahlung einleiten** auf und schickt sie an die Händlerkarte. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'00'	P1 für **Zahlung einleiten**
4	1	‘00’	P2, fester Wert
5	1	'2A'	L_c
6	1	'41'	Nachrichten-ID für **Abbuchen einleiten** der GeldKarte
7-8	2	‘XX XX’	Sequenznummer BSEQ der GeldKarte
9-16	8	'XX..XX'	RND
17-24	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 6-16|'00 00 00 00 00'
25-46	22	'XX..XX'	EF_ID der GeldKarte
47	1	'XX'	logische Schlüsselnummer KID des KGK_{RD}
48	1	‘1D’	L_e

Bei erfolgreicher Verarbeitung des Kommandos gibt die Händlerkarte die folgenden Antwortdaten aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’50’	Nachrichten-ID für **Abbuchen** der GeldKarte
2-3	2	'XX XX'	Sequenznummer BSEQ aus der Kommandonachricht
4-13	10	'nn..nD'	Händlerkartennummer aus Byte 1-10 des EF_ID der Händlerkarte ('D' Hexziffer)
14-17	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
18-21	4	'XX..XX'	Summensequenznummer SSEQ der Händlerkarte
22-29	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 1-21|'00 00 00'

Das Terminal speichert in diesem Fall die Sequenznummer HSEQ und fährt mit Schritt 9. der Zahlung fort.

Tritt bei der Kommandoausführung ein Fehler auf, bricht das Terminal die Zahlung für den Karteninhaber mit der folgenden Meldung ab

Systemfehler, bitte Karte entnehmen

Wurde das Kommando mit einem negativen Returncode der Gruppe A beendet, wird die Händlerkarte außer Betrieb genommen.

Wurde der Returncode '96 C5' ausgegeben, muß ein Kassenschnitt durchgeführt werden, bevor weitere Transaktionen erfolgen können.

Wenn der Returncode '9F 01' oder '9F 05' ausgegeben wurde, wird eine Fehlzahlung durchgeführt (Schritt 1 und 2 in 3.5.), bevor weitere Transaktionen durchgeführt werden.

Wenn der Returncode '66 16' zurückgegeben wird, werden vor der nächsten Transaktion Schritt 1-3 aus 3.3. erneut ausgeführt.

Wenn ein anderer Returncode der Gruppe B ausgegeben wurde, werden vor der nächsten Transaktion ein Reset der Händlerkarte und die erneute Selektion des DF_BSAM durchgeführt (Schritt 1 und 2 in 3.3.).

In allen anderen Fehlerfällen der Gruppe B besteht Ungewißheit über den Ausgang der Kommandoausführung. In diesem Fall wird nach einem Reset der Händlerkarte und Selektion des DF_BSAM eine Fehlzahlung durchgeführt, bevor weitere Transaktionen mit der Händlerkarte durchgeführt werden.

9. Mit den Antwortdaten des **Zahlung einleiten** (Byte 6-34), dem gespeicherten Zahlungsbetrag (Byte 35-37), Datum und Uhrzeit (Byte 38-44) und der Schlüsselnummer KID des KGK_{RD} (Byte 45) baut das Akzeptanzterminal die Kommandonachricht des **Abbuchen** auf und sendet sie an die GeldKarte. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘34’	INS
3	1	'80'	P1 für **Abbuchen**
4	1	‘00’	P2, fester Wert
5	1	‘28’	L_c
6	1	'50'	Nachrichten-ID für **Abbuchen**
7-8	2	'XX XX'	Sequenznummer BSEQ der Transaktion
9-18	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer)
19-22	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
23-26	4	'XX..XX'	Sequenznummer SSEQ der Händlerkarte
27-34	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 6-26|'00 00 00'
35-37	3	'nn..nn'	Zahlungsbetrag des Händlerterminals
38-41	4	JJJJ MM TT	Datum des Händlerterminals
42-44	3	HH MM SS	Uhrzeit des Händlerterminals
45	1	'XX'	logische Schlüsselnummer KID des K_{RD}
46	1	‘2B’	L_e

Datum und Uhrzeit können mit 0 belegt werden, wenn das Akzeptanzterminal keine Uhr besitzt.

Bei erfolgreicher Kommandoausführung gibt die GeldKarte die folgenden Antwortdaten aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’51’	Nachrichten-ID für **Abbuchen**
2-3	2	'XX XX'	Sequenznummer BSEQ der Transaktion
4-5	2	'XX XX'	Sequenznummer LSEQ des letzten erfolgreichen Ladens
6-8	3	'nn..nn'	geprüfter Zahlungsbetrag
9-18	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer) aus der Kommandonachricht
19-22	4	'XX..XX'	Sequenznummer HSEQ aus der Kommandonachricht
23-32	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
33-40	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über Byte 1-32
41-43	3	'nn..nn'	neuer aktueller Betrag

In diesem Fall speichert das Terminal den neuen aktuellen Betrag (Restguthaben) und fährt

mit Schritt 10. der Zahlung fort.

Wenn die GeldKarte einen negativen Returncode ≠ ’65 81’ ausgibt, wurde das Abbuchen nicht ausgeführt. In diesem Fall beendet das Terminal die Zahlung für den Karteninhaber mit der Anzeige

Chipfehler, bitte Karte entnehmen

In allen anderen Fehlerfällen besteht Ungewißheit über den Ausgang des **Abbuchen**. In diesen Fällen führt das Terminal ein Reset der GeldKarte aus und selektiert anschließend das DF_BÖRSE erneut (vgl. Schritt 2. und 3. der Zahlung).

Hierbei wird im Fehlerfall die Zahlung für den Karteninhaber mit der folgenden Meldung beendet

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Anschließend fordert das Terminal mit **Zahlungsdaten wiederholen** die erneute Ausgabe der Antwortdaten des **Abbuchen** an. Die folgende Tabelle zeigt die Command APDU von **Zahlungsdaten wiederholen**:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘38’	INS
3	1	'20'	P1 für **Zahlungsdaten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_e

Wenn das Kommando nicht erfolgreich ausgeführt wird, besteht weiterhin Ungewißheit darüber, ob der Zahlungsbetrag abgebucht wurde. Daher beendet das Terminal die Zahlung für den Karteninhaber mit der Meldung

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Wenn das Kommando erfolgreich durchgeführt wird, prüft das Terminal, ob es sich um die Antwortdaten des von ihm abgesetzten Kommandos **Abbuchen** handelt:

- Byte 1 muß den Wert '51' haben,
- in Byte 2-3 muß die von dem Terminal in Schritt 7 gespeicherte Sequenznummer BSEQ enthalten sein.

Hat Byte 1 nicht den Wert '51' oder ist in Byte 2-3 nicht die gespeicherte BSEQ enthalten, wurde das Kommando **Abbuchen** durch die GeldKarte nicht ausgeführt. In diesem Fall wird die Zahlung für den Karteninhaber mit folgender Meldung beendet

Chipfehler, bitte Karte entnehmen

In jedem der beschriebenen Fehlerfälle wird anschließend ein Reset der Händlerkarte, die erneute Selektion des DF_BSAM und eine Fehlzahlung durchgeführt.

10. Mit dem Kommando **Zahlung prüfen** übergibt das Akzeptanzterminal die Antwortdaten des **Abbuchen** (Byte 6-45 der Command APDU) der Händlerkarte zur Prüfung, ob tatsächlich eine Abbuchung aus der GeldKarte erfolgt ist. Hierzu baut es die folgende Command APDU auf und schickt sie an die Händlerkarte:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'20'	P1 für **Zahlung prüfen**
4	1	‘00’	P2, fester Wert
5	1	'28'	L_c
6	1	'51'	Nachrichten-ID für **Abbuchen** der GeldKarte
7-8	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
9-10	2	'XX XX'	Sequenznummer LSEQ der GeldKarte
11-13	3	'nn..nn'	durch die GeldKarte geprüfter Zahlungsbetrag
14-23	10	'nn..nD'	Händlerkartennummer ('D' Hexziffer)
24-27	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
28-37	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos der GeldKarte ('D' Hexziffer)
38-45	8	‘XX..XX’	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über Byte 6-37

Wenn die Händlerkarte einen positiven Returncode zurückgibt, fährt das Terminal mit Schritt 11. der Zahlung fort.

Wurde das Kommando mit einem negativen Returncode der Gruppe A beendet, kann keine Rückbuchung des abgebuchten Betrages erfolgen, da die Händlerkarte außer Betrieb genommen wird. Daher wird die Zahlung für den Karteninhaber mit der Anzeige beendet

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Gibt die Händlerkarte den Returncode '97 02' (Transaktionsbetrag zu groß) aus, wird eine Fehlzahlung mit Rückbuchung durchgeführt.

Liegt ein anderer Fehlerfall der Gruppe B vor, führt das Terminal ein Reset der Händlerkarte durch, selektiert das DF_BSAM erneut und schickt das Kommando **Zahlung prüfen** erneut an die Händlerkarte.

Wenn die Händlerkarte jetzt einen positiven Returncode zurückgibt, fährt das Terminal mit Schritt 11. der Zahlung fort.

Wird die Wiederholung des Kommandos mit einem negativen Returncode der Gruppe A beendet, wird die Zahlung für den Karteninhaber mit der Anzeige beendet

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Wird bei Ausführung der Wiederholung von **Zahlung prüfen** der Returncode ’9F 05’ ausgegeben, ist zuvor ein **Zahlung prüfen** erfolgreich ausgeführt worden.

Das Terminal liest dann mittels READ RECORD die Protokolldaten der letzten Transaktion im Record '01' des EF_HLOG der Händlerkarte (SFI '1C'), um festzustellen, ob es sich um das **Zahlung prüfen** der aktuellen Zahlung handelt. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'E4'	P2, Reference Control Byte
5	1	'38'	L_e, Recordlänge des EF_HLOG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'XX'	Statusbyte
2-5	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
6-9	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
10-31	22	'XX..XX'	EF_ID der GeldKarte
32-33	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
34-35	2	'XX XX'	Sequenznummer LSEQ der GeldKarte
36-38	3	'nn..nn'	Transaktionsbetrag
39-48	10	'nn..nD'	Daten des Börsenverrechnungskontos der GeldKarte
49-52	4	JJJJ MM TT	Datum der Transaktion
53-55	3	HH MM SS	Uhrzeit der Transaktion
56	1	'XX'	Schlüsselnummer KID des zur Zertifikatsberechnung verwendeten KGK_{RD}

Das Terminal wertet Byte 1 und Byte 6-9 des Records aus.

- Byte 1 muß den Wert '05' haben.
- Byte 6-9 müssen die in Schritt 8. gespeicherte Sequenznummer HSEQ enthalten.

Ist das der Fall, fährt das Terminal mit Schritt 11. der Zahlung fort.

In allen Fehlerfällen, führt das Händlerterminal ein Reset der Händlerkarte durch, selektiert das DF_BSAM und führt eine Fehlzahlung mit Rückbuchung durch (Kapitel 3.5.).

11. Wenn durch das Kommando **Zahlung prüfen** angezeigt wird, daß der Zahlungsbetrag aus der GeldKarte abgebucht wurde und mit dem Kommando **Zahlung** eine Zahlung mit

Zertifikat für den Händler erzeugt werden kann, ist die Zahlung für den Karteninhaber beendet.

Wenn die Ware ausgegeben werden kann, wird dem Karteninhaber angezeigt

Zahlung erfolgt, Restguthaben DM: <Restguthaben>

Bitte Karte entnehmen

Wenn durch Position oder Größe der Anzeige nicht gewährleistet werden kann, daß nur der Karteninhaber das Restguthaben sieht, sollte das Restguthaben nicht angezeigt werden:

Zahlung erfolgt, bitte Karte entnehmen

Eventuell wird ein Beleg gedruckt. Anschließend wird mit Schritt 12. der Zahlung fortgefahren.

Wenn die Ausgabe der Ware nicht möglich ist, wird eine Fehlzahlung mit Rückbuchung durchgeführt (Kapitel 3.5.).

12. Das Akzeptanzterminal baut die Kommandonachricht für **Zahlung** auf und sendet sie an die Händlerkarte. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'80'	P1 für **Zahlung**
4	1	‘00’	P2, fester Wert
5	1	'07'	L_c
6-9	4	JJJJ MM TT	Datum des Händlerterminals
10-12	3	HH MM SS	Uhrzeit des Händlerterminals
13	1	‘37’	L_e

Datum und Uhrzeit können mit 0 belegt werden, wenn das Akzeptanzterminal keine Uhr besitzt.

Wenn das Kommando erfolgreich ausgeführt wird, gibt die Händlerkarte die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'E9'	EBCDIC-Kodierung des Buchstaben Z, Kennung für Zahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	'nn..nD'	GeldKartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
32-33	2	'XX XX'	Sequenznummer LSEQ der GeldKarte
34-36	3	'nn..nn'	Zahlungsbetrag
37-46	10	'nn..nD'	Daten des Börsenverrechnungskontos ('D': Hexziffer)
47-54	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 48 Byte Byte 1-46|'00 00'
55	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

In diesem Fall wird mit Schritt 13. der Zahlung fortgefahren.

Wenn die Händlerkarte mit einem Returncode der Gruppe A antwortet, ist die Händlerkarte nicht mehr funktionsfähig und wird außer Betrieb genommen.

Wenn die Händlerkarte mit dem Returncode '9F 01', '9F 31' oder '9F 35' antwortet, ist eine Zahlung nicht durchführbar. Diese Returncodes dürften nach einer erfolgreichen Durchführung der vorherigen Schritte nicht auftreten, so daß sie auf eine Fehlfunktion des Terminals oder der Händlerkarte hinweisen. Über das Auftreten eines solchen Returncodes muß daher an der Serviceschnittstelle informiert werden.

Das Terminal bricht die Zahlung ab und führt vor dem Beginn weiterer Transaktionen die Vorbereitung des Akzeptanzterminals (vgl. Kapitel 3.3.) durch.

In allen anderen Fehlerfällen der Gruppe B wird nach einem Reset der Händlerkarte und erneuter Selektion des DF_BSAM versucht, das Kommando **Zahlung** erneut auszuführen, indem die Kommandonachricht noch einmal an die Händlerkarte geschickt wird.

Lassen sich Reset oder SELECT FILE auch nach Wiederholung nicht erfolgreich ausführen, muß die Händlerkarte ausgetauscht werden.

Wird das Kommando **Zahlung** bei erneutem Aufruf erfolgreich ausgeführt, wird mit Schritt 13. der Zahlung fortgefahren.

Bei Ausgabe des Returncodes ’9F 31’ (vorher hat **Zahlung** stattgefunden) läßt sich das Terminal mit dem Kommando **Antwort wiederholen** die Antwortdaten dieser **Zahlung** erneut ausgeben. Die folgende Tabelle zeigt den Aufbau der Command APDU von **Antwort wiederholen**:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'60'	P1 für **Antwort wiederholen**
4	1	‘01’	P2, Recordnummer '01'
5	1	‘37’	L_e

Wenn das Kommando erfolgreich bearbeitet wird, fährt das Terminal mit Schritt 13. der Zahlung fort.

Führen diese Maßnahmen nicht zur Ausgabe einer Zahlung mit Zertifikat, kann durch weitere terminalspezifische Maßnahmen versucht werden, die Zahlung erfolgreich zu beenden. Mißlingen diese Maßnahmen, ist das Akzeptanzterminal bis zum Einstecken einer neuen Händlerkarte nicht mehr zur Abwicklung von Zahlungen mit der elektronischen Geldbörse einsetzbar. Hierüber muß an der Serviceschnittstelle informiert werden.

13. Das Akzeptanzterminal speichert die Antwortdaten des Kommandos **Zahlung** verlustfrei. Hierzu bereitet es die Daten eventuell durch Hinzufügen von Terminal-ID, Datum und Uhrzeit auf (vgl. Kapitel 3.2.).

3.4.2. Warenausgabe vor Abbuchung

Im folgenden Diagramm wird der Ablauf einer Zahlung aus einer GeldKarte dargestellt, bei der die Ware ausgegeben wird, bevor die Abbuchung aus der GeldKarte erfolgt.

GeldKarte			Händlerterminal		Händlerkarte	
			A1	Anzeige: **Bitte Karte einstecken**		
		<---	C2	Reset GeldKarte		
R2	ATR der GeldKarte	--->				
		<---	C3	SELECT FILE DF_BÖRSE		
R3	OK	--->				
		<---	C4	READ RECORD EF_ID		
R4	Daten aus EF_ID	--->				
			A4	Daten prüfen und speichern		
			C5	READ RECORD EF_BETRAG		
		<---				
R5	Beträge aus EF_BETRAG	--->				
			A5	Beträge prüfen und speichern Anzeige: aktueller Betrag		

GeldKarte			Händlerterminal			Händlerkarte	
			C6	GET CHALLENGE	--->		
					<---	R6	RND
		<---	C7	**Abbuchen einleiten** mit RND, KID des K_{RD}			
R7	BSEQ, RND, K_{RD}-Zertifikat	--->					
			C8	**Zahlung einleiten** mit BSEQ, RND, K_{RD}-Zertifikat, EF_ID, KID des K_{RD}	--->		
						R8	BSEQ, Händlerkartennummer,HSEQ, SSEQ, K_{RD}-Zertifikat
					<---		
			A9	Ware ausgeben, Zahlungsbetrag bestimmen Anzeige: Zahlungsbetrag			
			C9	**Abbuchen** mit BSEQ, Händlerkartennummer, HSEQ, SSEQ, K_{RD}-Zertifikat, Zahlungsbetrag, Datum, Uhrzeit KID des K_{RD}			
		<---					
R9	BSEQ, LSEQ, Zahlungsbetrag, Händlerkartennummer, HSEQ, Börsenverrechnungskonto, K_{RD}-Zertifikat, Restguthaben						
		--->					
			A9	Restguthaben speichern			

GeldKarte		Händlerterminal			Händlerkarte	
		C10	**Zahlung prüfen** mit BSEQ, LSEQ Zahlungsbetrag, Händlerkartennummer HSEQ, Börsenverrechnungskonto, K_{RD}-Zertifikat	--> <---	R10	OK
		A11	Anzeige: (Restguthaben) **Bitte Karte entnehmen** (Beleg drucken)			
		C12	**Zahlung** mit Datum, Uhrzeit	---> <---	R12	Zahlung mit K_{ZD}-Zertifikat, KV des K_{ZD}
		A13	Zahlung mit Zertifikat verlustfrei speichern			

Erläuterung

Im folgenden werden nur die Abweichungen von dem bereits für eine Zahlung mit vorher feststehendem Zahlungsbetrag beschriebenen Verfahren erläutert.

1. Es wird zu Beginn angezeigt

   Bitte Karte einstecken

5. Das Terminal prüft, welcher Betrag zur Zahlung zur Verfügung steht und speichert ihn gegebenenfalls. Dies ist der aktuelle Betrag bzw. der maximale Transaktionsbetrag, wenn dieser kleiner ist als der aktuelle Betrag. Ist der zur Verfügung stehende Betrag nicht ausreichend, bricht das Akzeptanzterminal die Zahlung mit der entsprechenden Meldung ab

   Guthaben DM: <aktueller Betrag> reicht nicht zur Zahlung

Bitte Karte entnehmen

bzw.

Maximaler Zahlungsbetrag reicht nicht zur Zahlung

Bitte Karte entnehmen

Andernfalls speichert das Terminal den aktuellen Betrag, zeigt ihn an:

Guthaben DM: <aktueller Betrag>

und fährt mit Schritt 6. der Zahlung fort.

9. Nachdem sich die Händlerkarte in Schritt 8. von der Authentizität der GeldKarte überzeugt hat, kann Ware im Wert bis zu dem zur Verfügung stehenden Zahlungsbetrag ausgegeben werden. Der so ermittelte Zahlungsbetrag wird dem Karteninhaber angezeigt

Zahlungsbetrag DM: <Zahlungsbetrag>

und in Byte 35-37 der Kommandonachricht des **Abbuchen** eingestellt.

Da die Ware bereits ausgegeben wurde, wird bei diesem Verfahren die Zahlung nicht sofort abgebrochen, wenn bei **Abbuchen** ein Fehlerfall eintritt und die Daten der GeldKarte nicht geändert wurden. Statt dessen versucht das Terminal, das **Abbuchen** erneut durchzuführen:

Wenn die GeldKarte einen negativen Returncode ≠ '65 81' ausgibt, wurde das **Abbuchen** nicht ausgeführt. In diesem Fall schickt das Terminal das **Abbuchen** nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE noch einmal an die GeldKarte. Im Erfolgsfall wird mit Schritt 11. fortgefahren. Gibt die GeldKarte wieder einen negativen Returncode ≠ '65 81' aus, bricht das Terminal mit der Anzeige ab

Chipfehler, bitte Karte entnehmen

In allen anderen Fehlerfällen besteht bei der ersten und zweiten Ausführung des **Abbuchen** Ungewißheit über den Ausgang des **Abbuchen**. In diesen Fällen führt das Terminal ein Reset der GeldKarte aus und selektiert anschließend das DF_BÖRSE erneut (Schritt 2. und 3. der Zahlung mit der dort beschriebenen Reaktion auf Fehler).

Anschließend fordert das Terminal mit **Zahlungsdaten wiederholen** die erneute Ausgabe der Antwortdaten des **Abbuchen** an.

10. Die Kundenanzeige ist in diesem Fall

    Systemfehler, bitte Karte entnehmen

11. Wenn durch das Kommando **Zahlung prüfen** angezeigt wird, daß der Zahlungsbetrag aus der GeldKarte abgebucht wurde und mit dem Kommando **Zahlung** eine Zahlung mit Zertifikat für den Händler erzeugt werden kann, ist die Zahlung für den Karteninhaber beendet. Es wird angezeigt

    Zahlung erfolgt, Restguthaben DM: <Restguthaben>

    Bitte Karte entnehmen

    Wenn durch Position oder Größe der Anzeige nicht gewährleistet werden kann, daß nur der Karteninhaber das Restguthaben sieht, sollte das Restguthaben nicht angezeigt werden:

    Zahlung erfolgt, bitte Karte entnehmen

3.5. Fehlzahlung und Rückbuchung

Wenn es nicht möglich ist, eine von der Händlerkarte begonnene Transaktion regulär zu beenden, muß die Transaktion mit einer **Fehlzahlung** für die Händlerkarte zu Ende gebracht werden, bevor eine neue Transaktion begonnen werden kann.

Hat im Lauf der abzubrechenden Transaktion schon eine Abbuchung aus der beteiligten GeldKarte stattgefunden, muß der abgebuchte Zahlungsbetrag im Anschluß an die Fehlzahlung mit **Rückbuchen** in die GeldKarte rückgebucht werden. Hierzu werden die benötigten Kommandodaten mit **Daten für Rückbuchung** von der Händlerkarte angefordert.

Das folgende Diagramm zeigt die erforderlichen Schritte.

GeldKarte			Händlerterminal			Händlerkarte	
			C1	**Fehlzahlung** mit Datum, Uhrzeit	--->	R1	Fehlzahlung mit K_{ZD}-Zertifikat, KV des K_{ZD}
					<---		
			A2	Fehlzahlung mit Zertifikat verlustfrei speichern			
			C3	**Daten für Rückbuchung**	--->	R3	Händlerkartennummer, HSEQ, K_{RD}-Zertifikat
					<---		
		<---	C4	**Rückbuchen** mit Händlerkartennummer,HSEQ, K_{ZD}-Zertifikat, Datum, Uhrzeit			
R4	OK	--->					
			A5	Anzeige: **Zahlung nicht möglich**			

Erläuterung

Die im folgenden beschriebenen Anzeigetexte werden nur dann angezeigt, wenn es sich um eine Fehlzahlung mit Rückbuchung handelt. Wird **Fehlzahlung** nur ausgeführt, um eine Transaktion für die Händlerkarte zu beenden, entfallen die Anzeigetexte, da hieran keine GeldKarte beteiligt ist.

1. Das Akzeptanzterminal baut die Kommandonachricht für **Fehlzahlung** auf und sendet sie an die Händlerkarte. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'A0'	P1 für **Fehlzahlung**
4	1	‘00’	P2, fester Wert
5	1	'07'	L_c
6-9	4	JJJJ MM TT	Datum des Händlerterminals
10-12	3	HH MM SS	Uhrzeit des Händlerterminals
13	1	‘28’	L_e

Datum und Uhrzeit können mit 0 belegt werden, wenn das Akzeptanzterminal keine Uhr besitzt.

Bei erfolgreicher Kommandoausführung gibt die Händlerkarte die folgenden Antwortdaten aus:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'C6'	EBCDIC-Kodierung des Buchstaben F, Kennung für Fehlzahlung
2-11	10	'nn..nD'	Händlerkartennummer ('D': Hexziffer)
12-15	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes der Transaktion
16-19	4	'XX..XX'	Sequenznummer HSEQ der Transaktion
20-29	10	'nn..nD'	GeldKartennummer ('D': Hexziffer)
30-31	2	'XX XX'	Sequenznummer BSEQ der GeldKarte
32-39	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{ZD} über die 32 Byte Byte 1-31|'00'
40	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

In diesem Fall fährt das Terminal mit Schritt 2. von Fehlzahlung und Rückbuchung fort.

Wenn die Händlerkarte mit einem Returncode der Gruppe A abbricht, ist die Händlerkarte nicht mehr funktionsfähig und muß außer Betrieb genommen werden.

Wenn die Händlerkarte mit dem Returncode '9F 31' oder '9F 35' antwortet, befindet sie sich in einem Zustand, in dem neue Transaktionen, aber keine Fehlzahlung bearbeitet werden können.

In beiden Fällen wird die Transaktion für den Karteninhaber mit der Meldung abgebrochen

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

In allen anderen Fehlerfällen wird nach einem Reset und erneuter Selektion des DF_BSAM versucht, das Kommando **Fehlzahlung** erneut auszuführen, indem die Kommandonachricht noch einmal an die Händlerkarte geschickt wird.

Bei erfolgreicher Kommandoausführung des erneuten **Fehlzahlung** wird mit Schritt 2. von

Fehlzahlung und Rückbuchung fortgefahren.

Bei Ausgabe des Returncodes '9F 35' (vorher hat **Fehlzahlung** stattgefunden) läßt sich das Terminal mit dem Kommando **Antwort wiederholen** die Antwortdaten dieser **Fehlzahlung** erneut ausgeben. Die folgende Tabelle zeigt den Aufbau der Command APDU von **Antwort wiederholen**:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'60'	P1 für **Antwort wiederholen**
4	1	‘01’	P2, Recordnummer '01'
5	1	‘28’	L_e

Wenn das Kommando erfolgreich bearbeitet wird, fährt das Terminal mit Schritt 2. von Fehlzahlung und Rückbuchung fort.

Führen diese Maßnahmen nicht zur Ausgabe einer Fehlzahlung mit Zertifikat, kann durch weitere terminalspezifische Maßnahmen versucht werden, die Fehlzahlung erfolgreich zu beenden. Mißlingen diese Maßnahmen, ist das Akzeptanzterminal bis zum Einstecken einer neuen Händlerkarte nicht mehr zur Abwicklung von Zahlungen mit der elektronischen Geldbörse einsetzbar.

Die Transaktion wird für den Karteninhaber mit der Meldung abgebrochen

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

2. Das Akzeptanzterminal speichert die Antwortdaten des Kommandos **Fehlzahlung** verlustfrei. Eventuell bereitet es die Daten hierzu durch Hinzufügen von Zahlungsbetrag, Terminal-ID, Datum und Uhrzeit auf (vgl. Kapitel 3.2.).

3. Das Terminal fordert mit dem Kommando **Daten für Rückbuchung** Daten von der Händlerkarte an, mit denen ein **Rückbuchen** des abgebuchten Zahlungsbetrages in die GeldKarte erfolgen kann. Hierzu baut das Terminal die in der folgenden Tabelle dargestellte Command APDU auf und sendet sie an die Händlerkarte:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘40’	INS
3	1	'40'	P1 für **Daten für Rückbuchung**
4	1	‘00’	P2, fester Wert
5	1	‘17’	L_e

Bei erfolgreicher Kommandoausführung gibt die Händlerkarte die folgenden Antwortdaten aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’70’	Nachrichten-ID für **Rückbuchen** der GeldKarte
2-11	10	’nn..nD’	Händlerkartennummer Byte 1-10 des aus EF_ID ('D' Hexziffer)
12-15	4	'XX..XX'	Sequenznummer HSEQ aus Byte 6-9 des Record '01' im EF_HLOG
16-23	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 1-15|'00'

In diesem Fall fährt das Terminal mit Schritt 4. von Fehlzahlung und Rückbuchung fort.

Gibt die Händlerkarte einen Returncode der Gruppe A zurück, wird die Händlerkarte außer Betrieb genommen.

Die Transaktion wird für den Karteninhaber mit der Meldung abgebrochen

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

Bei einem Fehlerfall der Gruppe B wird das Kommando **Daten für Rückbuchung** nach einem Reset der Händlerkarte und erneuter Selektion des DF_BSAM erneut ausgeführt.

Im Erfolgsfall wird mit Schritt 4. von Fehlzahlung und Rückbuchung fortgefahren. Andernfalls wird die Transaktion für den Karteninhaber mit der Meldung abgebrochen

Systemfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

4. Mit den Antwortdaten des Kommandos **Daten für Rückbuchung** (Byte 6-28) sowie Datum und Uhrzeit (Byte 29-35) baut das Terminal die Kommandonachricht für **Rückbuchen** auf und sendet sie an die GeldKarte. Die folgende Tabelle zeigt die Struktur der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘36’	INS
3	1	'80'	P1, Control Byte
4	1	‘00’	P2, fester Wert
5	1	‘1E’	L_c
6	1	'70'	Nachrichten-ID für **Rückbuchen**
7-16	10	’nn..nD’	Händlerkartennummer ('D' Hexziffer)
17-20	4	'XX..XX'	Sequenznummer HSEQ der Händlerkarte
21-28	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 16 Byte Byte 6-20|'00'
29-32	4	JJJJ MM TT	Datum des Händlerterminals
33-35	3	HH MM SS	Uhrzeit des Händlerterminals
36	1	'04'	L_e

Datum und Uhrzeit können mit '00' belegt werden, wenn das Terminal keine Uhr besitzt.

Bei erfolgreicher Bearbeitung des Kommandos gibt die GeldKarte die folgenden Antwortdaten aus:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	’71’	Nachrichten-ID für **Rückbuchen**
2-4	3	’nn..nn’	neuer aktueller Betrag

In diesem Fall wird die Zahlung für den Karteninhaber mit der Meldung beendet

Zahlung nicht möglich, Guthaben DM: <neuer aktueller Betrag>

Bitte Karte entnehmen

Im Fehlerfall versucht das Terminal nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE, das Kommando **Rückbuchen** erneut aufzurufen.

Bei erfolgreicher Kommandoausführung wird dem Karteninhaber angezeigt

Zahlung nicht möglich, Guthaben DM: <neuer aktueller Betrag>

Bitte Karte entnehmen

Gibt die GeldKarte den Returncode '9F 71' zurück, wurde zuvor ein Rückbuchen ausgeführt. Mit dem Kommando **Zahlungsdaten wiederholen** läßt sich das Terminal die Antwortdaten dieses Rückbuchens erneut ausgeben. Die folgende Tabelle zeigt die Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	'E0'	CLA
2	1	'38'	INS
3	1	'20'	P1 für **Zahlungsdaten wiederholen**
4	1	'00'	P2, fester Wert
5	1	'04'	L_e

Wenn das Kommando erfolgreich ausgeführt wird, entnimmt das Terminal Byte 2-4 der Antwortdaten den durch das **Rückbuchen** berechneten neuen aktuellen Betrag. Das Terminal vergleicht diesen Betrag mit dem in Schritt 5. der Zahlung gespeicherten aktuellen Betrag vor der Abbuchung aus der GeldKarte. Sind die Beträge gleich, zeigt das Terminal an

Zahlung nicht möglich, Guthaben DM: <neuer aktueller Betrag>

Bitte Karte entnehmen

In allen anderen Fehlerfällen bei erneuter Ausführung des **Rückbuchen** bricht das Terminal die Transaktion mit der folgenden Meldung ab:

Chipfehler, wenden Sie sich an Ihr kartenausgebendes Institut

Bitte Karte entnehmen

4. Abläufe an einem Kassenschnitterminal

4.1. Regulärer Kassenschnitt

Wenn die seit dem letzten Kassenschnitt mit einer Händlerkarte erzeugten Einzeltransaktionen zur Abrechnung eingereicht werden sollen, muß hierzu ein Summensatz erzeugt und in einem neuen Record '01' des EF_SUMME der Händlerkarte die Summendaten für die Aufsummierung neuer Einzeltransaktionen mit einer um 1 inkrementierten Summensequenznummer SSEQ initialisiert werden. Dies wird in der Regel mittels des Kommandos **Kassenschnitt** der Händlerkarte

durchgeführt.

Das Kassenschnitterminal muß zur Durchführung des regulären Kassenschnittes eine Gruppe oder mehrere Gruppen von Zahlungs- und Fehlzahlungsdaten enthalten. Eine Gruppe von Zahlungs- und Fehlzahlungsdaten besteht aus den folgenden Datensätzen:

- Daten einer Händlerkarte, bestehend aus
  - Kartenidentifikationsdaten der Händlerkarte,
  - Händlerbankverbindung,
- Zahlungs- oder Fehlzahlungsdatensätzen mit
  - identischer Händlerkartennummer,
  - identischer Summensequenznummer SSEQ und
  - lückenlos aufsteigenden Sequenznummern HSEQ.

Eine Gruppe von Zahlungs- und Fehlzahlungsdaten enthält mindestens einen Zahlungs- oder Fehlzahlungsdatensatz. Die gemeinsame Händlerkartennummer in den Zahlungs- oder Fehlzahlungsdatensätzen muß mit der Händlerkartennummer in den Kartenidentifikationsdaten der zugehörigen Händlerkartendaten übereinstimmen.

Außerdem muß sich im Kassenschnitterminal eine Händlerkarte befinden.

In einem kombinierten Akzeptanz- und Kassenschnitterminal sind die Gruppen von Zahlungs- und Fehlzahlungsdaten bereits gespeichert. Die Händlerkarte befindet sich in der Regel bereits im Leser des kombinierten Akzeptanz- und Kassenschnitterminals.

In ein separates Kassenschnitterminal müssen die Gruppen von Zahlungs- und Fehlzahlungsdaten über eine geeignete Schnittstelle eingebracht werden. Eventuell ist es erforderlich, die eingegebenen Daten neu zu Gruppen zu sortieren, wenn mit derselben Händlerkarte an verschiedenen Akzeptanzterminals Zahlungs- oder Fehlzahlungsdaten erzeugt wurden. Die Händlerkarte ist in den Leser des Terminals einzustecken.

In den Vorgang Kassenschnitt werden alle Zahlungsdatengruppen einbezogen, zu denen bisher kein Summensatz erzeugt wurde. Zahlungsdatengruppen, zu denen ein Ersatzsummensatz gebildet wurde, ohne daß hierbei die Händlerkarte mit der zugehörigen Händlerkartennummer verwendet wurde, können ebenfalls in den Vorgang Kassenschnitt einbezogen werden.

Falls es sich bei dem Kassenschnitt um einen bedienten Vorgang handelt, kann das Kassenschnitterminal das sukzessive Einstecken von den Händlerkarten anfordern, die zur Summensatzbildung für die im Kassenschnitterminal enthaltenen Gruppen von Zahlungs- und Fehlzahlungsdaten benötigt werden. Die benötigten Händlerkarten werden durch ihre Händlerkartennummer eindeutig identifiziert.

Der im folgenden beschriebene Ablauf gilt gleichermaßen für separate Kassenschnitterminals wie für kombinierte Akzeptanz- und Kassenschnitterminals.

Zunächst wird ein Reset der Händlerkarte ausgeführt und das DF_BSAM selektiert (vgl. Schritt 1.

und 2 in Kapitel 3.3.).

Anschließend liest das Terminal mittels READ RECORD die Kartenidentifikationsdaten im Record '01' des EF_ID im MF der Händlerkarte (SFI '17'). Die folgende Tabelle zeigt den Aufbau der Command-APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’BC’	P2, Reference Control Byte
5	1	’16’	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’67’	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der Händlerkarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der Händlerkarte
16-17	2	'0280'	Ländercode
18-20	3	'nn nn nn'	Entgeltcode (Standardwert '00 00 00')
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Freier Zähler für die Schlüsselerzeugung

Das Terminal liest mittels READ RECORD die Summendaten im Record ’01’ des EF_SUMME (SFI ’19’). Die folgende Tabelle zeigt den Aufbau der Command-APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’CC’	P2, Reference Control Byte
5	1	’0D’	L_e, Recordlänge des EF_SUMME

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Händlerkarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	'XX..XX'	Sequenznummer SSEQ' des Summensatzes
5-8	4	'XX..XX'	Transaktionszähler TZ, Anzahl Transaktionen seit letztem **Kassenschnitt**
9-13	5	'nn..nn'	Summe der Zahlungsbeträge seit letztem **Kassenschnitt**

Für die Gruppen von Zahlungs- und Fehlzahlungsdaten, die die Kartennummer der Händlerkarte enthalten und für deren Summensequenznummer SSEQ gilt

SSEQ' - SSEQ > 2,

kann kein zertifizierter Summensatz mehr gebildet werden. Falls solche Gruppen im Terminal vorhanden sind, bildet das Kassenschnitterminal für sie Ersatzsummensätze, wie in Kapitel 4.2. beschrieben. Diese Zahlungsdatengruppen sollten besonders gekennzeichnet werden, da für sie eine erneute Einbeziehung in den Vorgang Kassenschnitt nicht sinnvoll ist. Bei jedem weiteren Kassenschnitt würde nur erneut der Ersatzsummensatz für diese Zahlungsdatengruppen erzeugt.

Falls eine der beiden Gruppen von Zahlungs- und Fehlzahlungsdaten, die die Kartennummer der Händlerkarte enthalten und für deren Summensequenznummer SSEQ gilt

SSEQ' = SSEQ +1 oder SSEQ' = SSEQ +2,

in dem Terminal vorhanden ist, werden mittels des Kommandos **Summensatz** die Summendaten mit Zertifikat für die Gruppe von der Händlerkarte angefordert. Die folgende Tabelle zeigt die Command-APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'E0'	CLA
2	1	'42'	INS
3	1	'20'	P1 für **Summensatz**
4	1	'02' oder '03'	P2, Recordnummer = SSEQ' - SSEQ + 1
5	1	'20'	L_e

Wenn das Kommando erfolgreich durchgeführt wird, gibt die Händlerkarte die folgenden Daten aus

Byte	Länge (in Byte)	Wert	Erläuterung
1-10	10	'nn..nD'	Händlerbankverbindung ('D' Hexziffer)
11-14	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
15-18	4	'XX..XX'	Anzahl aller zugehörigen Zahlungen und Fehlzahlungen
19-23	5	'nn..nn'	Summe der Zahlungsbeträge
24-31	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 1-23|'00'
32	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

Bei den im Erfolgsfall ausgegebenen Daten handelt es sich um die Summendaten mit Zertifikat, die die Positionen 3-6, 10 und 9 eines Summensatzes mit Zertifikat (vgl. Kapitel 2.4.2.) zur Einreichung bei der EZ bilden.

Wenn das Kommando **Summensatz** für eine der oben genannten beiden Gruppen von Zahlungs- und Fehlzahlungsdaten auch nach einem Reset der Händlerkarte, erneuter Selektion des DF_BSAM und Wiederholung des Kommandos nicht erfolgreich ausgeführt werden kann oder wenn die Anzahl der zugehörigen Zahlungen und Fehlzahlungen in Byte 15-18 der im Erfolgsfall ausgegeben Daten nicht mit der Anzahl der Zahlungen und Fehlzahlungen der Gruppe übereinstimmt, muß für die entsprechende Gruppe ein Ersatzsummensatz, wie in Kapitel 4.2. beschrieben, erzeugt werden.

Wenn der Transaktionszähler TZ aus dem gelesenen Record '01' des EF_SUMME den Wert 0 hat, kann der Vorgang Kassenschnitt für die Händlerkarte beendet werden. Eine eventuell noch nicht beendete erste Transaktion zu der aktuellen Summensequenznummer SSEQ' der Händlerkarte wird am Akzeptanzterminal in Schritt 8. einer Zahlung mit einer Fehlzahlung beendet.

Wenn der Transaktionszähler TZ aus dem gelesenen Record '01' des EF_SUMME verschieden von 0 ist, führt das Terminal das Kommando **Kassenschnitt** aus. Die folgende Tabelle zeigt die Command APDU des Kommandos:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'E0'	CLA
2	1	'42'	INS
3	1	'00'	P1 für **Kassenschnitt**
4	1	'00'	P2, fester Wert
5	1	'20'	L_e

Wenn das Kommando erfolgreich durchgeführt wird, gibt die Händlerkarte die folgenden Daten aus

Byte	Länge (in Byte)	Wert	Erläuterung
1-10	10	'nn..nD'	Händlerbankverbindung ('D' Hexziffer)
11-14	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
15-18	4	'XX..XX'	Anzahl aller zugehörigen Zahlungen und Fehlzahlungen
19-23	5	'nn..nn'	Summe der Zahlungsbeträge
24-31	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{RD} über die 24 Byte Byte 1-23|'00'
32	1	'XX'	Generationsnummer KV des verwendeten K_{ZD}

Gleichzeitig initialisiert die Händlerkarte in einem neuen Record '01' des EF_SUMME die Summendaten für die Aufsummierung neuer Einzeltransaktionen mit einer um 1 inkrementierten Summensequenznummer SSEQ.

Bei den im Erfolgsfall ausgegebenen Daten handelt es sich um die Summendaten mit Zertifikat, die die Positionen 3-6, 10 und 9 eines Summensatzes mit Zertifikat (vgl. Kapitel 2.4.2.) zur Einreichung der Gruppe von Zahlungs- und Fehlzahlungsdaten, die die Kartennummer der Händlerkarte enthält und für deren Summensequenznummer SSEQ gilt

SSEQ' = SSEQ,

bei der EZ bilden.

Wenn das Kommando **Kassenschnitt** mit einem Returncode der Gruppe A abgebrochen wird, muß die Händlerkarte außer Betrieb genommen werden. Falls die Gruppe von Daten mit der Kartennummer der Händlerkarte und mit SSEQ' = SSEQ im Terminal vorhanden ist, muß in diesem Fall ein Ersatzsummensatz für die Gruppe erzeugt werden (vgl. Kapitel 4.2.).

Wenn das Kommando mit einem der Returncodes '9F 01' oder '9F 05' abbricht, ist noch eine Transaktion in der Händlerkarte offen. Diese muß zunächst durch eine **Fehlzahlung** beendet werden (vgl. Kapitel 3.5.). Die erzeugten Fehlzahlungsdaten werden der Gruppe von Daten mit SSEQ' = SSEQ hinzugefügt. Anschließend wird nach einem Reset der Händlerkarte und erneuter Selektion des DF_BSAM das Kommando **Kassenschnitt** noch einmal ausgeführt.

Wenn ein anderer Returncode der Gruppe B ausgegeben wird, wird nach einem Reset der Händlerkarte und erneuter Selektion des DF_BSAM das Kommando **Kassenschnitt** noch einmal ausgeführt.

Tritt in einem der beiden beschriebenen Fälle bei der Wiederholung des Kommandos **Kassenschnitt** wieder ein Fehler auf und ist die Gruppe von Daten mit der Kartennummer der Händlerkarte und mit SSEQ' = SSEQ im Terminal vorhanden, wird ein Ersatzsummensatz für die

Gruppe erzeugt (vgl. Kapitel 4.2.). Anschließend wird wiederholt versucht, das Kommando **Kassenschnitt** durch die Händlerkarte ausführen zu lassen. Treten hierbei immer wieder Fehler auf, muß die Händlerkarte außer Betrieb genommen werden.

In allen anderen Fehlerfällen bei der ersten Ausführung von **Kassenschnitt** besteht Ungewißheit darüber, ob das Kommando **Kassenschnitt** ausgeführt wurde oder nicht. Vor einer erneuten Ausführung von **Kassenschnitt** muß geprüft werden, ob durch das Kommando bereits ein neuer Record angelegt wurde.

Dies erfolgt dadurch, daß nach einem Reset der Händlerkarte und erneuter Selektion des DF_BSAM mittels READ RECORD der Record '01' des EF_SUMME (SFI '19') erneut gelesen wird. Die folgende Tabelle zeigt den Aufbau der Command APDU für READ RECORD.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'CC'	P2, Reference Control Byte
5	1	'0D'	L_e, Recordlänge des EF_SUMME

Bei erfolgreicher Ausführung des Kommandos gibt die Händlerkarte einen Record der folgenden Struktur aus:

Byte	Länge (in Byte)	Wert	Erläuterung
1-4	4	'XX..XX'	Sequenznummer SSEQ des Summensatzes
5-8	4	'XX..XX'	Transaktionszähler TZ, Anzahl Transaktionen seit letztem **Kassenschnitt**
9-13	5	'nn..nn'	Summe der Zahlungsbeträge seit letztem **Kassenschnitt**

Hat der ausgegebene Transaktionszähler TZ den Wert 0, muß es sich hierbei um den neu initialisierten Record handeln, da TZ bei dem vorherigen Lesen des Record '01' von 0 verschieden war.

Ist das der Fall und ist die Gruppe von Daten mit der Kartennummer der Händlerkarte und mit SSEQ' = SSEQ im Terminal vorhanden, müssen die für die Gruppe benötigten Summendaten mit Zertifikat mittels **Summensatz** aus dem Record '02' des EF_SUMME entnommen werden:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘42’	INS
3	1	'20'	P1 für **Summensatz**
4	1	‘02’	P2, Recordnummer
5	1	‘20’	L_e

Andernfalls kann **Kassenschnitt** erneut ausgeführt werden.

Tritt bei **Summensatz** oder **Kassenschnitt** wieder ein Fehler auf und ist die Gruppe von Daten mit der Kartennummer der Händlerkarte und mit SSEQ' = SSEQ im Terminal vorhanden, wird für diese Gruppe ein Ersatzsummensatz gebildet (vgl. Kapitel 4.2.). Anschließend wird wiederholt versucht, das Kommando **Kassenschnitt** durch die Händlerkarte ausführen zu lassen. Treten hierbei immer wieder Fehler auf, muß die Händlerkarte außer Betrieb genommen werden.

Ist die Gruppe von Daten mit der Kartennummer der Händlerkarte und mit SSEQ' = SSEQ im Terminal vorhanden und stimmt die Anzahl der zugehörigen Zahlungen und Fehlzahlungen in Byte 15-18 der mittels **Summensatz** oder **Kassenschnitt** im Erfolgsfall ausgegeben Daten nicht mit der Anzahl der Zahlungen und Fehlzahlungen der Gruppe überein, muß für die Gruppe ein Ersatzsummensatz, wie in Kapitel 4.2. beschrieben, erzeugt werden.

Um aus den mittels **Summensatz** oder **Kassenschnitt** im Erfolgsfall ausgegebenen Summendaten mit Zertifikat zu einer Gruppe von Daten von Zahlungen und Fehlzahlungen einen kompletten S-Satz aufzubauen, sind die Positionen 1-2, 7-8 und 11 zu belegen.

Position 1 ist mit der Kennung 'E2' für Summensatz mit Zertifikat zu belegen. In Position 2 müssen die Daten des EF_ID der Händlerkarte eingestellt werden. Diese Daten müssen durch das Kassenschnitterminal hinzugefügt und zusammen mit den Summendaten gespeichert werden.

Wenn Position 7 und 8 mit Datum und Uhrzeit der Summensatzbildung belegt werden sollen, müssen Datum und Uhrzeit durch das Kassenschnitterminal hinzugefügt und mit den Summendaten gespeichert werden.

Position 7 und 8 können mit '00' belegt werden, so daß die Speicherung der entsprechenden Daten durch das Kassenschnitterminal nicht obligatorisch ist.

Position 11 ist mit '00' zu füllen und kann zu einem späteren Zeitpunkt hinzugefügt werden, beispielsweise durch das Einreichungsterminal.

Falls im Terminal Gruppen von Zahlungsdaten enthalten sind, deren Händlerkartennummer nicht mit der Kartennummer der Händlerkarte übereinstimmt, kann dies an der Händlerschnittstelle des Terminals angezeigt und zum Einstecken einer Händlerkarte mit einer passenden Händlerkartennummer aufgefordert werden.

Wenn das Terminal keine Aufforderung zum Wechsel der Händlerkarte ausgeben kann oder wenn nach der Aufforderung eine entsprechende Eingabe des Händlers erfolgt, kann das Terminal Ersatzsummensätze zu den Gruppen von Zahlungs- und Fehlzahlungsdaten mit abweichender Händlerkartennummer erzeugen (vgl. Kapitel 4.2.).

4.2. Erzeugen eines Ersatzsummensatzes

Ein Ersatzsummensatz (M-Satz) wird dann erzeugt, wenn kein regulärer Summensatz (S-Satz) erzeugt werden kann.

Zur Erzeugung eines Ersatzsummensatzes (vgl. Kapitel 2.4.2.) benötigt das Kassenschnitterminal die Daten der Händlerkarte (EF_ID und Händlerbankverbindung) und sämtliche Zahlungsdaten und Fehlzahlungsdaten (vgl. Kapitel 3.2.), zu denen der Ersatzsummensatz gebildet werden soll.

Wenn das Kassenschnitterminal nicht gleichzeitig das Akzeptanzterminal ist, durch das die Händlerkartendaten sowie Zahlungs- und Fehlzahlungsdaten gespeichert wurden, müssen diese Daten an das Kassenschnitterminal über eine geeignete Schnittstelle übergeben werden.

Das Kassenschnitterminal prüft, ob alle Zahlungs- und Fehlzahlungsdaten dieselbe Sequenznummer SSEQ enthalten und sortiert die Datensätze nach aufsteigender Sequenznummer HSEQ. Hierbei werden doppelt vorkommende Datensätze aussortiert.

Anschließend werden die Transaktionsbeträge der Z-Sätze aufsummiert und die Gesamtzahl der Datensätze festgestellt.

Das Kassenschnitterminal fügt den Daten die Kennung 'D4', das EF_ID der Händlerkarte und die Händlerbankverbindung hinzu. Optional ergänzt es die Daten um Datum und Uhrzeit.

5. Abläufe an einem Einreichungsterminal

Über eine geeignete Schnittstelle werden an das Einreichungsterminal die durch Akzeptanzterminal und Kassenschnitterminal vorbereiteten Zahlungs-, Fehlzahlungs- und Summendaten einer oder mehrerer Händlerkarten übergeben.

Wenn dies nicht schon vorher geschehen ist, erzeugt das Einreichungsterminal aus diesen Daten vollständige S-Sätze, M-Sätze, Z-Sätze, und F-Sätze (vgl. Kapitel 2.4.2. und 2.4.3.), indem es die Positionen in die korrekte Ordnung bringt und die fehlenden Positionen mit '00' belegt. Die Datensätze werden sortiert, wie in Kapitel 2.4. beschrieben. Anschließend werden V-Satz (Kapitel

2.4.1.) und E-Satz (Kapitel 2.4.4.) erzeugt.

Zur Einreichung von Daten nur einer Händlerkarte werden keine Kennungen und Namen zum Aufbau des V-Satzes benötigt, sondern es kann die in Kapitel 2.4.1. beschriebene vereinfachte Variante eines V-Satzes verwendet werden.

Die so generierte BZAHL-Datei wird bei der zuständigen Händler-EZ eingereicht.

Sofern dies technisch möglich ist, bestätigt die Händler-EZ unmittelbar nach dem Eintreffen einer Händler-Einreichungsdatei deren Empfang mit einer Quittungsdatei. Hiermit ist keine Zahlungsgarantie verbunden.

Die Händler-EZ erstellt die Quittungsdatei, nachdem sie mindestens die folgenden Prüfungen der empfangenen Datei erfolgreich durchgeführt hat:

- Die Datei enthält einen V-Satz und einen E-Satz (Kennungen 'V' und 'E' vorhanden)
- Die Datei hat den Namen BZAHL (Prüfung von Byte 2-6 des V-Satzes)

Die Quittungsdatei hat den folgenden Aufbau:

1. V-Satz: Der V-Satz der Quittungsdatei wird aus dem V-Satz der empfangenen BZAHL wie folgt generiert:

   Byte 7-26 und Byte 27-46 (Absender- und Empfängerangaben) des V-Satzes der empfangenen Datei werden vertauscht. Byte 47-53 des empfangenen V-Satzes werden mit aktuellem Datum und Uhrzeit überschrieben. Alle anderen Positionen des empfangenen V-Satzes werden unverändert übernommen.

2. Original V-Satz der empfangenen BZAHL,

3. Original E-Satz der empfangenen BZAHL,

4. E-Satz: Identisch mit V-Satz dieser Quittungsdatei, lediglich das erste Byte wird auf 'E' gesetzt.

GeldKarte
– Bankensonderfunktionsterminals –

Version 2.2
22.01.1997

Inhalt

1. Überblick

Ein Bankensonderfunktionsterminal ist ein in einem Kreditinstitut betriebenes Terminal. Es kann eine oder mehrere der im folgenden beschriebenen Funktionalitäten bereitstellen. Zu jeder Funktionalität werden Anforderungen formuliert, die das Bankensonderfunktionsterminal erfüllen muß, wenn es diese Funktionalität bereitstellt.

Die Funktionalität eines Bankensonderfunktionsterminals kann in die folgenden Funktionsgruppen unterteilt werden:

1. Echtheitsprüfung von Chipkarten
2. Ausgabe von Chipdaten

3. Laden/Entladen/Bezahlen (GeldKarte)
4. Ändern von Chipdaten
5. Nachladen von Daten/Kommandos in den Chip
6. Erweiterung der Funktionalität des Bankensonderfunktionsterminals

Die folgende Tabelle gibt einen Überblick über die Sicherheitsanforderungen an die Funktionen eines Bankensonderfunktionsterminals.

Funktionsgruppe	**Sicherheitsanforderungen**
Echtheitsprüfung	Schutz von kryptographischen Schlüsseln Zugriffskontrolle Protokollierung Ablaufkontrolle
Ausgabe von Daten: frei lesbare Daten	 keine besonderen Sicherheitsanforderungen
lesbare Daten nach Authentikation bzw. Daten lesen inkl. MAC-Prüfung	Schutz von kryptographischen Schlüsseln Zugriffskontrolle evtl. Protokollierung Ablaufkontrolle

Funktionsgruppe	**Sicherheitsanforderungen**
Laden/Entladen/Bezahlen	Schutz von kryptographischen Schlüsseln Zugriffskontrolle evtl. Anforderungen an Online-Verbindung Protokollierung Ablaufkontrolle
Ändern von Daten	Identifikation und Authentikation des Bedieners Schutz von kryptographischen Schlüsseln Zugriffskontrolle evtl. Anforderungen an Online-Verbindung Protokollierung Ablaufkontrolle
Nachladen von Daten/Kommandos	Identifikation und Authentikation des Bedieners[1] Schutz von kryptographischen Schlüsseln Zugriffskontrolle evtl. Anforderungen an Online-Verbindung Protokollierung Ablaufkontrolle
Erweiterung der Funktionalität des Bankensonderfunktionsterminals	Identifikation und Authentikation von zwei Bedienern (Vier-Augen-Prinzip) Schutz von kryptographischen Schlüsseln Zugriffskontrolle evtl. Anforderungen an Online-Verbindung Protokollierung Ablaufkontrolle

1) Mit dem Begriff "Bediener" wird ein Bankangestellter bezeichnet, der das Bankensonderfunktionsterminal bedient, nicht der Kunde, dessen Chipkarte mit dem Terminal bearbeitet wird.

2. Erläuterung der Anforderungen

Im folgenden werden die einzelnen Anforderungen an ein Bankensonderfunktionsterminal erläutert. Es wird dabei zwischen Sicherheitsanforderungen und sonstigen Anforderungen unterschieden.

2.1. Elektrische Spezifikation und Übertragungsprotokoll

Die elektrische Spezifikation der Schnittstelle zwischen einem Bankensonderfunktionsterminal und der GeldKarte muß den Vorgaben aus [ISO 4], [ISO 4'], [ISO 4"] und [EMV 1] genügen.

Das Übertragungsprotokoll zwischen Bankensonderfunktionsterminal und GeldKarte ist T=1 nach [ISO 4'], [ISO 4"] und [EMV 1].

Zur Behandlung der Unterschiede der Spezifikation von T=1 in [EMV 1] einerseits und [ISO 4']

andererseits werden die folgenden Festlegungen gemacht.

2.1.1. NAD

Gemäß [EMV 1] müssen Terminals NAD='00' verwenden. Andere Werte sind nicht zulässig. Die GeldKarte lehnt daher NAD≠'00' als Protokollfehler ab.

Daher müssen alle vom Terminal gesendeten Blöcke stets NAD='00' enthalten.

2.1.2. Länge (LEN)

Minimale Blocklänge

Gemäß [EMV 1] wird das Senden und Empfangen von I-Blöcken mit LEN=0 nicht unterstützt. Eine GeldKarte kann somit den Empfang eines I-Blocks mit LEN=0 als einen Protokollfehler abweisen.

Daher darf das Terminal keinen I-Block mit LEN=0 senden.

IFSC und IFSD

Die GeldKarte teilt ihr IFSC im ATR mit. Zur Zeit hat IFSC mindestens den Wert 60. Größere Werte von IFSC sind zulässig.

Gemäß [ISO 4'] muß ein Terminal nur den Standardwert IFSD=32 unterstützen und kann mit einem S(IFS request) IFSD<IFSC erreichen.

Gemäß [EMV 1] muß ein Terminal IFSD=254 für das Empfangen von Blöcken unterstützen. Ein S(IFS request) des Terminals mit IFSD<IFSC ist nicht zulässig. Die GeldKarte kann immer Blöcke mit LEN≤IFSC senden.

Daher muß das Terminal für Sitzungen mit GeldKarten IFSD≥IFSC für alle Typen von GeldKarten unterstützen, auch für solche, die im ATR den Wert IFSC=254 angeben.

Dies kann "stillschweigend" geschehen, indem das Terminal Blöcke mit 32<LEN≤IFSC ohne Fehlermeldung akzeptiert, oder indem es als ersten Block nach dem Empfang des ATR ein S(IFS request) sendet mit IFSC≤IFSD≤254.

Chaining

Gemäß [EMV 1] gilt für das Chaining vom Terminal zur Chipkarte, daß für alle Blöcke außer für den abschließenden LEN=IFSC gilt. In [ISO 4'] ist diese Forderung nicht explizit enthalten. Das Terminal darf während des Chaining grundsätzlich mit LEN≤IFSC senden. GeldKarten lehnen den Empfang von I-Blöcken außer des letzten Blockes mit LEN<IFSC während des Chainings als einen Protokollfehler ab.

Das Terminal muß daher bei dem Chaining zur GeldKarte alle I-Blöcke außer dem letzten mit LEN=IFSC senden.

2.1.3. Behandlung von S-Blöcken

Das Senden und Empfangen von S-Blöcken durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Für das Senden eines S(IFS request) gilt allerdings die oben gemachte Einschränkung, daß IFSD<IFSC hierbei nicht zulässig ist.

2.1.4. Fehlerbehandlung

Die Fehlerbehandlung durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Hierbei ist die folgende Anforderung zu beachten:

Empfängt das Terminal während der Protokollabwicklung einen R-Block, so ist nur der Sequenzzähler für den weiteren Protokollablauf ausschlaggebend. Die gemäß [ISO 4'] optionale Auswertung der Bit b1 bis b4 des PCB darf durch das Terminal nur zu informativen Zwecken erfolgen.

2.2. Sicherheitsanforderungen

2.2.1. Umgang mit kryptographischen Schlüsseln

Für die Ausführung der meisten Funktionen benötigt das Bankensonderfunktionsterminal die entsprechenden kryptographischen Schlüssel. Es muß sichergestellt sein, daß die dabei verwendeten kryptographischen Schlüssel bei Transport, Einbringung, Speicherung und Verarbeitung gegen das Ausspähen durch Unberechtigte geschützt sind. Dazu müssen in einem Sicherheitsmodul die entsprechenden Sicherheitsmechanismen realisiert werden. Dieses Sicherheitsmodul kann Bestandteil der Hardware des Bankensonderfunktionsterminals sein oder aber auch ein entfernt stehendes Sicherheitsmodul sein.

Kryptographische Schlüssel sind bestimmten Funktionen des Bankensonderfunktionsterminals zugeordnet. Es muß sichergestellt werden, daß nur bei der Ausführung dieser Funktionen auf die Schlüssel zugegriffen werden kann.

Für die Einbringung der kryptographischen Schlüssel in das Sicherheitsmodul des Bankensonderfunktionsterminals sind zwei unterschiedliche Wege denkbar:

1. Die Schlüssel werden von der jeweiligen Generierungsstelle (ZKA, Verlage, Bankinstitut) an den Terminalhersteller übermittelt. Der Terminalhersteller lädt die Schlüssel innerhalb einer gesicherten Umgebung in das Sicherheitsmodul.

Bei der Übermittlung der Schlüssel und bei ihrer Verarbeitung durch den Terminalhersteller sind die gleichen Sicherheitsanforderungen zu berücksichtigen, wie sie heute bei der Personalisierung von POS-Terminals bzw. wie sie bei der Personalisierung der Terminals für die Verarbeitung der ec-Karte mit Chip gelten.

2. Die Schlüssel werden von der jeweiligen Generierungsstelle an Beauftragte des Bankinstituts übermittelt, das das Bankensonderfunktionsterminal aufstellt. Dort werden die Schlüssel im Rahmen einer Initialisierung von den berechtigten Angestellten in das Bankensonderfunktionsterminal geladen und von diesem sicher in seinem Sicherheitsmodul gespeichert.

Für den institutseigenen Schlüssel KGK_{Card} und den Schlüssel K_{SIG} ist der zweite Weg vorgeschrieben.

In jedem Fall müssen die folgenden Sicherheitsanforderungen berücksichtigt werden:

- Ein Schlüssel wird grundsätzlich in zwei Schlüsselhälften transportiert. Eine Schlüsselhälfte hat dabei die gleiche Länge wie der eigentliche Schlüssel. Aus den Schlüsselhälften wird mittels XOR der eigentliche Schlüssel bestimmt.

- Bei dem Transport, der Verarbeitung und der Aufbewahrung der beiden Schlüsselhälften ist eine absolute Funktionstrennung der Schlüsselträger zu gewährleisten. In keinem Fall darf ein Schlüsselträger in den Besitz beider Schlüsselhälften kommen!

2.2.2. Identifikation und Authentikation des Bedieners

Vor der Ausführung der meisten Funktionen muß sich der Bediener gegenüber dem Bankensonderfunktionsterminal identifizieren (z. B. durch Eingabe einer Benutzerkennung) und authentisieren (z. B. durch die Eingabe eines Paßwortes).

Die eingegebene Bedieneridentität muß während der gesamten Arbeitssitzung manipulationssicher durch das Bankensonderfunktionsterminal gespeichert werden.

Es muß die Möglichkeit bestehen, daß mehrere Bediener für eine Arbeitssitzung eine Identifikation und Authentisierung durchführen. Dies wird für Funktionen benötigt, die nur von mehreren Bedienern ausgeführt werden dürfen (Vier-Augen-Prinzip).

2.2.3. Zugriffskontrolle

Vor jeder Ausführung einer Funktion des Bankensonderfunktionsterminals muß eine Zugriffskontrolle durchgeführt werden. Dabei muß überprüft werden,

- ob der für die aktuelle Arbeitssitzung identifizierte Bediener die gewünschte Funktion ausführen darf, und

- ob der für die aktuelle Arbeitssitzung identifizierte Bediener auf die gewünschten Daten der Chipkarte zugreifen darf.

Darf eine Funktion nur von zwei (bzw. mehreren) Bedienern ausgeführt werden, muß die Zugriffskontrolle überprüfen, ob die Kombination der für die aktuelle Sitzung identifizierten Bediener zur Ausführung der Funktion berechtigt.

Durch die Zugriffskontrolle muß ebenfalls gewährleistet sein, daß die kryptographischen Schlüssel nur durch die dafür berechtigten Funktionen verwendet werden können.

Funktionen, die keine vorherige Identifikation und Authentikation der Bediener verlangen, dürfen von jedem Bediener des Bankensonderfunktionsterminals ausgeführt werden.

2.2.4. Online-Verbindungen

Wird für die Ausführung einer Funktion eine Online-Verbindung zu einem entfernten Sicherheitsmodul benötigt, ist der Aufbau der Verbindung, die gesamte Kommunikation mit dem entfernten Sicherheitsmodul und der Abbau der Verbindung besonders zu schützen. Dabei ist insbesondere auf eine geeignete Ablaufkontrolle zu achten.

Es muß sichergestellt werden, daß die Daten bei ihrer Übertragung nicht ausgespäht bzw. unerkannt manipuliert werden können. Zusätzlich muß gewährleistet sein, daß das Bankensonderfunktionsterminal die von dem entfernten Sicherheitsmodul erhaltenen Daten nicht manipuliert an die Chipkarte weitergibt und so diese täuscht.

2.2.5. Protokollierung

Alle durch das Bankensonderfunktionsterminal ausgeführten Funktionen müssen (revisionsfähig) protokolliert werden. Neben der ausgeführten Funktion muß die Benutzerkennung des für die aktuelle Arbeitssitzung identifizierten Bedieners (soweit vorhanden) sowie ein Status für die Beendigung des Vorganges (ohne Fehler bzw. mit Fehlercode) protokolliert werden.

Die Daten der Protokollierung müssen manipulationssicher gespeichert werden. Die Protokollierungsdaten müssen der Zugriffskontrolle unterliegen. Ein Auslesen bzw. Löschen der Protokollierungsdaten darf nur besonders berechtigten Personen möglich sein.

2.2.6. Ablaufkontrolle

Besonders für die Funktionen, die den Zustand einer Chipkarte ändern, muß das Bankensonderfunktionsterminal eine Ablaufkontrolle durchführen. Dabei sind insbesondere alle Ausnahmesituationen wie zum Beispiel das Ziehen der Chipkarte vor Beendigung der Funktion zu berücksichtigen.

Durch die Ablaufkontrolle muß sichergestellt werden, daß die zu der Ausführung einer Funktion gehörende Zugriffskontrolle und Protokollierung nicht umgangen werden können. Die Durchführung der Zugriffskontrolle, die Ausführung der eigentlichen Funktion und die Protokollierung des Vorganges dürfen nicht durch andere Tätigkeiten des Bankensonderfunktionsterminals unterbrochen werden.

Wird für die Ausführung einer Funktion eine Online-Verbindung zu einem entfernten Sicherheitsmodul benötigt, ist der Aufbau der Verbindung, die gesamte ordnungsgemäße Kommunikation mit dem entfernten Sicherheitsmodul und der Abbau der Verbindung durch die Ablaufkontrolle zu überwachen.

2.3. Sonstige Anforderungen

2.3.1. Ausgabe über Bildschirm/Drucker

Die auszugebenden Daten müssen in einer für den Bediener bzw. dem Kunden lesbaren Form angezeigt werden. Es muß sichergestellt sein, daß die von der Chipkarte übermittelten Daten vor ihrer Ausgabe nicht manipuliert werden können.

2.3.2. Bedienerführung

Dem Bediener des Bankensonderfunktionsterminals muß eine menuegesteuerte Bedienerführung angeboten werden. Über Menues sollte der Bediener beispielsweise die folgenden Punkte auswählen können:

1. Funktionsgruppe
2. Auszuführende Funktion
3. Zu bearbeitende Daten der Chipkarte

3. Erläuterung der Funktionen des Bankensonderfunktionsterminals

Ein Bankensonderfunktionsterminal kann je nach Ausprägung unterschiedliche Funktionen ausführen. Die Anforderungen an die einzelnen Funktionen werden im folgenden erläutert.

3.1. Echtheitsprüfung des Chips

Mittels des Kommandos INTERNAL AUTHENTICATE kann sich die Applikation elektronische Geldbörse und/oder die Applikation electronic cash gegenüber der externen Welt authentisieren. Das ist dann relevant, wenn festgestellt werden soll, ob es sich bei einer vorgelegten Karte um eine

echte ec-Karte handelt.

Um die Echtheitsprüfung durchführen zu können, benötigt das Bankensonderfunktionsterminal

- einen Zufallszahlengenerator und
- den Schlüssel KGK_{CAM} und/oder einen Schlüssel KGK_{AK}.

Es muß sichergestellt werden, daß der Bediener über den Ausgang der Echtheitsprüfung nicht getäuscht werden kann.

Die Schlüssel KGK_{CAM} und KGK_{AK} dürfen nur durch diese Funktion verwendet werden.

Die Ausführung dieser Funktion benötigt keine vorherige Identifikation und Authentikation des Bedieners.

3.2. Ausgabe von Chipdaten

Mit dieser Funktionalität des Bankensonderfunktionsterminals werden Daten einer ec-Karte gelesen und in einer für den Bediener lesbaren Form am Bildschirm angezeigt oder ausgedruckt. Über die menuegesteuerte Bedienerführung muß der Bediener die anzuzeigenden Datenfelder des Chips auswählen können. Es muß sichergestellt sein, daß die Daten korrekt angezeigt bzw. ausgedruckt werden, so daß der Bediener nicht über die von der ec-Karte ausgegebenen Daten getäuscht werden kann.

3.2.1. Frei lesbare Daten

Für die Ausgabe frei lesbarer Daten (Zugriffsbedingung vom Typ ALW oder PRO für READ RECORD in der ec-Karte), werden keine zusätzlichen Sicherheitsanforderungen an das Bankensonderfunktionsterminal gestellt.

3.2.2. Nach Authentikation lesbare Daten

Die Kontodaten des Karteninhabers und die Online-Parameter für electronic cash sind nur lesbar, wenn das Terminal sich zuvor unter Verwendung des Schlüssels K_{INFO} authentisiert hat. Das Bankensonderfunktionsterminal benötigt dazu den Schlüssel KGK_{INFO}.

Der Schlüssel KGK_{INFO} darf nur durch diese Funktion verwendet werden.

3.2.3. Lesen von Daten mit MAC-Prüfung

Beim Lesen einiger Felder der Chipkarte liefert die Chipkarte neben den Daten einen von ihr berechneten MAC mit (Zugriffsbedingung vom Typ PRO für READ RECORD in der ec-Karte). Bei der Ausführung dieser Funktion überprüft das Bankensonderfunktionsterminal diesen MAC. Dadurch wird verhindert, daß eine manipulierte Chipkarte Daten vortäuscht. Dies kann zum Beispiel sinnvoll sein, falls der Inhalt einer elektronischen Geldbörse ausgelesen werden soll, um den entsprechenden Betrag dem Kunden zu erstatten.

Es muß sichergestellt werden, daß der Bediener über den Ausgang der MAC-Überprüfung nicht getäuscht werden kann.

Um diese Funktion durchführen zu können, benötigt das Bankensonderfunktionsterminal

- einen Zufallszahlengenerator und
- den für die MAC-Bildung verwendeten Schlüssel. Für Daten der Applikation elektronische Geldbörse ist dies der Schlüssel KGK_{CD}.

Der Schlüssel KGK_{CD} darf nur durch diese Funktion verwendet werden.

Ist der von der Chipkarte ausgegebene MAC falsch, muß dies protokolliert werden.

3.3. Laden/Entladen/Bezahlen (GeldKarte)

Ein Bankensonderfunktionsterminal kann die Funktionalität eines Ladeterminals besitzen. In diesem Fall muß das Bankensonderfunktionsterminal der Spezifikation für GeldKarte-Ladeterminals ([LIT 4D]) genügen.

Zu dieser Funktionsgruppe gehören die folgenden Funktionen:

- Laden und Entladen der elektronischen Geldbörse einer ec-Karte,
- Laden der elektronischen Geldbörse einer Wertkarte.

Die elektronische Geldbörse einer Wertkarte ist dabei nicht kontobezogen.

Für die Ladefunktion muß das Bankensonderfunktionsterminal in der Lage sein, die vom Chipkartenbesitzer eingegebene PIN sicher zu verarbeiten (siehe 2.2) und diese verschlüsselt oder im Klartext an die Chipkarte zu übermitteln. Die für die Lade- und Entladefunktion benötigten Schlüssel dürfen nur durch diese Funktion verwendet werden. Es muß eine Online-Verbindung zur Ladezentrale aufgebaut werden.

Die Bedeutung der Antwortcodes der Antwortnachrichten von Lade- und Entladedialogen sind dem Bediener unverändert anzuzeigen.

Ein Bankensonderfunktionsterminal kann die Funktionalität zum Bezahlen aus der elektronischen Geldbörse einer GeldKarte besitzen. In diesem Fall muß das Bankensonderfunktionsterminal der Spezifikation für Akzeptanzterminals in [LIT 4E] genügen. Falls die Zahlungsfunktion am Bankensonderfunktionsterminals verwendet werden soll, um GeldKarten zu entladen, können der in [LIT 4E] definierte Ablauf für das Bestätigen des Zahlungsbetrages nach Einstecken der GeldKarte und die dort definierten Anzeigetexte entsprechend modifiziert werden.

3.4. Beenden eines offenen Entladen auf das Kartenkonto

Bei einem Online-Entladen auf das Kartenkonto können die folgenden Probleme auftreten:

- Bei Ausführung des **Entladen** im Entladedialog oder des **Entladen einleiten** im Bestätigungsdialog treten Fehler auf, so daß die GeldKarte möglicherweise entladen ist, aber keine Buchung des entladenen Betrages erfolgen kann.

- Im Bestätigungsdialog trifft keine, eine fehlerhafte oder eine negative Entladebestätigung von der Ladezentrale ein, so daß die GeldKarte zwar entladen ist, aber zumindest Ungewißheit darüber herrscht, ob die Buchung des entladenen Betrages erfolgt ist.

Gemäß der Spezifikation für das Entladen in [LIT 4D] wird dem Karteninhaber in den meisten Fehlersituationen angezeigt, daß ein Fehler aufgetreten ist und daß er sich an sein kartenausgebendes Institut wenden soll. In einigen Fällen stellt der Karteninhaber später fest, daß die Buchung auf sein Konto nicht erfolgt ist.

In einem Bankensonderfunktionsterminal, das auch Ladeterminal ist, kann eine spezielle Funktion implementiert werden, durch die die Buchung des entladenen Betrages mittels einer sogenannten impliziten Entladequittung nachträglich veranlaßt wird. Auf diese Weise können Reklamationen, die aufgrund eines nicht beendeten Online-Entladens entstehen, bearbeitet werden. Mit dieser Funktion kann an einem Bankensonderfunktionsterminal auch sichergestellt werden, daß eine GeldKarte, die einbehalten werden soll, entladen ist.

Bei dieser Funktion zur Reklamationsbearbeitung handelt es sich um eine Variante des regulären Entladens auf das Kartenkonto. Hierbei wird der Betrag 0 entladen, um eine implizite Entladequittung zu generieren. In den vorbereitenden Schritten wird geprüft, ob die GeldKarte tatsächlich leer ist. Ist das nicht der Fall, wird der Vorgang abgebrochen. Im folgenden werden die Schritte der Funktion detailliert erläutert.

Die Anzeigetexte können sinngemäß verwendet werden. Es handelt sich um Anzeigetexte für die Bedieneranzeige.

3.4.1. Vorbereitende Schritte

Das folgende Diagramm zeigt die vorbereitenden Schritte der impliziten Entladequittung, die stattfinden nachdem der Karteninhaber seine Karte eingesteckt hat, wird geprüft,

- ob die Applikation elektronische Geldbörse auf der Karte vorhanden ist,
- ob die GeldKarte für das Entladen gültig ist,
- ob es sich um eine kontobezogene GeldKarte handelt,
- ob das Entladen mit dem Kommando **Entladen einleiten** oder **Entladen einleiten wiederholen** an die GeldKarte begonnen werden muß,
- ob der aktuelle Betrag der GeldKarte 0 ist.

In Abhängigkeit von den Ergebnissen der Prüfung wird am Bankensonderfunktionsterminal eine implizite Entladequittung durch das Entladen des Betrags 0 ausgelöst.

GeldKarte			Terminal		Ladezentrale	
			A1	Anzeige: **Bitte Karte einstecken**		
		<---	C2	Reset GeldKarte		
R2	ATR der GeldKarte	--->				
			C3	SELECT FILE DF_BÖRSE		
		<---				
R3	OK	--->				
		<---	C4	READ RECORD EF_ID		
R4	Daten aus EF_ID	--->				
			A4	Daten prüfen und speichern		
			C5	READ RECORD EF_BÖRSE		
		<---				
R5	Daten aus EF_BÖRSE	--->				
			A5	Kartentyp auswerten		
			C6	READ RECORD '01' EF_LLOG		
		<---				
R6	Record '01' aus EF_LLOG	--->				
			A6	Statusbyte auswerten und speichern		
			C7	READ RECORD EF_BETRAG		
		<---				
R7	Beträge aus EF_BETRAG	--->				
			A7	Anzeige: Entladen des aktuellen Betrags		
				Beträge speichern		

Erläuterung

1. Am Display wird angezeigt:

   Bitte Karte einstecken

2. Nachdem die GeldKarte eingesteckt ist, wird durch das Bankensonderfunktionsterminal ein Reset der Karte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer GeldKarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   ATR falsch, bitte Karte entnehmen

3. Die Applikation elektronische Geldbörse wird geöffnet, indem das ADF der Applikation DF_BÖRSE durch das Terminal mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 45 50 01 00'	AID der elektronischen Geldbörse

   Im Fehlerfall wird mit der folgenden Anzeige abgebrochen

   Applikation nicht selektierbar, bitte Karte entnehmen

   Nachdem der Applikationskontext geöffnet ist, können die AEFs der Applikation mittels SFI referenziert werden.

4. Das Terminal liest mittels READ RECORD die Kartenidentifikationsdaten im Record '01' des EF_ID im MF der GeldKarte (SFI '17'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'BC'	P2, Reference Control Byte
5	1	'16'	L_e, Recordlänge des EF_ID

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'67'	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der GeldKarte
13-15	3	'JJ MM TT'	Aktivierungsdatum der GeldKarte
16-17	2	'0280'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp

Die empfangenen Daten werden geprüft.

Wenn die Prüfziffer nicht korrekt ist oder wenn Branchenhauptschlüssel, erste Ziffer der Kurz-BLZ, Verfalldatum, Aktivierungsdatum, Ländercode, Währungskennzeichen oder Wertigkeit der Währung nicht korrekt kodiert sind, bricht das Bankensonderfunktionsterminal mit der folgenden Meldung ab:

Kartendaten falsch, bitte Karte entnehmen

Das Bankensonderfunktionsterminal überprüft Verfalldatum und Aktivierungsdatum. Wenn aktuelles Datum < Aktivierungsdatum oder aktuelles Datum > Verfalldatum + 3 Monate, bricht das Bankensonderfunktionsterminal mit der Meldung ab

Karte ungültig, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Kartendaten nicht lesbar, bitte Karte entnehmen

Wenn im Bankensonderfunktionsterminal gespeichert ist, welche Karten als institutseigene Karten zum Entladen akzeptiert werden, prüft es, ob es sich um eine institutseigene Karte handelt, indem es die Kurz-BLZ aus Byte 2-4 des gelesenen Records auswertet. Stellt es hierbei fest, daß die GeldKarte nicht zum Entladen akzeptiert wird, bricht es mit der Meldung ab

Institutsfremde Karte, bitte Karte entnehmen

Verfügt das Bankensonderfunktionsterminal nicht über die benötigte Information, wird diese Prüfung übersprungen. Sie muß dann in einer nachgelagerten Komponente des Terminalbetreibers erfolgen.

Die Daten werden gespeichert.

Währungskennzeichen und Wertigkeit der Währung werden verwendet, um den Gegenwert der in der GeldKarte gespeicherten Betragswerte errechnen und anzeigen zu können. Die möglichen Belegungen für die Wertigkeit der Währung und deren Bedeutung sind in Kapitel 6.9 von [LIT 1] beschrieben.

Anhand der Kurz-BLZ kann die für die GeldKarte zuständige Ladezentrale identifiziert werden.

5. In diesem Schritt stellt das Bankensonderfunktionsterminal fest, ob es sich um eine kontobezogene GeldKarte handelt. Hierzu liest es mittels READ RECORD den Record '01' des EF_BÖRSE (SFI '19'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'CC'	P2, Reference Control Byte
5	1	'1B'	L_e, Recordlänge des EF_BÖRSE

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Kennung für den Kartentyp '00': kontobezogene GeldKarte 'FF': Wertkarte ohne Kontobezug
2-5	4	'nn..nn'	Bankleitzahl kontoführendes Institut für Börsenverrechnungskonto
6-10	5	'nn..nn'	Kontonummer Börsenverrechnungskonto
11	1	'nD'	Prüfziffer über Byte 1-9
12-27	16	'XX..XX'	(Triple-)DES-Verschlüsselung im CBC-Mode mit K_{LD} und ICV = '00..00' der 16 Byte 4 Byte Bankleitzahl|5 Byte Kontonummer des Kartenkontos|2 Byte Kartenfolgenummer| 1 Byte Freizügigkeitsschlüssel|'00 00 00 00'

Wenn Byte 1 des Records keinen der Werte ’00’ oder ’FF’ hat, bricht das Bankensonderfunktionsterminal mit der Meldung ab

Kontodaten falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Kontodaten nicht lesbar, bitte Karte entnehmen

Das Bankensonderfunktionsterminal wertet den Kartentyp aus.

Wenn es sich um eine Wertkarte handelt, bricht das Terminal mit der Meldung ab

Karte nicht kontobezogen, bitte Karte entnehmen

6. Das Terminal liest mittels READ RECORD die Protokolldaten des letzten Lade-/Entladeschritts der GeldKarte im Record '01' des EF_LLOG der GeldKarte (SFI '1C'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’01’	P1, Recordnummer
4	1	’E4’	P2, Reference Control Byte
5	1	’21’	L_e, Recordlänge des EF_LLOG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-3	2	’XX XX’	Sequenznummer LSEQ der Lade-/Entladetransaktion
4	1	’XX’	Wiederholungszähler WZ für das **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geladener bzw. entladener Betrag
8-10	3	'nn..nn'	(neuer) aktueller Betrag
11-13	3	'nn..nn'	AS-ID der Ladezentrale
14-21	8	'nn..nn'	Terminal-ID des Bankensonderfunktionsterminals
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Bankensonderfunktionsterminals
25-27	4	JJJJ MM TT	Datum der Lade-/Entladetransaktion
28-31	3	HH MM SS	Uhrzeit der Lade-/Entladetransaktion
32-33	2	'XX XX'	Sequenznummer BSEQ des letzten Abbuchens bzw. Rückbuchens

Wenn das Statusbyte in Byte 1 einen der Werte '11', '13', '15', '17', '31' oder '35' hat, muß das erste Kommando der impliziten Entladequittung ein **Entladen einleiten** sein.

Wenn das Statusbyte in Byte 1 einen der Werte '01', '03', '05', '07', '21' oder '25' hat, muß das erste Kommando der impliziten Entladequittung ein **Entladen einleiten wiederholen** sein.

Das Ladeterminal speichert den gelesenen Record und hält fest, mit welchem Kommando zu beginnen ist.

Wenn das Statusbyte einen anderen Wert hat, wird mit

Laden/Entladen gesperrt, bitte Karte entnehmen

abgebrochen.

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Ladeprotokoll nicht lesbar, bitte Karte entnehmen

7. Das Terminal liest den aktuellen Betrag aus Record '01' des EF_BETRAG mit dem Kommando READ RECORD (SFI '18'). Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'09'	L_e, Recordlänge des EF_BETRAG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Wenn einer der drei Beträge nicht BCD-kodiert ist, gibt das Bankensonderfunktionsterminal die Meldung aus:

Beträge falsch, bitte Karte entnehmen

In den übrigen Fehlerfällen wird mit der folgenden Anzeige abgebrochen

Beträge nicht lesbar, bitte Karte entnehmen

Das Bankensonderfunktionsterminal prüft den aktuellen Betrag. Hat <aktueller Betrag> einen von 0 verschiedenen Wert, bricht das Terminal mit der Anzeige ab

Karte nicht entladen, bitte Karte entnehmen

Andernfalls speichert das Bankensonderfunktionsterminal die Beträge und fährt mit Schritt 1. der impliziten Entladequittung fort.

3.4.2. Implizite Entladequittung

Das folgende Diagramm zeigt die Schritte des Entladedialogs zur impliziten Entladequittung, die nachfolgend erläutert werden:

GeldKarte			**Terminal**			**Ladezentrale**	
			A1	Transaktion beginnen			
		<---	C2	**Entladen einleiten** oder **Entladen einleiten wiederholen**			
R2	Antwortdaten des **Entladen einleiten (wiederholen)** mit Betrag 0	--->					
			A2	Antwortdaten prüfen			
			A3	K_T ableiten			
			C3	Entladeanfrage 0200 Abwkz. 290000 mit Daten der GeldKarte, Zertifikat der GeldKarte K_T-MAC des Terminals	--->		
						R3	Entladeantwort 0210 Abwkz. 290000 mit Daten für GeldKarte Zertifikat für GeldKarte K_T-MAC für Terminal
					<---		
			A3	Entladeantwort und K_T-MAC prüfen			
		<---	C4	**Entladen**			
R4	Antwortdaten des **Entladen**	--->					
			A4	Antwortdaten prüfen eventuell Anzeige: **Entladen abgebrochen**			

Erläuterung

1. Das Bankensonderfunktionsterminal beginnt eine neue Transaktion. Hierbei muß sichergestellt sein, daß

   - Trace-Nummer TSEQ und
   - Schlüsselindex SI

   gegenüber der letzten Transaktion inkrementiert sind.

   Hierbei wird die Trace-Nummer nach Erreichen des maximal möglichen Wertes auf 0 zurückgesetzt.

   Hat der Schlüsselindex den Maximalwert erreicht, wird das Terminal für den weiteren Betrieb als Bankensonderfunktionsterminal gesperrt.

   In allen Schritten der Transaktion müssen durch das Bankensonderfunktionsterminal dieselbe Trace-Nummer und derselbe Schlüsselindex verwendet werden.

2. In Abhängigkeit vom Ausgang der Prüfungen in Schritt 6. der vorbereitenden Schritte wird die Kommandonachricht für das **Entladen einleiten** oder das **Entladen einleiten wiederholen** ohne Secure Messaging aufgebaut. Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	**Länge** (in Byte)	**Wert**	**Erläuterung**
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'00' '20'	P1 für **Entladen einleiten** P1 für **Entladen einleiten wiederholen**
4	1	‘00’	P2, fester Wert
5	1	‘1C’	L_c
6	1	'20' '24'	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
7-9	3	'00..00'	Filler
10-12	3	'00..00'	Filler
13-15	3	'00..00'	Filler
16-23	8	'nn..nn'	Terminal-ID des Bankensonderfunktionsterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Bankensonderfunktionsterminals
27-29	4	JJJJ MM TT	Datum des Bankensonderfunktionsterminals
30-33	3	HH MM SS	Uhrzeit des Bankensonderfunktionsterminals
34	1	‘3A’	L_e

Terminal-ID und Trace-Nummer müssen die korrekten Werte haben.

Das Kommando wird an die GeldKarte gesandt.

Bei erfolgreicher Ausführung gibt die GeldKarte die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'21' '25'	Nachrichten-ID für **Entladen einleiten** Nachrichten-ID für **Entladen einleiten wiederholen**
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des Kommandos
5-7	3	'00 00 00'	Entladebetrag
8-10	3	'00 00 00'	aktueller Betrag (identisch mit Byte 5-7)
11-20	10	'nn..nD'	Kontodaten des Börsenverrechnungskontos aus Byte 2-11 des EF_BÖRSE ('D' Hexziffer)
21-36	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Unter K_{LD} verschlüsselte Konto- und Kartendaten
37	1	'31' oder '35'	Statusbyte der letzten erfolgreichen Lade-/Entladetransaktion
38-39	2	'XX XX'	Sequenznummer LSEQ der letzten erfolgreichen Lade-/Entladetransaktion
40	1	'XX'	Wiederholungszähler WZ der letzten erfolgreichen Lade-/Entladetransaktion
41-43	3	'nn..nn'	Transaktionsbetrag der letzten erfolgreichen Lade-/Entladetransaktion
44-46	3	'nn..nn'	Maximalbetrag aus Byte 4-6 des EF_BETRAG
47-49	3	'nn..nn'	maximaler Transaktionsbetrag aus Byte 7-9 des EF_BETRAG
50	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
51-58	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 56 Byte Byte 1-50|'00 00 00 00 00 00'

Hat die Nachrichten-ID weder den Wert '21' noch '25' wird mit der folgenden Anzeige abgebrochen

Chipfehler bei Entladen einleiten, bitte Karte entnehmen

Das Terminal analysiert das Statusbyte der letzten erfolgreichen Transaktion in Byte 37 der Antwortdaten.

Wenn das Statusbyte einen der Werte '31' oder '35' hat, wurde zuletzt ein Entladen ausgeführt. In diesem Fall werden Byte 41-43 des gelesenen Records ausgewertet.

- Wenn Byte 41-43 des gelesenen Record den Wert 0 haben, wurde zuletzt der Betrag 0 entladen. Hierdurch wurde bereits ein eventuell vorher erfolgtes Entladen bestätigt. Eine implizite Entladequittung ist daher nicht sinnvoll.

Die Funktion wird abgebrochen. Dies kann mit der Anzeige erfolgen

Karte bereits erfolgreich entladen, bitte Karte entnehmen

- Wenn Byte 41-43 des gelesenen Record einen von 0 verschiedenen Wert haben, wurde zuletzt ein Betrag aus der GeldKarte entladen, der eventuell noch nicht gebucht wurde. Eine implizite Entladequittung ist sinnvoll. Die Sequenznummer LSEQ, der Wiederholungszähler WZ werden gespeichert und es wird mit Schritt 3. fortgefahren.

Wenn das Statusbyte in Byte 37 einen der Werte '11', '13', '15' oder '17' hat, wurde die GeldKarte zuletzt erfolgreich geladen. Hierdurch wurde bereits implizit ein eventuell vorher erfolgtes Entladen bestätigt. Eine implizite Entladequittung ist daher nicht sinnvoll.

Die Funktion wird abgebrochen. Dies kann mit der Anzeige erfolgen

Karte zuletzt geladen, bitte Karte entnehmen

Hat das Statusbyte in Byte 37 keinen der oben genannten Werte oder treten bei dem Lesen des Records andere Fehler auf wird mit der folgenden Anzeige abgebrochen

Chipfehler bei Entladen einleiten, bitte Karte entnehmen

3. Das Bankensonderfunktionsterminal zeigt an:

Vorgang wird bearbeitet

Eine Entladeanfrage mit Nachrichtentyp 0200 und Abwicklungskennzeichen 290000 an die Ladezentrale wird aufgebaut. Der an der Schnittstelle zur Ladezentrale erwartete Aufbau entspricht ISO 8583 (1987) und ist in Kapitel 3. von [LIT 4D] spezifiziert. Falls das Terminal nur eine Teilnachricht aufbaut, sind die Vorgaben in Kapitel 3.2. von [LIT 4D] zu beachten.

Die in Kapitel 3.4.2.1. von [LIT 4D] formulierten Konsistenzanforderungen an die Inhalte von BMP 62 und den übrigen Daten der (Teil-)Anfragenachricht müssen erfüllt sein.

Terminal-ID (BMP 41 und BMP 42) und Trace-Nummer (BMP 11) müssen die korrekten Werte enthalten.

In BMP 62 der Entladeanfrage stellt das Bankensonderfunktionsterminal den Inhalt des EF_ID der GeldKarte und die Antwortdaten der GeldKarte zu **Entladen einleiten (wiederholen)** ein.

Der Transaktionsschlüssel K_T wird wie in Kapitel 3. von [LIT 4D] spezifiziert aus dem K_{MAC}

unter Verwendung des Schlüsselindex SI abgeleitet. Die Entladeanfrage wird mit dem K_T MAC-gesichert.

Das Bankensonderfunktionsterminal stellt den Konditionscode 16 in die Entladeanfrage ein, so daß die Ladezentrale erkennen kann, daß die Anfrage von einem Bankensonderfunktionsterminal stammt.

Das Bankensonderfunktionsterminal schickt die Entladeanfrage an die Ladezentrale.

Wenn die Entladeanfrage nicht an die Ladezentrale gesendet werden kann, wenn die Entladeantwort der Ladezentrale nicht innerhalb eines definierten Zeitraums eintrifft, bricht das Bankensonderfunktionsterminal mit der Meldung ab

Kommunikationsstörung, bitte Karte entnehmen

Wenn durch eine nachgelagerte Komponente des Terminalbetreibers festgestellt wird, daß die Karte als institutsfremd nicht zum Entladen akzeptiert wird, bricht das Bankensonderfunktionsterminal mit der Meldung ab

Institutsfremde Karte, bitte Karte entnehmen

Wenn das Bankensonderfunktionsterminal eine Antwortnachricht erhält, prüft es mindestens anhand von Abwicklungskennzeichen, Nachrichtentyp, Terminal-ID und Trace-Nummer, ob diese Antwortnachricht zu der Anfragenachricht gehört.

Es prüft ob Form und Inhalt der Antwortnachricht und der MAC der Antwortnachricht korrekt sind. Das Bankensonderfunktionsterminal prüft, ob die Konsistenzanforderungen an die Daten in BMP 62 und die übrigen Daten der (Teil-)Antwortnachricht aus Kapitel 3.4.2.2. aus [LIT 4D] erfüllt sind.

Stellt das Terminal hierbei Fehler fest, bricht es mit der Meldung ab

Fehlerhafte Online-Antwort, bitte Karte entnehmen

Wenn die Antwortnachricht formal korrekt ist und einen fehlerfreien MAC aber einen negativen Antwortcode ($\neq$ 0) enthält, bricht das Bankensonderfunktionsterminal mit der

folgenden Meldung ab:

<Text>, bitte Karte entnehmen

<Text> ist der folgenden Tabelle zu entnehmen:

Antwortcode	Text
4	Die Karte ist nicht zugelassen, Karte einziehen
5	Karte aus Sicherheitsgründen abweisen
30	Formatfehler
54	Gültigkeitsdauer überschritten oder Karte nicht aktiviert
56	Entschlüsselte Kartendaten unbekannt oder fehlerhaft
58	Entladen an institutsfremdem Terminal
62	Die Karte ist gesperrt
76	Sequenznummer in BMP 57 falsch
77	Fehler in BMP 62, insbesondere MAC-Fehler
78	Sequenzfehler in BMP 62
91	Zuständiges AS nicht erreichbar
92	Falsches Routing
96	Zuständiges AS nicht verfügbar
97	MAC falsch (BMP 64)
98	Datum/Uhrzeit nicht plausibel

4. Wenn die Antwortnachricht formal korrekt ist, einen fehlerfreien MAC und den Antwortcode 0 enthält, baut das Bankensonderfunktionsterminal aus den Daten in BMP 62 der Antwortnachricht die Kommandonachricht eines **Entladen** auf.

   Die folgende Tabelle zeigt den Aufbau der Command APDU:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	‘E0’	CLA
2	1	‘32’	INS
3	1	'80'	P1 für **Entladen** ohne Änderung der Maximalbeträge
4	1	‘00’	P2, fester Wert
5	1	‘2B’	L_c
6	1	'30'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge
7-8	2	'XX XX'	Sequenznummer LSEQ der Transaktion
9	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
10-12	3	'00..00'	Filler
13-15	3	'nn..nn'	AS-ID der Ladezentrale
16-23	8	'nn..nn'	Terminal-ID des Bankensonderfunktionsterminals
24-26	3	'nn..nn'	Trace-Nummer TSEQ des Bankensonderfunktionsterminals
27-29	4	JJJJ MM TT	Datum des Bankensonderfunktionsterminals
30-33	3	HH MM SS	Uhrzeit des Bankensonderfunktionsterminals
34-36	3	'nn..nn'	neuer Maximalbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
37-39	3	'nn..nn'	neuer maximaler Transaktionsbetrag wird bei **Entladen** ohne Änderung der Maximalbeträge durch das Kommando nicht ausgewertet
40	1	‘XX’	Schlüssel-Version KV des verwendeten K_{LD}
41-48	8	'XX..XX'	Zertifikat: (Retail-)CBC-MAC mit K_{LD} über die 40 Byte Byte 6-40|'00 00 00 00 00'
49	1	‘0A’	L_e

Byte 6-48 der Command APDU entnimmt das Bankensonderfunktionsterminal BMP 62 der geprüften Antwortnachricht. Es stellt Byte 1-5 und Byte 49 wie spezifiziert ein.

Das Terminal speichert die verwendete Nachrichten-ID.

Das Bankensonderfunktionsterminal sendet die Kommandonachricht an die GeldKarte.

Wenn die GeldKarte einen negativen Returncode zurückgibt, bricht das Bankensonderfunktionsterminal mit der Meldung ab

Buchung erfolgt

Chipfehler, bitte Karte entnehmen

Wenn die GeldKarte das Kommando erfolgreich bearbeitet hat, gibt sie die folgenden Antwortdaten zurück:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'31'	Nachrichten-ID für **Entladen** ohne Änderung der Maximalbeträge
2-3	2	'XX XX'	Sequenznummer LSEQ der Transaktion
4	1	'XX'	Wiederholungszähler WZ des zugehörigen **Entladen einleiten**
5-7	3	'nn..nn'	Entladebetrag
8-10	3	'00..00'	neuer aktueller Betrag

Das Bankensonderfunktionsterminal führt die folgenden Prüfungen durch:

- Ist die Nachrichten-ID korrekt?
- Stimmen LSEQ und WZ mit den durch das Bankensonderfunktionsterminal gespeicherten Werten überein?
- Ist der Entladebetrag korrekt?
- Hat der neue aktuelle Betrag den Wert 0?

Wenn die Prüfungen der Antwortdaten positiv ausfallen, zeigt das Terminal an

Buchung erfolgt, bitte Karte entnehmen

Wenn das **Entladen** weder zu einem negativen Returncode noch zur Ausgabe korrekter Antwortdaten geführt hat, versucht das Bankensonderfunktionsterminal, das **Entladen** erneut auszuführen, eventuell nach einem Reset der GeldKarte und erneuter Selektion des DF_BÖRSE.

Wenn die GeldKarte das **Entladen** jetzt erfolgreich ausführt, zeigt das Terminal an

Buchung erfolgt

andernfalls

Buchung erfolgt

Chipfehler, bitte Karte entnehmen

3.5. Ändern von Chipdaten

Das Ändern von Chipdaten ist ein sicherheitskritischer Vorgang. Dies geschieht mittels des Kartenkommandos UPDATE RECORD. Dazu wird ein Zugriff auf den Schlüssel KGK_{Card} benötigt.

Beispiele für Funktionen dieser Funktionsgruppe sind:

- Hochsetzen des PIN-FBZ
- Ändern des Händlerkontos

Die Ausführung dieser Funktion benötigt eine vorherige Identifikation und Authentikation des Bedieners. Durch die Zugriffskontrolle muß sichergestellt werden, daß nur berechtigte Bediener die entsprechenden Daten der Chipkarte ändern können. Dabei ist insbesondere auf einen Schutz der in der Chipkarte gespeicherten kryptographischen Schlüssel zu achten.

Der Schlüssel KGK_{Card} darf nur durch diese Funktion und die Funktion "Nachladen" (siehe 3.6) verwendet werden.

Über die menuegesteuerte Bedienerführung muß der Bediener die Datenfelder des Chips auswählen können, deren Inhalt er ändern möchte. Dabei ist darauf zu achten, daß ein Bediener nur Zugriff auf Datenfelder des Chips erhält, für die er berechtigt ist. Beispiele:

- Kassierer darf PIN-FBZ hochsetzen, andere Datenfelder darf er nicht ändern.
- Chipkarten-Sicherheitsbeauftragter darf Datenfelder ändern, die kryptographische Schlüssel enthalten.

Alle Ausführungen von Funktionen dieser Funktionsgruppe sind zu protokollieren.

Im folgenden Kapitel wird beispielhaft erläutert, wie der PIN-Fehlbedienungszähler im Chip mittels einer Online-Kommunikation zwischen Bankensonderfunktionsterminal und Ladezentrale hochgesetzt werden kann.

3.6. Hochsetzen des PIN-Fehlbedienungszählers mittels ISO-Nachrichten

3.6.1. Schnittstelle zur Ladezentrale

3.6.1.1. Nachrichtenaufbau

Die folgenden Tabellen geben eine Übersicht über den Aufbau der an der Schnittstelle der

Ladezentralen verwendeten Nachrichten. Hierbei bedeutet

Typ: Nachrichtentyp
Lg: Länge in Byte

POV = BCD-kodiert, rechtsbündig mit führenden 0
P = BCD-kodiert, linksbündig bei ungerader Ziffernzahl rechts mit Füllzeichen 'F'
CLx = x EBCDIC-Zeichen, Ziffern ohne Vorzeichen
BIN = Binär

BMP	Typ	Typ	Bezeichnung	Inhalt	Lg	Format
	0322	0332				
	x	x	Nachrichtentyp		2	POV
	x	x	Primary Bitmap		8	BIN
2	x x	= =x	Längenfeld Kartennummer	'F1F0'	2 10	CL2 P
3	x	=x	Abwicklungskennzeichen für FBZ-Hochsetzen	‘990000’	3	POV
11	x	=x	Trace-Nummer		3	POV
12	x	=x	Uhrzeit	HHMMSS	3	POV
13	x	=x	Datum	MMTT	2	POV
14	x	=x	Verfalldatum der Börsenkarte	JJMM	2	POV
25	x	x	Konditionscode Sonderfunktionsterminal =	 ‘16’	1	POV
33	x x	x x	Längenfeld ID zwischengeschalteter Rechner AS-ID oder LZ-ID	'F0F3'	2 3	CL2 POV
39		x	Antwortcode		1	POV
41	x	=x	Terminal-ID		4	POV
42	x	=x	Betreiber-BLZ		8	POV
57	x x	x x	Längenfeld Verschlüsselungsparameter	'F0F0F9'	3 9	CL3 BIN
62	x x	x x	Längenfeld Parameterwerte	‘F0FnFn’	3 nn	CL3 BIN
64	x	x	MAC		8	BIN

Die BMP 62 ist in der Antwortnachricht nur dann vorhanden, wenn der Antwortcode in BMP 39 = 00 ist.

3.6.1.2. Daten zwischen Terminal und Ladezentrale

BMP 02: Kartennummer

Das Terminal stellt die Kartennummer aus Byte 1-9 und die Prüfziffer aus dem linken Halbbyte von Byte 10 des EF_ID einer Börsenkarte in dieses Feld einer Anfragenachricht ein:

Branchenhauptschlüssel	'67'	Byte 1 aus EF_ID
Kurz-Bankleitzahl	'2n nn nn'	Byte 2-4 aus EF_ID
individuelle Kartennummer	'nn nn nn nn nn'	Byte 5-9 aus EF_ID
Prüfziffer	'n'	linkes Halbbyte von Byte 10 aus EF_ID
Filler	'F'	durch das Terminal einzustellen.

BMP 03: Abwicklungskennzeichen

Es wird folgendes Abwicklungskennzeichen verwendet:

990000: FBZ-Hochsetzen

Das Abwicklungskennzeichen muß in allen Nachrichten, die zu einer Transaktion gehören identisch sein.

BMP 11: Trace-Nummer

Die Trace-Nummer kennzeichnet alle zu einer Transaktion gehörenden Nachrichten eines Terminals. Sie wird im Terminal generiert und vor einer Transaktion um 1 inkrementiert.

Anfrage eines Terminals und Antwort einer Ladezentrale enthalten dieselbe Trace-Nummer.

BMP 12: Uhrzeit

Dieses Nachrichtenfeld enthält die aktuelle Uhrzeit des Terminals zu Beginn der Transaktion, d. h. bei Erstellung der Anfrage.

BMP 13: Datum

Dieses Nachrichtenfeld enthält das aktuelle Datum des Terminals zu Beginn der Transaktion.

BMP 14: Verfalldatum der Börsenkarte

Dieses Nachrichtenfeld enthält das Verfalldatum aus Byte 11-12 des EF_ID einer Börsenkarte.

BMP 25: Konditionscode

Durch den Konditionscode 16 wird in allen Anfragenachrichten angezeigt, daß es sich um ein Terminal mit Bankensonderfunktionen handelt.

BMP 33: ID zwischengeschalteter Rechner oder LZ-ID

Bei der Weiterleitung von Transaktionen trägt die weiterleitende Host oder Ladezentrale ihre ID ein.

In Antwortnachrichten wird hier durch die Ladezentrale immer die LZ-ID eingestellt.

Die Belegung erfolgt analog zu der im GA-Verbund.

BMP 39: Antwortcodes

In der folgenden Tabelle wird die Bedeutung der verwendeten Antwortcodes erläutert.

Code	Bedeutung
00	Transaktion ok, FBZ wird hochgesetzt
04	Die Karte ist gesperrt, Karte einziehen
05	Karte aus Sicherheitsgründen abweisen (z. B. bei mehrfacher Transaktion pro Karte pro Kalendertag)
30	Formatfehler
54	Gültigkeitsdauer überschritten oder Karte nicht aktiviert
56	Entschlüsselte Kartendaten unbekannt oder fehlerhaft
58	Terminal nicht zugelassen oder Karte an falschen Terminal
77	Fehler in BMP 62, insbesondere MAC-Fehler
79	Nachricht formal korrekt aber zuständige Ladezentrale nicht erreichbar
91	Zuständige LZ nicht erreichbar
92	Falsches Routing
96	Zuständige LZ nicht verfügbar
97	MAC falsch (BMP 64)
98	Datum/Uhrzeit nicht plausibel

BMP 41: Terminal-ID

Ein Terminal im kreditwirtschaftlichen Bereich muß durch die Betreiber-BLZ in den rechten 4 Byte von BMP 42 und Terminal-ID eindeutig in der Ladezentrale identifiziert werden können.

BMP 42: Betreiber-BLZ

In die rechten 4 Byte wird die BCD-kodierte BLZ des Terminal-betreibenden Kreditinstituts rechtsbündig eingestellt. Zusammen mit der Terminal-ID in BMP 41 wird hierdurch das Terminal eindeutig identifiziert.

BMP 57: Verschlüsselungsparameter

Die Verschlüsselungsparameter bestehen aus 9 Byte: 8 Byte Schlüsselindex zur Bildung des Sessionkeys und 1 Byte Schlüsselgeneration.

Für den Datenaustausch zwischen Terminals, die direkt an einer Ladezentrale angebunden oder über einen z. B. electronic cash-Netzbetreiber mit der Ladezentrale verbunden sind, und der Ladezentrale werden beim Nachrichtenfluß über den Netzbetreiber die electronic cash-Schlüssel verwendet und es gelten die Regeln zur Bildung der Verschlüsselungsparameter aus electronic cash.

Für direkt an eine Ladezentrale angeschlossene Terminals gelten dieselben Regeln, es wird jedoch der separat definierte Schlüssel mit der Generationsnummer 45 verwendet.

Werden Terminals innerhalb eines Netzes eines Verbandsbereiches betrieben (und dort z. B. durch bestehende Client/Server-Strukturen unterstützt), so ist die Definition eigener gruppenspezifischer Schlüssel für die MAC-Sicherung dieser Transaktionen möglich.

BMP 62: Daten zwischen Börsenkarte und Ladezentrale

In BMP 62 sind die Daten enthalten, die Börsenkarte und Ladezentrale miteinander austauschen.

In der Antwortnachricht ist die BMP 62 nur dann vorhanden, wenn der Antwortcode = 00 ist.

BMP 64: Terminal-MAC

Der Terminal-MAC wird wie beim GA-Verbund berechnet und geprüft. Zur MAC-Berechnung und -Prüfung wird ein Schlüssel K_{MAC} verwendet, der durch seine Generationsnummer (Byte 9 in BMP 57) eindeutig identifiziert wird.

Aus K_{MAC} und dem aktuellen Schlüsselindex SI (Byte 1-8 in BMP 57) wird der Transaktionsschlüssel K_T berechnet:

$K_T = dK_{MAC}(SI)$.

Der MAC in BMP 64 ist der CBC-MAC mit K_T zu der Nachricht ohne BMP 64. Bezeichnet man die Nachricht ohne BMP 64 mit N, dann berechnet sich der MAC wie folgt:

$MAC = mK_T(N)$.

Die MAC-Sicherung muß auf der gesamten Strecke zwischen Terminal-Sicherheitsmodul und Ladezentrale erfolgen. Auf dieser Strecke darf durch Sicherheitskomponenten umgeschlüsselt werden.

3.6.1.3. Daten zwischen Börsenkarte und Ladezentrale

Börsenkarte und Ladezentrale tauschen in BMP 62 Daten MAC-gesichert miteinander aus.

Das Terminal stellt hier die Daten des EF_ID der Börsenkarte und die von der Börsenkarte generierte Zufallszahl ein.

Die Ladezentrale prüft den MAC der eingestellten Daten und die Konsistenz der übrigen Nachrichtendaten mit diesen Daten.

Die Ladezentrale baut eine Antwortnachricht auf, in deren BMP 62 sie im positiven Fall wiederum Daten einstellt. Diese Daten entnimmt das Terminal der Antwortnachricht, und prüft die Konsistenz der übrigen Nachrichtendaten mit diesen Daten. Es prüft die Nachricht anhand der von ihm gespeicherten Daten.

Nach erfolgreicher Prüfung erzeugt das Terminal gemäß den Vorgaben, die sich aus Abwicklungskennzeichen, Nachrichtentyp, Antwortcode und Konditionscode der Nachricht ergeben eine Kommandonachricht an die Börsenkarte.

Das Längenfeld in BMP 62 einer Anfragenachricht ist immer mit F0F5F0 zu belegen.

Die Daten in BMP 62 einer Anfragenachricht haben immer den folgenden Aufbau:

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'67'	Branchenhauptschlüssel
2-4	3	'2n nn nn'	"Kurz-BLZ" kartenausgebendes Institut
5-9	5	'nn..nn'	individuelle Kartennummer
10	1	'nD'	Prüfziffer für Byte 1 - 9
11-12	2	'JJ MM'	Verfalldatum der ec-Karte
13-15	3	'JJ MM TT'	Aktivierungsdatum der ec-Karte
16-17	2	'02 80'	Ländercode
18-20	3	'44 45 4D'	Währungskennzeichen 'DEM'
21	1	'01'	Wertigkeit der Währung
22	1	'XX'	Chiptyp
23-38	16	'XX..XX'	Byte 12-27 des EF_BÖRSE Bei kontobezogenen GeldKarten: Unter K_{LD} CBC-verschlüsselte Karten- und Kontodaten
39-40	2	'030n'	Wert des PIN-FBZ im Chip (n =0-3)
41	1	'XX'	Schlüssel-Version KV des verwendeten K_{LD} aus dem zugehörigem EF_KEYD
42	1	'XX'	Schlüssel-Version KV des verwendeten K_{Card} aus dem zugehörigem EF_KEYD
43-50	8	'XX..XX'	Zufallszahl, Antwortdaten auf das Kommando GET CHALLENGE

Das Längenfeld in BMP 62 einer Antwortnachricht ist immer mit 'F0F1F5' zu belegen.

Die Daten in BMP 62 einer Antwortnachricht haben den folgenden Aufbau

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	'04'	CLA
2	1	'DC'	INS
3	1	'01'	P1, Recordnummer in EF_FBZ
4	1	'04'	P2
5	1	'0A'	Lc
6-7	2	'0303'	PIN-FBZ Neu
8-15	8	'XX..XX'	Zertifikat: (Retail-)CFB-MAC mit K_{Card} über die Byte 1-7|'00' mit ICV = Zufallszahl aus Anfrage

3.6.2. Abläufe in der Ladezentrale

Eine Ladezentrale prüft Terminal-MAC sowie die formale Korrektheit einer eintreffenden Anfragenachricht.

Die Karte wird daraufhin geprüft, ob sie gültig und nicht gesperrt ist.

Es wird geprüft, ob das Terminal für die entsprechende Transaktion zugelassen ist.

Im folgenden werden die einzelnen Prüf- und Verarbeitungsschritte näher erläutert.

3.6.2.1. Eingangsprüfungen

Bei dem Routen einer Nachricht an die für die Börsenkarte zuständige Ladezentrale kann die Nachricht mit Antwortcodes 91, 92 oder 96 abgewiesen werden.

Wenn Datum und Uhrzeit nicht in einem Zeitfenster von ± 2 Stunden um den aktuellen Zeitpunkt liegen, wird die Nachricht mit Antwortcode 98 abgewiesen..

Wenn versucht wird, den FBZ an einem institutsfremden Terminal zu erhöhen, weist die Ladezentrale die Karte mit Antwortcode 58 ab.

3.6.2.2. Prüfung des Terminal-MAC

Der Terminal-MAC in BMP 64 der Anfragenachricht wird mittels der Verschlüsselungsparameter in BMP 57 und der Beschreibung zu BMP 57 geprüft. Im Fehlerfall wird die Nachricht mit Antwortcode 97 abgewiesen.

3.6.2.3. Schlüsselableitung

Der 16 Byte lange kartenindividuelle Schlüssel K_{LD} bzw. K_{Card} wird aus dem jeweiligen Masterkey KGK_{LD} bzw. KGK_{Card} und den Kartenidentifikationsdaten abgeleitet. Welcher KGK_{LD} bzw. KGK_{Card} hierzu zu verwenden ist, wird durch die jeweilige Generationsnummer KV aus Byte 41 bzw. 42 der BMP 62 festgelegt. Die Kartenidentifikationsdaten sind immer Byte 1-22 der BMP 62 zu entnehmen.

Der kartenindividuelle Schlüssel K (K_{LD} bzw. $K_{Card)}$ von 16 Byte Länge wird aus

- KGK (K_{LD} bzw. KGK_{Card}) (16 Byte),
- CID (Kartenidentifikationsdaten aus Byte 1-22 der BMP 62 mit '00 00' auf die Länge von 24 Byte gepaddet), und
- dem Initialwert I = '52 52 52 52 52 52 52 52 25 25 25 25 25 25 25 25' (16 Byte)

zu

$$K = P(d*KGK(H(I,CID))).$$

berechnet. Hierbei bezeichnen

- **P** die Funktion "Parity Adjustment", die wie folgt definiert ist:

  Sei $b_1,..,b_8$ die Darstellung eines Byte als Folge von 8 Bit. Dann setzt P das niedrigstwertige Bit b_8 jedes Byte auf ungerade Parität, d. h. b_8 wird in jedem Byte so gesetzt, daß es eine ungerade Anzahl von 1 enthält.

- **d*KGK** die Triple-DES Entschlüsselung im ECB-Mode mit dem Schlüssel KGK,

- **H** die in ISO 10118-2 definierte Hash-Funktion, die Werte X mit einer Länge, die ein Vielfaches von 8 Byte ist, mittels des Startwertes I (gemäß Anhang A von ISO 10118-2) auf einen Wert von 16 Byte Länge abbildet. Es werden die um das Parity Adjustment P erweiterten Transformationen u (Ad10) und u' (Ad 01) aus Anhang A von ISO 10118-2 verwendet. H ist rekursiv definiert:

  - Sei $X = x_1|...|x_n$ die Zerlegung des Wertes X in 8 Byte lange Blöcke und $L_0|R_0$ die Zerlegung des vorgegebenen Startwerts I in zwei 8 Byte Blöcke.

  - eK(X) ist die DES-Verschlüsselung eines 8 Byte Wertes mit einem 8 Byte Schlüssel.

  - $\oplus$ sei die bitweise Addition modulo 2 (XOR).

  - Die Transformationen Ad10 und Ad01 transformieren 8 Byte Werte K wie folgt:

    Sei $K = k_1,...,k_{64}$ die Darstellung von K als Folge von 64 Bit. Dann ist

    $Ad10(K) = [P](k_1,1,0,k_4,...,k_{64})$

    $Ad01(K) = [P](k_1,0,1,k_4,...,k_{64})$

    [P]: Das Parity Adjustment in Ad10 und Ad01 ist optional. Wenn vor der DES-Verschlüsselung eK(X) keine Paritätsprüfung des Schlüssels K erfolgt, kann P entfallen.

  - Dann errechnet sich $L_i|R_i$ aus $L_{i-1}|R_{i-1}$ und x_i wie folgt:

    L_{i-1} bestehe aus den Bits $l_1,...,l_{64}$ und R_{i-1} bestehe aus den Bits $r_1,...,r_{64}$,

    Schritt 1: $L'_i := Ad10(L_{i-1}) = [P](l_1,1,0,l_4,...,l_{64})$

    $R'_i := Ad01(R_{i-1}) = [P](r_1,0,1,r_4,...,r_{64})$

    Schritt 2: $A_i = A_{i[links]}|A_{i[rechts]} = eL'_i(x_i) \oplus x_i.$

    $B_i = B_{i[links]}|B_{i[rechts]} = eR'_i(x_i) \oplus x_i.$

    Schritt 3: $L_i = A_{i[links]}|B_{i[rechts]}$

$$R_i = B_{i[links]} | A_{i[rechts]}$$

- $L_n|R_n$ ist dann der Hash-Wert von X unter H:

$$H(I,X) = L_n|R_n$$

3.6.2.4. Gültigkeitsprüfung der Börsenkarte

Eine Börsenkarte ist für das Hochsetzen des FBZ bis zu ihrem Verfalldatum gültig. Ist die Börsenkarte für die gewünschte Transaktion nicht gültig, wird die Nachricht mit dem Antwortcode 54 abgewiesen.

Wenn das Aktivierungsdatum der Börsenkarte in Byte 13-15 der BMP 62 in der Zukunft liegt, wird die Nachricht ebenfalls mit Antwortcode 54 abgewiesen.

3.6.2.5. Sperrenprüfung der Börsenkarte

Wenn die Börsenkarte gesperrt ist, wird die Nachricht (je nach Art der Sperre) mit Antwortcode 04 oder 05 abgewiesen.

3.6.2.6. Prüfung der Terminaldaten

Es wird geprüft, ob das Terminal für die jeweilige Transaktion zugelassen ist.

Wird bei einer dieser Prüfungen ein Fehler festgestellt, wird die Nachricht mit dem Antwortcode 58 abgelehnt.

3.6.2.7. Formale Prüfung der Anfragenachricht

Es wird geprüft, ob Aufbau und Inhalt der Nachricht den Anforderungen der Nachrichtenbeschreibung genügen. Werden hierbei Fehler festgestellt, wird die Nachricht mit Antwortcode 30 abgelehnt.

3.6.2.8. Entschlüsselung und Prüfung der Konto- und Kartendaten

Die Byte 23-38 der BMP 62 enthalten die unter K_{LD} verschlüsselten Karten- und Kontodaten der GeldKarte, wenn es sich um eine kontobezogene GeldKarte handelt. Werden diese Daten im

Klartext benötigt, sind diese 16 Byte der Nachricht zu entnehmen und mittels K_{LD} Triple-DES zu entschlüsseln (CBC-Mode mit ICV = '00..00', vgl. Kapitel 2.3.2 in [LIT 1]). Als Ergebnis erhält man

- 4 Byte BCD-kodierte Bankleitzahl des Kartenkontos,
- 5 Byte BCD-kodierte Kontonummer des Kartenkontos,
- 2 Byte BCD-kodierte rechtsbündig eingestellte 3-stellige Kartenfolgenummer der Karte mit vorangestellter '0',
- 1 Byte BCD-kodierter rechtsbündig eingestellter 1-stelliger Freizügigkeitsschlüssel der Karte mit vorangestellter '0' und
- 4 Byte Filler '00 00 00 00'.

Die entschlüsselten Daten werden geprüft. Werden hierbei Fehler oder Inkonsistenzen festgestellt, wird die Nachricht mit dem Antwortcode 56 abgewiesen.

3.6.2.9. Berechnung des Börsenkarten-MAC

Der MAC für die Börsenkarte wird durch die Ladezentrale mit K_{Card} berechnet und in Byte 8-15 der BMP 62 einer positiven Antwortnachricht eingestellt.

Es wird der Retail-CFB-MAC mit K_{Card} über die Daten in BMP 62 Byte 1-7 berechnet und in Byte 8-15 eingestellt.

3.6.2.10. Berechnung des Terminal-MAC

Der Terminal-MAC in BMP 64 der Antwortnachricht wird mittels der Verschlüsselungsparameter in BMP 57 und nach den dort beschriebenen Regeln gebildet und geprüft.

3.7. Nachladen

Das Nachladen ist wie das Ändern von Chipdaten ein sicherheitskritischer Vorgang. Es wird unterschieden zwischen Nachladen von Daten und Nachladen von Kommandos.

3.7.1. Nachladen von Daten

Das Nachladen von Daten geschieht mittels der Kartenkommandos APPEND, CREATE, DELETE,

INCLUDE und EXCLUDE. Es wird dazu ein Zugriff auf den Schlüssel KGK_{Card} benötigt.

Die Ausführung dieser Funktion benötigt eine vorherige Identifikation und Authentikation des Bedieners. Durch die Zugriffskontrolle muß sichergestellt werden, daß nur berechtigte Bediener diese Funktion ausführen dürfen.

Der Schlüssel KGK_{Card} darf nur durch diese Funktion, die Funktion "Nachladen von Kommandos" (siehe 3.6.2) und die Funktion "Ändern von Chipdaten" (siehe 3.5) verwendet werden.

Über die menuegesteuerte Bedienerführung muß der Bediener die Art des Nachladens von Daten (je nach gewünschtem Kartenkommando) auswählen können.

Alle Ausführungen von Funktionen dieser Funktionsgruppe sind zu protokollieren.

3.7.2. Nachladen von Kommandos

Das Nachladen von Kommandos geschieht mittels des Kartenkommandos LOAD COMMAND. Es wird dazu ein Zugriff auf die Schlüssel KGK_{Card} und K_{SIG} benötigt.

Die Ausführung dieser Funktion benötigt eine vorherige Identifikation und Authentikation des Bedieners. Durch die Zugriffskontrolle muß sichergestellt werden, daß nur berechtigte Bediener diese Funktion ausführen dürfen. Diese Funktion sollte nur von wenigen besonders berechtigten Bedienern ausgeführt werden können.

Der Schlüssel KGK_{Card} darf nur durch diese Funktion, die Funktion "Nachladen von Daten" (siehe 3.6.1) und die Funktion "Ändern von Chipdaten" (siehe 3.5) verwendet werden.

Der Schlüssel K_{SIG} darf nur durch diese Funktion verwendet werden.

Alle Ausführungen dieser Funktion sind zu protokollieren.

Vor dem Nachladen eines Kommandos muß die Signatur des Kommandocodes überprüft werden. Dazu wird der Retail-CBC-MAC über den Kommandocode unter Verwendung von K_{SIG} berechnet und mit der Signatur verglichen.

Es muß sichergestellt werden, daß der Bediener über den Ausgang der Überprüfung der Signatur nicht getäuscht werden kann.

Nach einer erfolgreichen Überprüfung der Signatur muß der Kommandocode in Blöcke mit einer Länge von maximal 247 Byte aufgeteilt werden. Die einzelnen Blöcke werden dann jeweils mit einem LOAD COMMAND Kommando an die Chipkarte gegeben.

Bei dieser Funktion ist die Ablaufkontrolle von besonderer Bedeutung. Es muß insbesondere sichergestellt werden, daß zwischen der Überprüfung der Signatur des Kommandocodes und dem Laden des letzten Blocks mittels LOAD COMMAND keine Manipulationen an dem Kommandocode (bzw. den einzelnen Blöcken) oder an dem Ablauf der Funktion stattfinden können.

3.8. Erweiterung der Funktionalität

Die Funktionalität des Bankensonderfunktionsterminals kann durch das nachträgliche Einbringen von Software erweitert werden. Dadurch darf jedoch die Sicherheit der bisher beschriebenen Funktionalität nicht beeinträchtigt werden.

Das nachträgliche Einbringen von Software darf nur von zwei berechtigten Bedienern nach dem Vier-Augen-Prinzip durchgeführt werden. Die Ausführung dieser Funktion benötigt eine vorherige Identifikation und Authentikation der beiden Bediener. Durch die Zugriffskontrolle muß sichergestellt werden, daß nur berechtigte Bediener diese Funktion ausführen dürfen. Diese Funktion sollte nur von wenigen besonders berechtigten Bedienern ausgeführt werden können.

Alle Ausführungen dieser Funktion sind zu protokollieren.

Die Echtheit der nachträglich einzubringenden Software muß überprüfbar sein.

Die Ausführung der neu eingebrachten Funktionen muß der Zugriffskontrolle unterliegen. Dabei ist darauf zu achten, daß die bisher gültigen Einschränkungen der Zugriffe auf Funktionen und Chipdaten nicht umgangen bzw. "aufgelockert" werden.

4. Zulassung von Bankensonderfunktionsterminals

Es dürfen nur durch den ZKA zugelassene Bankensonderfunktionsterminals eingesetzt werden. Bedingung für die Zulassung ist die Vorlage eines Sicherheitsgutachtens eines Sicherheitsgutachters aus der Liste der vom ZKA benannten Gutachter für das Sicherheitsmodul des Bankensonderfunktionsterminals. Hierbei wird überprüft, ob das Sicherheitsmodul die in Kriterium VII aus "Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen" in [LIT K] formulierten Anforderungen an die Hardware erfüllt.

Ergebnis eines positiven Sicherheitsgutachtens ist eine Typzulassung des entsprechenden Sicherheitsmoduls.

Die funktionale Abnahme eines Bankensonderfunktionsterminals obliegt dem betreibenden Kreditinstitut.

GeldKarte
– Taschenkartenleser –

Version 2.2

22.01.1997

Inhalt

1. Funktionalität eines Taschenkartenlesers

Mit einem Taschenkartenleser muß der Benutzer den aktuell in der elektronischen Geldbörse gespeicherten Betrag lesen können.

Optional können weitere frei lesbare Daten, beispielsweise die Protokolldaten der letzten Lade-, Entlade- oder Zahlungs-Transaktionen angezeigt werden.

In einem Taschenkartenleser dürfen keine kryptographischen Schlüssel gespeichert werden.

Im folgenden wird anhand eines Diagramms erläutert, mit welchen Kommandos durch einen Taschenkartenleser die Beträge und die Protokolldaten der Applikation elektronische Geldbörse einer GeldKarte gelesen werden können. Hierbei muß nur die Reihenfolge der ersten beiden Schritte eingehalten werden. Die übrigen Schritte können in beliebiger, für den Karteninhaber nachvollziehbarer Weise erfolgen. Weitere Schritte und Anzeigen sind zulässig.

Die Fehlermeldungen können durch den Hersteller des Taschenkartenlesers festgelegt werden. Die im folgenden verwendeten Begriffe **erhalten** und **gegeben** sind sinngemäß zu verwenden.

Ferner wird festgelegt, wie die Transaktionsbeträge aus den Protokolldaten bei der Anzeige durch einen Taschenkartenleser zu qualifizieren sind.

GeldKarte			Taschenkartenleser	
		<---	C1	Reset GeldKarte
R1	ATR der GeldKarte	--->		
		<---	C2	SELECT FILE DF_BÖRSE
R2	OK	--->		
		<---	C3	READ RECORD EF_BETRAG
R3	Beträge aus EF_BETRAG	--->		
			A3	Beträge auswerten und anzeigen
		<---	C4	READ RECORD 'XX' EF_BLOG
R4	Record 'XX' aus EF_BLOG	--->		
			A4	Zahlungs-Protokolldaten auswerten und anzeigen
		<---	C5	READ RECORD 'XX' EF_LLOG
R5	Record 'XX' aus EF_LLOG	--->		
			A5	Lade-Protokolldaten auswerten und anzeigen

Erläuterung

1. Nachdem die GeldKarte eingesteckt ist, wird durch den Taschenkartenleser ein Reset der Karte durchgeführt. Hierbei wird verfahren, wie es für das Kommunikationsprotokoll T = 1 festgelegt ist.

   Der korrekte ATR einer GeldKarte ist in Kapitel 7 von [LIT 1] spezifiziert.

   Im Fehlerfall wird mit einer Fehlermeldung abgebrochen.

2. Die Applikation elektronische Geldbörse wird geöffnet, indem das ADF der Applikation DF_BÖRSE durch den Taschenkartenleser mittels des Kommandos SELECT FILE mit der Option "Keine FCI ausgeben" selektiert wird. Die folgende Tabelle zeigt den Aufbau der

Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'A4'	INS
3	1	'04'	P1, Selektion mit DF-Name
4	1	'0C'	P2, Keine Antwortdaten
5	1	'09'	L_c
6-14	9	'D2 76 00 00 25 45 50 01 00'	AID der elektronischen Geldbörse

Im Fehlerfall wird mit einer Fehlermeldung abgebrochen.

Nachdem der Applikationskontext geöffnet ist, können die AEFs der Applikation mittels SFI referenziert werden.

3. Der aktuelle Betrag und die zulässigen Maximalbeträge der GeldKarte können aus Record '01' des EF_BETRAG mit dem Kommando READ RECORD (SFI '18') gelesen werden. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'01'	P1, Recordnummer
4	1	'C4'	P2, Reference Control Byte
5	1	'09'	L_e, Recordlänge des EF_BETRAG

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1-3	3	'nn..nn'	aktueller Betrag
4-6	3	'nn..nn'	Maximalbetrag
7-9	3	'nn..nn'	maximaler Transaktionsbetrag

Im Fehlerfall wird mit einer Fehlermeldung abgebrochen.

Die Beträge sind 6-stellige, BCD-kodierte Pfennigbeträge. Bei der Anzeige des Betrages '02 34 56' aus der GeldKarte in DM ist daher anzuzeigen

234,56 (DM)

4. Mittels READ RECORD können die Protokolldaten der zuletzt mit der GeldKarte getätigten Zahlungen aus den Records des EF_BLOG der GeldKarte (SFI '1D') gelesen werden. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	'00'	CLA ohne Secure Messaging
2	1	'B2'	INS
3	1	'XX'	P1, Recordnummer
4	1	'EC'	P2, Reference Control Byte
5	1	'25'	L_e, Recordlänge des EF_BLOG

Das EF_BLOG einer GeldKarte hat zur Zeit 15 Records. In einer neu ausgegebenen GeldKarte ist nur der erste Record belegt. Durch Zahlungstransaktionen werden die Records sukzessive gefüllt. Nachdem 15 Records belegt sind, werden die Records zyklisch überschrieben.

Im Record mit der Recordnummer '01' ist immer die letzte Zahlung protokolliert, die vorletzte Zahlung in Record '02, etc.

Zum Lesen aller beschriebenen Records kann sukzessive das Kommando READ RECORD verwendet werden, wobei die Recordnummer, beginnend mit '01', solange inkrementiert wird, bis die GeldKarte mit dem Antwortcode '6A 83' (Record not found) anzeigt, daß keine weiteren beschriebenen Records vorhanden sind.

Es ist zu beachten, daß es GeldKarten gibt, die negative Returncodes, insbesondere '6A 83', in einem Fehler-Log protokollieren, so daß auch bei READ RECORD möglicherweise das EEPROM der GeldKarte beschrieben wird.

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-3	2	’XX XX’	Sequenznummer BSEQ der Abbuchungs-/Rückbuchungstransaktion
4-5	2	'XX XX'	Sequenznummer LSEQ des letzten erfolgreichen Ladens
6-8	3	'nn..nn'	abgebuchter/rückgebuchter Betrag
9-18	10	'nn..nD'	Händlerkartennummer der an der Transaktion beteiligten Händlerkarte
19-22	4	'XX..XX'	Sequenznummer HSEQ der beteiligten Händlerkarte
23-26	4	'XX..XX'	Sequenznummer SSEQ der beteiligten Händlerkarte
27-29	3	'nn..nn'	durch die Abbuchungs-/Rückbuchungstransaktion berechneter aktueller Betrag
30-33	4	JJJJ MM TT	Datum der Abbuchungs-/Rückbuchungstransaktion
34-36	3	HH MM SS	Uhrzeit der Abbuchungs-/Rückbuchungstransaktion
37	1	'XX'	Schlüsselnummer KID des zur Zertifikatsberechnung verwendeten K_{RD}

Das Statusbyte in Byte 1 eines Records des EF_BLOG kann bei einer korrekt funktionierenden GeldKarte die folgenden Werte haben:

Statusbyte	Bedeutung
’51’	Abbuchung
’71’	Rückbuchung

Falls das Statusbyte den Wert ’51’ hat, ist der Transaktionsbetrages aus Byte 6-8 des Records als **gegeben** anzuzeigen.

Falls das Statusbyte den Wert ’71’ hat, ist der Transaktionsbetrages aus Byte 6-8 zurückgebucht worden. Durch die protokollierte Transaktion wurde der Betrag in der GeldKarte nicht geändert.

Als Transaktionsbetrag für eine Rückbuchung (Statusbyte '71') ist daher **0** anzuzeigen.

Hat das Statusbyte einen anderen Wert, ist ein Fehler anzuzeigen.

5. Mittels READ RECORD können die Protokolldaten der letzten Lade-/Entladeschritte der GeldKarte aus den Records des EF_LLOG der GeldKarte (SFI '1C') gelesen werden. Die folgende Tabelle zeigt den Aufbau der Command APDU.

Byte	Länge	Wert	Erläuterung
1	1	’00’	CLA ohne Secure Messaging
2	1	’B2’	INS
3	1	’XX’	P1, Recordnummer
4	1	’E4’	P2, Reference Control Byte
5	1	’21’	L_e, Recordlänge des EF_LLOG

Das EF_LLOG einer GeldKarte hat zur Zeit 3 Records. In einer neu ausgegebenen GeldKarte ist nur der erste Record belegt. Durch Lade- oder Entladetransaktionen werden die Records sukzessive gefüllt. Nachdem 3 Records belegt sind, werden die Records zyklisch überschrieben.

Im Record mit der Recordnummer '01' ist immer die letzte Lade-/Entladetransaktion protokolliert, die vorletzte Zahlung in Record '02, etc.

Zum Lesen aller beschriebenen Records kann sukzessive das Kommando READ RECORD verwendet werden, wobei die Recordnummer, beginnend mit '01', solange inkrementiert wird, bis die GeldKarte mit dem Antwortcode '6A 83' (Record not found) anzeigt, daß keine weiteren beschriebenen Records vorhanden sind.

Es ist zu beachten, daß es GeldKarten gibt, die negative Returncodes, insbesondere '6A 83', in einem Fehler-Log protokollieren, so daß auch bei READ RECORD möglicherweise das EEPROM der GeldKarte beschrieben wird.

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die GeldKarte einen Record mit der folgenden Struktur zurück.

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-3	2	’XX XX’	Sequenznummer LSEQ der Lade-/Entladetransaktion
4	1	’XX’	Wiederholungszähler WZ für das **Laden einleiten** und **Entladen einleiten**
5-7	3	'nn..nn'	geladener bzw. entladener Betrag
8-10	3	'nn..nn'	(neuer) aktueller Betrag
11-13	3	'nn..nn'	AS-ID der Ladezentrale
14-21	8	'nn..nn'	Terminal-ID des Ladeterminals
22-24	3	'nn..nn'	Trace-Nummer TSEQ des Ladeterminals
25-27	4	JJJJ MM TT	Datum der Lade-/Entladetransaktion
28-31	3	HH MM SS	Uhrzeit der Lade-/Entladetransaktion
32-33	2	'XX XX'	Sequenznummer BSEQ des letzten Abbuchens bzw. Rückbuchens

Das Statusbyte in Byte 1 des Record ’01’ des EF_LLOG kann bei einer korrekt funktionierenden GeldKarte die folgenden Werte haben:

Statusbyte	Bedeutung (protokolliertes Kommando)
’01’	**Laden einleiten** ohne Secure Messaging
’03’	**Laden einleiten** mit Secure Messaging
’05’	**Laden einleiten wiederholen** ohne Secure Messaging
’07’	**Laden einleiten wiederholen** mit Secure Messaging
’11’	**Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge
'13'	**Laden** mit Secure Messaging ohne Änderung der Maximalbeträge
'15'	**Laden** ohne Secure Messaging mit Änderung der Maximalbeträge
'17'	**Laden** mit Secure Messaging mit Änderung der Maximalbeträge
'21'	**Entladen einleiten**
'25'	**Entladen einleiten wiederholen**
'31'	**Entladen** ohne Änderung der Maximalbeträge
'35'	**Entladen** mit Änderung der Maximalbeträge

Das Statusbyte in Byte 1 der Records ’02’ und ’03’ des EF_LLOG kann bei einer korrekt funktionierenden GeldKarte die folgenden Werte haben:

Statusbyte	Bedeutung (protokolliertes Kommando)
’11’	**Laden** ohne Secure Messaging ohne Änderung der Maximalbeträge
'13'	**Laden** mit Secure Messaging ohne Änderung der Maximalbeträge
'15'	**Laden** ohne Secure Messaging mit Änderung der Maximalbeträge
'17'	**Laden** mit Secure Messaging mit Änderung der Maximalbeträge
'31'	**Entladen** ohne Änderung der Maximalbeträge
'35'	**Entladen** mit Änderung der Maximalbeträge

Falls das Statusbyte einen der Werte ’11’, ’13’, ’15’ oder ’17’ hat, ist der Transaktionsbetrages aus Byte 5-7 des Records als **erhalten** anzuzeigen.

Falls das Statusbyte einen der Werte ’31’, oder ’35’ hat, ist der Transaktionsbetrages aus Byte 5-7 des Records als **gegeben** anzuzeigen.

Falls das Statusbyte in Record ’01’ einen der Werte ’01’, ’03’, ’05’. ’07’, ’21’ oder ’25’ hat, ist der Transaktionsbetrages aus Byte 5-7 noch nicht gebucht worden. Durch die protokollierte Transaktion wurde der Betrag in der GeldKarte nicht geändert.

Als Transaktionsbetrag ist in diesen Fällen daher **0** anzuzeigen.

Hat das Statusbyte einen anderen Wert, ist ein Fehler anzuzeigen.

2. Elektrische Spezifikation und Übertragungsprotokoll

Die elektrische Spezifikation der Schnittstelle zwischen einem Terminal und der GeldKarte muß den Vorgaben aus [ISO 4], [ISO 4'], [ISO 4''] und [EMV 1] genügen.

Das Übertragungsprotokoll zwischen Terminal und ZKA-Chipkarten ist T=1 nach [ISO 4'], [ISO 4''] und [EMV 1].

Zur Behandlung der Unterschiede der Spezifikation von T=1 in [EMV 1] einerseits und [ISO 4'] andererseits werden die folgenden Festlegungen gemacht. Die hier formulierten Anforderungen muß ein Terminal erfüllen, um die ZKA-Zulassung zu erhalten.

Zusätzlich werden für Taschenkartenleser zulässige Einschränkungen des Protokolls definiert.

Im folgenden wird unter der Unterstützung des Empfangens eines Blocktyps verstanden, daß die T=1-Implementierung einen bestimmten Blocktyp nach dem Empfangen als ein dem Protokoll entsprechenden Blocktyp erkennt, verarbeitet und eventuell beantwortet.

2.1. NAD

Gemäß [EMV 1] müssen Terminals NAD='00' verwenden. Andere Werte sind nicht zulässig. Die GeldKarte lehnt daher NAD≠'00' als Protokollfehler ab.

Daher müssen alle vom Terminal gesendeten Blöcke stets NAD='00' enthalten.

2.2. Länge (LEN)

2.2.1. Minimale Blocklänge

Gemäß [EMV 1] wird das Senden und Empfangen von I-Blöcken mit LEN=0 nicht unterstützt. Eine GeldKarte kann somit den Empfang eines I-Blocks mit LEN=0 als einen Protokollfehler abweisen.

Daher darf das Terminal keinen I-Block mit LEN=0 senden.

2.2.2. IFSC und IFSD

Die GeldKarte teilt ihr IFSC im ATR mit. Zur Zeit hat IFSC mindestens den Wert 60. Größere Werte

von IFSC sind zulässig.

Gemäß [ISO 4'] muß ein Terminal nur den Standardwert IFSD=32 unterstützen und kann mit einem S(IFS request) IFSD<IFSC erreichen.

Gemäß [EMV 1] muß ein Terminal IFSD=254 für das Empfangen von Blöcken unterstützen. Ein S(IFS request) des Terminals mit IFSD<IFSC ist nicht zulässig. Die Chipkarte kann immer Blöcke mit LEN≤IFSC senden.

Daher muß das Terminal für Sitzungen mit GeldKarten IFSD≥IFSC für alle Typen von GeldKarten unterstützen, auch für solche GeldKarten, die im ATR den Wert IFSC=254 angeben.

Dies kann "stillschweigend" geschehen, indem das Terminal Blöcke mit 32<LEN≤IFSC ohne Fehlermeldung akzeptiert, oder indem es als ersten Block nach dem Empfang des ATR ein S(IFS request) sendet mit IFSC≤IFSD≤254.

Regelung für Taschenkartenleser

Für Taschenkartenleser, die das Empfangen von I-Blöcken mit LEN=254 nicht unterstützen, kann eine zeitlich befristete Zulassung ausgesprochen werden, wenn die zur Zeit maximale Antwortlänge der durch den Taschenkartenleser verwendeten Kommandos verarbeitet werden kann.

2.2.3. Chaining

Gemäß [EMV 1] gilt für das Chaining vom Terminal zur Chipkarte, daß für alle Blöcke außer für den abschließenden LEN=IFSC gilt. In [ISO 4'] ist diese Forderung nicht explizit enthalten. Das Terminal darf während des Chaining grundsätzlich mit LEN≤IFSC senden. GeldKarten lehnen den Empfang von I-Blöcken außer des letzten Blockes mit LEN<IFSC während des Chainings als einen Protokollfehler ab.

Das Terminal muß daher bei dem Chaining zur GeldKarte alle I-Blöcke außer dem letzten mit LEN=IFSC senden.

Regelung für Taschenkartenleser

Für Taschenkartenleser, die das Chaining von der Chipkarte zum Terminal nicht unterstützen, kann eine zeitlich befristete Zulassung ausgesprochen werden, falls keine der sich im Feld befindlichen GeldKarten beim Senden der Antwortnachrichten zu den vom Taschenkartenleser verwendeten Kommandos ein Chaining durchführt.

2.3. Behandlung von R-Blöcken

Regelung für Taschenkartenleser

R-Blöcke mit b4-b1 = '0' werden für die Bestätigung beim Chaining benötigt. Taschenkartenleser

müssen solche R-Blöcke nur dann senden können, falls sie das Chaining zwischen Chipkarte und Terminal unterstützen. Taschenkartenleser müssen solche R-Blöcke nur dann empfangen können, falls sie das Chaining zwischen Terminal und Chipkarte unterstützen.

2.4. Behandlung von S-Blöcken

Das Senden und Empfangen von S-Blöcken durch das Terminal darf gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Für das Senden eines S(IFS request) gilt allerdings die oben gemachte Einschränkung, daß IFSD<IFSC hierbei nicht zulässig ist.

Regelungen für Taschenkartenleser

Taschenkartenleser müssen S(IFS request) und S(ABORT request) weder senden noch empfangen können.

Taschenkartenleser müssen nicht S(RESYNCH request) senden können.

Für Taschenkartenleser, die S(WTX request) nicht verarbeiten können, kann eine zeitlich befristete Zulassung ausgesprochen werden, falls keine der sich im Feld befindlichen GeldKarten bei der Verarbeitung der vom Taschenkartenleser verwendeten Kommandos eine Verlängerung der Blockwartezeit benötigt.

2.5. Fehlerbehandlung

Die Fehlerbehandlung durch das Terminal kann gemäß [ISO 4'] realisiert werden, auch wenn sich hieraus Abweichungen von [EMV 1] ergeben. Hierbei ist die folgende Anforderung zu beachten:

Empfängt das Terminal während der Protokollabwicklung einen R-Block, so ist nur der Sequenzzähler für den weiteren Protokollablauf ausschlaggebend. Die gemäß [ISO 4'] optionale Auswertung der Bit b1 bis b4 des PCB darf durch das Terminal nur zu informativen Zwecken erfolgen.

Regelung für Taschenkartenleser

Taschenkartenleser müssen nicht die vollständige Fehlerbehandlung gemäß [EMV 1] implementieren. Für Taschenkartenleser ist es zulässig, nicht nur bei dem Erkennen von Protokollfehlern sondern auch bei dem Erkennen von Übertragungs- und Synchronisationsfehlern die Chipkarten-Kontakte sofort zu deaktivieren.

3. Zulassung von Taschenkartenlesern

Es dürfen nur vom ZKA zugelassene Taschenkartenleser eingesetzt werden. Bedingung für die

Zulassung ist die funktionale Abnahme durch eine Stelle des Kreditgewerbes.

Gegenstand der Abnahme sind insbesondere

- elektromechanische Eigenschaften der Schnittstelle zwischen Taschenkartenleser und Chipkarte und
- korrekte Auswertung der Statusbytes bei der Anzeige von Protokolldaten.

Die Abnahme wird

- anhand von Testfällen,
- mit einer Chipkartensimulation der GeldKarte und
- mittels Checklisten, die durch den Hersteller auszufüllen sind,

durchgeführt.

Ergebnis der Abnahme ist eine Typzulassung.

Anforderungen an die elektromechanischen Eigenschaften von Chipkarten-Terminals

Version 2.2
22.01.1997

Inhalt

2.1 Input/Output (I/O)

2.2 Programmierspannung (VPP)

2.3 Clock (CLK)

2.4 Reset (RST)

2.5 Versorgungsspannung (VCC)

2.6 Kontaktwiderstand

2.7 Kurzschlußverträglichkeit

2.8 Ein- und Ausschalten des Terminals mit einsteckender ICC

2.9 ICC-Einbringungs- und Kontaktaktivierungs-Sequenz

2.10 Physikalischer Transport von Zeichen

Checkliste zur Überprüfung der Anforderungen an die elektromechanischen Eigenschaften von Chipkarten-Terminals

A. Allgemeine Bemerkungen

Die nachfolgenden Tabellen dienen den Herstellern von Terminals zur Überprüfung der geforderten mechanischen und elektrischen Eigenschaften ihrer Terminals. Kann eine Anforderung durch die empfohlenen Testmethoden und -werkzeuge erfüllt werden, so ist diese in der Spalte "Überprüfung erfolgt" abzuhaken. Bei Verwendung abweichender Testmethoden und -werkzeuge oder Abweichungen von den Anforderungen sind diese in einer Anlage zu beschreiben und zu begründen. Durch Vermerke in der Spalte "Verweis auf Abweichungen" ist auf diese Anlagen zu verweisen.

1 Mechanische Eigenschaften

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Akzeptanz von ICCs mit den physikalischen Eigenschaften entsprechend ISO 7816-1	EMV 1.1 Part I, 5.3.1	Überprüfung mit Referenzkarten			
2	Akzeptanz von ICCs mit den Kontakten an der Position entsprechend Figure 2 in ISO 7816-2	EMV 1.1 Part I, 5.3.1	Überprüfung mit Referenzkarten			
3	Akzeptanz von ICCs mit Hochprägung entsprechend ISO 7811-1	EMV 1.1 Part I, 5.3.1	Überprüfung mit Referenzkarten			
4	Positionierhilfen und Klammern dürfen keine Gefahr für die ICC darstellen	EMV 1.1 Part I, 5.3.1	visuelle und praktische Kontrolle			
5	Kontakte C1, C2, C3, C5 und C7 belegt nach Table 6	EMV 1.1 Part I, 5.3.2	visuelle Kontrolle und Zuordnung der Signale	Oszilloskop, Logikanalysator		

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
6	Kontakt C6 elektrisch isoliert	EMV 1.1 Part I, 5.3.2	Widerstandsmessung	Multimeter		
7	Empfehlung: Kontakte C4 und C8, falls unterstützt, nicht treibend	nicht definiert	Strommessung gegen GND	Multimeter		
8	Anpreßdruck für alle unterstützten Kontakte nicht stärker als 0,6 N Empfehlung: Anpreßdruck für alle unterstützten Kontakte zwischen 0,2 und 0,6 N	EMV 1.1 Part I, 5.3.2 EMV 3.0 Part I, 1.3.2	Adapterkarte mit Drucksensoren (Piezos) auf den ISO-Kontaktfeldern	seperate Druckaufnehmer für jeden Kontakt bzw ein Druckaufnehmer für gesamtes Kontaktfeld, wobei die einzelnen Kontaktfeder nacheinander gemessen werden		

2 Elektrische Eigenschaften

Alle Maße sind am Kontaktpunkt zwischen ICC und IFD in bezug auf den GND-Kontakt definiert über den Umgebungstemperaturbereich von 0º C bis 50º C.

Aus dem Terminal herausfließende Ströme werden als positiv betrachtet.

2.1 Input/Output (I/O)

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Überprüfung erfolgt
1	keine Gefahr für das IFD, im Falle, daß sich IFD und ICC gleichzeitig im Sendemodus befinden	EMV 1.1 Part I, 5.4.2	während ATR-Sequenz wird I/O des IFD auf H gesetzt	Modifikation des IFD		
2	im Falle, daß sich IFD und ICC gleichzeitig im Empfangsmodus befinden, befindet sich der Kontakt im High-Zustand (H)	EMV 1.1 Part I, 5.4.2	Pegelmessung an I/O	Multimeter oder Logiktester über Adapterkarte im Kartenleser		
3	High-State erreicht durch pull-up-Widerstand zu VCC mit Nominalwert 20 k↓.	EMV 1.1 Part I, 5.4.2	Strom an I/O gegen GND zwischen 0,2 und 0,3 mA	Multimeter über Adapterkarte im Kartenleser		
4	Kontakt befindet sich nicht im High-Zustand, solange nicht VCC unter Spannung ist.	EMV 1.1 Part I, 5.4.2	Pegelmessung an I/O und VCC	Multimeter oder Logiktester über Adapterkarte im Kartenleser		
5	Empfehlung: max. Stromfluß an I/O ⇧5 mA	EMV 3.0 Part I, 1.4.2.1	Strommessung an I/O	Multimeter über Adapterkarte im Kartenleser		
6	im Sende-Modus gilt für V_{OH}: $0.8*V_{CC}$ ⇔V_{OH} ⇔V_{CC} bei 0 A < I_{OH} < 20 ↘A bei V_{CC}=max Empfehlung: $0.8*V_{CC}$ ⇔V_{OH} ⇔V_{CC} bei -20 ↘A < I_{OH} < 20 ↘A bei V_{CC}=min	EMV 1.1 Part I, 5.4.2 EMV 3.0 Part I, 1.4.2.1	Strom- und Spannungsmessung am I/O	Multimeter, Logiktester oder Oszilloskop über eine Adapterkarte im Kartenleser		
7	im Sende-Modus gilt für V_{OL}: 0 V ⇔V_{OL} ⇔0,3 V bei 0 A > I_{OL} > -1 mA bei V_{CC}=max	EMV 1.1 Part I, 5.4.2.1	Strom- und Spannungsmessung am I/O	Multimeter, Logiktester oder Oszilloskop über eine Adapterkarte im Kartenleser		
8	t_R ⇔0,8 ↘s und t_F ⇔0,8 ↘s bei $C_{IN(ICC)}$ = 30pF max	EMV 1.1 Part I, 5.4.2.1	Messung der Flankensteilheit zwischen V_{OLmax} und $V_{$	Oszilloskop über Adapterkarte im Kartenleser angeschlossen		

9	Unterschwinger nicht unter -0,25 V	EMV 1.1 Part I, 5.4.2.1	Spitzenwertmessung der Unterschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
10	Überschwinger nicht über V_{CC} + 0,25 V	EMV 1.1 Part I, 5.4.2.1	Spitzenwertmessung der Überschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
11	Außerhalb von Sendephasen ist der I/O-Treiber auf Empfangs-Modus gesetzt.	EMV 1.1 Part I, 5.4.2.1	Strom an I/O gegen GND zwischen 0,2 und 0,3 mA	Multimeter über Adapterkarte im Kartenleser		
12	im Empfangs-Modus wird V_{IH} als H akzeptiert, wenn: $0.6*V_{CC} \Leftrightarrow V_{IH} \Leftrightarrow V_{CC}$	EMV 1.1 Part I, 5.4.2.2	Pegelmessung an I/O und Auswertung der vom IFD gelesenen Date	Multimeter oder Logiktester und Interpretation der gelesenen Daten von der Software		
13	im Empfangs-Modus wird V_{IL} als L akzeptiert, wenn: $0\ V \Leftrightarrow V_{IL} \Leftrightarrow 0,5\ V$	EMV 1.1 Part I, 5.4.2.2	Pegelmessung an I/O und Auswertung der vom IFD gelesenen Date	Multimeter oder Logiktester und Interpretation der gelesenen Daten von der Software		
14	im Empfangs-Modus werden $t_R \Leftrightarrow 1,2$ µs und $t_F \Leftrightarrow 1,2$ µs akzeptiert	EMV 1.1 Part I, 5.4.2.2	Messung der Flankensteilheit zwischen V_{OLmax} und V_{OHmin}	Oszilloskop über Adapterkarte im Kartenleser ange-schlossen		

2.2 Programmierspannung (VPP)

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	IFD generiert keine VPP	EMV 1.1 Part I, 5.4.3	Pegelmessung an VPP	Multimeter oder Logiktester über Adapterkarte im Kartenleser		

2.3 Clock (CLK)

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Terminal generiert V_{OH} mit: V_{CC} - 0,5 V ⇔V_{OH} ⇔ V_{CC} bei 0 A < I_{OH} < 50 µA bei V_{CC}=min	EMV 1.1 Part I, 5.4.4	Strom- und Spannungsmessung am CLK	Multimeter oder Logiktester über Adapterkarte im Kartenleser		
2	Terminal generiert V_{OL} mit: 0 V ⇔V_{OL} ⇔0,4 V bei 0 A > I_{OL} > -50 µA bei V_{CC}=max	EMV 1.1 Part I, 5.4.4	Strom- und Spannungsmessung am CLK	Multimeter oder Logiktester über Adapterkarte im Kartenleser		
3	t_R ⇔8 % und t_F ⇔8 % des Taktzyklus bei $C_{IN(ICC)}$ = 30pF max	EMV 1.1 Part I, 5.4.4	Messung der Flankensteilheit zwischen V_{OLmax} und V_{OHmin}	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
4	Unterschwinger nicht unter -0,25 V	EMV 1.1 Part I, 5.4.4	Spitzenwertmessung der Unterschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
5	Überschwinger nicht über V_{CC} + 0,25 V	EMV 1.1 Part I, 5.4.4	Spitzenwertmessung der Überschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. messung mit Oszilloskop		
6	Tastverhältnis im Bereich zwischen 45 % und 55 % des Taktzyklus	EMV 1.1 Part I, 5.4.4	Frequenzähler oder Oszilloskop	Frequenzzähler mit Pulsbreitenerkenung bzw. Oszilloskop		
7	1 MHz ⇔Taktfrequenz ⇔5 MHz	EMV 1.1 Part I, 5.4.4	Frequenzmessung	Frequenzzähler		
8	Taktfrequenzabweichung während einer Sitzung unter ⇧1 %	EMV 1.1 Part I, 5.4.4	Protokollierung der Abweichung während Frequenzmessung	Frequenzzähler mit max./ min.-Erkennung		

2.4 Reset (RST)

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Terminal generiert V_{OH} mit: V_{CC} - 0,5 V ⇔V_{OH} ⇔ V_{CC} bei 0 A < I_{OH} < 50 µA bei V_{CC}=min	EMV 1.1 Part I, 5.4.5	Strom- und Spannungsmessung am RST	Multimeter, Logiktester oder Oszilloskop über eine Adapterkarte im Kartenleser		
2	Terminal generiert V_{OL} mit: 0 V ⇔V_{OL} ⇔0,4 V bei 0 A > I_{OL} > -50 µA bei V_{CC}=max	EMV 1.1 Part I, 5.4.5	Strom- und Spannungsmessung am RST	Multimeter, Logiktester oder Oszilloskop über eine Adapterkarte im Kartenleser		
3	t_R ⇔0,8 µs und t_F ⇔0,8 µs bei $C_{IN(ICC)}$ = 30pF max	EMV 1.1 Part I, 5.4.5	Messung der Flankensteilheit zwischen V_{OLmax} und V_{OHmin}	Oszilloskop über Adapterkarte im Kartenleser angeschlossen		
4	Unterschwinger nicht unter -0,25 V	EMV 1.1 Part I, 5.4.5	Spitzenwertmessung der Unterschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
5	Überschwinger nicht über V_{CC} + 0,25 V	EMV 1.1 Part I, 5.4.5	Spitzenwertmessung der Überschwinger bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		

2.5 Versorgungsspannung (VCC)

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Terminal generiert V_{CC} = 5 V ⇧0,25 V	EMV 1.1 Part I, 5.4.6	Spannungsmessung am VCC	Multimeter, Logiktester oder Oszilloskop über eine Adapterkarte im Kartenleser		
2	Strom an VCC ⇦50 mA Dieser Strom muß vom Terminal geliefert werden können. Empfehlung: Strom an VCC ⇦55 mA	EMV 1.1 Part I, 5.4.6 EMV 3.0 Part I, 1.4.6	Messung des Kurzschlußstrom gegen GND.	Multimeter über eine Adapterkarte im Kartenleser		
3	Schutzvorkehrungen gegen intern oder extern verursachte Spannungsschwankungen und Spannungsspitzen	EMV 1.1 Part I, 5.4.6	Spitzenwertmessung der Schwankungen bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		
4	Entkopplung von Strom-spitzen bis zu einer Ladung von 40 nAs innerhalb einer max. Verzögerung von 400 ns und einer max. Amplitude bis 100 mA bei Einhaltung von V_{CC} (Punkt 1)	EMV 1.1 Part I, 5.4.6	Spitzenwertmessung der Schwankungen bzw. visuelle Kontrolle	Multimeter mit Peak Detection bzw. Messung mit Oszilloskop		

2.6 Kontaktwiderstand

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Kontaktwiderstand zwischen sauberen IFD- und ICC-Kontakten < 100 mΩ während garantierten Lebenszeit des Terminals	EMV 1.1 Part I, 5.4.7	Widerstandsmessung bei min. und max. Anpressdruck der Kontaktfedern	Multimeter oder Widerstandsmeßbrücke		

2.7 Kurzschlußverträglichkeit

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Kurzschlüsse zwischen den Kontaktflächen führen beim Terminal zu keiner Gefahr oder Fehlfunktion .	EMV 1.1 Part I, 5.4.8	Überprüfung der Funktionsfähigkeit des Terminals nach paarweisen Kurzschlüssen von Kontakten	Adapterkarte im Kartenleser mit den jeweilig kurzgeschlossenen ISO-Kontakten		

2.8 Ein- und Ausschalten des Terminals mit einsteckender ICC

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	keine Signalverfälschungen an den Kontakten	EMV 1.1 Part I, 5.4.9	visuelle Kontrolle und Messung der Übergangs-widerständen und kapazitäten der ISO-Kontakte zum Kartenleser	Oszilloskop und Multimeter		

2.9 ICC-Einbringungs- und Kontaktaktivierungs-Sequenz

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
1	Bei Einbringung der ICC befinden sich alle Kontakte im Zustand L. V_{CC} ⇔0,4 V	EMV 1.1 Part I, 6.1.2	Pegelmessung an Adapterkarte mit ausgeschalter Karten-Einsteckkontrolle	Multimeter über Adapterkarte im Kartenleser		
2	Terminal erkennt die korrekte Positionierung der ICC bis auf 0,5 mm genau in Richtung der Einbringung.	EMV 1.1 Part I, 6.1.2	Abstandsmessung zwischen Ansprechen der Einsteckkontrolle und Endanschlag einer Refenzkarte			
3	Kontaktaktivierungs-Sequenz: RST verbleibt im Zustand L während der Aktivierungssequenz. Vor Aktivierung von I/O oder CLK muß VCC aktiviert werden. Terminal verfiziert, daß V_{CC} stabil ist. I/O wird auf Empfangs-Modus gesetzt (spätestens 200 Taktzyklen nach .Zuschaltung von CLK). CLK wird mit stabilem Takt versorgt.	EMV 1.1 Part I, 6.1.2	visuelle Beobachtung und Auszählung am Oszilloskop. Bezugspunkt für die Triggerung ist der 0-1 Übergang an VCC	Oszilloskop oder Logikanalysator		

4	Initiierung eines Cold-Resets nach Beendigung der Kontaktaktivierungs-Sequenz: T0 = Zeitpunkt der Aktivierung von CLK Spätestens 200 Takzyklen nach T0 befindet sich I/O im Zustand H (ICC und IFD beide im Empfangs-Modus). RST verbleibt im Zustand L bis 40000 Taktzyklen* nach T0. Anschließend wird RST auf H gesetzt (T1). Startet ICC nicht zwischen 400 und 40000 Taktzyklen nach T1 mit dem Aussenden des ATR, initiiert das IFD die Kontaktdeaktivierungs-Sequenz. *Empfehlung: mindestens 40000 Taktzyklen	EMV 1.1 Part I, 6.1.3.1 ISO 7816-3: 5.2	visuelle Beobachtung und Auszählung am Oszilloskop. Bezugspunkt für die Triggerung ist der 0-1 Übergang an VCC	Oszilloskop oder Logikanalysator		

5	Initiierung eines Warm-Resets, wenn der während eines Warm-Resets empfangene ATR nicht der Spezifikation in I -8 entspricht: T0': IFD setzt RST erneut auf L. VCC und CLK bleiben stabil. Spätestens 200 Taktzyklen nach T0' wird I/O auf Empfangs-Modus gesetzt. Daher muß sich I/O ab diesem Zeitpunkt im Zustand H befinden. RST verbleibt im Zustand L bis 40000 Taktzyklen* nach T0'. Anschließend wird RST auf H gesetzt (T1'). Startet ICC nicht zwischen 400 und 40000 Taktzyklen nach T1' mit dem Aussenden des ATR, initiiert das IFD die Deaktivierungs-Sequenz. *Empfehlung: mindestens 40000 Taktzyklen	EMV 1.1 Part I, 6.1.3.2 ISO 7816-3: 5.2	visuelle Beobachtung und Auszählung am Oszilloskop. Bezugspunkt für die Triggerung ist der 0-1 Übergang an VCC.	Oszilloskop		
6	Initiierung der Kontaktdeaktivierungs-Sequenz in der Reihenfolge: Setze RST in den Zustand L. Setze I/O und CLK in den Zustand L. Setze VCC auf V_{CC} ⇔0,4 V.	EMV 1.1 Part I, 6.1.5	visuelle Beobachtung und Auszählung am Oszilloskop.	Oszilloskop oder Logikanalysator		
7	Erkennen eines vorzeitigen Entfernens der ICC und Beendigung der Deaktivierungs-Sequenz bevor die ICC um 1 mm herausgezogen wurde bis zu einer Geschwindigkeit von 1 m/s	EMV 1.1 Part I, 6.2	Einsteckkontrolle des IFD muß innerhalb von 0,5 ms (siehe Punkt 2) die Kontaktiereinheit deaktivieren.	Oszilloskop oder Logikanalysator, Referenzkarte		

2.10 Physikalischer Transport von Zeichen

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Überprüfung erfolgt
1	etu an I/O in Abhängigkeit der Frequenz f an CLK während des Empfangs des ATR: etu = 372 / f	EMV 1.1 Part I, 7.1	Frequenzmessung an CLK	Frequenzzähler		
2	Vor der Übertragung eines Zeichens befindet sich I/O im Zustand H	EMV 1.1 Part I, 7.2	Pegelmessung	Multimeter oder Logiktester über eine Adapterkarte im Kartenleser		
3	Ein Character (Zeichen) besteht aus 10 Bits: 1 Start-Bit, 8 Daten-Bits und 1 Paritäts-Bit	EMV 1.1 Part I, 7.2	Analyse der vom IFD gesendeten Zeichen	Protokollanalysator, Terminal oder Oszilloskop Senden des ATR mit Protokollanalysator		
4	Die verwendete Abtastrate zur Detektion des Start-Bits im Polling-Betrieb beträgt ⇔0,2 etu.	EMV 1.1 Part I, 7.2	Softwareanalyse der Abtastrate bzw. Verarbeitungsdauer einer Interruptanforderung	Bereitstellung der Quellcodes und Hardwareschnittstelle der phys. Transportschicht des IFD		
5	Anzahl der logischen 1 entsprechend der eingestellten Konvention in den 8 Daten-Bits einschließlich des Paritäts-Bits muß gerade sein.	EMV 1.1 Part I, 7.2	Analyse der vom IFD gesendeten Zeichen Senden des ATR mit korrekter und falscher Parität an das IFD	Protokollanalysator, Terminal oder Oszilloskop Senden des ATR mit Protokollanalysator		

Nr.	Anforderung	Referenz	empfohlene Testmethode	empfohlenes Testwerkzeug	Verweis auf Abweichungen	Über-prüfung erfolgt
6	Start-Bit wird innerhalb von 0,7 etu erkannt. (Zeitpunkt 0 in der zeitlichen Mitte zwischen dem zuletzt beobachteten H und dem dedektierten L)	EMV 1.1 Part I, 7.2	Softwareanalyse	Bereitstellung der Quellcodes und Hardwareschnittstelle der phys. Transportschicht des IFD		
7	Nachfolgende Bits werden empfangen in den Intervallen (n + 0,5 ⇧0,2) etu. (Start-Bit ist Bit 1)	EMV 1.1 Part I, 7.2	Variieren der Baudrate des IFD bis erste Fehler auftreten	Protokollanalysator, Terminal oder Oszilloskop		
8	Von Start-Flanke des Start-Bits bis zur Ende-Flanke des n-ten Bits ist eine Übertragungszeit von (n ⇧0,2) etu einzuhalten.	EMV 1.1 Part I, 7.2	Variieren der Baudrate des IFD bis erste Fehler auftreten	Protokollanalysator, Terminal oder Oszilloskop		
9	IFD befindet sich während der Guardtime in Empfangs-Modus (I/O in Zustand H).	EMV 1.1 Part I, 7.2	Pegel-und Strommessung gegen GND	Multimeter		

Unterschiede EMV 1.1 EMV 3.0

EMV 1.1	EMV 3.0
5.4.2: -	1.4.2: Strom an C7 in beide Richtungen begrenzt auf ⇧5 mA
alle Vorgaben für I_{OL} gelten bei V_{CC}=max	alle Vorgaben für I_{OL} gelten bei V_{CC}=min
5.4.2.1: 0.8*V_{CC} ⇔V_{OH} ⇔V_{CC} bei 0 A < I_{OH} < 20 ↘A bei V_{CC}=min	1.4.2.1: 0.8*V_{CC} ⇔V_{OH} ⇔V_{CC} bei -20 ↘A < I_{OH} < 20 ↘A bei V_{CC}=min
5.4.6: Terminal generiert VCC = 5 V ⇧0,25 V	1.4.6: Terminal generiert VCC = 5 V ⇧0,4 V
5.4.6: Strom an VCC ⇔50 mA	1.4.6: Strom an VCC ⇔55 mA
	3.1: Definition von current etu

Datei-Übertragungsprotokolle

Version 2.2
22.01.1997, Ausgabedatum: 23.01.1996

Datei-Übertragungsprotokolle für die Einreichung von BZAHL-Dateien bei den HEZen

Die Händler-Evidenzzentralen des Zahlungssystems GeldKarte unterstützen mindestens die folgenden Datei-Übertragungsprotokolle zur Einreichung von BZAHL-Dateien durch Händlersysteme:

1. FTAM, Transportprotokoll X.25

   FTAM mit Transportprotokoll X.25 soll zur Übertragung der BZAHL-Dateien von Sammel-Einreichungsterminals (Konzentrator-Terminals) an die HEZen verwendet werden.

   Die Dateinamenskonvention für die Benennung der BZAHL-Datei bei der Übertragung von Einreicherterminals an die HEZ ist beigefügt.

   FTAM wird vom Beginn des Feldversuchs in Ravensburg/Weingarten an durch die HEZen unterstützt.

2. ISO-Dateitransfer

   ISO-Dateitransfer soll zur direkten Übertragung der BZAHL-Dateien von Akzeptanzterminals an die HEZen verwendet werden. Dieses Datei-Übertragungsprotokoll ist in erster Linie für solche Akzeptanzterminals vorgesehen, die auch electronic cash-Terminals sind.

   Eine Beschreibung des Datei-Übertragungsprotokolls ISO-Dateitransfer ist beigefügt.

   ISO-Dateitransfer wird vom Beginn des Feldversuchs in Ravensburg/Weingarten an durch die HEZen unterstützt.

3. Z-Modem

   Z-Modem soll ebenfalls zur direkten Übertragung der BZAHL-Dateien von Akzeptanzterminals an die HEZen verwendet werden. Dieses Datei-Übertragungsprotokoll ist in erster Linie für reine Akzeptanzterminals des Zahlungssystems GeldKarte vorgesehen.

   Eine Beschreibung des Datei-Übertragungsprotokolls Z-Modem sowie Beispielprogramme sind beigefügt.

   Z-Modem wird ab 2. Quartal 1996 durch die HEZen unterstützt.

Der Einsatz weiterer Datei-Übertragungsprotokolle ist aufgrund bilateraler Absprachen zwischen Einreicher und zuständiger HEZ möglich.

Dateinamenskonvention für den Dateitransfer mittels FTAM

23.01.1996

Dateinamenskonvention für die Kommunikation zwischen Einreichungsterminal und HEZ

Für die Kommunikation zwischen Einreichungsterminal und HEZ wurden die folgenden Namenskonventionen festgelegt:

Tdhhmmss.kBZ, wobei

T = konstant

d = letzte Stelle des aktuellen Tages bei Dateierstellung im Maschinenformat

hh = Stunden der aktuellen Uhrzeit bei Dateierstellung

mm = Minuten der aktuellen Uhrzeit bei Dateierstellung

ss = Sekunden der aktuellen Uhrzeit bei Dateierstellung

k = Kennzeichen für Komprimierung

- F FLAM-komprimiert
- X nicht komprimiert

BZ = Kennzeichen BZAHL-Datei bzw. Quittungsdatei

Funktionsbeschreibung ISO-Dateitransfer für Händler-Einreichungsdateien

Version 1.0
07.12.1995

Inhalt

1. Einleitung

Dieses Dokument beschreibt den Dateitransfer einer Händler-Einreichungsdatei, im folgenden BZAHL-Datei genannt, von einem Einreichungsterminal (ET) zu einer Händler-Evidenzzentrale (HEZ) unter Verwendung von Nachrichten, die gemäß ISO 8583 formatiert sind. Der Aufbau der zu übertragenden BZAHL-Datei ist in Kapitel 2.2.3. von [1] festgelegt.

Ein Dateitransfer gemäß dieser Spezifikation besteht aus

- einer Initialisierungsphase (Beginndialog mit Anfrage und Antwort),
- dem eigentlichen Transfer der BZAHL-Datei (Nutzdaten) und
- einer Abschlußphase (Schlußdialog mit Anfrage und Antwort).

Diese drei Phasen müssen grundsätzlich eingehalten werden.

2. DFÜ-Schnittstelle

2.1. Verbindungsauf- und -abbau

Die Übertragung einer BZAHL-Datei wird grundsätzlich vom Einreichungsterminal angestoßen und benötigt ein transparentes 8 Bit Übertragungsmedium. Mit der eigentlichen Datenübermittlung muß gewartet werden, bis eine Bestätigung des Verbindungsaufbaus durch die DFÜ-Komponente eintrifft.

Für den Dateitransfer von BZAHL-Dateien wird ein einheitlicher Zeitraum bis zum Timeout (Timeout-Zeitraum) festgelegt. Der Timeout-Zeitraum beträgt zur Zeit 10 Sekunden, muß aber durch alle Komponenten variabel gehandhabt werden können.

Der Verbindungsabbau durch das ET erfolgt,

- wenn innerhalb des Timeout-Zeitraums nach Absetzen der Beginn- oder Schlußanfrage keine Antwort der HEZ eintrifft oder
- wenn das ET eine fehlerhafte Beginnantwort oder eine Beginnantwort mit einem von '00' verschiedenen Antwortcode erhält oder
- wenn das ET eine fehlerhafte Datenablehnung oder eine Datenablehnung mit einem von '06' verschiedenen Antwortcode erhält oder
- wenn die Schlußantwort eingetroffen ist.

Im Regelfall erfolgt der Abbau der Verbindung durch das ET. Die HEZ kann die Verbindung abbauen,

- wenn innerhalb des Timeout-Zeitraums nach Verbindungsaufbau keine Beginnanfrage eintrifft oder
- wenn sie eine Beginnantwort mit negativem Antwortcode, eine Datenablehnung mit einem von '06' verschiedenen Antwortcode oder die Schlußantwort abgesetzt hat oder
- wenn innerhalb des Timeout-Zeitraums nach Absetzen der Beginnantwort keine Datentransfer-Nachricht eintrifft oder
- wenn innerhalb des Timeout-Zeitraums nach Eintreffen einer Datentransfer-Nachricht weder eine weitere Datentransfer-Nachricht noch eine Schlußanfrage eintrifft.

2.2. Nachrichtenarten

Die verschiedenen für die Dateiübertragung verwendeten Nachrichtenarten werden anhand der Einzelfelder "Nachrichtentyp" und des 1. Byte des "Abwicklungskennzeichen" (BMP 3) der ISO-Nachrichten identifiziert.

Im einzelnen sind definiert:

Nachr.-Typ	1. Byte, Abw. kz.	Name der Nachricht
9100	80	Datentransfer-Nachricht
9110	80	Datenablehnung
9100	81	Beginnanfrage Dateitransfer
9110	81	Beginnantwort Dateitransfer
9100	82	Schlußanfrage Dateitransfer
9110	82	Schlußantwort Dateitransfer

2.3. ISO-Tabelle

Das Format der für die Dateiübertragung verwendeten Nachrichten basiert auf der ISO-Norm 8583. Im Feld "Bitmap" ist Bit Nr. i genau dann gesetzt, wenn Feld BMP i in der Nachricht vorkommt; dabei sind die Bits von links nach rechts mit 1..64 numeriert.

BMP	Bezeichnung	Format	Länge in Byte	9100	9110	Inhalt
-	Nachrichtentyp	NP	2	x	x	
-	Bitmap	B	8	x	x	
3	Abwicklungskennzeichen	NP	3	x	x	
11	Trace-Nummer	NP	3	a	=	
12	Lokalzeit	NP	3	b	b	HHMMSS
13	Lokaldatum	NP	2	b	b	MMTT
39	Antwortcode	NP	1		x	
41	Terminal-ID	NP	4	x	=	
60	Längenfeld spez. Dialogdaten	CL3 B	3 ..999	x x		
-	CRC	B	2	x	x	

Legende

a	nur im Beginn- und Schlußdialog vorhanden, in allen vier Nachrichten identisch
b	nur im Beginn- und Schlußdialog vorhanden
=	Wert aus der zugehörigen Anfrage
x	Pflichtfeld
NP	BCD, rechtsbündig mit führenden Nullen
B	binär
CL3	3 EBCDIC-Ziffern ohne Vorzeichen
..999	maximal 999, genauer Wert in LLL festgelegt

Die Gesamtlänge einer Datentransfer-Nachricht (inkl. CRC) kann gemäß dieser Definition 1021 Byte nicht überschreiten. Die maximale Größe einer übertragbaren BZAHL-Datei wird durch die Längenangabe LLL und die Blocknumerierung (BMP 3) auf 999 x 9999 Byte begrenzt. Eine Überlauf der Blocknumerierung (Wert '0000') ist zur Zeit nicht definiert.

2.4. Inhalt der Datenfelder

Im folgenden wird Inhalt und Aufbau der Datenfelder der Nachrichten vom Typ 9100 und 9110 beschrieben. Die Verwendung der Datenfelder

- Abwicklungskennzeichen,
- Antwortcode und
- Spezielle Dialogdaten

wird im Kapitel 2.5. näher erläutert.

2.4.1. BMP 3: Abwicklungskennzeichen

Das erste Byte dieses Feldes dient zusammen mit dem Feld "Nachrichtentyp" zur weiteren Unterscheidung der einzelnen Nachrichtenarten. Byte 2 und 3 haben je nach Nachrichtenart eine unterschiedliche Funktion.

‘811nnn’	Beginnanfrage/-antwort
nnn	Maximale Größe der Nutzdaten in BMP 60 für den Datentransfer, BCD
‘80nnnn’	Datentransfer/Datenablehnung
nnnn	Blocknummer, BCD, beginnend mit 0001
‘82nnnn’	Schlußanfrage/-antwort
nnnn	Blocknummer des letzten Blockes, BCD

2.4.2. BMP 11: Trace-Nummer

Das Führen einer Trace-Nummer in einem ET sowie die Überprüfung der Trace-Nummer in einer HEZ ist optional. Um das Zusammenspiel von ETs mit/ohne Führen einer Trace-Nummer und HEZen mit/ohne Prüfung einer Trace-Nummer zu ermöglichen, wird wie folgt verfahren:

Die Trace-Nummer muß in den Nachrichten des Beginn- und Schlußdialogs enthalten sein. Hierbei muß in allen vier Nachrichten dieselbe Trace-Nummer verwendet werden.

Wenn ein ET eine Trace-Nummer führt, wird diese vom ET, beginnend mit '000001', unter fortlaufendem Inkrementieren generiert; nach '999999' folgt wieder '000001'. Die gegenüber der letzten Transaktion des Terminals inkrementierte Trace-Nummer wird in die BMP 11 der Beginnanfrage eingestellt und in die Schlußanfrage übernommen.

Wenn ein ET keine Trace-Nummer führt, stellt es den Wert '000000' in die BMP 11 von Beginn- und Schlußanfrage ein.

Wenn eine HEZ die Trace-Nummer prüft, prüft sie, ob die Trace-Nummer in der Beginnanfrage den Wert '000000' hat oder gegenüber der letzten Transaktion inkrementiert ist, wobei sie um mehr als 1 inkrementiert sein darf. Die HEZ prüft weiterhin, ob die Trace-Nummer in Beginnanfrage und Schlußanfrage identisch ist und übernimmt sie in die jeweilige Antwort.

2.4.3. BMP 12: Lokalzeit der Transaktion

Die bei Aufbau der jeweiligen Nachricht aktuelle Lokalzeit X'HHMMSS' wird

- vom ET in die Beginn- und Schlußanfrage und

- vom HEZ in die Beginn- und Schlußantwort

eingestellt. Hierbei wird folgendes Format verwendet:

HH	Stundenangabe	2-stellig
MM	Minutenangabe	2-stellig
SS	Sekundenangabe	2-stellig

Wenn das Einreichungsterminal über keine Uhr verfügt, wird dieses Feld mit dem Wert X'000000' gefüllt. Die HEZ prüft die Lokalzeit des ET nur dann auf Plausibilität, wenn das Lokaldatum in BMP 13 verschieden von 0 ist.

2.4.4. BMP 13: Lokaldatum der Transaktion

Das bei Aufbau der jeweiligen Nachricht aktuelle Lokaldatum X'MMTT' wird

- vom ET in die Beginn- und Schlußanfrage und
- vom HEZ in die Beginn- und Schlußantwort

eingestellt. Hierbei wird folgendes Format verwendet:

MM	Monatsangabe	2-stellig
TT	Tagesangabe	2-stellig

Wenn das Einreichungsterminal über keine Uhr verfügt, wird dieses Feld mit dem Wert X'000000' gefüllt. Die HEZ prüft das Lokaldatum des ET nur dann auf Plausibilität, wenn es verschieden von 0 ist.

2.4.5. BMP 39: Antwortcode

Das Datenfeld enthält den Antwortcode der HEZ. Die folgenden Antwortcodes werden bei der Übertragung von BZAHL-Dateien mit der jeweiligen Bedeutung verwendet:

Code	Bedeutung
00	Funktion fehlerfrei durchgeführt
06	Fehler, optional Wiederaufsetzen des Datentransfers nur in Datenablehnung
29	Dateitransfer nicht erfolgreich nur in Datenablehnung und Schlußantwort
30	Formatfehler
89	CRC falsch
98	Datum/Uhrzeit nicht plausibel, nur wenn Datum in BMP 13 ≠ 0, nur in Beginn- oder Schlußantwort

2.4.6. BMP 41: Terminal-Identifikationsnummer

Die Terminal-ID muß in allen Nachrichten der Dateiübertragung enthalten sein. Hierbei muß in allen Nachrichten dieselbe Terminal-ID verwendet werden.

Im Rahmen des Zahlungssystems GeldKarte wird einem ET keine Terminal-ID zugeordnet.

Ein Terminal, das nicht nur ET im Zahlungssystem GeldKarte ist, sondern auch als Akzeptanzterminal für ein anderes Zahlungssystem des Kreditgewerbes (electronic cash, POZ) zugelassen ist und in diesem Zusammenhang eine Terminal-ID besitzt, kann diese Terminal-ID in die BMP 41 der Anfragenachrichten einstellen. Andernfalls stellt das ET in die BMP 41 der Nachrichten den Wert '000000' ein.

Die HEZ prüft bei der Übertragung von BZAHL-Dateien nur, ob die Anfragenachrichten des Dateitransfers dieselbe Terminal-ID enthalten. Die HEZ übernimmt die Terminal-ID aus der Anfrage in die jeweilige Antwort.

2.4.7. BMP 60: Spezielle Dialogdaten

Beginnanfrage

Das Feld wird immer mit der Konstanten 'F0F0F10A' belegt.

Datentransfer-Nachricht

Das Feld enthält 3 Byte Längenangabe (LLL) und nnn Byte der BZAHL-Datei (Nutzdaten), wobei LLL='FnFnFn' mit $0 \leq n \leq 9$ ist.

Schlußanfrage

Das Feld wird immer mit der Konstanten ’F0F0F0’ belegt.

2.5. Ablauf des Dateitransfers

Der Dateitransfer dient dazu, eine BZAHL-Datei vom Einreichungsterminal (ET) zur Händler-Evidenzzentrale (HEZ) zu übertragen. Anhand des Nachrichtentyps und des Abwicklungskennzeichens in BMP 3 der jeweiligen ISO-Nachricht wird zwischen Beginnanfrage/-antwort, dem eigentlichen Datentransfer und der Schlußanfrage/-antwort unterschieden.

2.5.1. Beginndialog

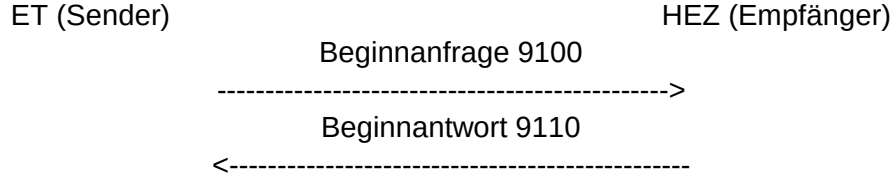
2.5.1.1. Nachrichtenaufbau des Beginndialogs

Der Beginndialog wird mittels der Nachrichten Beginnanfrage (Nachrichtentyp 9100) und Beginnantwort (Nachrichtentyp 9110) durchgeführt. Diese Nachrichten haben den folgenden Aufbau:

BMP	Bezeichnung	Format	Länge in Byte	9100	9110	Inhalt
-	Nachrichtentyp	NP	2	x	x	
-	Bitmap	B	8	x	x	
3	Abwicklungskennzeichen	NP	3	x	x	’811nnn’
11	Trace-Nummer	NP	3	x	=	
12	Lokalzeit	NP	3	x	x	HHMMSS
13	Lokaldatum	NP	2	x	x	MMTT
39	Antwortcode	NP	1		x	
41	Terminal-ID	NP	4	x	=	
60	Längenfeld spez. Dialogdaten	CL3 B	3 1	x x		'F0F0F1' '0A'
-	CRC	B	2	x	x	

2.5.1.2. Ablauf des Beginndialogs

Der Beginndialog dient dazu, zwischen ET und HEZ die maximale Größe der Nutzdaten im Feld BMP 60 für den nachfolgenden Datentransfer festzulegen.



Festlegen der maximalen Nutzdatengröße

Das Abwicklungskennzeichen in Beginnanfrage und -antwort besteht immer aus den 3 Byte '811nnn', wobei die 3 BCD-Ziffern nnn die maximale Größe der Nutzdaten in BMP 60 für den Datentransfer angeben, so daß Größenangaben zwischen 001 und 999 Byte gemacht werden können.

In nnn des Abwicklungskennzeichens der Beginnanfrage teilt das ET mit, wieviel Byte Nutzdaten es

in BMP 60 maximal übertragen kann. Die HEZ teilt die von ihr maximal unterstützte Größe der Nutzdaten in nnn des Abwicklungskennzeichens der Beginnantwort mit.

Eine HEZ unterstützt mindestens eine Nutzdatengröße von 236 Byte.

Das Minimum der Größenangaben von ET und HEZ in den Abwicklungskennzeichen des Beginndialogs wird als Maximalgröße für die Nutzdaten festgelegt.

Beispiel:

Wenn das ET eine maximale Nutzdatengröße von 260 Byte angibt und die HEZ mit einer maximalen Größe von 999 Byte antwortet, darf das ET maximal 260 Byte Nutzdaten in die BMP 60 einstellen.

Wenn das ET eine maximale Nutzdatengröße von 999 Byte angibt und die HEZ mit einer maximalen Größe von 260 Byte antwortet, darf das ET ebenfalls maximal 260 Byte Nutzdaten in die BMP 60 einstellen.

Prüfungen und Antwortcodes

Die HEZ prüft eine eintreffende Beginnanfrage und verwendet in der Beginnantwort die Antwortcodes 00, 30, 89, 98 mit der folgenden Bedeutung:

00: Die Beginnanfrage ist korrekt.

30: Aufbau, Format und/oder Inhalt der Beginnanfrage entsprechen nicht der Spezifikation.

89: Der CRC der Beginnanfrage ist falsch.

98: Datum und Uhrzeit ≠ 0 und nicht plausibel.

Wenn das ET eine korrekte Beginnantwort mit Antwortcode 00 erhält, kann es mit dem Datentransfer fortfahren.

Wenn das ET eine korrekte Beginnantwort mit Antwortcode ≠ 00 erhält, protokolliert es den Antwortcode und bricht die Dateiübertragung ab.

Wenn das ET eine fehlerhafte Beginnantwort oder innerhalb des Timeout-Zeitraums keine Beginnantwort erhält, bricht es die Dateiübertragung ebenfalls ab.

2.5.2. Datentransfer

2.5.2.1. Nachrichtenaufbau des Datentransfers

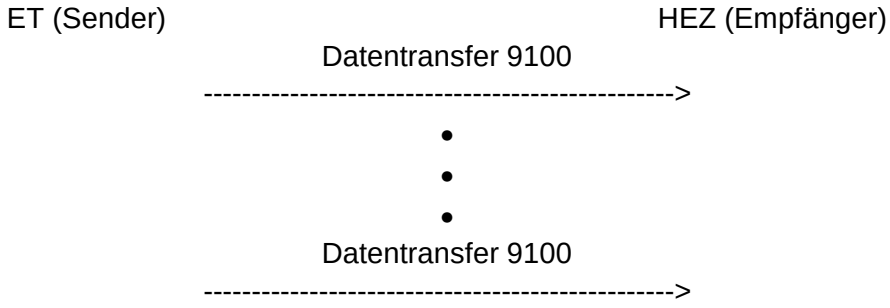
Die Übertragung der BZAHL-Datei wird mittels Datentransfer-Nachrichten (Nachrichtentyp 9100) durchgeführt. Die HEZ kann in einer Datenablehnung (Nachrichtentyp 9110) festgestellte Fehler

mitteilen. Die Nachrichten haben den folgenden Aufbau:

BMP	Bezeichnung	Format	Länge in Byte	9100	9110	Inhalt
-	Nachrichtentyp	NP	2	x	x	
-	Bitmap	B	8	x	x	
3	Abwicklungskennzeichen	NP	3	x	x	’80nnnn’
39	Antwortcode	NP	1		x	
41	Terminal-ID	NP	4	x	=	
60	Längenfeld spez. Dialogdaten	CL3 B	3 ..999	x x		LLL
-	CRC	B	2	x	x	

2.5.2.2. Ablauf des Datentransfers

Bei der Übertragung einer BZAHL-Datei nach dem Beginndialog ist das ET der Sender und die HEZ der Empfänger. Das ET schickt mittels Datentransfer-Nachrichten (Typ 9100) die durchnumerierten Datenblöcke der BZAHL-Datei als Nutzdaten in BMP 60, wobei die maximale Nutzdatenlänge im Beginndialog vereinbart wurde.



Zerlegung einer BZAHL-Datei

Die tatsächliche Länge der Datenblöcke, in die die BZAHL-Datei zerlegt wird, wird dem jeweiligen Datenblock immer in den ersten drei Byte der BMP 60 vorangestellt.

Beispiel:

Wurde im Beginndialog eine maximale Nutzdatenlänge von 236 Byte ausgehandelt, kann eine 320 Byte lange BZAHL-Datei in zwei Datentransfer-Nachrichten übertragen werden, deren BMP 60 folgenden Inhalt hat:

Nachricht 1: Länge X'F2F3F6' gefolgt von den ersten 236 Byte der BZAHL-Datei

Nachricht 2: Länge X'F0F8F4' gefolgt von den übrigen 84 Byte der BZAHL-Datei

Wenn das Terminal so ausgelegt ist, daß es nur einen oder mehrere komplette 80 Byte lange Datensätze der BZAHL-Datei als Nutzdaten in die Datentransfer-Nachrichten einstellt, kann es die 320 Byte lange BZAHL auch folgendermaßen übertragen:

Nachricht 1: Länge X'F1F6F0' gefolgt von den ersten beiden Datensätzen der BZAHL-Datei

Nachricht 2: Länge X'F1F6F0' gefolgt von den übrigen beiden Datensätzen der BZAHL-Datei

Jede weitere Zerlegung der BZAHL-Datei in Datenblöcke, die der Bedingung genügt, daß der einzelne Datenblock nicht länger als 236 Byte ist, wird bei der Übertragung in Datentransfer-Nachrichten durch die HEZ akzeptiert.

Blocknumerierung

Innerhalb des Datentransfers einer BZAHL-Datei wird eine fortlaufende, lückenlose Blocknummer, beginnend mit 0001, in Byte 2 und 3 des Abwicklungskennzeichens der Datentransfer-Nachrichten durch das ET eingestellt. Hierbei müssen die Datenblöcke der BZAHL-Datei gemäß ihrer Reihenfolge in der BZAHL-Datei numeriert werden.

Beispiel:

Wenn eine BZAHL-Datei von 3200 Byte Länge übertragen werden soll und im Beginndialog eine maximale Nutzdatenlänge von 999 Byte vereinbart wurde, kann die BZAHL-Datei, zerlegt in 4 Datenblöcke, übertragen werden. Hierzu werden 4 Datentransfer-Nachrichten verwendet, die gemäß ihrer Position in der BZAHL-Datei die fortlaufenden Blocknummern 0001 bis 0004 im Abwicklungskennzeichen enthalten.

Prüfungen und Datenablehnung

Die HEZ kann eine Datenablehnung (Typ 9110) an das ET schicken, wenn sie einen Fehler in der Übertragung feststellt.

In Byte 2 und 3 des Abwicklungskennzeichens der Datenablehnung stellt die HEZ die Nummer des ersten von ihr als fehlerhaft erkannten Datenblocks ein.

In der Datenablehnung werden die Antwortcodes 06, 29, 30 und 89 mit der folgenden Bedeutung verwendet:

29: Die Datentransfer-Nachrichten sind nicht in der Reihenfolge der Blocknummern eingetroffen. In Byte 2 und 3 des Abwicklungskennzeichens wird die Blocknummer des ersten fehlenden Blockes eingestellt, d. h. die Blocknummer, die auf die letzte in korrekter Reihenfolge empfangene Blocknummer folgte.

Beispiel:

Wenn die Datenblöcke mit den Nummern 0001 bis 0004 und anschließend mit der Nummer

0006 eintreffen, wird in das Abwicklungskennzeichen der Datenablehnung durch die HEZ die Nummer 0005 eingestellt.

30: Aufbau, Form und/oder Inhalt der Datentransfer-Nachricht, deren Blocknummer in Byte 2 und 3 des Abwicklungskennzeichen der Datenablehnung enthalten ist, entsprechen nicht der Spezifikation.

89: Der CRC der Datentransfer-Nachricht, deren Blocknummer in Byte 2 und 3 des Abwicklungskennzeichens der Datenablehnung enthalten ist, ist falsch.

Durch diese Antwortcodes in der Datenablehnung zeigt die HEZ an, daß das ET die laufende Dateiübertragung ohne Schlußdialog abbrechen soll. Die BZAHL-Datei muß dann innerhalb eines neuen Dateitransfers übertragen werden.

Eine besondere Bedeutung hat der Antwortcode 06:

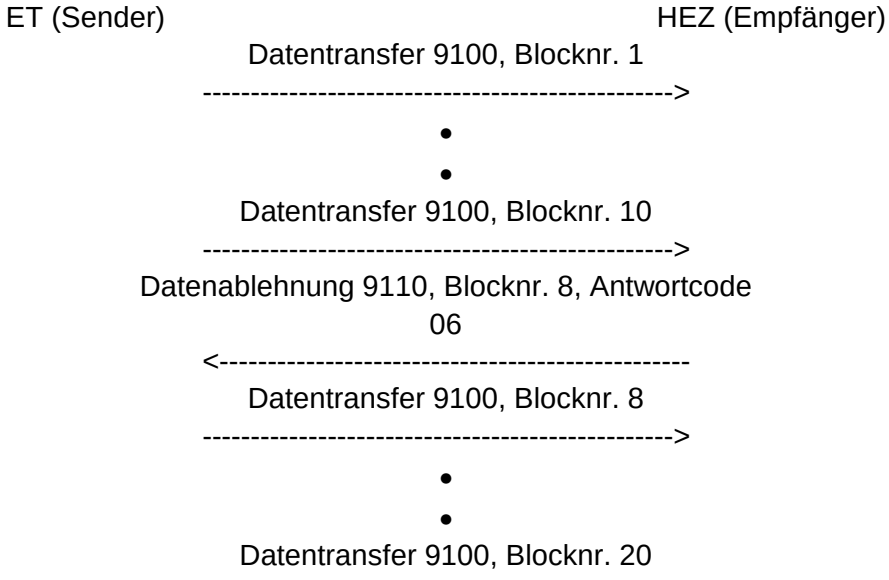
Durch den Antwortcode '06' kann die HEZ in der Datenablehnung mitteilen, daß das ET den Dateitransfer ab dem ersten fehlerhaften Datenblock, dessen Datenblocknummer in der Datenablehnung enthalten ist, wiederholen kann.

Bei Antwortcode 06 kann das ET entscheiden, ob es die Datenübertragung ab der abgelehnten Nachricht (identifiziert durch die Blocknummer) oder den gesamten Dateitransfer wiederholt. Zur Wiederholung des gesamten Dateitransfers muß das ET in diesem Fall zunächst die laufende Datenübertragung mittels des Schlußdialogs beenden.

Wenn das ET in diesem Fall die Dateiübertragung mit dem Schlußdialog beendet, verwendet die HEZ den negativen Antwortcode 29 in der Schlußantwort, auch wenn die Schlußantwort selbst korrekt war.

Beispiel:

Eine in 20 Datenblöcke zerlegte BZAHL-Datei soll übertragen werden. Nach der Übertragung des 10. Blocks erhält das ET eine Datenablehnung mit Antwortcode 06 und Angabe des 8. Blocks zum Wiederaufsetzen.



-->

2.5.3. Schlußdialog

2.5.3.1. Nachrichtenaufbau des Schlußdialogs

Der Schlußdialog wird mittels der Nachrichten Schlußanfrage (Nachrichtentyp 9100) und Schlußantwort (Nachrichtentyp 9110) durchgeführt. Diese Nachrichten haben den folgenden Aufbau:

BMP	Bezeichnung	Format	Länge in Byte	9100	9110	Inhalt
-	Nachrichtentyp	NP	2	x	x	
-	Bitmap	B	8	x	x	
3	Abwicklungskennzeichen	NP	3	x	x	’82nnnn’
11	Trace-Nummer	NP	3	x	=	
12	Lokalzeit	NP	3	x	x	HHMMSS
13	Lokaldatum	NP	2	x	x	MMTT
39	Antwortcode	NP	1		x	
41	Terminal-ID	NP	4	x	=	
60	Längenfeld	CL3	3	x		'F0F0F0'
-	CRC	B	2	x	x	

2.5.3.2. Ablauf des Schlußdialogs

Mittels der Schlußanfrage teilt das ET der HEZ mit, daß der Datentransfer beendet ist. Durch die Schlußantwort informiert die HEZ das ET über Erfolg oder Mißerfolg der Dateiübertragung.

ET (Sender) HEZ (Empfänger)

Schlußanfrage 9100

-->

Schlußantwort 9110

<--

Verwendung der Blocknummer

In Byte 2 und 3 des Abwicklungskennzeichen der Schlußanfrage stellt das ET die Blocknummer des letzten an die HEZ übertragenen Datenblocks (Anzahl der übertragenen Datenblöcke). Dies ist auch dann der Fall, wenn das ET den Schlußdialog aufgrund einer Datenablehnung mit Antwortcode 06 durchführt.

Die HEZ sendet daraufhin eine Schlußantwort, in deren Abwicklungskennzeichen die Blocknummer

aus dem Abwicklungskennzeichens der Schlußanfrage übernommen wird.

Prüfungen und Antwortcodes

Durch den Antwortcode in der Schlußantwort teilt die HEZ dem ET den Ausgang der Datenübertragung mit.

In der Schlußantwort werden die Antwortcodes 00, 29, 30, 89 und 98 mit der folgenden Bedeutung verwendet:

00: Mit Antwortcode 00 quittiert die HEZ, daß die Anzahl der von ihr empfangenen Datenblöcke mit der Angabe in der Schlußanfrage übereinstimmt. Alle durch das ET gesendeten Nachrichten waren korrekt.

29: Die HEZ verwendet den Antwortcode 29, wenn die Blocknummer in der sonst korrekten Schlußanfrage nicht der Zahl der von ihr empfangenen Datenblöcke entspricht.

Die HEZ muß diesen Antwortcode auch verwenden, wenn sie in einer der Datentransfer-Nachrichten einen Fehler festgestellt hat. Insbesondere ist dieser Antwortcode immer dann in die Schlußantwort einzustellen, wenn die HEZ bei dem vorhergehenden Datentransfer eine Datenablehnung geschickt hat.

30: Aufbau, Format und/oder Inhalt der Schlußanfrage entsprechen nicht der Spezifikation.

89: Der CRC der Schlußanfrage ist falsch.

98: Datum und Uhrzeit ≠ 0 und nicht plausibel.

Wenn das ET eine korrekte Schlußantwort mit Antwortcode 00 erhält, ist die Dateiübertragung erfolgreich beendet.

Wenn das ET eine korrekte Schlußantwort mit Antwortcode ≠ 00 erhält, protokolliert es den Antwortcode.

Wenn das ET eine fehlerhafte Schlußantwort oder innerhalb des Timeout-Zeitraums keine Schlußantwort erhält, bricht es die Dateiübertragung ebenfalls ab.

Ein negativer Antwortcode in der Schlußantwort, eine fehlerhafte Schlußantwort oder das Ausbleiben der Schlußantwort bedeuten für das ET immer, daß die Dateiübertragung nicht erfolgreich war. Es muß die BZAHL-Datei erneut übertragen.

3. Datenformat-Beschreibung

B	binär
CLx	x EBCDIC-Zeichen, Ziffern ohne Vorzeichen
NP	numerisch gepackt (BCD).
X	Hexadezimal-Ziffern (nachfolgende Konstanten in Apostrophs)
..	Der Wert der Längenangabe kann bis zu dem angegebenen Wert variieren.

4. Abkürzungen

BMP	Bit map position (ISO); gibt durch eine binäre "1" zu erkennen, daß das betreffende, durch ISO definierte Feld in der Nachricht enthalten ist. Wird synonym für das jeweilige Feld selbst benutzt.
CCITT	Comite Consultatif International Telephonique et Telegraphique; Internationaler beratender Ausschuß für den Telegraphen- und Fernsprechdienst
CRC	Prüfsumme für eine Nachricht, basierend auf der CCITT V.41-Norm mit dem Polynom x^16 + x^12 + x^5 + 1 und Initialwert 0000000000000000
EBCDIC	Extended Binary Coded Decimal Interchange Code; von IBM definiertes binäres Codierungsverfahren für Ziffern, Buchstaben, Sonderzeichen und Steuerzeichen.
ET	Einreichungsterminal
EZ	Evidenzzentrale
HEZ	Händler-Evidenzzentrale
ISO	International Standardization Organization; Internationale Normierungsbehörde.
ZKA	Zentraler Kreditausschuß

5. Referenzen

[1] ZKA, Schnittstellenspezifikation für die ec-Karte mit Chip, Ladeterminals, Händlersysteme, Bankensonderfunktionsterminals und Taschenkartenleser der elektronischen Geldbörse, Version 2.1.1, 09.08.1995, letztes Änderungsdatum 27.10.1995

Funktionsbeschreibung ZMODEM-Übertragungsprotokoll Version 8-3-87

23.11.1995

Inhalt

1. Einleitung

Die folgende Beschreibung des ZMODEM-Übertragungsprotokolls ist in [3] enthalten. Die Beispielprogramme in C sind der Datei "RZSZ0589.ARC" entnommen, die in vielen Bulletin Boards in INTERNET oder Compuserve zu finden ist.

ZMODEM ist Public Domain:

"ZMODEM was developed for the public domain under a Telenet contract. The ZMODEM protocol descriptions and the Unix rz/sz program source code are public domain. No licensing, trademark, or copyright restrictions apply to the use of the protocol, the Unix rz/sz source code and the ZMODEM name." (Kapitel 22. in [3])

Fragen können gerichtet werden an

Chuck Forsberg
Omen Technology Inc
17505-V Sauvie Island Road
Portland Oregon 97231
VOICE: 503-621-3406 :VOICE
Modem (TeleGodzilla): 503-621-3746
Usenet: ...!tektronix!reed!omen!caf
Compuserve: 70007,2304
Source: TCE022

2. Funktionsbeschreibung

2.1. ROSETTA STONE

Here are some definitions which reflect current vernacular in the computer media. The attempt here is identify the file transfer protocol rather than specific programs.

FRAME — A ZMODEM frame consists of a header and 0 or more data subpackets.

XMODEM — refers to the original 1977 file transfer etiquette introduced by Ward Christensen's MODEM2 program. It's also called the MODEM or MODEM2 protocol. Some who are unaware of MODEM7's unusual batch file mode call it MODEM7. Other aliases include "CP/M Users's Group" and "TERM II

FTP 3". This protocol is supported by most communications programs because it is easy to implement.

XMODEM/CRC replaces XMODEM's 1 byte checksum with a two byte Cyclical Redundancy Check (CRC-16), improving error detection.

XMODEM-1k Refers to XMODEM-CRC with optional 1024 byte blocks.

YMODEM refers to the XMODEM/CRC protocol with batch transmission and optional 1024 byte blocks as described in YMODEM.DOC.[1(1)]

2.2. ZMODEM REQUIREMENTS

ZMODEM requires an 8 bit transfer medium.[2(2)] ZMODEM escapes network control characters to allow operation with packet switched networks. In general, ZMODEM operates over any path that supports XMODEM, and over many that don't.

To support full streaming[3(3)],the transmission path should either assert flow control or pass full speed transmission without loss of data. Otherwise the ZMODEM sender must manage the window size.

2.2.1. File Contents

2.2.1.1. Binary Files

ZMODEM places no constraints on the information content of binary files, except that the number of bits in the file must be a multiple of 8.

2.2.1.2. Text Files

Since ZMODEM is used to transfer files between different types of computer systems, text files must meet minimum requirements if they are to be readable on a wide variety of systems and environments.

Text lines consist of printing ASCII characters, spaces, tabs, and backspaces.

ASCII End of Line

The ASCII code definition allows text lines terminated by a CR/LF (015, 012) sequence, or by a NL (012) character. Lines logically terminated by a lone CR (013) are not ASCII text.

A CR (013) without a linefeed implies overprinting, and is not acceptable as a logical line separator. Overprinted lines should print all important characters in the last pass to allow CRT displays to display meaningful text. Overstruck characters may be generated by backspacing or by overprinting

the line with CR (015) not followed by LF.

Overstruck characters generated with backspaces should be sent with the most important character last to accommodate CRT displays that cannot overstrike. The sending program may use the ZCNL bit to force the receiving program to convert the received end of line to its local end of line convention.[4]
(4)

2.3. ZMODEM BASICS

2.3.1. Packetization

ZMODEM frames differ somewhat from XMODEM blocks. XMODEM blocks are not used for the following reasons:

- Block numbers are limited to 256
- No provision for variable length blocks
- Line hits corrupt protocol signals, causing failed file transfers. In particular, modem errors sometimes generate false block numbers, false EOTs and false ACKs. False ACKs are the most troublesome as they cause the sender to lose synchronization with the receiver.

  State of the art programs such as Professional-YAM and ZCOMM overcome some of these weaknesses with clever proprietary code, but a stronger protocol is desired.
- It is difficult to determine the beginning and ends of XMODEM blocks when line hits cause a loss of synchronization. This precludes rapid error recovery.

2.3.2. Link Escape Encoding

ZMODEM achieves data transparency by extending the 8 bit character set (256 codes) with escape sequences based on the ZMODEM data link escape character ZDLE.[5](5)

Link Escape coding permits variable length data subpackets without the overhead of a separate byte count. It allows the beginning of frames to be detected without special timing techniques, facilitating rapid error recovery.

Link Escape coding does add some overhead. The worst case, a file consisting entirely of escaped characters, would incur a 50% overhead.

The ZDLE character is special. ZDLE represents a control sequence of some sort. If a ZDLE character appears in binary data, it is prefixed with ZDLE, then sent as ZDLEE.

The value for ZDLE is octal 030 (ASCII CAN). This particular value was chosen to allow a string of 5 consecutive CAN characters to abort a ZMODEM session, compatible with YMODEM session abort.

Since CAN is not used in normal terminal operations, interactive applications and communications programs can monitor the data flow for ZDLE. The following characters can be scanned to detect the ZRQINIT header, the invitation to automatically download commands or files.

Receipt of five successive CAN characters will abort a ZMODEM session. Eight CAN characters are sent.

The receiving program decodes any sequence of ZDLE followed by a byte with bit 6 set and bit 5 reset (upper case letter, either parity) to the equivalent control character by inverting bit 6. This allows the transmitter to escape any control character that cannot be sent by the communications medium. In addition, the receiver recognizes escapes for 0177 and 0377 should these characters need to be escaped.

ZMODEM software escapes ZDLE, 020, 0220, 021, 0221, 023, and 0223. If preceded by 0100 or 0300 (@), 015 and 0215 are also escaped to protect the Telenet command escape CR-@-CR. The receiver ignores 021, 0221, 023, and 0223 characters in the data stream.

The ZMODEM routines in zm.c accept an option to escape all control characters, to allow operation with less transparent networks. This option can be given to either the sending or receiving program.

2.3.3. Header

All ZMODEM frames begin with a header which may be sent in binary or HEX form. ZMODEM uses a single routine to recognize binary and hex headers. Either form of the header contains the same raw information:

- A type byte[6(6)]
- Four bytes of data indicating flags and/or numeric quantities depending on the frame type

Figure 1. Order of Bytes in Header

TYPE: frame type
F0: Flags least significant byte
P0: file Position least significant
P3: file Position most significant

TYPE F3 F2 F1 F0

TYPE P0 P1 P2 P3

2.3.3.1. 16 Bit CRC Binary Header

A binary header is sent by the sending program to the receiving program. ZDLE encoding

accommodates XON/XOFF flow control.

A binary header begins with the sequence ZPAD, ZDLE, ZBIN.

The frame type byte is ZDLE encoded.

The four position/flags bytes are ZDLE encoded.

A two byte CRC of the frame type and position/flag bytes is ZDLE encoded.

0 or more binary data subpackets with 16 bit CRC will follow depending on the frame type.

The function zsbhdr transmits a binary header. The function zgethdr receives a binary or hex header.

Figure 2. 16 Bit CRC Binary Header

* ZDLE A TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2

2.3.3.2. 32 Bit CRC Binary Header

A "32 bit CRC" Binary header is similar to a Binary Header, except the ZBIN (A) character is replaced by a ZBIN32 (C) character, and four characters of CRC are sent. 0 or more binary data subpackets with 32 bit CRC will follow depending on the frame type.

The common variable Txfcs32 may be set TRUE for 32 bit CRC iff the receiver indicates the capability with the CANFC32 bit. The zgethdr, zsdata and zrdata functions automatically adjust to the type of Frame Check Sequence being used.

Figure 3. 32 Bit CRC Binary Header

* ZDLE C TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CRC-3 CRC-4

2.3.3.3. HEX Header

The receiver sends responses in hex headers. The sender also uses hex headers when they are not followed by binary data subpackets.

Hex encoding protects the reverse channel from random control characters. The hex header receiving routine ignores parity.

Use of Kermit style encoding for control and paritied characters was considered and rejected because of increased possibility of interacting with some timesharing systems' line edit functions. Use of HEX headers from the receiving program allows control characters to be used to interrupt the sender when errors are detected. A HEX header may be used in place of a binary header

wherever convenient. If a data packet follows a HEX header, it is protected with CRC-16.

A hex header begins with the sequence ZPAD, ZPAD, ZDLE, ZHEX. The zgethdr routine synchronizes with the ZPAD-ZDLE sequence. The extra ZPAD character allows the sending program to detect an asynchronous header (indicating an error condition) and then call zgethdr to receive the header.

The type byte, the four position/flag bytes, and the 16 bit CRC thereof are sent in hex using the character set 01234567890abcdef. Upper case hex digits are not allowed; they false trigger XMODEM and YMODEM programs. Since this form of hex encoding detects many patterns of errors, especially missing characters, a hex header with 32 bit CRC has not been defined.

A carriage return and line feed are sent with HEX headers. The receive routine expects to see at least one of these characters, two if the first is CR. The CR/LF aids debugging from printouts, and helps overcome certain operating system related problems.

An XON character is appended to all HEX packets except ZACK and ZFIN. The XON releases the sender from spurious XOFF flow control characters generated by line noise, a common occurrence. XON is not sent after ZACK headers to protect flow control in streaming situations. XON is not sent after a ZFIN header to allow clean session cleanup.

0 or more data subpackets will follow depending on the frame type.

The function zshhdr sends a hex header.

Figure 4. HEX Header

* * ZDLE B TYPE F3/P0 F2/P1 F1/P2 F0/P3 CRC-1 CRC-2 CR LF XON

(TYPE, F3...F0, CRC-1, and CRC-2 are each sent as two hex digits.)

2.3.4. Binary Data Subpackets

Binary data subpackets immediately follow the associated binary header packet. A binary data packet contains 0 to 1024 bytes of data. Recommended length values are 256 bytes below 2400 bps, 512 at 2400 bps, and 1024 above 4800 bps or when the data link is known to be relatively error free.[7(7)]

No padding is used with binary data subpackets. The data bytes are ZDLE encoded and transmitted. A ZDLE and frameend are then sent, followed by two or four ZDLE encoded CRC bytes. The CRC accumulates the data bytes and frameend.

The function zsdata sends a data subpacket. The function zrdata receives a data subpacket.

2.3.5. ASCII Encoded Data Subpacket

The format of ASCII Encoded data subpackets is not currently specified. These could be used for server commands, or main transfers in 7 bit environments.

2.4. PROTOCOL TRANSACTION OVERVIEW

As with the XMODEM recommendation, ZMODEM timing is receiver driven. The transmitter should not time out at all, except to abort the program if no headers are received for an extended period of time, say one minute.[8(8)]

2.4.1. Session Startup

To start a ZMODEM file transfer session, the sending program is called with the names of the desired file(s) and option(s).

The sending program may send the string "rz\r" to invoke the receiving program from a possible command mode. The "rz" followed by carriage return activates a ZMODEM receive program or command if it were not already active.

The sender may then display a message intended for human consumption, such as a list of the files requested, etc.

Then the sender may send a ZRQINIT header. The ZRQINIT header causes a previously started receive program to send its ZRINIT header without delay.

In an interactive or conversational mode, the receiving application may monitor the data stream for ZDLE. The following characters may be scanned for B00 indicating a ZRQINIT header, a command to download a command or data.

The sending program awaits a command from the receiving program to start file transfers. If a "C", "G", or NAK is received, an XMODEM or YMODEM file transfer is indicated, and file transfer(s) use the YMODEM protocol.

Note: With ZMODEM and YMODEM, the sending program provides the file name, but not with XMODEM.

In case of garbled data, the sending program can repeat the invitation to receive a number of times until a session starts.

When the ZMODEM receive program starts, it immediately sends a ZRINIT header to initiate ZMODEM file transfers, or a ZCHALLENGE header to verify the sending program. The receive program resends its header at response time (default 10 second) intervals for a suitable period of time (40 seconds total) before falling back to YMODEM protocol.

If the receiving program receives a ZRQINIT header, it resends the ZRINIT header. If the sending program receives the ZCHALLENGE header, it places the data in ZP0...ZP3 in an answering ZACK

header.

If the receiving program receives a ZRINIT header, it is an echo indicating that the sending program is not operational.

Eventually the sending program correctly receives the ZRINIT header.

The sender may then send an optional ZSINIT frame to define the receiving program's Attn sequence, or to specify complete control character escaping.[9(9)]

If the ZSINIT header specifies ESCCTL or ESC8, a HEX header is used, and the receiver activates the specified ESC modes before reading the following data subpacket.

The receiver sends a ZACK header in response, optionally containing the serial number of the receiving program, or 0.

2.4.2. File Transmission

The sender then sends a ZFILE header with ZMODEM Conversion, Management, and Transport options[10(10)] followed by a ZCRCW data subpacket containing the file name, file length, modification date, and other information identical to that used by YMODEM Batch.

The receiver examines the file name, length, and date information provided by the sender in the context of the specified transfer options, the current state of its file system(s), and local security requirements. The receiving program should insure the pathname and options are compatible with its operating environment and local security requirements.

The receiver may respond with a ZSKIP header, which makes the sender proceed to the next file (if any) in the batch.

> If the receiver has a file with the same name and length, it may respond with a ZCRC header, which requires the sender to perform a 32 bit CRC on the file and transmit the complement of the CRC in a ZCRC header.[11(11)] The receiver uses this information to determine whether to accept the file or skip it. This sequence is triggered by the ZMCRC Management Option.

A ZRPOS header from the receiver initiates transmission of the file data starting at the offset in the file specified in the ZRPOS header. Normally the receiver specifies the data transfer to begin begin at offset 0 in the file.

> The receiver may start the transfer further down in the file. This allows a file transfer interrupted by a loss or carrier or system crash to be completed on the next connection without requiring the entire file to be retransmitted.[12(12)] If downloading a file from a timesharing system that becomes sluggish, the transfer can be interrupted and resumed later with no loss of data.

The sender sends a ZDATA binary header (with file position) followed by one or more data

subpackets.

The receiver compares the file position in the ZDATA header with the number of characters successfully received to the file. If they do not agree, a ZRPOS error response is generated to force the sender to the right position within the file.[13(13)]

A data subpacket terminated by ZCRCG and CRC does not elicit a response unless an error is detected; more data subpacket(s) follow immediately.

> ZCRCQ data subpackets expect a ZACK response with the receiver's file offset if no error, otherwise a ZRPOS response with the last good file offset. Another data subpacket continues immediately. ZCRCQ subpackets are not used if the receiver does not indicate FDX ability with the CANFDX bit.

ZCRCW data subpackets expect a response before the next frame is sent. If the receiver does not indicate overlapped I/O capability with the CANOVIO bit, or sets a buffer size, the sender uses the ZCRCW to allow the receiver to write its buffer before sending more data.

> A zero length data frame may be used as an idle subpacket to prevent the receiver from timing out in case data is not immediately available to the sender.

In the absence of fatal error, the sender eventually encounters end of file. If the end of file is encountered within a frame, the frame is closed with a ZCRCE data subpacket which does not elicit a response except in case of error.

The sender sends a ZEOF header with the file ending offset equal to the number of characters in the file. The receiver compares this number with the number of characters received. If the receiver has received all of the file, it closes the file. If the file close was satisfactory, the receiver responds with ZRINIT. If the receiver has not received all the bytes of the file, the receiver ignores the ZEOF because a new ZDATA is coming. If the receiver cannot properly close the file, a ZFERR header is sent.

> After all files are processed, any further protocol errors should not prevent the sending program from returning with a success status.

2.4.3. Session Cleanup

The sender closes the session with a ZFIN header. The receiver acknowledges this with its own ZFIN header.

When the sender receives the acknowledging header, it sends two characters, "OO" (Over and Out) and exits to the operating system or application that invoked it. The receiver waits briefly for the "O" characters, then exits whether they were received or not.

2.4.4. Session Abort Sequence

If the receiver is receiving data in streaming mode, the Attn sequence is executed to interrupt data transmission before the Cancel sequence is sent. The Cancel sequence consists of eight CAN characters and ten backspace characters. ZMODEM only requires five Cancel characters, the other three are "insurance".

The trailing backspace characters attempt to erase the effects of the CAN characters if they are received by a command interpreter.

```
static char canistr[] = {
24,24,24,24,24,24,24,24,8,8,8,8,8,8,8,8,8,8,0
};
```

2.5. STREAMING TECHNIQUES / ERROR RECOVERY

It is a fact of life that no single method of streaming is applicableto a majority of today's computing and telecommunicationsenvironments. ZMODEM provides several data streaming methodsselected according to the limitations of the sending environment,receiving environment, and transmission channel(s).

2.5.1. Full Streaming with Sampling

If the receiver can overlap serial I/O with disk I/O, and if the sender can sample the reverse channel for the presence of data without having to wait, full streaming can be used with no Attn sequence required. The sender begins data transmission with a ZDATA header and continuous ZCRCG data subpackets. When the receiver detects an error, it executes the Attn sequence and then sends a ZRPOS header with the correct position within the file.

At the end of each transmitted data subpacket, the sender checks for the presence of an error header from the receiver. To do this, the sender samples the reverse data stream for the presence of either a ZPAD or CAN character.[14(14)] Flow control characters (if present) are acted upon.

Other characters (indicating line noise) increment a counter which is reset whenever the sender waits for a header from the receiver. If the counter overflows, the sender sends the next data subpacket as ZCRCW, and waits for a response.

ZPAD indicates some sort of error header from the receiver. A CAN suggests the user is attempting to "stop the bubble machine" by keyboarding CAN characters. If one of these characters is seen, an empty ZCRCE data subpacket is sent. Normally, the receiver will have sent an ZRPOS or other error header, which will force the sender to resume transmission at a different address, or take other action. In the unlikely event the ZPAD or CAN character was spurious, the receiver will time out and send a ZRPOS header.[15(15)]

Then the receiver's response header is read and acted upon.[16(16)]

A ZRPOS header resets the sender's file offset to the correct position. If possible, the sender should purge its output buffers and/or networks of all unprocessed output data, to minimize the amount of unwanted data the receiver must discard before receiving data starting at the correct file offset. The next transmitted data frame should be a ZCRCW frame followed by a wait to guarantee complete flushing of the network's memory.

If the receiver gets a ZACK header with an address that disagrees with the sender address, it is ignored, and the sender waits for another header. A ZFIN, ZABORT, or TIMEOUT terminates the session; a ZSKIP terminates the processing of this file.

The reverse channel is then sampled for the presence of another header from the receiver.[17(17)] if one is detected, the getinsync() function is again called to read another error header. Otherwise, transmission resumes at the (possibly reset) file offset with a ZDATA header followed by data subpackets.

2.5.1.1. Window Management

When sending data through a network, some nodes of the network store data while it is transferred to the receiver. 7000 bytes and more of transient storage have been observed. Such a large amount of storage causes the transmitter to "get ahead" of the reciever. This can be fatal with MEGAlink and other protocols that depend on timely notification of errors from the receiver. This condition is not fatal with ZMODEM, but it does slow error recovery.

To manage the window size, the sending program uses ZCRCQ data subpackets to trigger ZACK headers from the receiver. The returning ZACK headers inform the sender of the receiver's progress. When the window size (current transmitter file offset - last reported receiver file offset) exceeds a specified value, the sender waits for a ZACK[18(18)] packet with a receiver file offset that reduces the window size.

Unix sz versions beginning with May 9 1987 control the window size with the "-w N" option, where N is the maximum window size. Pro-YAM, ZCOMM and DSZ versions beginning with May 9 1987 control the window size with "zmodem pwN". This is compatible with previous versions of these programs.[19(19)]

2.5.2. Full Streaming with Reverse Interrupt

The above method cannot be used if the reverse data stream cannot be sampled without entering an I/O wait. An alternate method is to instruct the receiver to interrupt the sending program when an error is detected.

The receiver can interrupt the sender with a control character, break signal, or combination thereof, as specified in the Attn sequence. After executing the Attn sequence, the receiver sends a hex ZRPOS header to force the sender to resend the lost data.

When the sending program responds to this interrupt, it reads a HEX header (normally ZRPOS)

from the receiver and takes the action described in the previous section. The Unix sz.c program uses a setjmp/longjmp call to catch the interrupt generated by the Attn sequence. Catching the interrupt activates the getinsync() function to read the receiver's error header and take appropriate action.

When compiled for standard SYSTEM III/V Unix, sz.c uses an Attn sequence of Ctrl-C followed by a 1 second pause to interrupt the sender, then give the sender (Unix) time to prepare for the receiver's error header.

2.5.3. Full Streaming with Sliding Window

If none of the above methods is applicable, hope is not yet lost. If the sender can buffer responses from the receiver, the sender can use ZCRCQ data subpackets to get ACKs from the receiver without interrupting the transmission of data. After a sufficient number of ZCRCQ data subpackets have been sent, the sender can read one of the headers that should have arrived in its receive interrupt buffer.

A problem with this method is the possibility of wasting an excessive amount of time responding to the receiver's error header. It may be possible to program the receiver's Attn sequence to flush the sender's interrupt buffer before sending the ZRPOS header.

2.5.4. Full Streaming over Error Free Channels

File transfer protocols predicated on the existence of an error free end to end communications channel have been proposed from time to time. Such channels have proven to be more readily available in theory than in actuality. The frequency of undetected errors increases when modem scramblers have more bits than the error detecting CRC.

A ZMODEM sender assuming an error free channel with end to end flow control can send the entire file in one frame without any checking of the reverse stream. If this channel is completely transparent, only ZDLE need be escaped. The resulting protocol overhead for average long files is less than one per cent.[20(20)]

2.5.5. Segmented Streaming

If the receiver cannot overlap serial and disk I/O, it uses the ZRINIT frame to specify a buffer length which the sender will not overflow. The sending program sends a ZCRCW data subpacket and waits for a ZACK header before sending the next segment of the file.

If the sending program supports reverse data stream sampling or interrupt, error recovery will be faster (on average) than a protocol (such as YMODEM) that sends large blocks.

A sufficiently large receiving buffer allows throughput to closely approach that of full streaming. For example, 16kb segmented streaming adds about 3 per cent to full streaming ZMODEM file transfer times when the round trip delay is five seconds.

2.6. ATTENTION SEQUENCE

The receiving program sends the Attn sequence whenever it detects an error and needs to interrupt the sending program.

The default Attn string value is empty (no Attn sequence). The receiving program resets Attn to the empty default before each transfer session.

The sender specifies the Attn sequence in its optional ZSINIT frame. The Attn string is terminated with a null.

Two meta-characters perform special functions:

- \335 (octal) Send a break signal
- \336 (octal) Pause one second

2.7. FRAME TYPES

The numeric values for the values shown in boldface are given in zmodem.h. Unused bits and unused bytes in the header (ZP0...ZP3) are set to 0.

2.7.1. ZRQINIT

Sent by the sending program, to trigger the receiving program to send its ZRINIT header. This avoids the aggravating startup delay associated with XMODEM and Kermit transfers. The sending program may repeat the receive invitation (including ZRQINIT) if a response is not obtained at first.

ZF0 contains ZCOMMAND if the program is attempting to send a command, 0 otherwise.

2.7.2. ZRINIT

Sent by the receiving program. ZF0 and ZF1 contain the bitwise or of the receiver capability flags:

```
#define CANCRY      8 /* Receiver can decrypt */
#define CANFDX      01 /* Rx can send and receive true FDX */
#define CANOVIO     02 /* Rx can receive data during disk I/O */
```

```
#define CANBRK      04 /* Rx can send a break signal */
#define CANCRY      010 /* Receiver can decrypt */
#define CANLZW      020 /* Receiver can uncompress */
#define CANFC32     040 /* Receiver can use 32 bit Frame Check */
#define ESCCTL      0100 /* Receiver expects ctl chars to be escaped */
#define ESC8        0200 /* Receiver expects 8th bit to be escaped */
```

ZP0 and ZP1 contain the size of the receiver's buffer in bytes, or 0 if nonstop I/O is allowed.

2.7.3. ZSINIT

The Sender sends flags followed by a binary data subpacket terminated with ZCRCW.

```
/* Bit Masks for ZSINIT flags byte ZF0 */
#define TESCCTL     0100 /* Transmitter expects ctl chars to be escaped */
#define TESC8       0200 /* Transmitter expects 8th bit to be escaped */
```

The data subpacket contains the null terminated Attn sequence, maximum length 32 bytes including the terminating null.

2.7.4. ZACK

Acknowledgment to a ZSINIT frame, ZCHALLENGE header, ZCRCQ or ZCRCW data subpacket. ZP0 to ZP3 contain file offset. The response to ZCHALLENGE contains the same 32 bit number received in the ZCHALLENGE header.

2.7.5. ZFILE

This frame denotes the beginning of a file transmission attempt. ZF0, ZF1, and ZF2 may contain options. A value of 0 in each of these bytes implies no special treatment. Options specified to the receiver override options specified to the sender with the exception of ZCBIN which overrides any other Conversion Option given to the sender or receiver.

2.7.5.1. ZF0: Conversion Option

If the receiver does not recognize the Conversion Option, an application dependent default conversion may apply.

ZCBIN "Binary" transfer - inhibit conversion unconditionally

ZCNL Convert received end of line to local end of line convention. The supported end of line conventions are CR/LF (most ASCII based operating systems except Unix and Macintosh), and NL (Unix). Either of these two end of line conventions meet the permissible ASCII definitions for Carriage Return and Line Feed/New Line. Neither the ASCII code nor ZMODEM ZCNL encompass lines separated only by carriage returns. Other processing appropriate to ASCII text files and the local operating system may also be applied by the receiver.[21(21)]

ZCRECOV Recover/Resume interrupted file transfer. ZCREVOV is also useful for updating a remote copy of a file that grows without resending of old data. If the destination file exists and is no longer than the source, append to the destination file and start transfer at the offset corresponding to the receiver's end of file. This option does not apply if the source file is shorter. Files that have been converted (e.g., ZCNL) or subject to a single ended Transport Option cannot have their transfers recovered.

2.7.5.2. ZF1: Management Option

If the receiver does not recognize the Management Option, the file should be transferred normally.

The ZMSKNOLOC bit instructs the receiver to bypass the current file if the receiver does not have a file with the same name.

Five bits (defined by ZMMASK) define the following set of mutually exclusive management options.

ZMNEWL Transfer file if destination file absent. Otherwise, transfer file overwriting destination if the source file is newer or longer.

ZMCRC Compare the source and destination files. Transfer if file lengths or file polynomials differ.

ZMAPND Append source file contents to the end of the existing destination file (if any).

ZMCLOB Replace existing destination file (if any).

ZMDIFF Transfer file if destination file absent. Otherwise, transfer file overwriting destination if files have different lengths or dates.

ZMPROT Protect destination file by transferring file only if the destination file is absent.

ZMNEW Transfer file if destination file absent. Otherwise, transfer file overwriting destination if the source file is newer.

2.7.5.3. ZF2: Transport Option

If the receiver does not implement the particular transport option, the file is copied without

conversion for later processing.

ZTLZW Lempel-Ziv compression. Transmitted data will be identical to that produced by compress 4.0 operating on a computer with VAX byte ordering, using 12 bit encoding.

ZTCRYPT Encryption. An initial null terminated string identifies the key. Details to be determined.

ZTRLE Run Length encoding, Details to be determined.

A ZCRCW data subpacket follows with file name, file length, modification date, and other information described in a later chapter.

2.7.5.4. ZF3: Extended Options

The Extended Options are bit encoded.

ZTSPARS Special processing for sparse files, or sender managed selective retransmission. Each file segment is transmitted as a separate frame, where the frames are not necessarily contiguous. The sender should end each segment with a ZCRCW data subpacket and process the expected ZACK to insure no data is lost. ZTSPARS cannot be used with ZCNL.

2.7.6. ZSKIP

Sent by the receiver in response to ZFILE, makes the sender skip to the next file.

2.7.7. ZNAK

Indicates last header was garbled. (See also ZRPOS).

2.7.8. ZABORT

Sent by receiver to terminate batch file transfers when requested by the user. Sender responds with a ZFIN sequence.[22(22)]

2.7.9. ZFIN

Sent by sending program to terminate a ZMODEM session. Receiver responds with its own ZFIN.

2.7.10. ZRPOS

Sent by receiver to force file transfer to resume at file offset given in ZP0...ZP3.

2.7.11. ZDATA

ZP0...ZP3 contain file offset. One or more data subpackets follow.

2.7.12. ZEOF

Sender reports End of File. ZP0...ZP3 contain the ending file offset.

2.7.13. ZFERR

Error in reading or writing file, protocol equivalent to ZABORT.

2.7.14. ZCRC

Request (receiver) and response (sender) for file polynomial. ZP0...ZP3 contain file polynomial.

2.7.15. ZCHALLENGE

Request sender to echo a random number in ZP0...ZP3 in a ZACK frame. Sent by the receiving program to the sending program to verify that it is connected to an operating program, and was not activated by spurious data or a Trojan Horse message.

2.7.16. ZCOMPL

Request now completed.

2.7.17. ZCAN

This is a pseudo frame type returned by gethdr() in response to a Session Abort sequence.

2.7.18. ZFREECNT

Sending program requests a ZACK frame with ZP0...ZP3 containing the number of free bytes on the current file system. A value of 0 represents an indefinite amount of free space.

2.7.19. ZCOMMAND

ZCOMMAND is sent in a binary frame. ZF0 contains 0 or ZCACK1 (see below).

A ZCRCW data subpacket follows, with the ASCII text command string terminated with a NULL character. If the command is intended to be executed by the operating system hosting the receiving program (e.g., "shell escape"), it must have "!" as the first character. Otherwise the command is meant to be executed by the application program which receives the command.

If the receiver detects an illegal or badly formed command, the receiver immediately responds with a ZCOMPL header with an error code in ZP0...ZP3.

If ZF0 contained ZCACK1, the receiver immediately responds with a ZCOMPL header with 0 status.

Otherwise, the receiver responds with a ZCOMPL header when the operation is completed. The exit status of the completed command is stored in ZP0...ZP3. A 0 exit status implies nominal completion of the command.

If the command causes a file to be transmitted, the command sender will see a ZRQINIT frame from the other computer attempting to send data.

The sender examines ZF0 of the received ZRQINIT header to verify it is not an echo of its own ZRQINIT header. It is illegal for the sending program to command the receiving program to send a command.

If the receiver program does not implement command downloading, it may display the command to the standard error output, then return a ZCOMPL header.

2.8. SESSION TRANSACTION EXAMPLES

2.8.1. A simple file transfer

A simple transaction, one file, no errors, no CHALLENGE, overlapped I/O:

Sender	**Receiver**
"rz\r"	
ZRQINIT(0)	
	ZRINIT
ZFILE	
	ZRPOS
ZDATA data ...	
ZEOF	
	ZRINIT
ZFIN	
	ZFIN
OO	

2.8.2. Challenge and Command Download

Sender	**Receiver**
"rz\r"	
ZRQINIT(ZCOMMAND)	
	ZCHALLENGE(random-number)
ZACK(same-number)	
	ZRINIT
ZCOMMAND, ZDATA	
	(Performs Command) ZCOMPL
ZFIN	
	ZFIN
OO	

2.9. ZFILE FRAME FILE INFORMATION

ZMODEM sends the same file information with the ZFILE frame data that YMODEM Batch sends in its block 0.

N.B.: The pathname (file name) field is mandatory.

Pathname The pathname (conventionally, the file name) is sent as a null terminated ASCII string. This is the filename format used by the handle oriented MSDOS(TM)

functions and C library fopen functions. An assembly language example follows:

```
DB      'foo.bar',0
```

No spaces are included in the pathname. Normally only the file name stem (no directory prefix) is transmitted unless the sender has selected YAM's f option to send the full absolute or relative pathname. The source drive designator (A:, B:, etc.) usually is not sent.

Filename Considerations:

- File names should be translated to lower case unless the sending system supports upper/lower case file names. This is a convenience for users of systems (such as Unix) which store filenames in upper and lower case
- The receiver should accommodate file names in lower and upper case.
- When transmitting files between different operating systems, file names must be acceptable to both the sender and receiving operating systems. If not, transformations should be applied to make the file names acceptable. If the transformations are unsuccessful, a new file name may be invented be the receiving program.

If directories are included, they are delimited by /; i.e., "subdir/foo" is acceptable, "subdir\foo" is not.

Length The file length and each of the succeeding fields are optional.[23(23)] The length field is stored as a decimal string counting the number of data bytes in the file.

The ZMODEM receiver uses the file length as an estimate only. It may be used to display an estimate of the transmission time, and may be compared with the amount of free disk space. The actual length of the received file is determined by the data transfer. A file may grow after transmission commences, and all the data will be sent.

Modification Date A single space separates the modification date from the file length.

The mod date is optional, and the filename and length may be sent without requiring the mod date to be sent.

The mod date is sent as an octal number giving the time the contents of the file were last changed measured in seconds from Jan 1 1970 Universal Coordinated Time (GMT). A date of 0 implies the modification date is unknown and should be left as the date the file is received.

This standard format was chosen to eliminate ambiguities arising from transfers between different time zones.

File Mode A single space separates the file mode from the modification date. The file mode is stored as an octal string. Unless the file originated from a Unix system, the file mode is set to 0. rz(1) checks the file mode for the 0x8000 bit which indicates a Unix type regular file. Files with the 0x8000 bit set are assumed to have been sent

from another Unix (or similar) system which uses the same file conventions. Such files are not translated in any way.

Serial Number A single space separates the serial number from the file mode. The serial number of the transmitting program is stored as an octal string. Programs which do not have a serial number should omit this field, or set it to 0. The receiver's use of this field is optional.

The file information is terminated by a null. If only the pathname is sent, the pathname is terminated with two nulls. The length of the file information subpacket, including the trailing null, must not exceed 1024 bytes; a typical length is less than 64 bytes.

3. Programmbeispiele

3.1. Definition der ZMODEM-Konstanten

```
/*
 *  Z M O D E M . H     Manifest constants for ZMODEM
 *   application to application file transfer protocol
 *   04-17-89  Chuck Forsberg Omen Technology Inc
 */
#define ZPAD '*'        /* 052 Padding character begins frames */
#define ZDLE 030        /* Ctrl-X Zmodem escape - `ala BISYNC DLE */
#define ZDLEE (ZDLE^0100)    /* Escaped ZDLE as transmitted */
#define ZBIN 'A'        /* Binary frame indicator (CRC-16) */
#define ZHEX 'B'        /* HEX frame indicator */
#define ZBIN32 'C'      /* Binary frame with 32 bit FCS */
#define ZBINR32 'D'     /* RLE packed Binary frame with 32 bit FCS */
#define ZVBIN 'a'       /* Binary frame indicator (CRC-16) */
#define ZVHEX 'b'       /* HEX frame indicator */
#define ZVBIN32 'c'     /* Binary frame with 32 bit FCS */
#define ZVBINR32 'd'    /* RLE packed Binary frame with 32 bit FCS */
#define ZRESC 0176      /* RLE flag/escape character */
#define ZMAXHLEN 16 /* Max header information length  NEVER CHANGE */
#define ZMAXSPLEN 1024      /* Max subpacket length  NEVER CHANGE */

/* Frame types (see array "frametypes" in zm.c) */
#define ZRQINIT 0       /* Request receive init */
#define ZRINIT 1        /* Receive init */
#define ZSINIT 2        /* Send init sequence (optional) */
#define ZACK 3/* ACK to above */
#define ZFILE 4         /* File name from sender */
#define ZSKIP 5         /* To sender: skip this file */
#define ZNAK 6/* Last packet was garbled */
#define ZABORT 7        /* Abort batch transfers */
#define ZFIN 8  /* Finish session */
#define ZRPOS 9         /* Resume data trans at this position */
#define ZDATA 10        /* Data packet(s) follow */
#define ZEOF 11         /* End of file */
#define ZFERR 12        /* Fatal Read or Write error Detected */
#define ZCRC 13         /* Request for file CRC and response */
#define ZCHALLENGE 14       /* Receiver's Challenge */
#define ZCOMPL 15    /* Request is complete */
#define ZCAN 16         /* Other end canned session with CAN*5 */
#define ZFREECNT 17 /* Request for free bytes on filesystem */
#define ZCOMMAND 18         /* Command from sending program */
#define ZSTDERR 19   /* Output to standard error, data follows */
```

```
/* ZDLE sequences */
#define ZCRCE 'h'       /* CRC next, frame ends, header packet follows */
#define ZCRCG 'i'       /* CRC next, frame continues nonstop */
#define ZCRCQ 'j'       /* CRC next, frame continues, ZACK expected */
#define ZCRCW 'k'       /* CRC next, ZACK expected, end of frame */
#define ZRUB0 'l'       /* Translate to rubout 0177 */
#define ZRUB1 'm'       /* Translate to rubout 0377 */

/* zdlread return values (internal) */
/* -1 is general error, -2 is timeout */
#define GOTOR 0400
#define GOTCRCE (ZCRCE|GOTOR)   /* ZDLE-ZCRCE received */
#define GOTCRCG (ZCRCG|GOTOR)  /* ZDLE-ZCRCG received */
#define GOTCRCQ (ZCRCQ|GOTOR)  /* ZDLE-ZCRCQ received */
#define GOTCRCW (ZCRCW|GOTOR) /* ZDLE-ZCRCW received */
#define GOTCAN (GOTOR|030)          /* CAN*5 seen */

/* Byte positions within header array */
#define ZF0 3   /* First flags byte */
#define ZF1 2
#define ZF2 1
#define ZF3 0
#define ZP0 0   /* Low order 8 bits of position */
#define ZP1 1
#define ZP2 2
#define ZP3 3   /* High order 8 bits of file position */

/* Bit Masks for ZRINIT flags byte ZF0 */
#define CANFDX 01     /* Rx can send and receive true FDX */
#define CANOVIO 02    /* Rx can receive data during disk I/O */
#define CANBRK 04     /* Rx can send a break signal */
#define CANRLE 010    /* Receiver can decode RLE */
#define CANLZW 020    /* Receiver can uncompress */
#define CANFC32 040   /* Receiver can use 32 bit Frame Check */
#define ESCCTL 0100   /* Receiver expects ctl chars to be escaped */
#define ESC8   0200   /* Receiver expects 8th bit to be escaped */

/* Bit Masks for ZRINIT flags byte ZF1 */
#define CANVHDR 01   /* Variable headers OK */

/* Parameters for ZSINIT frame */
#define ZATTNLEN 32  /* Max length of attention string */
#define ALTCOFF ZF1  /* Offset to alternate canit string, 0 if not used */

/* Bit Masks for ZSINIT flags byte ZF0 */
#define TESCCTL 0100/* Transmitter expects ctl chars to be escaped */
#define TESC8   0200   /* Transmitter expects 8th bit to be escaped */

/* Parameters for ZFILE frame */
/* Conversion options one of these in ZF0 */
#define ZCBIN 1          /* Binary transfer - inhibit conversion */
#define ZCNL 2/* Convert NL to local end of line convention */
#define ZCRESUM 3     /* Resume interrupted file transfer */
/* Management include options, one of these ored in ZF1 */
#define ZMSKNOLOC 0200      /* Skip file if not present at rx */
/* Management options, one of these ored in ZF1 */
#define ZMMASK 037   /* Mask for the choices below */
#define ZMNEWL 1      /* Transfer if source newer or longer */
#define ZMCRC 2       /* Transfer if different file CRC or length */
#define ZMAPND 3      /* Append contents to existing file (if any) */
#define ZMCLOB 4      /* Replace existing file */
```

```
#define ZMNEW 5         /* Transfer if source newer */
        /* Number 5 is alive ... */
#define ZMDIFF 6        /* Transfer if dates or lengths different */
#define ZMPROT 7        /* Protect destination file */
/* Transport options, one of these in ZF2 */
#define ZTLZW 1         /* Lempel-Ziv compression */
#define ZTRLE 3         /* Run Length encoding */
/* Extended options for ZF3, bit encoded */
#define ZXSPARS 64      /* Encoding for sparse file operations */
#define ZCANVHDR 01 /* Variable headers OK */
/* Receiver window size override */
#define ZRWOVR 4        /* byte position for receive window override/256 */

/* Parameters for ZCOMMAND frame ZF0 (otherwise 0) */
#define ZCACK1 1        /* Acknowledge, then do command */

long rclhdr();

/* Globals used by ZMODEM functions */
extern Rxframeind;      /* ZBIN ZBIN32, or ZHEX type of frame */
extern Rxtype;  /* Type of header received */
extern Rxcount;/* Count of data bytes received */
extern Rxtimeout;       /* Tenths of seconds to wait for something */
extern long Rxpos;      /* Received file position */
extern long Txpos;      /* Transmitted file position */
extern Txfcs32; /* TURE means send binary frames with 32 bit FCS */
extern Crc32t;  /* Display flag indicating 32 bit CRC being sent */
extern Crc32;   /* Display flag indicating 32 bit CRC being received */
extern Znulls;  /* Number of nulls to send at beginning of ZDATA hdr */
extern char Attn[ZATTNLEN+1];/* Attention string rx sends to tx on err */
extern char *Altcan;    /* Alternate canit string */

/* End of ZMODEM.H */
```

3.2. C-Programmbeispiele der ZMODEM-Basisfunktionen

```
/*
 *  Z M . C
 *   ZMODEM protocol primitives
 *   05-24-89  Chuck Forsberg Omen Technology Inc
 *
 * Entry point Functions:
 *      zsbhdr(type, hdr) send binary header
 *      zshhdr(type, hdr) send hex header
 *      zgethdr(hdr, eflag) receive header - binary or hex
 *      zsdata(buf, len, frameend) send data
 *      zrdata(buf, len) receive data
 *      stohdr(pos) store position data in Txhdr
 *      long rclhdr(hdr) recover position offset from header
 *
 *      This version implements ZMODEM Run Length Encoding, Comparision,
 *      and variable length headers.  These features were not funded
 *      by the original Telenet development contract.  This software,
 *      including these features, may be freely used for non
 *      commercial and educational purposes.  This software may also
 *      be freely used to support file transfer operations to or from
 *      licensed Omen Technology products.  Contact Omen Technology
 *      for licensing for other uses.  Any programs which use part or
 *      all of this software must be provided in source form with this
```

```
 *      notice intact except by written permission from Omen
 *      Technology Incorporated.
 *
 *              Omen Technology Inc             FAX: 503-621-3745
 *              Post Office Box 4681
 *              Portland OR 97208
 *
 *      Previous versions of this program (not containing the extensions
 *      listed above) remain in the public domain.
 *
 *      This code is made available in the hope it will be useful,
 *      BUT WITHOUT ANY WARRANTY OF ANY KIND OR LIABILITY FOR ANY
 *      DAMAGES OF ANY KIND.
 *
 */

#ifndef CANFDX
#include "zmodem.h"
int Rxtimeout = 100;            /* Tenths of seconds to wait for something */
#endif

#ifndef UNSL
#define UNSL
#endif


/* Globals used by ZMODEM functions */
int Rxframeind;         /* ZBIN ZBIN32, or ZHEX type of frame */
int Rxtype;             /* Type of header received */
int Rxhlen;             /* Length of header received */
int Rxcount;            /* Count of data bytes received */
char Rxhdr[ZMAXHLEN];       /* Received header */
char Txhdr[ZMAXHLEN];       /* Transmitted header */
long Rxpos;             /* Received file position */
long Txpos;             /* Transmitted file position */
int Txfcs32;            /* TURE means send binary frames with 32 bit FCS */
int Crc32t;             /* Controls 32 bit CRC being sent */
                        /* 1 == CRC32,  2 == CRC32 + RLE */
int Crc32r;             /* Indicates/controls 32 bit CRC being received */
                        /* 0 == CRC16,  1 == CRC32,  2 == CRC32 + RLE */
int Usevhdrs;           /* Use variable length headers */
int Znulls;             /* Number of nulls to send at beginning of ZDATA hdr */
char Attn[ZATTNLEN+1];      /* Attention string rx sends to tx on err */
char *Altcan;           /* Alternate canit string */

static lastsent;   /* Last char we sent */
static Not8bit;         /* Seven bits seen on header */

static char *frametypes[] = {
        "No Response to Error Correction Request",      /* -4 */
        "No Carrier Detect",                /* -3 */
        "TIMEOUT",          /* -2 */
        "ERROR",            /* -1 */
#define FTOFFSET 4
        "ZRQINIT",
        "ZRINIT",
        "ZSINIT",
        "ZACK",
        "ZFILE",
        "ZSKIP",
        "ZNAK",
        "ZABORT",
```

```
        "ZFIN",
        "ZRPOS",
        "ZDATA",
        "ZEOF",
        "ZFERR",
        "ZCRC",
        "ZCHALLENGE",
        "ZCOMPL",
        "ZCAN",
        "ZFREECNT",
        "ZCOMMAND",
        "ZSTDERR",
        "xxxxx"
#define FRTYPES 22    /* Total number of frame types in this array */
                      /*  not including psuedo negative entries */
};

static char badcrc[] = "Bad CRC";

/* Send ZMODEM binary header hdr of type type */
zsbhdr(len, type, hdr)
register char *hdr;
{
        register int n;
        register unsigned short crc;

#ifndef DSZ
        vfile("zsbhdr: %c %d %s %lx", Usevhdrs?'v':'f', len,
          frametypes[type+FTOFFSET], rclhdr(hdr));
#endif
        if (type == ZDATA)
                for (n = Znulls; --n >=0; )
                        xsendline(0);

        xsendline(ZPAD); xsendline(ZDLE);

        switch (Crc32t=Txfcs32) {
        case 2:
                zsbh32(len, hdr, type, Usevhdrs?ZVBINR32:ZBINR32);
                flushmo();  break;
        case 1:
                zsbh32(len, hdr, type, Usevhdrs?ZVBIN32:ZBIN32);  break;
        default:
                if (Usevhdrs) {
                        xsendline(ZVBIN);
                        zsendline(len);
                }
                else
                        xsendline(ZBIN);
                zsendline(type);
                crc = updcrc(type, 0);

                for (n=len; --n >= 0; ++hdr) {
                        zsendline(*hdr);
                        crc = updcrc((0377& *hdr), crc);
                }
                crc = updcrc(0,updcrc(0,crc));
                zsendline(crc>>8);
                zsendline(crc);
        }
        if (type != ZDATA)
                flushmo();
```

```
}


/* Send ZMODEM binary header hdr of type type */
zsbh32(len, hdr, type, flavour)
register char *hdr;
{
        register int n;
        register UNSL long crc;

        xsendline(flavour);
        if (Usevhdrs)
                zsendline(len);
        zsendline(type);
        crc = 0xFFFFFFFFL; crc = UPDC32(type, crc);

        for (n=len; --n >= 0; ++hdr) {
                crc = UPDC32((0377 & *hdr), crc);
                zsendline(*hdr);
        }
        crc = ~crc;
        for (n=4; --n >= 0;) {
                zsendline((int)crc);
                crc >>= 8;
        }
}

/* Send ZMODEM HEX header hdr of type type */
zshhdr(len, type, hdr)
register char *hdr;
{
        register int n;
        register unsigned short crc;

#ifndef DSZ
        vfile("zshhdr: %c %d %s %lx", Usevhdrs?'v':'f', len,
          frametypes[type+FTOFFSET], rclhdr(hdr));
#endif
        sendline(ZPAD); sendline(ZPAD); sendline(ZDLE);
        if (Usevhdrs) {
                sendline(ZVHEX);
                zputhex(len);
        }
        else
                sendline(ZHEX);
        zputhex(type);
        Crc32t = 0;

        crc = updcrc(type, 0);
        for (n=len; --n >= 0; ++hdr) {
                zputhex(*hdr); crc = updcrc((0377 & *hdr), crc);
        }
        crc = updcrc(0,updcrc(0,crc));
        zputhex(crc>>8); zputhex(crc);

        /* Make it printable on remote machine */
        sendline(015); sendline(0212);
        /*
         * Uncork the remote in case a fake XOFF has stopped data flow
         */
        if (type != ZFIN && type != ZACK)
                sendline(021);
```

```
	flushmo();
}

/*
 * Send binary array buf of length length, with ending ZDLE sequence frameend
 */
static char *Zendnames[] = { "ZCRCE", "ZCRCG", "ZCRCQ", "ZCRCW"};
zsdata(buf, length, frameend)
register char *buf;
{
	register unsigned short crc;

#ifndef DSZ
	vfile("zsdata: %d %s", length, Zendnames[frameend-ZCRCE&3]);
#endif
	switch (Crc32t) {
	case 1:
		zsda32(buf, length, frameend);  break;
	case 2:
		zsdar32(buf, length, frameend);  break;
	default:
		crc = 0;
		for (;--length >= 0; ++buf) {
			zsendline(*buf); crc = updcrc((0377 & *buf), crc);
		}
		xsendline(ZDLE); xsendline(frameend);
		crc = updcrc(frameend, crc);

		crc = updcrc(0,updcrc(0,crc));
		zsendline(crc>>8); zsendline(crc);
	}
	if (frameend == ZCRCW) {
		xsendline(XON);  flushmo();
	}
}

zsda32(buf, length, frameend)
register char *buf;
{
	register int c;
	register UNSL long crc;

	crc = 0xFFFFFFFFL;
	for (;--length >= 0; ++buf) {
		c = *buf & 0377;
		if (c & 0140)
			xsendline(lastsent = c);
		else
			zsendline(c);
		crc = UPDC32(c, crc);
	}
	xsendline(ZDLE); xsendline(frameend);
	crc = UPDC32(frameend, crc);

	crc = ~crc;
	for (c=4; --c >= 0;) {
		zsendline((int)crc);  crc >>= 8;
	}
}

/*
 * Receive array buf of max length with ending ZDLE sequence
```

```
 *  and CRC.  Returns the ending character or error code.
 *  NB: On errors may store length+1 bytes!
 */
zrdata(buf, length)
register char *buf;
{
        register int c;
        register unsigned short crc;
        register char *end;
        register int d;

        switch (Crc32r) {
        case 1:
                return zrdat32(buf, length);
        case 2:
                return zrdatr32(buf, length);
        }

        crc = Rxcount = 0;  end = buf + length;
        while (buf <= end) {
                if ((c = zdlread()) & ~0377) {
crcfoo:
                        switch (c) {
                        case GOTCRCE:
                        case GOTCRCG:
                        case GOTCRCQ:
                        case GOTCRCW:
                                crc = updcrc((d=c)&0377, crc);
                                if ((c = zdlread()) & ~0377)
                                        goto crcfoo;
                                crc = updcrc(c, crc);
                                if ((c = zdlread()) & ~0377)
                                        goto crcfoo;
                                crc = updcrc(c, crc);
                                if (crc & 0xFFFF) {
                                        zperr(badcrc);
                                        return ERROR;
                                }
                                Rxcount = length - (end - buf);
#ifndef DSZ
                                vfile("zrdata: %d  %s", Rxcount,
                                 Zendnames[d-GOTCRCE&3]);
#endif
                                return d;
                        case GOTCAN:
                                zperr("Sender Canceled");
                                return ZCAN;
                        case TIMEOUT:
                                zperr("TIMEOUT");
                                return c;
                        default:
                                garbitch(); return c;
                        }
                }
                *buf++ = c;
                crc = updcrc(c, crc);
        }
#ifdef DSZ
        garbitch();
#else
        zperr("Data subpacket too long");
#endif
```

```
	return ERROR;
}

zrdat32(buf, length)
register char *buf;
{
	register int c;
	register UNSL long crc;
	register char *end;
	register int d;

	crc = 0xFFFFFFFFL;  Rxcount = 0;  end = buf + length;
	while (buf <= end) {
		if ((c = zdlread()) & ~0377) {
crcfoo:
			switch (c) {
			case GOTCRCE:
			case GOTCRCG:
			case GOTCRCQ:
			case GOTCRCW:
				d = c;  c &= 0377;
				crc = UPDC32(c, crc);
				if ((c = zdlread()) & ~0377)
					goto crcfoo;
				crc = UPDC32(c, crc);
				if ((c = zdlread()) & ~0377)
					goto crcfoo;
				crc = UPDC32(c, crc);
				if ((c = zdlread()) & ~0377)
					goto crcfoo;
				crc = UPDC32(c, crc);
				if ((c = zdlread()) & ~0377)
					goto crcfoo;
				crc = UPDC32(c, crc);
				if (crc != 0xDEBB20E3) {
					zperr(badcrc);
					return ERROR;
				}
				Rxcount = length - (end - buf);
#ifndef DSZ
				vfile("zrdat32: %d %s", Rxcount,
				 Zendnames[d-GOTCRCE&3]);
#endif
				return d;
			case GOTCAN:
				zperr("Sender Canceled");
				return ZCAN;
			case TIMEOUT:
				zperr("TIMEOUT");
				return c;
			default:
				garbitch(); return c;
			}
		}
		*buf++ = c;
		crc = UPDC32(c, crc);
	}
	zperr("Data subpacket too long");
	return ERROR;
}

garbitch()
```

```
{
        zperr("Garbled data subpacket");
}

/*
 * Read a ZMODEM header to hdr, either binary or hex.
 *  eflag controls local display of non zmodem characters:
 *        0:  no display
 *        1:  display printing characters only
 *        2:  display all non ZMODEM characters
 *
 *   Set Rxhlen to size of header (default 4) (valid iff good hdr)
 *  On success, set Zmodem to 1, set Rxpos and return type of header.
 *   Otherwise return negative on error.
 *   Return ERROR instantly if ZCRCW sequence, for fast error recovery.
 */
zgethdr(hdr, eflag)
char *hdr;
{
        register int c, n, cancount;

        n = Zrwindow + Effbaud;                 /* Max bytes before start of frame */
        Rxframeind = Rxtype = 0;

startover:
        cancount = 5;
again:
        /* Return immediate ERROR if ZCRCW sequence seen */
        switch (c = readline(Rxtimeout)) {
        case RCDO:
        case TIMEOUT:
                goto fifi;
        case CAN:
gotcan:
                if (--cancount <= 0) {
                        c = ZCAN; goto fifi;
                }
                switch (c = readline(1)) {
                case TIMEOUT:
                        goto again;
                case ZCRCW:
                        switch (readline(1)) {
                        case TIMEOUT:
                                c = ERROR; goto fifi;
                        case RCDO:
                                goto fifi;
                        default:
                                goto agn2;
                        }
                case RCDO:
                        goto fifi;
                default:
                        break;
                case CAN:
                        if (--cancount <= 0) {
                                c = ZCAN; goto fifi;
                        }
                        goto again;
                }
        /* **** FALL THRU TO **** */
        default:
agn2:
```

```
                if ( --n == 0) {
                        c = GCOUNT;  goto fifi;
                }
                if (eflag && ((c &= 0177) & 0140))
                        bttyout(c);
                else if (eflag > 1)
                        bttyout(c);
#ifdef UNIX
                fflush(stderr);
#endif
                goto startover;
        case ZPAD|0200:                 /* This is what we want. */
                Not8bit = c;
        case ZPAD:              /* This is what we want. */
                break;
        }
        cancount = 5;
splat:
        switch (c = noxrd7()) {
        case ZPAD:
                goto splat;
        case RCDO:
        case TIMEOUT:
                goto fifi;
        default:
                goto agn2;
        case ZDLE:              /* This is what we want. */
                break;
        }


        Rxhlen = 4;             /* Set default length */
        Rxframeind = c = noxrd7();
        switch (c) {
        case ZVBIN32:
                if ((Rxhlen = c = zdlread()) < 0)
                        goto fifi;
                if (c > ZMAXHLEN)
                        goto agn2;
                Crc32r = 1;  c = zrbhd32(hdr); break;
        case ZBIN32:
                if (Usevhdrs)
                        goto agn2;
                Crc32r = 1;  c = zrbhd32(hdr); break;
        case ZVBINR32:
                if ((Rxhlen = c = zdlread()) < 0)
                        goto fifi;
                if (c > ZMAXHLEN)
                        goto agn2;
                Crc32r = 2;  c = zrbhd32(hdr); break;
        case ZBINR32:
                if (Usevhdrs)
                        goto agn2;
                Crc32r = 2;  c = zrbhd32(hdr); break;
        case RCDO:
        case TIMEOUT:
                goto fifi;
        case ZVBIN:
                if ((Rxhlen = c = zdlread()) < 0)
                        goto fifi;
                if (c > ZMAXHLEN)
                        goto agn2;
```

```
                Crc32r = 0;  c = zrbhdr(hdr); break;
        case ZBIN:
                if (Usevhdrs)
                        goto agn2;
                Crc32r = 0;  c = zrbhdr(hdr); break;
        case ZVHEX:
                if ((Rxhlen = c = zgethex()) < 0)
                        goto fifi;
                if (c > ZMAXHLEN)
                        goto agn2;
                Crc32r = 0;  c = zrhhdr(hdr); break;
        case ZHEX:
                if (Usevhdrs)
                        goto agn2;
                Crc32r = 0;  c = zrhhdr(hdr); break;
        case CAN:
                goto gotcan;
        default:
                goto agn2;
        }
        Rxpos = hdr[ZP3] & 0377;
        Rxpos = (Rxpos<<8) + (hdr[ZP2] & 0377);
        Rxpos = (Rxpos<<8) + (hdr[ZP1] & 0377);
        Rxpos = (Rxpos<<8) + (hdr[ZP0] & 0377);
fifi:
        switch (c) {
        case GOTCAN:
                c = ZCAN;
        /* **** FALL THRU TO **** */
        case ZNAK:
        case ZCAN:
        case ERROR:
        case TIMEOUT:
        case RCDO:
        case GCOUNT:
                zperr("Got %s", frametypes[c+FTOFFSET]);
        /* **** FALL THRU TO **** */
#ifndef DSZ
        default:
                if (c >= -4 && c <= FRTYPES)
                        vfile("zgethdr: %c %d %s %lx", Rxframeind, Rxhlen,
                          frametypes[c+FTOFFSET], Rxpos);
                else
                        vfile("zgethdr: %c %d %lx", Rxframeind, c, Rxpos);
#endif
        }
        /* Use variable length headers if we got one */
        if (c >= 0 && c <= FRTYPES && Rxframeind & 040)
                Usevhdrs = 1;
        return c;
}

/* Receive a binary style header (type and position) */
zrbhdr(hdr)
register char *hdr;
{
        register int c, n;
        register unsigned short crc;

        if ((c = zdlread()) & ~0377)
                return c;
        Rxtype = c;
```

```
	crc = updcrc(c, 0);

	for (n=Rxhlen; --n >= 0; ++hdr) {
		if ((c = zdlread()) & ~0377)
			return c;
		crc = updcrc(c, crc);
		*hdr = c;
	}
	if ((c = zdlread()) & ~0377)
		return c;
	crc = updcrc(c, crc);
	if ((c = zdlread()) & ~0377)
		return c;
	crc = updcrc(c, crc);
	if (crc & 0xFFFF) {
		zperr(badcrc);
		return ERROR;
	}
#ifdef ZMODEM
	Protocol = ZMODEM;
#endif
	Zmodem = 1;
	return Rxtype;
}

/* Receive a binary style header (type and position) with 32 bit FCS */
zrbhd32(hdr)
register char *hdr;
{
	register int c, n;
	register UNSL long crc;

	if ((c = zdlread()) & ~0377)
		return c;
	Rxtype = c;
	crc = 0xFFFFFFFFL; crc = UPDC32(c, crc);
#ifdef DEBUGZ
	vfile("zrbhd32 c=%X  crc=%lX", c, crc);
#endif

	for (n=Rxhlen; --n >= 0; ++hdr) {
		if ((c = zdlread()) & ~0377)
			return c;
		crc = UPDC32(c, crc);
		*hdr = c;
#ifdef DEBUGZ
		vfile("zrbhd32 c=%X  crc=%lX", c, crc);
#endif
	}
	for (n=4; --n >= 0;) {
		if ((c = zdlread()) & ~0377)
			return c;
		crc = UPDC32(c, crc);
#ifdef DEBUGZ
		vfile("zrbhd32 c=%X  crc=%lX", c, crc);
#endif
	}
	if (crc != 0xDEBB20E3) {
		zperr(badcrc);
		return ERROR;
	}
#ifdef ZMODEM
```

```
	Protocol = ZMODEM;
#endif
	Zmodem = 1;
	return Rxtype;
}


/* Receive a hex style header (type and position) */
zrhhdr(hdr)
char *hdr;
{
	register int c;
	register unsigned short crc;
	register int n;

	if ((c = zgethex()) < 0)
		return c;
	Rxtype = c;
	crc = updcrc(c, 0);

	for (n=Rxhlen; --n >= 0; ++hdr) {
		if ((c = zgethex()) < 0)
			return c;
		crc = updcrc(c, crc);
		*hdr = c;
	}
	if ((c = zgethex()) < 0)
		return c;
	crc = updcrc(c, crc);
	if ((c = zgethex()) < 0)
		return c;
	crc = updcrc(c, crc);
	if (crc & 0xFFFF) {
		zperr(badcrc); return ERROR;
	}
	switch ( c = readline(1)) {
	case 0215:
		Not8bit = c;
		/* **** FALL THRU TO **** */
	case 015:
		/* Throw away possible cr/lf */
		switch (c = readline(1)) {
		case 012:
			Not8bit |= c;
		}
	}
#ifdef ZMODEM
	Protocol = ZMODEM;
#endif
	Zmodem = 1; return Rxtype;
}

/* Send a byte as two hex digits */
zputhex(c)
register int c;
{
	static char	digits[]  = "0123456789abcdef";

#ifdef DEBUGZ
	if (Verbose>8)
		vfile("zputhex: %02X", c);
#endif
```

```
	sendline(digits[(c&0xF0)>>4]);
	sendline(digits[(c)&0xF]);
}

/*
 * Send character c with ZMODEM escape sequence encoding.
 *  Escape XON, XOFF. Escape CR following @ (Telenet net escape)
 */
zsendline(c)
{

	/* Quick check for non control characters */
	if (c & 0140)
		xsendline(lastsent = c);
	else {
		switch (c &= 0377) {
		case ZDLE:
			xsendline(ZDLE);
			xsendline (lastsent = (c ^= 0100));
			break;
		case 015:
		case 0215:
			if (!Zctlesc && (lastsent & 0177) != '@')
				goto sendit;
		/* **** FALL THRU TO **** */
		case 020:
		case 021:
		case 023:
		case 0220:
		case 0221:
		case 0223:
			xsendline(ZDLE);
			c ^= 0100;
	sendit:
			xsendline(lastsent = c);
			break;
		default:
			if (Zctlesc && ! (c & 0140)) {
				xsendline(ZDLE);
				c ^= 0100;
			}
			xsendline(lastsent = c);
		}
	}
}

/* Decode two lower case hex digits into an 8 bit byte value */
zgethex()
{
	register int c;

	c = zgeth1();
#ifdef DEBUGZ
	if (Verbose>8)
		vfile("zgethex: %02X", c);
#endif
	return c;
}
zgeth1()
{
	register int c, n;
```

```
	if ((c = noxrd7()) < 0)
		return c;
	n = c - '0';
	if (n > 9)
		n -= ('a' - ':');
	if (n & ~0xF)
		return ERROR;
	if ((c = noxrd7()) < 0)
		return c;
	c -= '0';
	if (c > 9)
		c -= ('a' - ':');
	if (c & ~0xF)
		return ERROR;
	c += (n<<4);
	return c;
}

/*
 * Read a byte, checking for ZMODEM escape encoding
 *  including CAN*5 which represents a quick abort
 */
zdlread()
{
	register int c;

again:
	/* Quick check for non control characters */
	if ((c = readline(Rxtimeout)) & 0140)
		return c;
	switch (c) {
	case ZDLE:
		break;
	case 023:
	case 0223:
	case 021:
	case 0221:
		goto again;
	default:
		if (Zctlesc && !(c & 0140)) {
			goto again;
		}
		return c;
	}
again2:
	if ((c = readline(Rxtimeout)) < 0)
		return c;
	if (c == CAN && (c = readline(Rxtimeout)) < 0)
		return c;
	if (c == CAN && (c = readline(Rxtimeout)) < 0)
		return c;
	if (c == CAN && (c = readline(Rxtimeout)) < 0)
		return c;
	switch (c) {
	case CAN:
		return GOTCAN;
	case ZCRCE:
	case ZCRCG:
	case ZCRCQ:
	case ZCRCW:
		return (c | GOTOR);
	case ZRUB0:
```

```
                return 0177;
        case ZRUB1:
                return 0377;
        case 023:
        case 0223:
        case 021:
        case 0221:
                goto again2;
        default:
                if (Zctlesc && ! (c & 0140)) {
                        goto again2;
                }
                if ((c & 0140) ==  0100)
                        return (c ^ 0100);
                break;
        }
        if (Verbose>1)
                zperr("Bad escape sequence %x", c);
        return ERROR;
}

/*
 * Read a character from the modem line with timeout.
 *  Eat parity, XON and XOFF characters.
 */
noxrd7()
{
        register int c;

        for (;;) {
                if ((c = readline(Rxtimeout)) < 0)
                        return c;
                switch (c &= 0177) {
                case XON:
                case XOFF:
                        continue;
                default:
                        if (Zctlesc && !(c & 0140))
                                continue;
                case '\r':
                case '\n':
                case ZDLE:
                        return c;
                }
        }
}

/* Store long integer pos in Txhdr */
stohdr(pos)
long pos;
{
        Txhdr[ZP0] = pos;
        Txhdr[ZP1] = pos>>8;
        Txhdr[ZP2] = pos>>16;
        Txhdr[ZP3] = pos>>24;
}

/* Recover a long integer from a header */
long
rclhdr(hdr)
register char *hdr;
{
```

```
        register long l;

        l = (hdr[ZP3] & 0377);
        l = (l << 8) | (hdr[ZP2] & 0377);
        l = (l << 8) | (hdr[ZP1] & 0377);
        l = (l << 8) | (hdr[ZP0] & 0377);
        return l;
}

/* End of zm.c */
```

3.3. C-Programmbeispiele der ZMODEM Sende- und Empfangsroutine

```
#define VERSION "sz 3.03 5-09-89"
#define PUBDIR "/usr/spool/uucppublic"

/*% cc -compat -M2 -Ox -K -i -DTXBSIZE=16384  -DNFGVMIN -DREADCHECK sz.c -lx -o sz; size sz

/*% cc -Zi -DXX -DNFGVMIN -DREADCHECK sz.c -lx -o xsz; size xsz
<-xtx-*> cc -Osal -DTXBSIZE=32768  -DSV sz.c -lx -o $B/sz; size $B/sz

 ***************************************************************************
 *
 * sz.c By Chuck Forsberg,  Omen Technology INC
 *
 ***************************************************************************
 *
 * Typical Unix/Xenix/Clone compiles:
 *
 *      cc -O sz.c -o sz          USG (SYS III/V) Unix
 *      cc -O -DSV sz.c -o sz           Sys V Release 2 with non-blocking input
 *                                      Define to allow reverse channel checking
 *      cc -O -DV7  sz.c -o sz          Unix Version 7, 2.8 - 4.3 BSD
 *
 *      cc -O -K -i -DNFGVMIN -DREADCHECK sz.c -lx -o sz    Classic Xenix
 *
 *      ln sz sb                  **** All versions ****
 *      ln sz sx                  **** All versions ****
 *
 ***************************************************************************
 *
 * Typical VMS compile and install sequence:
 *
 *              define LNK$LIBRARY   SYS$LIBRARY:VAXCRTL.OLB
 *              cc sz.c
 *              cc vvmodem.c
 *              link sz,vvmodem
 *      sz :== $disk$user2:[username.subdir]sz.exe
 *
 *  If you feel adventureous, remove the #define BADSEEK line
 *  immediately following the #ifdef vax11c line!  Some VMS
 *  systems know how to fseek, some don't.
 *
 ***************************************************************************
 *
 *
 * A program for Unix to send files and commands to computers running
 *  Professional-YAM, PowerCom, YAM, IMP, or programs supporting Y/XMODEM.
 *
```

```
 *  Sz uses buffered I/O to greatly reduce CPU time compared to UMODEM.
 *
 *  USG UNIX (3.0) ioctl conventions courtesy Jeff Martin
 *
 *      This version implements ZMODEM Run Length Encoding, Comparision,
 *      and variable length headers.  These features were not funded
 *      by the original Telenet development contract.  This software,
 *      including these features, may be freely used for non
 *      commercial and educational purposes.  This software may also
 *      be freely used to support file transfer operations to or from
 *      licensed Omen Technology products.  Contact Omen Technology
 *      for licensing for other uses.  Any programs which use part or
 *      all of this software must be provided in source form with this
 *      notice intact except by written permission from Omen
 *      Technology Incorporated.
 *
 *              Omen Technology Inc             FAX: 503-621-3745
 *              Post Office Box 4681
 *              Portland OR 97208
 *
 *      Previous versions of this program (not containing the extensions
 *      listed above) remain in the public domain.
 *
 *      This code is made available in the hope it will be useful,
 *      BUT WITHOUT ANY WARRANTY OF ANY KIND OR LIABILITY FOR ANY
 *      DAMAGES OF ANY KIND.
 *
 *  2.1x hacks to avoid VMS fseek() bogosity, allow input from pipe
 *     -DBADSEEK -DTXBSIZE=32768
 *  2.x has mods for VMS flavor
 *
 * 1.34 implements tx backchannel garbage count and ZCRCW after ZRPOS
 * in accordance with the 7-31-87 ZMODEM Protocol Description
 */

#ifdef XX
#define XARGSFILE "args"
long Thisflen;
#endif

char *substr(), *getenv();

#ifdef vax11c
#define STATIC
#define BADSEEK
#define TXBSIZE 32768                   /* Must be power of two, < MAXINT */
#include <types.h>
#include <stat.h>
#define STAT
#define LOGFILE "szlog.tmp"
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <ctype.h>
#include <errno.h>
#define OS "VMS"
#define ROPMODE "r"
#define READCHECK
#define BUFWRITE
extern int errno;
#define SS_NORMAL SS$_NORMAL
#define xsendline(c) sendline(c)
```

```
#ifndef PROGNAME
#define PROGNAME "sz"
#endif


#else    /* vax11c */

#ifdef GENIE
#define STATIC static
#define LOGFILE "szlog"
#define BADSEEK
#define TXBSIZE 32768                    /* Must be power of two, < MAXINT */
#define OS "GEnie"
#define SS_NORMAL 0
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <ctype.h>
#include <errno.h>
#include <stdlib.h>
#include <fildes.h>
FILDES fdes;
extern int errno;
int Binfile;
long Thisflen;

#define sendline(c) putchar(c & 0377)
#define xsendline(c) putchar(c)

#else    /* GENIE */

#define LOGFILE "/tmp/szlog"
#define SS_NORMAL 0
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <ctype.h>
#include <errno.h>
extern int errno;
#define STATIC

#define sendline(c) putchar(c & 0377)
#define xsendline(c) putchar(c)

#endif
#endif

#define PATHLEN 256
#define OK 0
#define FALSE 0
#ifdef TRUE
#undef TRUE
#endif
#define TRUE 1
#define ERROR (-1)
/* Ward Christensen / CP/M parameters - Don't change these! */
#define ENQ 005
#define CAN ('X'&037)
#define XOFF ('s'&037)
#define XON ('q'&037)
#define SOH 1
```

```
#define STX 2
#define EOT 4
#define ACK 6
#define NAK 025
#define CPMEOF 032
#define WANTCRC 0103        /* send C not NAK to get crc not checksum */
#define WANTG 0107   /* Send G not NAK to get nonstop batch xmsn */
#define TIMEOUT (-2)
#define RCDO (-3)
#define GCOUNT (-4)
#define RETRYMAX 10


#define HOWMANY 2
STATIC int Zmodem=0;        /* ZMODEM protocol requested by receiver */
unsigned Baudrate=4800;             /* Default, set by first mode() call */
STATIC unsigned Effbaud = 4800;
STATIC unsigned Txwindow;    /* Control the size of the transmitted window */
STATIC unsigned Txwspac;     /* Spacing between zcrcq requests */
STATIC unsigned Txwcnt;      /* Counter used to space ack requests */
STATIC long Lrxpos;          /* Receiver's last reported offset */
STATIC int errors;

#ifdef vax11c
#include "vrzsz.c"       /* most of the system dependent stuff here */
#else
#ifdef GENIE
#include "genie.c"       /* most of the system dependent stuff here */
#else
#include "rbsb.c"        /* most of the system dependent stuff here */
#ifdef XX
#undef STAT
#endif
#endif
#endif

#include "crctab.c"

STATIC int Filesleft;
STATIC long Totalleft;

/*
 * Attention string to be executed by receiver to interrupt streaming data
 *  when an error is detected.  A pause (0336) may be needed before the
 *  ^C (03) or after it.
 */
#ifdef READCHECK
STATIC char Myattn[] = { 0 };
#else
#ifdef USG
STATIC char Myattn[] = { 03, 0336, 0 };
#else
#ifndef GENIE
STATIC char Myattn[] = { 0 };
#endif
#endif
#endif

FILE *in;

#ifdef BADSEEK
STATIC int Canseek = 0;         /* 1: Can seek 0: only rewind -1: neither (pipe) */
```

```
#ifndef TXBSIZE
#define TXBSIZE 16384                   /* Must be power of two, < MAXINT */
#endif
#else
STATIC int Canseek = 1;         /* 1: Can seek 0: only rewind -1: neither (pipe) */
#endif

#ifdef TXBSIZE
#define TXBMASK (TXBSIZE-1)
STATIC char Txb[TXBSIZE];               /* Circular buffer for file reads */
STATIC char *txbuf = Txb;               /* Pointer to current file segment */
#else
STATIC char txbuf[1024];
#endif
STATIC long vpos = 0;                   /* Number of bytes read from file */

STATIC char Lastrx;
STATIC char Crcflg;
STATIC int Verbose=0;
STATIC int Modem2=0;            /* XMODEM Protocol - don't send pathnames */
STATIC int Restricted=0;        /* restricted; no /.. or ../ in filenames */
STATIC int Quiet=0;             /* overrides logic that would otherwise set verbose */
STATIC int Ascii=0;             /* Add CR's for brain damaged programs */
STATIC int Fullname=0;                  /* transmit full pathname */
STATIC int Unlinkafter=0;       /* Unlink file after it is sent */
STATIC int Dottoslash=0;        /* Change foo.bar.baz to foo/bar/baz */
STATIC int firstsec;
STATIC int errcnt=0;            /* number of files unreadable */
STATIC int blklen=128;          /* length of transmitted records */
STATIC int Optiong;             /* Let it rip no wait for sector ACK's */
STATIC int Eofseen;             /* EOF seen on input set by zfilbuf */
STATIC int BEofseen;            /* EOF seen on input set by fooseek */
STATIC int Totsecs;             /* total number of sectors this file */
STATIC int Filcnt=0;            /* count of number of files opened */
STATIC int Lfseen=0;
STATIC unsigned Rxbuflen = 16384;     /* Receiver's max buffer length */
STATIC int Tframlen = 0;        /* Override for tx frame length */
STATIC int blkopt=0;            /* Override value for zmodem blklen */
STATIC int Rxflags = 0;
STATIC long bytcnt;
STATIC int Wantfcs32 = TRUE;/* want to send 32 bit FCS */
STATIC char Lzconv;    /* Local ZMODEM file conversion request */
STATIC char Lzmanag; /* Local ZMODEM file management request */
STATIC int Lskipnocor;
STATIC char Lztrans;
STATIC int Command;             /* Send a command, then exit. */
STATIC char *Cmdstr;            /* Pointer to the command string */
STATIC int Cmdtries = 11;
STATIC int Cmdack1;             /* Rx ACKs command, then do it */
STATIC int Exitcode;
STATIC int Test;                /* 1= Force receiver to send Attn, etc with qbf. */
                        /* 2= Character transparency test */
STATIC char *qbf=
 "The quick brown fox jumped over the lazy dog's back 1234567890\r\n";
STATIC long Lastsync;           /* Last offset to which we got a ZRPOS */
STATIC int Beenhereb4;                  /* How many times we've been ZRPOS'd same place */

STATIC jmp_buf tohere;                  /* For the interrupt on RX timeout */
STATIC jmp_buf intrjmp;         /* For the interrupt on RX CAN */

#ifdef XARGSFILE
char *
```

```
mystrsave(s)
char *s;
{
        register char *p;
        char *malloc();

        if (p = malloc(strlen(s)+1) ) {
                strcpy(p, s); return p;
        }
        fprintf(stderr, "No memory for mystrsave!\n");
        exit(1);
}

/* Remove (presumably) terminating CR and/or LF from string */
uncrlf(s)
register char *s;
{
        for ( ; *s; ++s)
                switch (*s) {
                case '\r':
                case '\n':
                        *s = 0;  return;
                }
}
#endif


/* called by signal interrupt or terminate to clean things up */
bibi(n)
{
        canit(); fflush(stdout); mode(0);
        fprintf(stderr, "sz: caught signal %d; exiting\n", n);
#ifndef GENIE
        if (n == SIGQUIT)
                abort();
#endif
        if (n == 99)
                fprintf(stderr, "mode(2) in rbsb.c not implemented!!\n");
        cucheck();
        exit(128+n);
}
/* Called when ZMODEM gets an interrupt (^X) */
onintr()
{
        signal(SIGINT, SIG_IGN);
        longjmp(intrjmp, -1);
}

STATIC int Zctlesc;       /* Encode control characters */
STATIC int Nozmodem = 0;       /* If invoked as "sb" */
STATIC char *Progname = "sz";
STATIC int Zrwindow = 1400;    /* RX window size (controls garbage count) */
#include "zm.c"

#include "zmr.c"

#ifdef XARGSFILE
#define XARGSMAX 256
char *xargv[XARGSMAX+1];
#endif

main(argc, argv)
```

```
char *argv[];
{
        register char *cp;
        register npats;
        int dm;
        char **patts;
        static char xXbuf[BUFSIZ];

        if ((cp = getenv("ZNULLS")) && *cp)
                Znulls = atoi(cp);
        if ((cp=getenv("SHELL")) && (substr(cp, "rsh") || substr(cp, "rksh")))
                Restricted=TRUE;
        from_cu();
#ifdef vax11c
        chkinvok(PROGNAME);
#else
        chkinvok(argv[0]);
#endif

        Rxtimeout = 600;
        npats=0;
        if (argc<2)
                usage();
        setbuf(stdout, xXbuf);
        while (--argc) {
                cp = *++argv;
                if (*cp++ == '-' && *cp) {
                        while ( *cp) {
                                switch(*cp++) {
                                case '\\':
                                        *cp = toupper(*cp);  continue;
                                case '+':
                                        Lzmanag = ZMAPND; break;
#ifdef CSTOPB
                                case '2':
                                        Twostop = TRUE; break;
#endif
                                case 'a':
                                        Lzconv = ZCNL;
                                        Ascii = TRUE; break;
                                case 'b':
                                        Lzconv = ZCBIN; break;
                                case 'C':
                                        if (--argc < 1) {
                                                usage();
                                        }
                                        Cmdtries = atoi(*++argv);
                                        break;
                                case 'i':
                                        Cmdack1 = ZCACK1;
                                        /* **** FALL THROUGH TO **** */
                                case 'c':
                                        if (--argc != 1) {
                                                usage();
                                        }
                                        Command = TRUE;
                                        Cmdstr = *++argv;
                                        break;
                                case 'd':
                                        ++Dottoslash;
                                        /* **** FALL THROUGH TO **** */
                                case 'f':
```

```
                        Fullname=TRUE; break;
                case 'e':
                        Zctlesc = 1; break;
                case 'k':
                        blklen=1024; break;
                case 'L':
                        if (--argc < 1) {
                                usage();
                        }
                        blkopt = atoi(*++argv);
                        if (blkopt<24 || blkopt>1024)
                                usage();
                        break;
                case 'l':
                        if (--argc < 1) {
                                usage();
                        }
                        Tframlen = atoi(*++argv);
                        if (Tframlen<32 || Tframlen>1024)
                                usage();
                        break;
                case 'N':
                        Lzmanag = ZMNEWL;  break;
                case 'n':
                        Lzmanag = ZMNEW;  break;
                case 'o':
                        Wantfcs32 = FALSE; break;
                case 'p':
                        Lzmanag = ZMPROT;  break;
                case 'r':
                        if (Lzconv == ZCRESUM)
                                Lzmanag = (Lzmanag & ZMMASK) | ZMCRC;
                        Lzconv = ZCRESUM; break;
                case 'q':
                        Quiet=TRUE; Verbose=0; break;
                case 't':
                        if (--argc < 1) {
                                usage();
                        }
                        Rxtimeout = atoi(*++argv);
                        if (Rxtimeout<10 || Rxtimeout>1000)
                                usage();
                        break;
                case 'T':
                        if (++Test > 1) {
                                chartest(1); chartest(2);
                                mode(0);  exit(0);
                        }
                        break;
#ifndef vax11c
                case 'u':
                        ++Unlinkafter; break;
#endif
                case 'v':
                        ++Verbose; break;
                case 'w':
                        if (--argc < 1) {
                                usage();
                        }
                        Txwindow = atoi(*++argv);
                        if (Txwindow < 256)
                                Txwindow = 256;
```

```
                                        Txwindow = (Txwindow/64) * 64;
                                        Txwspac = Txwindow/4;
                                        if (blkopt > Txwspac
                                         || (!blkopt && Txwspac < 1024))
                                                blkopt = Txwspac;
                                        break;
                                case 'X':
                                        ++Modem2; break;
                                case 'Y':
                                        Lskipnocor = TRUE;
                                        /* **** FALLL THROUGH TO **** */
                                case 'y':
                                        Lzmanag = ZMCLOB; break;
                                case 'Z':
                                case 'z':
                                        Lztrans = ZTRLE;  break;
                                default:
                                        usage();
                                }
                        }
                }
                else if ( !npats && argc>0) {
                        if (argv[0][0]) {
                                npats=argc;
                                patts=argv;
                        }
                }
        }
        if (npats < 1 && !Command && !Test)
                usage();
        if (Verbose) {
                if (freopen(LOGFILE, "a", stderr)==NULL) {
                        printf("Can't open log file %s\n",LOGFILE);
                        exit(0200);
                }
                setbuf(stderr, NULL);
        }
        if (Fromcu && !Quiet) {
                if (Verbose == 0)
                        Verbose = 2;
        }
        vfile("%s %s for %s\n", Progname, VERSION, OS);

#ifdef XARGSFILE
        vfile("npats=%d *patts=%s", npats, *patts);
        if (npats == 1 && !strcmp(XARGSFILE, *patts)) {
                in = fopen(XARGSFILE, "r");
                if (!in) {
                        printf(stderr, "Can't open / control file!\n");
                        exit(2);
                }
                for (npats=0,argv=patts=xargv; npats<XARGSMAX; ++npats,++argv) {
                        if (fgets(txbuf, 1024, in) <= 0)
                                break;
                        uncrlf(txbuf);
                        *argv = mystrsave(txbuf);
                }
                fclose(in);
        }
#endif

        mode(1);
```

```
#ifdef GENIE
        signal(SIGINT, SIG_IGN);
#else
        if (signal(SIGINT, bibi) == SIG_IGN) {
                signal(SIGINT, SIG_IGN); signal(SIGKILL, SIG_IGN);
        } else {
                signal(SIGINT, bibi); signal(SIGKILL, bibi);
        }
#endif
#ifdef SIGQUIT
        if ( !Fromcu)
                signal(SIGQUIT, SIG_IGN);
#endif
#ifdef SIGTERM
        signal(SIGTERM, bibi);
#endif

        if ( !Modem2) {
                if (!Nozmodem) {
                        printf("rz\r");  fflush(stdout);
                }
                countem(npats, patts);
                if (!Nozmodem) {
                        stohdr(0L);
                        if (Command)
                                Txhdr[ZF0] = ZCOMMAND;
                        zshhdr(4, ZRQINIT, Txhdr);
                }
        }
        fflush(stdout);

        if (Command) {
                if (getzrxinit()) {
                        Exitcode=0200; canit();
                }
                else if (zsendcmd(Cmdstr, 1+strlen(Cmdstr))) {
                        Exitcode=0200; canit();
                }
        } else if (wcsend(npats, patts)==ERROR) {
                Exitcode=0200;
                canit();
        }
        fflush(stdout);
        mode(0);
        dm = ((errcnt != 0) | Exitcode);
        if (dm) {
                cucheck();  exit(dm);
        }
        exit(SS_NORMAL);
        /*NOTREACHED*/
}

wcsend(argc, argp)
char *argp[];
{
        register n;

        Crcflg=FALSE;
        firstsec=TRUE;
        bytcnt = -1;
        if (Nozmodem) {
```

```
                printf("Start your YMODEM receive. ");  fflush(stdout);
        }
        for (n=0; n<argc; ++n) {
                Totsecs = 0;
                if (wcs(argp[n])==ERROR)
                        return ERROR;
        }
        Totsecs = 0;
        if (Filcnt==0) {   /* bitch if we couldn't open ANY files */
                if (!Nozmodem && !Modem2) {
                        Command = TRUE;
                        Cmdstr = "echo \"sz: Can't open any requested files\"";
                        if (getnak()) {
                                Exitcode=0200; canit();
                        }
                        if (!Zmodem)
                                canit();
                        else if (zsendcmd(Cmdstr, 1+strlen(Cmdstr))) {
                                Exitcode=0200; canit();
                        }
                        Exitcode = 1; return OK;
                }
                canit();
                fprintf(stderr,"\r\nCan't open any requested files.\r\n");
                return ERROR;
        }
        if (Zmodem)
                saybibi();
        else if ( !Modem2)
                wctxpn("");
        return OK;
}

wcs(oname)
char *oname;
{
        register c;
        register char *p, *q;
#ifdef STAT
        struct stat f;
#endif
        char name[PATHLEN];

        strcpy(name, oname);

#ifdef XARGSFILE
        /* Parse GEniename:REALname:length pathname syntax */
        Thisflen = -1;
        for (p = oname; *p; ++p) {
                if (*p == ':') {
                        *p++ = 0;
                        q = p;
                        for (++p; *p; ++p) {
                                if (*p == ':') {
                                        *p++ = 0;
                                        Thisflen = atol(p);
                                        break;
                                }
                        }
                        strcpy(name, q);
                        break;
                }
```

```
        }
#endif

#ifdef GENIE
        _describe(oname,&fdes);                 /* An undocumented goodie */
        if (fdes.type_file == 1) {  /* Fortran Sequential Binary */
                Binfile = 1;
                in = fopen(oname,"rB");
        }
        else if (fdes.type_file == 0)  { /* Ascii */
                Binfile = 0;
                in = fopen(oname,"r");
        }
        else {                                  /* not a SL filetype */
                fprintf(stderr, "\nUnknown file type %d\n",fdes.type_file);
                ++errcnt;
                return OK;                      /* pass over it, there may be others */
        }
#else
        if (Restricted) {
                /* restrict pathnames to current tree or uucppublic */
                if ( substr(name, "../")
                 || (name[0]== '/' && strncmp(name, PUBDIR, strlen(PUBDIR))) ) {
                        canit();
                        fprintf(stderr,"\r\nsz:\tSecurity Violation\r\n");
                        return ERROR;
                }
        }

        in=fopen(oname, ROPMODE);
#endif

        if (in==NULL) {
                ++errcnt;
                return OK;      /* pass over it, there may be others */
        }
        BEofseen = Eofseen = 0;  vpos = 0;

#ifdef STAT
        /* Check for directory or block special files */
        fstat(fileno(in), &f);
        c = f.st_mode & S_IFMT;
        if (c == S_IFDIR || c == S_IFBLK) {
                fclose(in);
                return OK;
        }
#endif

        ++Filcnt;
        switch (wctxpn(name)) {
        case ERROR:
                return ERROR;
        case ZSKIP:
                return OK;
        }
#ifdef STAT
        if (!Zmodem && wctx(f.st_size)==ERROR)
                return ERROR;
#else
        if (!Zmodem && wctx(1000000000L)==ERROR)
                return ERROR;
#endif
```

```
#ifndef vax11c
#ifndef GENIE
        if (Unlinkafter)
                unlink(oname);
#endif
#endif

        return 0;
}

/*
 * generate and transmit pathname block consisting of
 *  pathname (null terminated),
 *  file length, mode time and file mode in octal
 *  as provided by the Unix fstat call.
 *  N.B.: modifies the passed name, may extend it!
 */
wctxpn(name)
char *name;
{
        register char *p, *q;
        char name2[PATHLEN];
#ifdef STAT
        struct stat f;
#endif

        if (Modem2) {
#ifdef STAT
                if (*name && fstat(fileno(in), &f)!= -1) {
                        fprintf(stderr, "Sending %s, %ld XMODEM blocks. ",
                          name, (127+f.st_size)>>7);
                }
#endif
                fprintf(stderr, "Give your local XMODEM receive command now.\r\n");
                fflush(stderr);
                return OK;
        }
        zperr("Awaiting pathname nak for %s", *name?name:"<END>");
        if ( !Zmodem)
                if (getnak())
                        return ERROR;

        q = (char *) 0;
        if (Dottoslash) {               /* change . to . */
                for (p=name; *p; ++p) {
                        if (*p == '/')
                                q = p;
                        else if (*p == '.')
                                *(q=p) = '/';
                }
                if (q && strlen(++q) > 8) {          /* If name>8 chars */
                        q += 8;                      /*   make it .ext */
                        strcpy(name2, q);            /* save excess of name */
                        *q = '.';
                        strcpy(++q, name2);          /* add it back */
                }
        }

        for (p=name, q=txbuf ; *p; )
                if ((*q++ = *p++) == '/' && !Fullname)
                        q = txbuf;
```

```
        *q++ = 0;
        p=q;
        while (q < (txbuf + 1024))
                *q++ = 0;
        if (*name) {
#ifdef XX
                if (Thisflen >= 0) {
                        sprintf(p, "%u 0 0 0 %d %ld",
                          Thisflen, Filesleft, Totalleft);
                        Totalleft -= Thisflen;
                }
#endif
#ifdef GENIE
                else
                        sprintf(p, "%d", fdes.current_file_size * 1260);
                vfile("%s open Binfile=%d size=%ld", name, Binfile,
                  fdes.current_file_size * 1260);
#endif

#ifdef STAT
#ifndef XX
                if (fstat(fileno(in), &f)!= -1)
                        sprintf(p, "%lu %lo %o 0 %d %ld", f.st_size, f.st_mtime,
                          f.st_mode, Filesleft, Totalleft);
                Totalleft -= f.st_size;
#endif
#endif

        }
        if (--Filesleft <= 0)
                Totalleft = 0;
        if (Totalleft < 0)
                Totalleft = 0;

#ifdef STAT
        /* force 1k blocks if name won't fit in 128 byte block */
        if (txbuf[125])
                blklen=1024;
        else {          /* A little goodie for IMP/KMD */
                txbuf[127] = (f.st_size + 127) >>7;
                txbuf[126] = (f.st_size + 127) >>15;
        }
#endif
        if (Zmodem)
                return zsendfile(txbuf, 1+strlen(p)+(p-txbuf));
        if (wcputsec(txbuf, 0, 128)==ERROR)
                return ERROR;
        return OK;
}

getnak()
{
        register firstch;

        Lastrx = 0;
        for (;;) {
                switch (firstch = readline(800)) {
                case ZPAD:
                        if (getzrxinit())
                                return ERROR;
                        Ascii = 0;      /* Receiver does the conversion */
                        return FALSE;
```

```
                case TIMEOUT:
                        zperr("Timeout on pathname");
                        return TRUE;
                case WANTG:
#ifdef MODE2OK
                        mode(2);        /* Set cbreak, XON/XOFF, etc. */
#endif
                        Optiong = TRUE;
                        blklen=1024;
                case WANTCRC:
                        Crcflg = TRUE;
                case NAK:
                        return FALSE;
                case CAN:
                        if ((firstch = readline(20)) == CAN && Lastrx == CAN)
                                return TRUE;
                default:
                        break;
                }
                Lastrx = firstch;
        }
}


wctx(flen)
long flen;
{
        register int thisblklen;
        register int sectnum, attempts, firstch;
        long charssent;

        charssent = 0;  firstsec=TRUE;  thisblklen = blklen;
        vfile("wctx:file length=%ld", flen);

        while ((firstch=readline(Rxtimeout))!=NAK && firstch != WANTCRC
          && firstch != WANTG && firstch!=TIMEOUT && firstch!=CAN)
                ;
        if (firstch==CAN) {
                zperr("Receiver CANcelled");
                return ERROR;
        }
        if (firstch==WANTCRC)
                Crcflg=TRUE;
        if (firstch==WANTG)
                Crcflg=TRUE;
        sectnum=0;
        for (;;) {
                if (flen <= (charssent + 896L))
                        thisblklen = 128;
                if ( !filbuf(txbuf, thisblklen))
                        break;
                if (wcputsec(txbuf, ++sectnum, thisblklen)==ERROR)
                        return ERROR;
                charssent += thisblklen;
        }
        fclose(in);
        attempts=0;
        do {
                purgeline();
                sendline(EOT);
                flushmo();
                ++attempts;
```

```
	}
		while ((firstch=(readline(Rxtimeout)) != ACK) && attempts < RETRYMAX);
	if (attempts == RETRYMAX) {
		zperr("No ACK on EOT");
		return ERROR;
	}
	else
		return OK;
}

wcputsec(buf, sectnum, cseclen)
char *buf;
int sectnum;
int cseclen;	/* data length of this sector to send */
{
	register checksum, wcj;
	register char *cp;
	unsigned oldcrc;
	int firstch;
	int attempts;

	firstch=0;		/* part of logic to detect CAN CAN */

	if (Verbose>2)
		fprintf(stderr, "Sector %3d %2dk\n", Totsecs, Totsecs/8 );
	else if (Verbose>1)
		fprintf(stderr, "\rSector %3d %2dk ", Totsecs, Totsecs/8 );
	for (attempts=0; attempts <= RETRYMAX; attempts++) {
		Lastrx= firstch;
		sendline(cseclen==1024?STX:SOH);
		sendline(sectnum);
		sendline(-sectnum -1);
		oldcrc=checksum=0;
		for (wcj=cseclen,cp=buf; --wcj>=0; ) {
			sendline(*cp);
			oldcrc=updcrc((0377& *cp), oldcrc);
			checksum += *cp++;
		}
		if (Crcflg) {
			oldcrc=updcrc(0,updcrc(0,oldcrc));
			sendline((int)oldcrc>>8);
			sendline((int)oldcrc);
		}
		else
			sendline(checksum);
		flushmo();

		if (Optiong) {
			firstsec = FALSE; return OK;
		}
		firstch = readline(Rxtimeout);
gotnak:
		switch (firstch) {
		case CAN:
			if(Lastrx == CAN) {
cancan:
				zperr("Cancelled");  return ERROR;
			}
			break;
		case TIMEOUT:
			zperr("Timeout on sector ACK"); continue;
		case WANTCRC:
```

```
                        if (firstsec)
                                Crcflg = TRUE;
                case NAK:
                        zperr("NAK on sector"); continue;
                case ACK:
                        firstsec=FALSE;
                        Totsecs += (cseclen>>7);
                        return OK;
                case ERROR:
                        zperr("Got burst for sector ACK"); break;
                default:
                        zperr("Got %02x for sector ACK", firstch); break;
                }
                for (;;) {
                        Lastrx = firstch;
                        if ((firstch = readline(Rxtimeout)) == TIMEOUT)
                                break;
                        if (firstch == NAK || firstch == WANTCRC)
                                goto gotnak;
                        if (firstch == CAN && Lastrx == CAN)
                                goto cancan;
                }
        }
        zperr("Retry Count Exceeded");
        return ERROR;
}

/* fill buf with count chars padding with ^Z for CPM */
filbuf(buf, count)
register char *buf;
{
        register c, m;

        if ( !Ascii) {
                m = read(fileno(in), buf, count);
                if (m <= 0)
                        return 0;
                while (m < count)
                        buf[m++] = 032;
                return count;
        }
        m=count;
        if (Lfseen) {
                *buf++ = 012; --m; Lfseen = 0;
        }
        while ((c=getc(in))!=EOF) {
                if (c == 012) {
                        *buf++ = 015;
                        if (--m == 0) {
                                Lfseen = TRUE; break;
                        }
                }
                *buf++ =c;
                if (--m == 0)
                        break;
        }
        if (m==count)
                return 0;
        else
                while (--m>=0)
                        *buf++ = CPMEOF;
        return count;
```

```
}

/* Fill buffer with blklen chars */
zfilbuf()
{
        int n;

#ifdef TXBSIZE
        vfile("zfilbuf: bytcnt =%lu vpos=%lu blklen=%d", bytcnt, vpos, blklen);
        /* We assume request is within buffer, or just beyond */
        txbuf = Txb + (bytcnt & TXBMASK);
        if (vpos <= bytcnt) {
#ifdef GENIE
                if (Binfile) {
                        long l, m;  char *p;

                        for (p=txbuf, n=0, l=blklen;  l;  l -= 128, p+= 128) {
                                n += m = fgetb(p, 128, in);
                                if (m == 0)
                                        break;
                        }
                } else
#endif
                n = fread(txbuf, 1, blklen, in);

                vpos += n;
                if (n < blklen)
                        Eofseen = 1;
                vfile("zfilbuf: n=%d vpos=%lu Eofseen=%d", n, vpos, Eofseen);
                return n;
        }
        if (vpos >= (bytcnt+blklen))
                return blklen;
        /* May be a short block if crash recovery etc. */
        Eofseen = BEofseen;
        return (vpos - bytcnt);
#else
        n = fread(txbuf, 1, blklen, in);
        if (n < blklen)
                Eofseen = 1;
        return n;
#endif
}

#ifdef TXBSIZE
/* Replacement for brain damaged fseek function.  Returns 0==success */
fooseek(fptr, pos, whence)
FILE *fptr;
long pos;
{
        long m, n;
#ifdef GENIE
        long l, k;  char *p;
#endif

        vfile("fooseek: pos =%lu vpos=%lu Canseek=%d", pos, vpos, Canseek);
        /* Seek offset < current buffer */
        if (pos < (vpos -TXBSIZE +1024)) {
                BEofseen = 0;
                if (Canseek > 0) {
                        vpos = pos & ~TXBMASK;
                        if (vpos >= pos)
```

```
                        vpos -= TXBSIZE;
                if (fseek(fptr, vpos, 0))
                        return 1;
        }
        else if (Canseek == 0) {
#ifdef GENIE
                if (Binfile) {
                        if (fseekb(fptr, vpos = 0L, 0))
                                return 1;
                        } else
#endif
                if (fseek(fptr, vpos = 0L, 0))
                        return 1;
        } else
                return 1;
        while (vpos < pos) {
#ifdef GENIE
                if (Binfile) {
                        for (p=Txb,n=0,l=TXBSIZE; l; l -= 128,p+= 128) {
                                n += (k = fgetb(p, 128, fptr));
                                vfile("bsk1: l=%d k=%d", l, k);
                                if (k == 0)
                                        break;
                        }
                } else
#endif
                n = fread(Txb, 1, TXBSIZE, fptr);
                vpos += n;
                vfile("n=%d vpos=%ld", n, vpos);
                if (n < TXBSIZE) {
                        BEofseen = 1;
                        break;
                }
        }
        vfile("vpos=%ld", vpos);
        return 0;
}
/* Seek offset > current buffer (Crash Recovery, etc.) */
if (pos > vpos) {
        if (Canseek)
                if (fseek(fptr, vpos = (pos & ~TXBMASK), 0))
                        return 1;
        while (vpos <= pos) {
                txbuf = Txb + (vpos & TXBMASK);
                m = TXBSIZE - (vpos & TXBMASK);
                vfile("m=%ld vpos=%ld", m,vpos);
#ifdef GENIE
                if (Binfile) {
                        for (p=txbuf,n=0,l=m; l; l -= 128,p+= 128) {
                                n += (k = fgetb(p, 128, fptr));
                                vfile("bsk2: l=%d k=%d n=%d", l, k, n);
                                if (k == 0)
                                        break;
                        }
                } else
#endif
                        n = fread(txbuf, 1, m, fptr);
                vfile("n=%ld vpos=%ld", n,vpos);
                vpos += n;
                vfile("bo=%d m=%ld vpos=%ld", txbuf-Txb,m,vpos);
                if (n < m) {
                        BEofseen = 1;
```

```
                                        break;
                                }
                        }
                        return 0;
                }
                /* Seek offset is within current buffer */
                vfile("within buffer: vpos=%ld", vpos);
                return 0;
}
#define fseek fooseek
#endif


/* VARARGS1 */
vfile(f, a, b, c, d)
register char *f;
{
        if (Verbose > 2) {
                fprintf(stderr, f, a, b, c, d);
                fprintf(stderr, "\n");
        }
}


alrm()
{
        longjmp(tohere, -1);
}


#ifndef GENIE
#ifndef vax11c
/*
 * readline(timeout) reads character(s) from file descriptor 0
 * timeout is in tenths of seconds
 */
readline(timeout)
{
        register int c;
        static char byt[1];

        fflush(stdout);
        if (setjmp(tohere)) {
                zperr("TIMEOUT");
                return TIMEOUT;
        }
        c = timeout/10;
        if (c<2)
                c=2;
        if (Verbose>5) {
                fprintf(stderr, "Timeout=%d Calling alarm(%d) ", timeout, c);
        }
        signal(SIGALRM, alrm); alarm(c);
        c=read(0, byt, 1);
        alarm(0);
        if (Verbose>5)
                fprintf(stderr, "ret %x\n", byt[0]);
        if (c<1)
                return TIMEOUT;
        return (byt[0]&0377);
}
```

```
flushmo()
{
	fflush(stdout);
}


purgeline()
{
#ifdef USG
	ioctl(0, TCFLSH, 0);
#else
	lseek(0, 0L, 2);
#endif
}
#endif
#endif

/* send cancel string to get the other end to shut up */
canit()
{
	static char canistr[] = {
	 24,24,24,24,24,24,24,24,24,8,8,8,8,8,8,8,8,8,0
	};

#ifdef vax11c
	raw_wbuf(strlen(canistr), canistr);
	purgeline();
#else
	printf(canistr);
	fflush(stdout);
#endif
}


/*
 * Log an error
 */
/*VARARGS1*/
zperr(s,p,u)
char *s, *p, *u;
{
	if (Verbose <= 0)
		return;
	fprintf(stderr, "Retry %d: ", errors);
	fprintf(stderr, s, p, u);
	fprintf(stderr, "\n");
}

/*
 * substr(string, token) searches for token in string s
 * returns pointer to token within string if found, NULL otherwise
 */
char *
substr(s, t)
register char *s,*t;
{
	register char *ss,*tt;
	/* search for first char of token */
	for (ss=s; *s; s++)
		if (*s == *t)
			/* compare token with substring */
			for (ss=s,tt=t; ;) {
```

```
                                        if (*tt == 0)
                                                return s;
                                        if (*ss++ != *tt++)
                                                break;
                                }
        return NULL;
}

char *babble[] = {
#ifdef vax11c
        "Send file(s) with ZMODEM/YMODEM/XMODEM Protocol",
        "       (Y) = Option applies to YMODEM only",
        "       (Z) = Option applies to ZMODEM only",
        "Usage:sz [-2+abdefkLlNnquvwYy] [-] file ...",
        "       sz [-2Ceqv] -c COMMAND",
        "       \\ Force next option letter to upper case",
        "       sb [-2adfkquv] [-] file ...",
        "       sx [-2akquv] [-] file",
#endif
#ifndef vax11c
        "Send file(s) with ZMODEM/YMODEM/XMODEM Protocol",
        "       (Y) = Option applies to YMODEM only",
        "       (Z) = Option applies to ZMODEM only",
        "Usage:sz [-2+abdefkLlNnquvwYy] [-] file ...",
        "       sz [-2Ceqv] -c COMMAND",
        "       sb [-2adfkquv] [-] file ...",
        "       sx [-2akquv] [-] file",
#endif
#ifdef CSTOPB
        "       2   Use 2 stop bits",
#endif
        "       +   Append to existing destination file (Z)",
        "       a   (ASCII) change NL to CR/LF",
        "       b   Binary file transfer override",
        "       c   send COMMAND (Z)",
#ifndef vax11c
        "       d   Change '.' to '/' in pathnames (Y/Z)",
#endif
        "       e   Escape all control characters (Z)",
        "       f   send Full pathname (Y/Z)",
        "       i   send COMMAND, ack Immediately (Z)",
        "       k   Send 1024 byte packets (Y)",
        "       L N Limit subpacket length to N bytes (Z)",
        "       l N Limit frame length to N bytes (l>=L) (Z)",
        "       n   send file only if source newer (Z)",
        "       N   send file only if source newer or longer (Z)",
        "       o   Use 16 bit CRC instead of 32 bit CRC (Z)",
        "       p   Protect existing destination file (Z)",
        "       r   Resume/Recover interrupted file transfer (Z)",
        "       q   Quiet (no progress reports)",
#ifndef vax11c
        "       u   Unlink (remove) file after transmission",
#endif
        "       v   Verbose - provide debugging information",
        "       w N restrict Window to N bytes (Z)",
        "       Y   Yes, overwrite existing file, skip if not present at rx (Z)",
        "       y   Yes, overwrite existing file (Z)",
        "       Z   Activate ZMODEM compression(Z)",
        ""
};

usage()
```

```
{
        char **pp;

        for (pp=babble; **pp; ++pp)
                fprintf(stderr, "%s\n", *pp);
        fprintf(stderr, "%s for %s by Chuck Forsberg, Omen Technology INC\n",
         VERSION, OS);
        fprintf(stderr, "\t\t\042The High Reliability Software\042\n");
        cucheck();
        exit(SS_NORMAL);
}

/*
 * Get the receiver's init parameters
 */
getzrxinit()
{
        register n;
#ifdef STAT
        struct stat f;
#endif

        for (n=10; --n>=0; ) {

                switch (zgethdr(Rxhdr, 1)) {
                case ZCHALLENGE:     /* Echo receiver's challenge numbr */
                        stohdr(Rxpos);
                        zshhdr(4, ZACK, Txhdr);
                        continue;
                case ZCOMMAND:                  /* They didn't see out ZRQINIT */
                        stohdr(0L);
                        zshhdr(4, ZRQINIT, Txhdr);
                        continue;
                case ZRINIT:
                        Rxflags = 0377 & Rxhdr[ZF0];
                        Usevhdrs = Rxhdr[ZF1] & CANVHDR;
                        Txfcs32 = (Wantfcs32 && (Rxflags & CANFC32));
                        Zctlesc |= Rxflags & TESCCTL;
                        Rxbuflen = (0377 & Rxhdr[ZP0])+((0377 & Rxhdr[ZP1])<<8);
                        if ( !(Rxflags & CANFDX))
                                Txwindow = 0;
                        vfile("Rxbuflen=%d Tframlen=%d", Rxbuflen, Tframlen);
                        if ( !Fromcu)
                                signal(SIGINT, SIG_IGN);
#ifdef MODE2OK
                        mode(2);          /* Set cbreak, XON/XOFF, etc. */
#endif

#ifndef READCHECK
#ifndef USG
#ifndef GENIE
                        /* Use 1024 byte frames if no sample/interrupt */
                        if (Rxbuflen < 32 || Rxbuflen > 1024) {
                                Rxbuflen = 1024;
                                vfile("Rxbuflen=%d", Rxbuflen);
                        }
#endif
#endif
#endif

                        /* Override to force shorter frame length */
                        if (Rxbuflen && (Rxbuflen>Tframlen) && (Tframlen>=32))
```

```
                        Rxbuflen = Tframlen;
                if ( !Rxbuflen && (Tframlen>=32) && (Tframlen<=1024))
                        Rxbuflen = Tframlen;
                vfile("Rxbuflen=%d", Rxbuflen);

#ifndef GENIE
#ifndef vax11c
#ifdef STAT
                /* If using a pipe for testing set lower buf len */
                fstat(0, &f);
                if ((f.st_mode & S_IFMT) != S_IFCHR) {
                        Rxbuflen = 1024;
                }
#endif
#endif
#endif

#ifdef BADSEEK
#ifdef GENIE
                if (Txwindow == 0) {
                        Txwspac = (Txwindow = 4096)/4;
                }
#else
                if (Txwindow == 0)
                        Txwindow = TXBSIZE - 1024;
                Txwspac = TXBSIZE/4;
#endif
                Canseek = 0;
#endif

                /*
                 * If input is not a regular file, force ACK's to
                 *  prevent running beyond the buffer limits
                 */
#ifdef STAT
                if ( !Command) {
                        fstat(fileno(in), &f);
                        if ((f.st_mode & S_IFMT) != S_IFREG) {
                                Canseek = -1;
#ifdef TXBSIZE
                                Txwindow = TXBSIZE - 1024;
                                Txwspac = TXBSIZE/4;
#else
                                return ERROR;
#endif
                        }
                }
#endif

                /* Set initial subpacket length */
                if (blklen < 1024) {        /* Command line override? */
                        if (Effbaud > 300)
                                blklen = 256;
                        if (Effbaud > 1200)
                                blklen = 512;
                        if (Effbaud > 2400)
                                blklen = 1024;
                }
                if (Rxbuflen && blklen>Rxbuflen)
                        blklen = Rxbuflen;
                if (blkopt && blklen > blkopt)
                        blklen = blkopt;
```

```
#ifdef GENIE
                        blklen /= 128;  blklen *= 128;
                        if (blklen < 128)
                                blklen = 128;
#endif
                        vfile("Rxbuflen=%d blklen=%d", Rxbuflen, blklen);
                        vfile("Txwindow = %u Txwspac = %d", Txwindow, Txwspac);


                        if (Lztrans == ZTRLE && (Rxflags & CANRLE))
                                Txfcs32 = 2;
                        else
                                Lztrans = 0;

                        return (sendzsinit());
                case ZCAN:
                case TIMEOUT:
                        return ERROR;
                case ZRQINIT:
                        if (Rxhdr[ZF0] == ZCOMMAND)
                                continue;
                default:
                        zshhdr(4, ZNAK, Txhdr);
                        continue;
                }
        }
        return ERROR;
}

/* Send send-init information */
sendzsinit()
{
        register c;

        if (Myattn[0] == '\0' && (!Zctlesc || (Rxflags & TESCCTL)))
                return OK;
        errors = 0;
        for (;;) {
                stohdr(0L);
#ifdef ALTCANOFF
                Txhdr[ALTCOFF] = ALTCANOFF;
#endif
                if (Zctlesc) {
                        Txhdr[ZF0] |= TESCCTL; zshhdr(4, ZSINIT, Txhdr);
                }
                else
                        zsbhdr(4, ZSINIT, Txhdr);
                zsdata(Myattn, ZATTNLEN, ZCRCW);
                c = zgethdr(Rxhdr, 1);
                switch (c) {
                case ZCAN:
                        return ERROR;
                case ZACK:
                        return OK;
                default:
                        if (++errors > 19)
                                return ERROR;
                        continue;
                }
        }
}
```

```
/* Send file name and related info */
zsendfile(buf, blen)
char *buf;
{
        register c;
        register UNSL long crc;
        long lastcrcrq = -1;
        char *p;

        for (;;) {
                Txhdr[ZF0] = Lzconv;    /* file conversion request */
                Txhdr[ZF1] = Lzmanag; /* file management request */
                if (Lskipnocor)
                        Txhdr[ZF1] |= ZMSKNOLOC;
                Txhdr[ZF2] = Lztrans;    /* file transport request */
                Txhdr[ZF3] = 0;
                zsbhdr(4, ZFILE, Txhdr);
                zsdata(buf, blen, ZCRCW);
again:
                c = zgethdr(Rxhdr, 1);
                switch (c) {
                case ZRINIT:
                        while ((c = readline(50)) > 0)
                                if (c == ZPAD) {
                                        goto again;
                                }
                        /* **** FALL THRU TO **** */
                default:
                        continue;
                case ZCAN:
                case TIMEOUT:
                case ZABORT:
                case ZFIN:
                        return ERROR;
                case ZCRC:
                        if (Rxpos != lastcrcrq) {
                                lastcrcrq = Rxpos;
                                crc = 0xFFFFFFFFL;
                                if (Canseek >= 0) {
                                        fseek(in, 0L, 0);
                                        while (((c = getc(in)) != EOF) && --lastcrcrq)
                                                crc = UPDC32(c, crc);
                                        crc = ~crc;
                                        clearerr(in);       /* Clear possible EOF */
                                        lastcrcrq = Rxpos;
                                }
                        }
                        stohdr(crc);
                        zsbhdr(4, ZCRC, Txhdr);
                        goto again;
                case ZSKIP:
                        fclose(in); return c;
                case ZRPOS:
                        /*
                         * Suppress zcrcw request otherwise triggered by
                         * lastyunc==bytcnt
                         */
#ifdef GENIE
                        /*
                         *  Special case - turn on RLE if not archive, etc.
                         *   otherwise turn off RLE unless cmd line specified
                         */
```

```
                        if (Rxflags & CANRLE) {                 /* RX can do it */
                                bytcnt = 0;
                                zfilbuf();
                                vfile("txbuf012: %x %x %x", txbuf[0], txbuf[1],
                                  txbuf[2]);
                                if ((txbuf[0] != 032)       /* .ARC file */
                                 && (txbuf[0] != 0x1f)      /* .Z file */
                                 && (txbuf[0] != 0x1c)      /* .LHZ file */
                                 && strncmp(txbuf, "ZOO", 3)
                                 && strncmp(txbuf, "GIF", 3)
                                 && (txbuf[2] != 3))        /* .ZIP file */
                                        Txfcs32 = 2;
                                else if ( !(Lztrans & ZTRLE))
                                        Txfcs32 = 1;
                        }
                        /* GEnie binary can't seek to byte */
                        if (Binfile) {
                                Rxpos &= ~127L;
                        }
#endif
                        if (fseek(in, Rxpos, 0))
                                return ERROR;
                        Lastsync = (bytcnt = Txpos = Lrxpos = Rxpos) -1;
                        return zsendfdata();
                }
        }
}

/* Send the data in the file */
zsendfdata()
{
        register c, e, n;
        register newcnt;
        register long tcount = 0;
        int junkcount;          /* Counts garbage chars received by TX */
        static int tleft = 6;   /* Counter for test mode */

        junkcount = 0;
        Beenhereb4 = FALSE;
somemore:
        if (setjmp(intrjmp)) {
waitack:
                junkcount = 0;
                c = getinsync(0);
gotack:
                switch (c) {
                default:
                case ZCAN:
                        fclose(in);
                        return ERROR;
                case ZSKIP:
                        fclose(in);
                        return c;
                case ZACK:
                case ZRPOS:
                        break;
                case ZRINIT:
                        return OK;
                }
#ifdef READCHECK
                /*
                 * If the reverse channel can be tested for data,
```

```
                 *  this logic may be used to detect error packets
                 *  sent by the receiver, in place of setjmp/longjmp
                 *  rdchk(fd) returns non 0 if a character is available
                 */
                while (rdchk(0)) {
#ifdef EATSIT
                        switch (checked)
#else
                        switch (readline(1))
#endif
                        {
                        case CAN:
                        case ZPAD:
                                c = getinsync(1);
                                goto gotack;
                        case XOFF:              /* Wait a while for an XON */
                        case XOFF|0200:
                                readline(100);
                        }
                }
#endif
        }

        if ( !Fromcu)
                signal(SIGINT, onintr);
        newcnt = Rxbuflen;
        Txwcnt = 0;
        stohdr(Txpos);
        zsbhdr(4, ZDATA, Txhdr);

        /*
         * Special testing mode.  This should force receiver to Attn,ZRPOS
         *  many times.  Each time the signal should be caught, causing the
         *  file to be started over from the beginning.
         */
        if (Test) {
                if ( --tleft)
                        while (tcount < 20000) {
                                printf(qbf); fflush(stdout);
                                tcount += strlen(qbf);
#ifdef READCHECK
                                while (rdchk(0)) {
#ifdef EATSIT
                                        switch (checked)
#else
                                        switch (readline(1))
#endif
                                        {
                                        case CAN:
                                        case ZPAD:
#ifdef TCFLSH
                                                ioctl(0, TCFLSH, 1);
#endif
                                                goto waitack;
                                        case XOFF:      /* Wait for XON */
                                        case XOFF|0200:
                                                readline(100);
                                        }
                                }
#endif
                        }
                signal(SIGINT, SIG_IGN); canit();
```

```
                sleep(3); purgeline(); mode(0);
                printf("\nsz: Tcount = %ld\n", tcount);
                if (tleft) {
                        printf("ERROR: Interrupts Not Caught\n");
                        exit(1);
                }
                exit(SS_NORMAL);
        }

        do {
                n = zfilbuf();
                if (Eofseen)
                        e = ZCRCE;
                else if (junkcount > 3)
                        e = ZCRCW;
                else if (bytcnt == Lastsync)
                        e = ZCRCW;
                else if (Rxbuflen && (newcnt -= n) <= 0)
                        e = ZCRCW;
                else if (Txwindow && (Txwcnt += n) >= Txwspac) {
                        Txwcnt = 0;  e = ZCRCQ;
                } else
                        e = ZCRCG;
                if (Verbose>1)
                        fprintf(stderr, "\r%7ld ZMODEM%s    ",
                          Txpos, Crc32t?" CRC-32":"");
                zsdata(txbuf, n, e);
                bytcnt = Txpos += n;
                if (e == ZCRCW)
                        goto waitack;
#ifdef READCHECK
                /*
                 * If the reverse channel can be tested for data,
                 *  this logic may be used to detect error packets
                 *  sent by the receiver, in place of setjmp/longjmp
                 *  rdchk(fd) returns non 0 if a character is available
                 */
                fflush(stdout);
                while (rdchk(0)) {
#ifdef EATSIT
                        switch (checked)
#else
                        switch (readline(1))
#endif
                        {
                        case CAN:
                        case ZPAD:
                                c = getinsync(1);
                                if (c == ZACK)
                                        break;
#ifdef TCFLSH
                                ioctl(0, TCFLSH, 1);
#endif
                                /* zcrce - dinna wanna starta ping-pong game */
                                zsdata(txbuf, 0, ZCRCE);
                                goto gotack;
                        case XOFF:              /* Wait a while for an XON */
                        case XOFF|0200:
                                readline(100);
                        default:
                                ++junkcount;
                        }
```

```
                }
#endif   /* READCHECK */
                if (Txwindow) {
                        while ((tcount = (Txpos - Lrxpos)) >= Txwindow) {
                                vfile("%ld window >= %u", tcount, Txwindow);
                                if (e != ZCRCQ)
                                        zsdata(txbuf, 0, e = ZCRCQ);
                                c = getinsync(1);
                                if (c != ZACK) {
#ifdef TCFLSH
                                        ioctl(0, TCFLSH, 1);
#endif
                                        zsdata(txbuf, 0, ZCRCE);
                                        goto gotack;
                                }
                        }
                        vfile("window = %ld", tcount);
                }
        } while (!Eofseen);
        if ( !Fromcu)
                signal(SIGINT, SIG_IGN);

        for (;;) {
                stohdr(Txpos);
                zsbhdr(4, ZEOF, Txhdr);
                switch (getinsync(0)) {
                case ZACK:
                        continue;
                case ZRPOS:
                        goto somemore;
                case ZRINIT:
                        return OK;
                case ZSKIP:
                        fclose(in);
                        return c;
                default:
                        fclose(in);
                        return ERROR;
                }
        }
}

/*
 * Respond to receiver's complaint, get back in sync with receiver
 */
getinsync(flag)
{
        register c;

        for (;;) {
                if (Test) {
                        printf("\r\n\n\n***** Signal Caught *****\r\n");
                        Rxpos = 0; c = ZRPOS;
                } else
                        c = zgethdr(Rxhdr, 0);
                switch (c) {
                case ZCAN:
                case ZABORT:
                case ZFIN:
                case TIMEOUT:
                        return ERROR;
                case ZRPOS:
```

```
                        /* ************************************ */
                        /*  If sending to a buffered modem, you  */
                        /*   might send a break at this point to */
                        /*   dump the modem's buffer.            */
                        clearerr(in);       /* In case file EOF seen */
                        if (fseek(in, Rxpos, 0))
                                return ERROR;
                        Eofseen = 0;
                        bytcnt = Lrxpos = Txpos = Rxpos;
#ifndef GENIE
                        if (Lastsync == Rxpos) {
                                if (++Beenhereb4 > 4)
                                        if (blklen > 32)
                                                blklen /= 2;
                        }
#endif
                        Lastsync = Rxpos;
                        return c;
                case ZACK:
                        Lrxpos = Rxpos;
                        if (flag || Txpos == Rxpos)
                                return ZACK;
                        continue;
                case ZRINIT:
                case ZSKIP:
                        fclose(in);
                        return c;
                case ERROR:
                default:
                        zsbhdr(4, ZNAK, Txhdr);
                        continue;
                }
        }
}


/* Say "bibi" to the receiver, try to do it cleanly */
saybibi()
{
        for (;;) {
                stohdr(0L);                     /* CAF Was zsbhdr - minor change */
                zshhdr(4, ZFIN, Txhdr); /*  to make debugging easier */
                switch (zgethdr(Rxhdr, 0)) {
                case ZFIN:
                        sendline('O'); sendline('O'); flushmo();
                case ZCAN:
                case TIMEOUT:
                        return;
                }
        }
}

/* Local screen character display function */
bttyout(c)
{
        if (Verbose)
                putc(c, stderr);
}

/* Send command and related info */
zsendcmd(buf, blen)
char *buf;
```

```
{
        register c;
        long cmdnum;

#ifdef GENIE
        cmdnum = 69;
#else
        cmdnum = getpid();
#endif
        errors = 0;
        for (;;) {
                stohdr(cmdnum);
                Txhdr[ZF0] = Cmdack1;
                zsbhdr(4, ZCOMMAND, Txhdr);
                zsdata(buf, blen, ZCRCW);
listen:
                Rxtimeout = 100;                /* Ten second wait for resp. */
                Usevhdrs = 0;           /* Allow rx to send fixed len headers */
                c = zgethdr(Rxhdr, 1);

                switch (c) {
                case ZRINIT:
                        goto listen;    /* CAF 8-21-87 */
                case ERROR:
                case GCOUNT:
                case TIMEOUT:
                        if (++errors > Cmdtries)
                                return ERROR;
                        continue;
                case ZCAN:
                case ZABORT:
                case ZFIN:
                case ZSKIP:
                case ZRPOS:
                        return ERROR;
                default:
                        if (++errors > 20)
                                return ERROR;
                        continue;
                case ZCOMPL:
                        Exitcode = Rxpos;
                        saybibi();
                        return OK;
                case ZRQINIT:
#ifdef vax11c           /* YAMP :== Yet Another Missing Primitive */
                        return ERROR;
#else
                        vfile("******** RZ *******");
                        system("rz");
                        vfile("******** SZ *******");
                        goto listen;
#endif
                }
        }
}

/*
 * If called as sb use YMODEM protocol
 */
chkinvok(s)
char *s;
{
```

```
	register char *p;

	p = s;
	while (*p == '-')
		s = ++p;
	while (*p)
		if (*p++ == '/')
			s = p;
	if (*s == 'v') {
		Verbose=1; ++s;
	}
	Progname = s;
	if (s[0]=='s' && s[1]=='b') {
		Nozmodem = TRUE; blklen=1024;
	}
	if (s[0]=='s' && s[1]=='x') {
		Modem2 = TRUE;
	}
}

#ifdef STAT
countem(argc, argv)
register char **argv;
{
	register c;
	struct stat f;

	for (Totalleft = 0, Filesleft = 0; --argc >=0; ++argv) {
		f.st_size = -1;
		if (Verbose>2) {
			fprintf(stderr, "\nCountem: %03d %s ", argc, *argv);
			fflush(stderr);
		}
		if (access(*argv, 04) >= 0 && stat(*argv, &f) >= 0) {
			c = f.st_mode & S_IFMT;
			if (c != S_IFDIR && c != S_IFBLK) {
				++Filesleft;  Totalleft += f.st_size;
			}
		}
		if (Verbose>2)
			fprintf(stderr, " %ld", f.st_size);
	}
	if (Verbose>2)
		fprintf(stderr, "\ncountem: Total %d %ld\n",
		  Filesleft, Totalleft);
}
#else
countem(argc, argv)
register char **argv;
{
	register c;
	register char *p;
	long size;

	for (Totalleft = 0, Filesleft = 0; --argc >=0; ++argv) {
		size = -1;
		if (Verbose>2) {
			fprintf(stderr, "\nCountem: %03d %s ", argc, *argv);
			fflush(stderr);
		}
		++Filesleft;
#ifdef XARGSFILE
```

```
                /* Look for file length in third colon sep field */
                for (p = *argv; *p; ++p) {
                        if (*p == ':') {
                                for (++p; *p; ++p) {
                                        if (*p == ':') {
                                                ++p;
                                                size = atol(p);
                                                Totalleft += size;
                                                break;
                                        }
                                }
                        break;
                        }
                }
#endif

                if (Verbose>2)
                        fprintf(stderr, " %ld", size);
        }
        if (Verbose>2)
                fprintf(stderr, "\ncountem: Total %d %ld\n",
                  Filesleft, Totalleft);
}
#endif

chartest(m)
{
        register n;

        mode(m);
        printf("\r\n\nCharacter Transparency Test Mode %d\r\n", m);
        printf("If Pro-YAM/ZCOMM is not displaying ^M hit ALT-V NOW.\r\n");
        printf("Hit Enter.\021");  fflush(stdout);
        readline(500);

        for (n = 0; n < 256; ++n) {
                if (!(n%8))
                        printf("\r\n");
                printf("%02x ", n);  fflush(stdout);
                sendline(n);       flushmo();
                printf("  ");  fflush(stdout);
                if (n == 127) {
                        printf("Hit Enter.\021");  fflush(stdout);
                        readline(500);
                        printf("\r\n");  fflush(stdout);
                }
        }
        printf("\021\r\nEnter Characters, echo is in hex.\r\n");
        printf("Hit SPACE or pause 40 seconds for exit.\r\n");

        while (n != TIMEOUT && n != ' ') {
                n = readline(400);
                printf("%02x\r\n", n);
                fflush(stdout);
        }
        printf("\r\nMode %d character transparency test ends.\r\n", m);
        fflush(stdout);
}

/* End of sz.c */
```

```
#define VERSION "3.01 5-25-89"
#define PUBDIR "/usr/spool/uucppublic"

/*% cc -compat -M2 -Ox -K -i -DMD % -o rz; size rz;
<-xtx-*> cc386 -Ox -DMD rz.c -o $B/rz;  size $B/rz
 *
 * rz.c By Chuck Forsberg
 *
 *      cc -O rz.c -o rz            USG (3.0) Unix
 *      cc -O -DV7  rz.c -o rz              Unix V7, BSD 2.8 - 4.3
 *
 *      ln rz rb;  ln rz rx                 For either system
 *
 *      ln rz /usr/bin/rzrmail              For remote mail.  Make this the
 *                                          login shell. rzrmail then calls
 *                                          rmail(1) to deliver mail.
 *
 * To compile on VMS:
 *
 *      define LNK$LIBRARY   SYS$LIBRARY:VAXCRTL.OLB
 *      cc rz.c
 *      cc vvmodem.c
 *      link rz,vvmodem
 *      rz :== $disk:[username.subdir]rz.exe
 *    For high speed, try increasing the SYSGEN parameter TTY_TYPAHDSZ to 256.
 *
 *
 *  Unix is a trademark of Western Electric Company
 *
 * A program for Unix to receive files and commands from computers running
 *  Professional-YAM, PowerCom, YAM, IMP, or programs supporting XMODEM.
 *  rz uses Unix buffered input to reduce wasted CPU time.
 *
 *      This version implements ZMODEM Run Length Encoding
 *      and variable length headers.  These features were not funded
 *      by the original Telenet development contract.  This software,
 *      including these features, may be freely used for non
 *      commercial and educational purposes.  This software may also
 *      be freely used to support file transfer operations to or from
 *      licensed Omen Technology products.  Contact Omen Technology
 *      for licensing for other uses.  Any programs which use part or
 *      all of this software must be provided in source form with this
 *      notice intact except by written permission from Omen
 *      Technology Incorporated.
 *
 *              Omen Technology Inc         FAX: 503-621-3745
 *              Post Office Box 4681
 *              Portland OR 97208
 *
 *      Previous versions of this program (not containing the extensions
 *      listed above) remain in the public domain.
 *
 *      This code is made available in the hope it will be useful,
 *      BUT WITHOUT ANY WARRANTY OF ANY KIND OR LIABILITY FOR ANY
 *      DAMAGES OF ANY KIND.
 *
 *
 * Iff the program is invoked by rzCOMMAND, output is piped to
 * "COMMAND filename"  (Unix only)
 *
 *  Some systems (Venix, Coherent, Regulus) may not support tty raw mode
 *  read(2) the same way as Unix. ONEREAD must be defined to force one
```

```
 *  character reads for these systems. Added 7-01-84 CAF
 *
 *  Alarm signal handling changed to work with 4.2 BSD 7-15-84 CAF
 *
 *  BIX added 6-30-87 to support BIX(TM) upload protocol used by the
 *  Byte Information Exchange.
 *
 *  NFGVMIN Updated 2-18-87 CAF for Xenix systems where c_cc[VMIN]
 *  doesn't work properly (even though it compiles without error!),
 *
 *  SEGMENTS=n added 2-21-88 as a model for CP/M programs
 *    for CP/M-80 systems that cannot overlap modem and disk I/O.
 *
 *  VMS flavor hacks begin with rz version 2.00
 *
 *  -DMD may be added to compiler command line to compile in
 *    Directory-creating routines from Public Domain TAR by John Gilmore
 *
 *  HOWMANY may be tuned for best performance
 *
 *  USG UNIX (3.0) ioctl conventions courtesy  Jeff Martin
 */

#ifdef vax11c
#include <types.h>
#include <stat.h>
#define LOGFILE "rzlog.tmp"
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <ctype.h>
#include <errno.h>
#define OS "VMS"
#define BUFREAD
extern int errno;
#define SS_NORMAL SS$_NORMAL

#ifndef PROGNAME
#define PROGNAME "rz"
#endif


#else


#define SS_NORMAL 0
#define LOGFILE "/tmp/rzlog"
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <ctype.h>
#include <errno.h>
extern int errno;
FILE *popen();
#endif

#define OK 0
#define FALSE 0
#define TRUE 1
#define ERROR (-1)

/*
```

```
 * Max value for HOWMANY is 255.
 *   A larger value reduces system overhead but may evoke kernel bugs.
 *   133 corresponds to an XMODEM/CRC sector
 */
#ifndef HOWMANY
#define HOWMANY 133
#endif

/* Ward Christensen / CP/M parameters - Don't change these! */
#define ENQ 005
#define CAN ('X'&037)
#define XOFF ('s'&037)
#define XON ('q'&037)
#define SOH 1
#define STX 2
#define EOT 4
#define ACK 6
#define NAK 025
#define CPMEOF 032
#define WANTCRC 0103          /* send C not NAK to get crc not checksum */
#define TIMEOUT (-2)
#define RCDO (-3)
#define GCOUNT (-4)
#define ERRORMAX 5
#define RETRYMAX 5
#define WCEOT (-10)
#define PATHLEN 257  /* ready for 4.2 bsd ? */
#define UNIXFILE 0xF000         /* The S_IFMT file mask bit for stat */

int Zmodem=0;               /* ZMODEM protocol requested */
int Nozmodem = 0;       /* If invoked as "rb" */
unsigned Baudrate = 2400;
unsigned Effbaud = 2400;
#ifdef vax11c
#include "vrzsz.c"          /* most of the system dependent stuff here */
#else
#include "rbsb.c"           /* most of the system dependent stuff here */
#endif
#include "crctab.c"

char *substr();
FILE *fout;

/*
 * Routine to calculate the free bytes on the current file system
 *  ~0 means many free bytes (unknown)
 */
long getfree()
{
        return(~0L);      /* many free bytes ... */
}

int Lastrx;
int Crcflg;
int Firstsec;
int Eofseen;                /* indicates cpm eof (^Z) has been received */
int errors;
int Restricted=0;           /* restricted; no /.. or ../ in filenames */
#ifdef ONEREAD
/* Sorry, Regulus and some others don't work right in raw mode! */
int Readnum = 1;            /* Number of bytes to ask for in read() from modem */
#else
```

```
int Readnum = HOWMANY;      /* Number of bytes to ask for in read() from modem */
#endif

#define DEFBYTL 2000000000L          /* default rx file size */
long Bytesleft;             /* number of bytes of incoming file left */
long Modtime;               /* Unix style mod time for incoming file */
int Filemode;               /* Unix style mode for incoming file */
char Pathname[PATHLEN];
char *Progname;                     /* the name by which we were called */

int Batch=0;
int Topipe=0;
int MakeLCPathname=TRUE;    /* make received pathname lower case */
int Verbose=0;
int Quiet=0;                /* overrides logic that would otherwise set verbose */
int Nflag = 0;              /* Don't really transfer files */
int Rxclob=FALSE;           /* Clobber existing file */
int Rxbinary=FALSE;         /* receive all files in bin mode */
int Rxascii=FALSE;          /* receive files in ascii (translate) mode */
int Thisbinary;             /* current file is to be received in bin mode */
int Blklen;                 /* record length of received packets */

#ifdef SEGMENTS
int chinseg = 0; /* Number of characters received in this data seg */
char secbuf[1+(SEGMENTS+1)*1024];
#else
char secbuf[1025];
#endif


char linbuf[HOWMANY];
int Lleft=0;                /* number of characters in linbuf */
time_t timep[2];
char Lzmanag;               /* Local file management request */
char zconv;                 /* ZMODEM file conversion request */
char zmanag;                /* ZMODEM file management request */
char ztrans;                /* ZMODEM file transport request */
int Zctlesc;                /* Encode control characters */
int Zrwindow = 1400;        /* RX window size (controls garbage count) */

jmp_buf tohere;             /* For the interrupt on RX timeout */

#define xsendline(c) sendline(c)

#include "zm.c"

#include "zmr.c"

int tryzhdrtype=ZRINIT; /* Header type to send corresponding to Last rx close */

alrm()
{
        longjmp(tohere, -1);
}

/* called by signal interrupt or terminate to clean things up */
bibi(n)
{
        if (Zmodem)
                zmputs(Attn);
        canit(); mode(0);
        fprintf(stderr, "rz: caught signal %d; exiting", n);
```

```
	cucheck();
	exit(128+n);
}

main(argc, argv)
char *argv[];
{
	register char *cp;
	register npats;
	char *virgin, **patts;
	char *getenv();
	int exitcode;

	Rxtimeout = 100;
	setbuf(stderr, NULL);
	if ((cp=getenv("SHELL")) && (substr(cp, "rsh") || substr(cp, "rksh")))
		Restricted=TRUE;

	from_cu();
#ifdef vax11c
	chkinvok(virgin = PROGNAME);
#else
	chkinvok(virgin=argv[0]);		/* if called as [-]rzCOMMAND set flag */
#endif
	npats = 0;
	while (--argc) {
		cp = *++argv;
		if (*cp == '-') {
			while( *++cp) {
				switch(*cp) {
				case '\\':
					 cp[1] = toupper(cp[1]);  continue;
				case '+':
					Lzmanag = ZMAPND; break;
				case 'a':
					Rxascii=TRUE;  break;
				case 'b':
					Rxbinary=TRUE; break;
				case 'c':
					Crcflg=TRUE; break;
#ifndef vax11c
				case 'D':
					Nflag = TRUE; break;
#endif
				case 'e':
					Zctlesc = 1; break;
				case 'p':
					Lzmanag = ZMPROT;  break;
				case 'q':
					Quiet=TRUE; Verbose=0; break;
				case 't':
					if (--argc < 1) {
						usage();
					}
					Rxtimeout = atoi(*++argv);
					if (Rxtimeout<10 || Rxtimeout>1000)
						usage();
					break;
				case 'w':
					if (--argc < 1) {
						usage();
					}
```

```
                                        Zrwindow = atoi(*++argv);
                                        break;
                                case 'u':
                                        MakeLCPathname=FALSE; break;
                                case 'v':
                                        ++Verbose; break;
                                case 'y':
                                        Rxclob=TRUE; break;
                                default:
                                        usage();
                                }
                        }
                }
                else if ( !npats && argc>0) {
                        if (argv[0][0]) {
                                npats=argc;
                                patts=argv;
                        }
                }
        }
        if (npats > 1)
                usage();
        if (Batch && npats)
                usage();
        if (Verbose) {
                if (freopen(LOGFILE, "a", stderr)==NULL) {
                        printf("Can't open log file %s\n",LOGFILE);
                        exit(0200);
                }
                setbuf(stderr, NULL);
                fprintf(stderr, "argv[0]=%s Progname=%s\n", virgin, Progname);
        }
        if (Fromcu && !Quiet) {
                if (Verbose == 0)
                        Verbose = 2;
        }
        vfile("%s %s for %s\n", Progname, VERSION, OS);
        mode(1);
        if (signal(SIGINT, bibi) == SIG_IGN) {
                signal(SIGINT, SIG_IGN); signal(SIGKILL, SIG_IGN);
        }
        else {
                signal(SIGINT, bibi); signal(SIGKILL, bibi);
        }
        signal(SIGTERM, bibi);
        if (wcreceive(npats, patts)==ERROR) {
                exitcode=0200;
                canit();
        }
        mode(0);
        if (exitcode && !Zmodem)        /* bellow again with all thy might. */
                canit();
        if (exitcode)
                cucheck();
        exit(exitcode ? exitcode:SS_NORMAL);
}


usage()
{
        cucheck();
        fprintf(stderr,"Usage:    rz [-abeuvy]                (ZMODEM)\n");
```

```
        fprintf(stderr,"or rb [-abuvy]                  (YMODEM)\n");
        fprintf(stderr,"or rx [-abcv] file     (XMODEM or XMODEM-1k)\n");
        fprintf(stderr,"      -a ASCII transfer (strip CR)\n");
        fprintf(stderr,"      -b Binary transfer for all files\n");
#ifndef vax11c
        fprintf(stderr,"      -c Use 16 bit CRC       (XMODEM)\n");
#endif
        fprintf(stderr,"      -e Escape control characters    (ZMODEM)\n");
        fprintf(stderr,"      -v Verbose more v's give more info\n");
        fprintf(stderr,"      -y Yes, clobber existing file if any\n");
        fprintf(stderr,"%s %s for %s by Chuck Forsberg, Omen Technology INC\n",
          Progname, VERSION, OS);
        fprintf(stderr, "\t\t\042The High Reliability Software\042\n");
        exit(SS_NORMAL);
}
/*
 *  Debugging information output interface routine
 */
/* VARARGS1 */
vfile(f, a, b, c)
register char *f;
{
        if (Verbose > 2) {
                fprintf(stderr, f, a, b, c);
                fprintf(stderr, "\n");
        }
}

/*
 * Let's receive something already.
 */

char *rbmsg =
"%s ready. To begin transfer, type \"%s file ...\" to your modem program\r\n\n";

wcreceive(argc, argp)
char **argp;
{
        register c;

        if (Batch || argc==0) {
                Crcflg=1;
                if ( !Quiet)
                        fprintf(stderr, rbmsg, Progname, Nozmodem?"sb":"sz");
                if (c=tryz()) {
                        if (c == ZCOMPL)
                                return OK;
                        if (c == ERROR)
                                goto fubar;
                        c = rzfiles();
                        if (c)
                                goto fubar;
                } else {
                        for (;;) {
                                if (wcrxpn(secbuf)== ERROR)
                                        goto fubar;
                                if (secbuf[0]==0)
                                        return OK;
                                if (procheader(secbuf) == ERROR)
                                        goto fubar;
                                if (wcrx()==ERROR)
                                        goto fubar;
```

```
                        }
                }
        } else {
                Bytesleft = DEFBYTL; Filemode = 0; Modtime = 0L;

                procheader(""); strcpy(Pathname, *argp); checkpath(Pathname);
                fprintf(stderr, "\nrz: ready to receive %s\r\n", Pathname);
                if ((fout=fopen(Pathname, "w")) == NULL)
                        return ERROR;
                if (wcrx()==ERROR)
                        goto fubar;
        }
        return OK;
fubar:
        canit();
#ifndef vax11c
        if (Topipe && fout) {
                pclose(fout);  return ERROR;
        }
#endif
        Modtime = 1;
        if (fout)
                fclose(fout);
#ifndef vax11c
        if (Restricted) {
                unlink(Pathname);
                fprintf(stderr, "\r\nrz: %s removed.\r\n", Pathname);
        }
#endif
        return ERROR;
}


/*
 * Fetch a pathname from the other end as a C ctyle ASCIZ string.
 * Length is indeterminate as long as less than Blklen
 * A null string represents no more files (YMODEM)
 */
wcrxpn(rpn)
char *rpn;      /* receive a pathname */
{
        register c;

#ifdef NFGVMIN
        readline(1);
#else
        purgeline();
#endif

et_tu:
        Firstsec=TRUE;  Eofseen=FALSE;
        sendline(Crcflg?WANTCRC:NAK);
        Lleft=0; /* Do read next time ... */
        while ((c = wcgetsec(rpn, 100)) != 0) {
                if (c == WCEOT) {
                        zperr( "Pathname fetch returned %d", c);
                        sendline(ACK);
                        Lleft=0; /* Do read next time ... */
                        readline(1);
                        goto et_tu;
                }
                return ERROR;
```

```
        }
        sendline(ACK);
        return OK;
}

/*
 * Adapted from CMODEM13.C, written by
 * Jack M. Wierda and Roderick W. Hart
 */

wcrx()
{
        register int sectnum, sectcurr;
        register char sendchar;
        register char *p;
        int cblklen;                    /* bytes to dump this block */

        Firstsec=TRUE;sectnum=0; Eofseen=FALSE;
        sendchar=Crcflg?WANTCRC:NAK;

        for (;;) {
                sendline(sendchar);     /* send it now, we're ready! */
                Lleft=0; /* Do read next time ... */
                sectcurr=wcgetsec(secbuf, (sectnum&0177)?50:130);
                report(sectcurr);
                if (sectcurr==(sectnum+1 &0377)) {
                        sectnum++;
                        cblklen = Bytesleft>Blklen ? Blklen:Bytesleft;
                        if (putsec(secbuf, cblklen)==ERROR)
                                return ERROR;
                        if ((Bytesleft-=cblklen) < 0)
                                Bytesleft = 0;
                        sendchar=ACK;
                }
                else if (sectcurr==(sectnum&0377)) {
                        zperr( "Received dup Sector");
                        sendchar=ACK;
                }
                else if (sectcurr==WCEOT) {
                        if (closeit())
                                return ERROR;
                        sendline(ACK);
                        Lleft=0; /* Do read next time ... */
                        return OK;
                }
                else if (sectcurr==ERROR)
                        return ERROR;
                else {
                        zperr( "Sync Error");
                        return ERROR;
                }
        }
}

/*
 * Wcgetsec fetches a Ward Christensen type sector.
 * Returns sector number encountered or ERROR if valid sector not received,
 * or CAN CAN received
 * or WCEOT if eot sector
 * time is timeout for first char, set to 4 seconds thereafter
 ********** NO ACK IS SENT IF SECTOR IS RECEIVED OK **********
 *    (Caller must do that when he is good and ready to get next sector)
```

```
 */

wcgetsec(rxbuf, maxtime)
char *rxbuf;
int maxtime;
{
        register checksum, wcj, firstch;
        register unsigned short oldcrc;
        register char *p;
        int sectcurr;

        for (Lastrx=errors=0; errors<RETRYMAX; errors++) {

                if ((firstch=readline(maxtime))==STX) {
                        Blklen=1024; goto get2;
                }
                if (firstch==SOH) {
                        Blklen=128;
get2:
                        sectcurr=readline(1);
                        if ((sectcurr+(oldcrc=readline(1)))==0377) {
                                oldcrc=checksum=0;
                                for (p=rxbuf,wcj=Blklen; --wcj>=0; ) {
                                        if ((firstch=readline(1)) < 0)
                                                goto bilge;
                                        oldcrc=updcrc(firstch, oldcrc);
                                        checksum += (*p++ = firstch);
                                }
                                if ((firstch=readline(1)) < 0)
                                        goto bilge;
                                if (Crcflg) {
                                        oldcrc=updcrc(firstch, oldcrc);
                                        if ((firstch=readline(1)) < 0)
                                                goto bilge;
                                        oldcrc=updcrc(firstch, oldcrc);
                                        if (oldcrc & 0xFFFF)
                                                zperr( "CRC");
                                        else {
                                                Firstsec=FALSE;
                                                return sectcurr;
                                        }
                                }
                                else if (((checksum-firstch)&0377)==0) {
                                        Firstsec=FALSE;
                                        return sectcurr;
                                }
                                else
                                        zperr( "Checksum");
                        }
                        else
                                zperr("Sector number garbled");
                }
                /* make sure eot really is eot and not just mixmash */
#ifdef NFGVMIN
                else if (firstch==EOT && readline(1)==TIMEOUT)
                        return WCEOT;
#else
                else if (firstch==EOT && Lleft==0)
                        return WCEOT;
#endif
                else if (firstch==CAN) {
                        if (Lastrx==CAN) {
```

```
                                zperr( "Sender CANcelled");
                                return ERROR;
                        } else {
                                Lastrx=CAN;
                                continue;
                        }
                }
                else if (firstch==TIMEOUT) {
                        if (Firstsec)
                                goto humbug;
bilge:
                        zperr( "TIMEOUT");
                }
                else
                        zperr( "Got 0%o sector header", firstch);

humbug:
                Lastrx=0;
                while(readline(1)!=TIMEOUT)
                        ;
                if (Firstsec) {
                        sendline(Crcflg?WANTCRC:NAK);
                        Lleft=0; /* Do read next time ... */
                } else {
                        maxtime=40; sendline(NAK);
                        Lleft=0; /* Do read next time ... */
                }
        }
        /* try to stop the bubble machine. */
        canit();
        return ERROR;
}

#ifndef vax11c
/*
 * This version of readline is reasoably well suited for
 * reading many characters.
 *  (except, currently, for the Regulus version!)
 *
 * timeout is in tenths of seconds
 */
readline(timeout)
int timeout;
{
        register n;
        static char *cdq;       /* pointer for removing chars from linbuf */

        if (--Lleft >= 0) {
                if (Verbose > 8) {
                        fprintf(stderr, "%02x ", *cdq&0377);
                }
                return (*cdq++ & 0377);
        }
        n = timeout/10;
        if (n < 2)
                n = 3;
        if (Verbose > 5)
                fprintf(stderr, "Calling read: alarm=%d  Readnum=%d ",
                  n, Readnum);
        if (setjmp(tohere)) {
#ifdef TIOCFLUSH
/*              ioctl(0, TIOCFLUSH, 0); */
```

```
#endif
                Lleft = 0;
                if (Verbose>1)
                        fprintf(stderr, "Readline:TIMEOUT\n");
                return TIMEOUT;
        }
        signal(SIGALRM, alrm); alarm(n);
        Lleft=read(0, cdq=linbuf, Readnum);
        alarm(0);
        if (Verbose > 5) {
                fprintf(stderr, "Read returned %d bytes\n", Lleft);
        }
        if (Lleft < 1)
                return TIMEOUT;
        --Lleft;
        if (Verbose > 8) {
                fprintf(stderr, "%02x ", *cdq&0377);
        }
        return (*cdq++ & 0377);
}



/*
 * Purge the modem input queue of all characters
 */
purgeline()
{
        Lleft = 0;
#ifdef USG
        ioctl(0, TCFLSH, 0);
#else
        lseek(0, 0L, 2);
#endif
}
#endif


/*
 * Process incoming file information header
 */
procheader(name)
char *name;
{
        register char *openmode, *p, **pp;

        /* set default parameters and overrides */
        openmode = "w";
        Thisbinary = (!Rxascii) || Rxbinary;
        if (Lzmanag)
                zmanag = Lzmanag;

        /*
         *  Process ZMODEM remote file management requests
         */
        if (!Rxbinary && zconv == ZCNL)          /* Remote ASCII override */
                Thisbinary = 0;
        if (zconv == ZCBIN)      /* Remote Binary override */
                Thisbinary = TRUE;
        else if (zmanag == ZMAPND)
                openmode = "a";
```

```
#ifndef BIX
        /* Check for existing file */
        if (!Rxclob && (zmanag&ZMMASK) != ZMCLOB && (fout=fopen(name, "r"))) {
                fclose(fout);  return ERROR;
        }
#endif

        Bytesleft = DEFBYTL; Filemode = 0; Modtime = 0L;

        p = name + 1 + strlen(name);
        if (*p) {  /* file coming from Unix or DOS system */
                sscanf(p, "%ld%lo%o", &Bytesleft, &Modtime, &Filemode);
#ifndef vax11c
                if (Filemode & UNIXFILE)
                        ++Thisbinary;
#endif
                if (Verbose) {
                        fprintf(stderr,  "Incoming: %s %ld %lo %o\n",
                          name, Bytesleft, Modtime, Filemode);
                }
        }

#ifdef BIX
        if ((fout=fopen("scratchpad", openmode)) == NULL)
                return ERROR;
        return OK;
#else

        else {                  /* File coming from CP/M system */
                for (p=name; *p; ++p)                   /* change / to _ */
                        if ( *p == '/')
                                *p = '_';

                if ( *--p == '.')                       /* zap trailing period */
                        *p = 0;
        }

#ifndef vax11c
        if (!Zmodem && MakeLCPathname && !IsAnyLower(name)
          && !(Filemode&UNIXFILE))
                uncaps(name);
#endif
        if (Topipe > 0) {
                sprintf(Pathname, "%s %s", Progname+2, name);
                if (Verbose)
                        fprintf(stderr,  "Topipe: %s %s\n",
                          Pathname, Thisbinary?"BIN":"ASCII");
#ifndef vax11c
                if ((fout=popen(Pathname, "w")) == NULL)
                        return ERROR;
#endif
        } else {
                strcpy(Pathname, name);
                if (Verbose) {
                        fprintf(stderr,  "Receiving %s %s %s\n",
                          name, Thisbinary?"BIN":"ASCII", openmode);
                }
                checkpath(name);
                if (Nflag)
                        name = "/dev/null";
#ifndef vax11c
                if (name[0] == '!' || name[0] == '|') {
```

```
                        if ( !(fout = popen(name+1, "w"))) {
                                return ERROR;
                        }
                        Topipe = -1;  return(OK);
                }
#endif
#ifdef MD
                fout = fopen(name, openmode);
                if ( !fout)
                        if (make_dirs(name))
                                fout = fopen(name, openmode);
#else
                fout = fopen(name, openmode);
#endif
                if ( !fout)
                        return ERROR;
        }
        return OK;
#endif /* BIX */
}

#ifdef MD
/*
 *  Directory-creating routines from Public Domain TAR by John Gilmore
 */

/*
 * After a file/link/symlink/dir creation has failed, see if
 * it's because some required directory was not present, and if
 * so, create all required dirs.
 */
make_dirs(pathname)
register char *pathname;
{
        register char *p;           /* Points into path */
        int madeone = 0;                /* Did we do anything yet? */
        int save_errno = errno;         /* Remember caller's errno */
        char *strchr();

        if (errno != ENOENT)
                return 0;               /* Not our problem */

        for (p = strchr(pathname, '/'); p != NULL; p = strchr(p+1, '/')) {
                /* Avoid mkdir of empty string, if leading or double '/' */
                if (p == pathname || p[-1] == '/')
                        continue;
                /* Avoid mkdir where last part of path is '.' */
                if (p[-1] == '.' && (p == pathname+1 || p[-2] == '/'))
                        continue;
                *p = 0;                         /* Truncate the path there */
                if ( !mkdir(pathname, 0777)) {  /* Try to create it as a dir */
                        vfile("Made directory %s\n", pathname);
                        madeone++;              /* Remember if we made one */
                        *p = '/';
                        continue;
                }
                *p = '/';
                if (errno == EEXIST)            /* Directory already exists */
                        continue;
                /*
                 * Some other error in the mkdir.  We return to the caller.
                 */
```

```
                break;
        }
        errno = save_errno;             /* Restore caller's errno */
        return madeone;                         /* Tell them to retry if we made one */
}

#if (MD != 2)
#define TERM_SIGNAL(status) ((status) & 0x7F)
#define TERM_COREDUMP(status)   (((status) & 0x80) != 0)
#define TERM_VALUE(status)  ((status) >> 8)
/*
 * Make a directory.  Compatible with the mkdir() system call on 4.2BSD.
 */
mkdir(dpath, dmode)
char *dpath;
int dmode;
{
        int cpid, status;
        struct stat statbuf;

        if (stat(dpath,&statbuf) == 0) {
                errno = EEXIST;                 /* Stat worked, so it already exists */
                return -1;
        }

        /* If stat fails for a reason other than non-existence, return error */
        if (errno != ENOENT) return -1;

        switch (cpid = fork()) {

        case -1:                        /* Error in fork() */
                return(-1);             /* Errno is set already */

        case 0:                         /* Child process */
                /*
                 * Cheap hack to set mode of new directory.  Since this
                 * child process is going away anyway, we zap its umask.
                 * FIXME, this won't suffice to set SUID, SGID, etc. on this
                 * directory.  Does anybody care?
                 */
                status = umask(0);      /* Get current umask */
                status = umask(status | (0777 & ~dmode)); /* Set for mkdir */
                execl("/bin/mkdir", "mkdir", dpath, (char *)0);
                _exit(-1);              /* Can't exec /bin/mkdir */

        default:                        /* Parent process */
                while (cpid != wait(&status)) ;     /* Wait for kid to finish */
        }

        if (TERM_SIGNAL(status) != 0 || TERM_VALUE(status) != 0) {
                errno = EIO;            /* We don't know why, but */
                return -1;              /* /bin/mkdir failed */
        }

        return 0;
}
#endif /* MD != 2 */
#endif /* MD */

/*
 * Putsec writes the n characters of buf to receive file fout.
 *  If not in binary mode, carriage returns, and all characters
```

```
 *  starting with CPMEOF are discarded.
 */
putsec(buf, n)
char *buf;
register n;
{
        register char *p;

        if (n == 0)
                return OK;
        if (Thisbinary) {
                for (p=buf; --n>=0; )
                        putc( *p++, fout);
        }
        else {
                if (Eofseen)
                        return OK;
                for (p=buf; --n>=0; ++p ) {
                        if ( *p == '\r')
                                continue;
                        if (*p == CPMEOF) {
                                Eofseen=TRUE; return OK;
                        }
                        putc(*p ,fout);
                }
        }
        return OK;
}

#ifndef vax11c
/*
 *  Send a character to modem.  Small is beautiful.
 */
sendline(c)
{
        char d;

        d = c;
        if (Verbose>6)
                fprintf(stderr, "Sendline: %x\n", c);
        write(1, &d, 1);
}

flushmo() {}
#endif




/* make string s lower case */
uncaps(s)
register char *s;
{
        for ( ; *s; ++s)
                if (isupper(*s))
                        *s = tolower(*s);
}
/*
 * IsAnyLower returns TRUE if string s has lower case letters.
 */
IsAnyLower(s)
```

```
register char *s;
{
        for ( ; *s; ++s)
                if (islower(*s))
                        return TRUE;
        return FALSE;
}

/*
 * substr(string, token) searches for token in string s
 * returns pointer to token within string if found, NULL otherwise
 */
char *
substr(s, t)
register char *s,*t;
{
        register char *ss,*tt;
        /* search for first char of token */
        for (ss=s; *s; s++)
                if (*s == *t)
                        /* compare token with substring */
                        for (ss=s,tt=t; ;) {
                                if (*tt == 0)
                                        return s;
                                if (*ss++ != *tt++)
                                        break;
                        }
        return NULL;
}

/*
 * Log an error
 */
/*VARARGS1*/
zperr(s,p,u)
char *s, *p, *u;
{
        if (Verbose <= 0)
                return;
        fprintf(stderr, "Retry %d: ", errors);
        fprintf(stderr, s, p, u);
        fprintf(stderr, "\n");
}

/* send cancel string to get the other end to shut up */
canit()
{
        static char canistr[] = {
         24,24,24,24,24,24,24,24,24,8,8,8,8,8,8,8,8,8,0
        };

#ifdef vax11c
        raw_wbuf(strlen(canistr), canistr);
        purgeline();
#else
        printf(canistr);
        Lleft=0; /* Do read next time ... */
        fflush(stdout);
#endif
}
```

```
report(sct)
int sct;
{
	if (Verbose>1)
		fprintf(stderr,"%03d%c",sct,sct%10? ' ' : '\r');
}

/*
 * If called as [-][dir/../]vrzCOMMAND set Verbose to 1
 * If called as [-][dir/../]rzCOMMAND set the pipe flag
 * If called as rb use YMODEM protocol
 */
chkinvok(s)
char *s;
{
	register char *p;

	p = s;
	while (*p == '-')
		s = ++p;
	while (*p)
		if (*p++ == '/')
			s = p;
	if (*s == 'v') {
		Verbose=1; ++s;
	}
	Progname = s;
	if (s[0]=='r' && s[1]=='z')
		Batch = TRUE;
	if (s[0]=='r' && s[1]=='b')
		Batch = Nozmodem = TRUE;
	if (s[2] && s[0]=='r' && s[1]=='b')
		Topipe = 1;
	if (s[2] && s[0]=='r' && s[1]=='z')
		Topipe = 1;
}

/*
 * Totalitarian Communist pathname processing
 */
checkpath(name)
char *name;
{
	if (Restricted) {
		if (fopen(name, "r") != NULL) {
			canit();
			fprintf(stderr, "\r\nrz: %s exists\n", name);
			bibi(-1);
		}
		/* restrict pathnames to current tree or uucppublic */
		if ( substr(name, "../")
		 || (name[0]== '/' && strncmp(name, PUBDIR, strlen(PUBDIR))) ) {
			canit();
			fprintf(stderr,"\r\nrz:\tSecurity Violation\r\n");
			bibi(-1);
		}
	}
}

/*
 * Initialize for Zmodem receive attempt, try to activate Zmodem sender
 *  Handles ZSINIT frame
```

```
 *  Return ZFILE if Zmodem filename received, -1 on error,
 *   ZCOMPL if transaction finished,  else 0
 */
tryz()
{
        register c, n;
        register cmdzack1flg;

        if (Nozmodem)           /* Check for "rb" program name */
                return 0;


        for (n=Zmodem?15:5; --n>=0; ) {
                /* Set buffer length (0) and capability flags */
#ifdef SEGMENTS
                stohdr(SEGMENTS*1024L);
#else
                stohdr(0L);
#endif
#ifdef CANBREAK
                Txhdr[ZF0] = CANFC32|CANFDX|CANOVIO|CANBRK;
#else
                Txhdr[ZF0] = CANFC32|CANFDX|CANOVIO;
#endif
                if (Zctlesc)
                        Txhdr[ZF0] |= TESCCTL;
                Txhdr[ZF0] |= CANRLE;
                Txhdr[ZF1] = CANVHDR;
                /* tryzhdrtype may == ZRINIT */
                zshhdr(4,tryzhdrtype, Txhdr);
                if (tryzhdrtype == ZSKIP)       /* Don't skip too far */
                        tryzhdrtype = ZRINIT;   /* CAF 8-21-87 */
again:
                switch (zgethdr(Rxhdr, 0)) {
                case ZRQINIT:
                        if (Rxhdr[ZF3] & 0x80)
                                Usevhdrs = 1;   /* we can var header */
                        continue;
                case ZEOF:
                        continue;
                case TIMEOUT:
                        continue;
                case ZFILE:
                        zconv = Rxhdr[ZF0];
                        zmanag = Rxhdr[ZF1];
                        ztrans = Rxhdr[ZF2];
                        if (Rxhdr[ZF3] & ZCANVHDR)
                                Usevhdrs = TRUE;
                        tryzhdrtype = ZRINIT;
                        c = zrdata(secbuf, 1024);
                        mode(3);
                        if (c == GOTCRCW)
                                return ZFILE;
                        zshhdr(4,ZNAK, Txhdr);
                        goto again;
                case ZSINIT:
                        Zctlesc = TESCCTL & Rxhdr[ZF0];
                        if (zrdata(Attn, ZATTNLEN) == GOTCRCW) {
                                stohdr(1L);
                                zshhdr(4,ZACK, Txhdr);
                                goto again;
                        }
```

```
                        zshhdr(4,ZNAK, Txhdr);
                        goto again;
                case ZFREECNT:
                        stohdr(getfree());
                        zshhdr(4,ZACK, Txhdr);
                        goto again;
                case ZCOMMAND:
#ifdef vax11c
                        return ERROR;
#else
                        cmdzack1flg = Rxhdr[ZF0];
                        if (zrdata(secbuf, 1024) == GOTCRCW) {
                                if (cmdzack1flg & ZCACK1)
                                        stohdr(0L);
                                else
                                        stohdr((long)sys2(secbuf));
                                purgeline();    /* dump impatient questions */
                                do {
                                        zshhdr(4,ZCOMPL, Txhdr);
                                }
                                while (++errors<20 && zgethdr(Rxhdr,1) != ZFIN);
                                ackbibi();
                                if (cmdzack1flg & ZCACK1)
                                        exec2(secbuf);
                                return ZCOMPL;
                        }
                        zshhdr(4,ZNAK, Txhdr); goto again;
#endif
                case ZCOMPL:
                        goto again;
                default:
                        continue;
                case ZFIN:
                        ackbibi(); return ZCOMPL;
                case ZCAN:
                        return ERROR;
                }
        }
        return 0;
}

/*
 * Receive 1 or more files with ZMODEM protocol
 */
rzfiles()
{
        register c;

        for (;;) {
                switch (c = rzfile()) {
                case ZEOF:
                case ZSKIP:
                        switch (tryz()) {
                        case ZCOMPL:
                                return OK;
                        default:
                                return ERROR;
                        case ZFILE:
                                break;
                        }
                        continue;
                default:
```

```
                        return c;
                case ERROR:
                        return ERROR;
                }
        }
}

/*
 * Receive a file with ZMODEM protocol
 *  Assumes file name frame is in secbuf
 */
rzfile()
{
        register c, n;
        long rxbytes;

        Eofseen=FALSE;
        if (procheader(secbuf) == ERROR) {
                return (tryzhdrtype = ZSKIP);
        }

        n = 20; rxbytes = 0l;

        for (;;) {
#ifdef SEGMENTS
                chinseg = 0;
#endif
                stohdr(rxbytes);
                zshhdr(4,ZRPOS, Txhdr);
nxthdr:
                switch (c = zgethdr(Rxhdr, 0)) {
                default:
                        vfile("rzfile: zgethdr returned %d", c);
                        return ERROR;
                case ZNAK:
                case TIMEOUT:
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
                        if ( --n < 0) {
                                vfile("rzfile: zgethdr returned %d", c);
                                return ERROR;
                        }
                case ZFILE:
                        zrdata(secbuf, 1024);
                        continue;
                case ZEOF:
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
                        if (rclhdr(Rxhdr) != rxbytes) {
                                /*
                                 * Ignore eof if it's at wrong place - force
                                 *  a timeout because the eof might have gone
                                 *  out before we sent our zrpos.
                                 */
                                errors = 0;  goto nxthdr;
                        }
                        if (closeit()) {
                                tryzhdrtype = ZFERR;
```

```
                        vfile("rzfile: closeit returned <> 0");
                        return ERROR;
                }
                vfile("rzfile: normal EOF");
                return c;
        case ERROR:   /* Too much garbage in header search error */
#ifdef SEGMENTS
                putsec(secbuf, chinseg);
                chinseg = 0;
#endif
                if ( --n < 0) {
                        vfile("rzfile: zgethdr returned %d", c);
                        return ERROR;
                }
                zmputs(Attn);
                continue;
        case ZSKIP:
#ifdef SEGMENTS
                putsec(secbuf, chinseg);
                chinseg = 0;
#endif
                Modtime = 1;
                closeit();
                vfile("rzfile: Sender SKIPPED file");
                return c;
        case ZDATA:
                if (rclhdr(Rxhdr) != rxbytes) {
                        if ( --n < 0) {
                                return ERROR;
                        }
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
                        zmputs(Attn);  continue;
                }
moredata:
                if (Verbose>1)
                        fprintf(stderr, "\r%7ld ZMODEM%s    ",
                          rxbytes, Crc32r?" CRC-32":"");
#ifdef SEGMENTS
                if (chinseg >= (1024 * SEGMENTS)) {
                        putsec(secbuf, chinseg);
                        chinseg = 0;
                }
                switch (c = zrdata(secbuf+chinseg, 1024))
#else
                switch (c = zrdata(secbuf, 1024))
#endif
                {
                case ZCAN:
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
                        vfile("rzfile: zgethdr returned %d", c);
                        return ERROR;
                case ERROR:   /* CRC error */
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
```

```
                        if ( --n < 0) {
                                vfile("rzfile: zgethdr returned %d", c);
                                return ERROR;
                        }
                        zmputs(Attn);
                        continue;
                case TIMEOUT:
#ifdef SEGMENTS
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#endif
                        if ( --n < 0) {
                                vfile("rzfile: zgethdr returned %d", c);
                                return ERROR;
                        }
                        continue;
                case GOTCRCW:
                        n = 20;
#ifdef SEGMENTS
                        chinseg += Rxcount;
                        putsec(secbuf, chinseg);
                        chinseg = 0;
#else
                        putsec(secbuf, Rxcount);
#endif
                        rxbytes += Rxcount;
                        stohdr(rxbytes);
                        zshhdr(4,ZACK, Txhdr);
                        sendline(XON);
                        goto nxthdr;
                case GOTCRCQ:
                        n = 20;
#ifdef SEGMENTS
                        chinseg += Rxcount;
#else
                        putsec(secbuf, Rxcount);
#endif
                        rxbytes += Rxcount;
                        stohdr(rxbytes);
                        zshhdr(4,ZACK, Txhdr);
                        goto moredata;
                case GOTCRCG:
                        n = 20;
#ifdef SEGMENTS
                        chinseg += Rxcount;
#else
                        putsec(secbuf, Rxcount);
#endif
                        rxbytes += Rxcount;
                        goto moredata;
                case GOTCRCE:
                        n = 20;
#ifdef SEGMENTS
                        chinseg += Rxcount;
#else
                        putsec(secbuf, Rxcount);
#endif
                        rxbytes += Rxcount;
                        goto nxthdr;
                }
        }
}
```

```
}

/*
 * Send a string to the modem, processing for \336 (sleep 1 sec)
 *   and \335 (break signal)
 */
zmputs(s)
char *s;
{
        register c;

        while (*s) {
                switch (c = *s++) {
                case '\336':
                        sleep(1); continue;
                case '\335':
                        sendbrk(); continue;
                default:
                        sendline(c);
                }
        }
}

/*
 * Close the receive dataset, return OK or ERROR
 */
closeit()
{
        time_t time();

#ifndef vax11c
        if (Topipe) {
                if (pclose(fout)) {
                        return ERROR;
                }
                return OK;
        }
#endif
        if (fclose(fout)==ERROR) {
                fprintf(stderr, "file close ERROR\n");
                return ERROR;
        }
#ifndef vax11c
        if (Modtime) {
                timep[0] = time(NULL);
                timep[1] = Modtime;
                utime(Pathname, timep);
        }
#endif
        if ((Filemode&S_IFMT) == S_IFREG)
                chmod(Pathname, (07777 & Filemode));
        return OK;
}

/*
 * Ack a ZFIN packet, let byegones be byegones
 */
ackbibi()
{
        register n;

        vfile("ackbibi:");
```

```
	Readnum = 1;
	stohdr(0L);
	for (n=3; --n>=0; ) {
		purgeline();
		zshhdr(4,ZFIN, Txhdr);
		switch (readline(100)) {
		case 'O':
			readline(1);	/* Discard 2nd 'O' */
			vfile("ackbibi complete");
			return;
		case RCDO:
			return;
		case TIMEOUT:
		default:
			break;
		}
	}
}



/*
 * Local console output simulation
 */
bttyout(c)
{
	if (Verbose || Fromcu)
		putc(c, stderr);
}

#ifndef vax11c
/*
 * Strip leading ! if present, do shell escape.
 */
sys2(s)
register char *s;
{
	if (*s == '!')
		++s;
	return system(s);
}
/*
 * Strip leading ! if present, do exec.
 */
exec2(s)
register char *s;
{
	if (*s == '!')
		++s;
	mode(0);
	execl("/bin/sh", "sh", "-c", s);
}
#endif
/* End of rz.c */
```

4. Bezugsdokumente

[1] "Bestzeit Schneller mit Enhanced Y-Modem", Herwig Feichtinger; c't 11/1995, Heise Verlag

[2] "XMODEM/YMODEM PROTOCOL REFERENCE", formattet 10-27-87, Edited by Chuck Forsberg, Omem Technology Inc

[3] "The ZMODEM Inter Application File Transfer Protocol", Rev 8-3-87, Chuck Forsberg Omem Technology Inc

[4] C-Sourcen einer ZMODEM Implementierung, Datei "RZSZ0589.ARC" vom 29.06.90, Chuck Forsberg, Omen Technology Inc.

[5] "DFÜ & BTX", Christian Spanik und Hannes Rügheimer, Markt&Technik Buch- und Software-Verlag GmbH, 1994

Einreicherkarte

Version 2.2
09.08.1995, Änderungsdatum: 22.01.1997

Inhalt

1. Einleitung

Die ***Einreicherkarte*** bietet denjenigen Händlern, die über keinen Online-Anschluß an ihren Händlersystemen verfügen, die Möglichkeit, die von der Händler-Evidenzzentrale erwartete BZAHL-Datei auf diese Einreicherkarte zu sichern, um sie dann bei einem Einreicherterminal, beispielsweise ihrer Händlerbank, einzureichen.

Die Einreicherkarte ist eine Chipkarte, deren Struktur der Daten, Sicherheitsarchitektur sowie Standard- und Administrationskommandos der Spezifikation in [LIT 1] entsprechen, und die die Applikation EINREICHERKARTE bestehend aus dem Applikationsverzeichnis (ADF) und den zugeordneten AEFs enthält. Das ADF der Applikation EINREICHERKARTE wird mit DF_EINREICH bezeichnet.

Spezifische Ergänzungskommandos für die Funktionen der Einreicherkarte werden nicht benötigt. Das Schreiben der Datensätze einer BZAHL-Datei sowie die Einreichung am Einreichungsterminal wird mit in [LIT 1] spezifizierten Standardkommandos durchgeführt.

Es hängt von der Speicherkapazität des EEPROM der Einreicherkarte ab, wie groß die maximale Anzahl von Datensätzen einer zu transportierenden BZAHL-Datei ist. An einem Händlersystem, das mit Einreicherkarten arbeitet, muß diese maximale Anzahl einstellbar sein, so daß flexibel auf eine Veränderung der Aufnahmekapazität der verwendeten Einreicherkarten reagiert werden kann.

2. Daten der Einreicherkarte

Die Einreicherkarte enthält, außer ggf. einem K_{RAND}, keine kryptographischen Schlüssel oder Paßwörter, so daß nur die Zugriffsbedingungen ALW und NEV für die Dateien der Einreicherkarte verwendet werden können.

Für das MF und das DF_EINREICH wird hierbei die ADMIN-AC NEV gesetzt. Da die Einreicherkarte nur das MF und das DF_EINREICH enthält, sind hiermit die Administrationskommandos CREATE FILE, DELETE FILE, INCLUDE und EXCLUDE für die Einreicherkarte gesperrt, so daß alle Dateien der Einreicherkarte bei der Initialisierung der Einreicherkarte angelegt bzw. in die Applikation eingebunden werden müssen.

Da die Einreicherkarte insbesondere keinen K_{Card} enthält (Schlüsselnummer '00' im EF_KEY des

MF), ist für die Einreicherkarte auch das Kommando LOAD COMMAND gesperrt, so daß das nachträgliche Einbringen oder Ändern von Kommandos nicht möglich ist.

Das MF der Einreicherkarte muß die folgenden EFs enthalten:

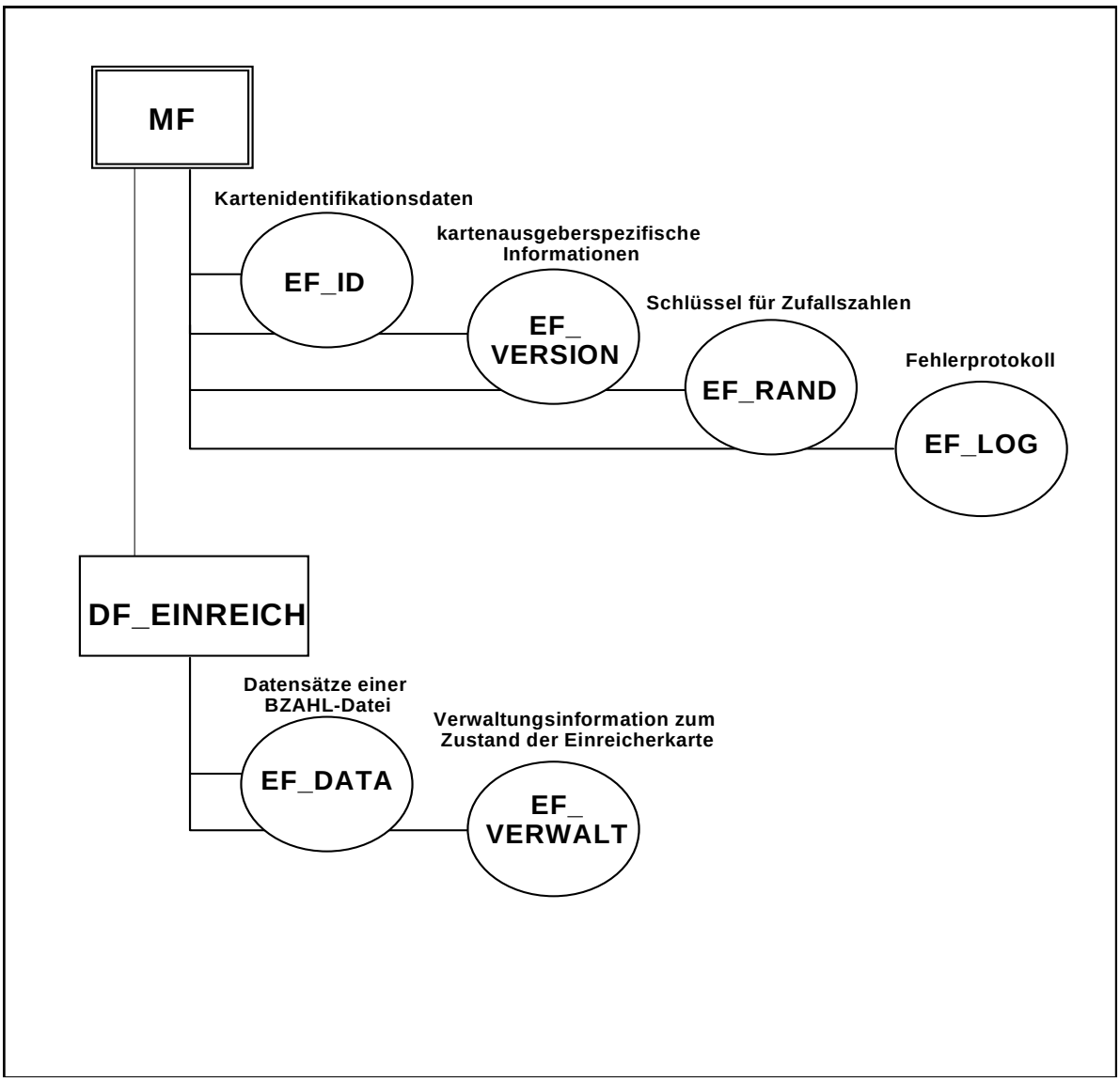
- das EF_ID und
- das EF_RAND, wenn der Zufallszahlengenerator wie in Kapitel 2.6 in [LIT 1] beschrieben realisiert wird.

Das MF der Einreicherkarte kann das EF_LOG und/oder das EF_VERSION enthalten.

Aufbau und Inhalt des EF_ID, EF_LOG und EF_RAND sowie der Aufbau des EF_VERSION im MF entsprechen den Spezifikationen in Kapitel 6. von [LIT 1]. In dem Record des EF_VERSION können Informationen zur Einreicherkarte abgelegt werden.

Im folgenden werden die Datei-Kontrollinformation des MF, des EF_ID, EF_LOG, EF_RAND und EF_VERSION im MF sowie Aufbau, Inhalt und Datei-Kontrollinformation des DF_EINREICH und der darin enthaltenen EFs beschrieben.

Die folgende Grafik gibt eine Übersicht über die Dateien der Einreicherkarte.



2.1. FCP des MF und der im MF enthaltenen EFs

2.1.1. FCP des MF

Für das MF der Einreicherkarte sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'11'		
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'3F 00'	Datei-ID des MF
'84'	'04'	'52 4F 4F 54'	DF-Name des MF
'86'	'02'	'00 F0'	ADMIN-AC für das MF

2.1.2. FCP des EF_ID im MF

Für das EF_ID im MF sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		
'81'	'02'	'00 16'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 16'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 03'	Datei-ID des EF_ID
'86'	'06'	'00 F0 00 00 00 F0'	ACs für das EF_ID

Erläuterung

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_ID die AC '00 F0' (NEV) festgelegt, so daß die Kartenidentifikationsdaten der Einreicherkarte personalisiert werden müssen und danach nicht mehr verändert werden können.

Für das Kommando READ RECORD wird für das EF_ID die AC '00 00' (ALW) festgelegt, so daß die Kartenidentifikationsdaten der Einreicherkarte frei lesbar sind.

2.1.3. FCP des EF_LOG im MF

Für das EF_LOG im MF sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		
'81'	'02'	'00 1E'	allokierter Speicherplatz in Byte
'82'	'03'	'06 41 0A'	Datei-Deskriptor für zyklisches EF
'83'	'02'	'00 04'	Datei-ID des EF_LOG
'86'	'06'	'00 F0 00 00 00 F0'	ACs für das EF_LOG

Erläuterung

Für die Kommandos APPEND RECORD und UPDATE RECORD wird für das EF_LOG die AC '00 F0' (NEV) festgelegt, so daß das EF_LOG nur durch die interne Schreibroutine zur Fehlerprotokollierung verändert werden kann.

Für das Kommando READ RECORD wird für das EF_LOG die AC '00 00' (ALW) festgelegt, so daß es frei lesbar ist.

2.1.4. FCP des EF_RAND im MF

Für das EF_RAND im MF sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
'62'	'15'		
'81'	'02'	'00 08'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 08'	Datei-Deskriptor für lineares EF
'83'	'02'	'00 05'	Datei-ID des EF_RAND
'86'	'06'	'00 F0 00 F0 00 F0'	ACs für das EF_RAND

Erläuterung

Für APPEND RECORD, READ RECORD und UPDATE RECORD wird für das EF_RAND die AC '00 F0' (NEV) festgelegt, so daß ein Schlüssel zur Bildung von Zufallszahlen nur bei der Personalisierung in die Einreicherkarte eingebracht und danach weder gelesen noch verändert werden kann.

2.1.5. FCP des EF_VERSION im MF

Für das EF_VERSION sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		
’81’	’02’	’00 08’	allokierter Speicherplatz in Byte
’82’	’03’	’02 41 08’	Datei-Deskriptor für lineares EF
'83'	'02'	'00 17'	Datei-ID des EF_VERSION
'86'	'06'	'00 00 00 00 00 00'	ACs für das EF_VERSION

Erläuterung

Für APPEND RECORD, READ RECORD und UPDATE RECORD wird für das EF_VERSION die AC '00 00' (ALW) festgelegt, so daß die Informationen zur Einreicherkarte frei lesbar und änderbar sind.

2.2. ADF der Applikation EINREICHERKARTE

Für das ADF der Applikation EINREICHERKARTE (DF_EINREICH) sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’16’		Tag und Länge für FCP
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A4 00'	Datei-ID des DF_EINREICH
'84'	'09'	'D2 76 00 00 25 45 4B 01 00'	DF-Name (AID) des DF_EINREICH
'86'	'02'	'00 F0'	AC für das DF_EINREICH

Wenn das DF_EINREICH mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FCP ausgegeben:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’1A’		Tag und Länge für FCP
'81'	'02'	'XX XX'	freier Speicherplatz in der Einreicherkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A4 00'	Datei-ID des DF_EINREICH
'84'	'09'	'D2 76 00 00 25 45 4B 01 00'	DF-Name (AID) des DF_EINREICH
'86'	'02'	'00 F0'	AC für das DF_EINREICH

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

In zwei Byte wird der in der Einreicherkarte für das Anlegen weiterer Dateien zur Verfügung stehende freie Speicherplatz in Byte angegeben.

Tag ’86’

Für die Kommandogruppe ADMIN wird für das DF_EINREICH die AC '00 F0' (NEV) festgelegt. Wenn das DF_EINREICH selektiert ist, kann also ein CREATE FILE, DELETE FILE, INCLUDE oder EXCLUDE nicht ausgeführt werden.

Der Applikation EINREICHERKARTE sind bei der Initialisierung 2 Dateien als AEFs zuzuordnen:

SFI '14': EF_DATA im DF_EINREICH,

SFI '15': EF_VERWALT im DF_EINREICH.

Wenn das DF_EINREICH mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, werden die folgenden FMD mit den Pfaden der AEFs ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_EINREICH direkt im MF befindet):

Tag	**Länge** (in Byte)	**Wert**	**Erläuterung**
’64’	’0E’		Tag und Länge für FMD
'85'	'05'	'14 A4 00 02 06'	Pfad für das AEF mit SFI '14' (EF_DATA im DF_EINREICH)
'85'	'05'	'15 A4 00 02 05'	Pfad für das AEF mit SFI '15' (EF_VERWALT im DF_EINREICH)

Wenn das DF_EINREICH mittels SELECT FILE selektiert wird und die entsprechende Option im Parameterbyte P2 des Kommandos gesetzt ist, wird die folgende FCI mit den ACs der AEFs im zusammengesetzten Datenobjekt mit Tag ’A5’ ausgegeben (hierbei wird vorausgesetzt, daß sich das DF_EINREICH direkt im MF befindet):

Tag	Länge (in Byte)	Wert	Erläuterung
’6F’	’2E’		Tag und Länge für FCI
'81'	'02'	'XX XX'	freier Speicherplatz in der Einreicherkarte in Byte
'82'	'01'	'38'	Datei-Deskriptor für DF
'83'	'02'	'A4 00'	Datei-ID des DF_EINREICH
'84'	'09'	'D2 76 00 00 25 45 4B 01 00'	DF-Name (AID) des DF_EINREICH
'86'	'02'	'00 F0'	AC für das DF_EINREICH
'A5'	'12'		Tag und Länge für ACs der AEFs
'86'	'07'	'14 00 00 00 00 00 00'	AC für das AEF mit SFI '14' (EF_DATA im DF_EINREICH)
'86'	'07'	'15 00 00 00 00 00 00'	AC für das AEF mit SFI '15' (EF_VERWALT im DF_EINREICH)

2.3. EF_VERWALT

Das EF_VERWALT enthält einen Record mit Verwaltungsinformationen über den derzeitigen Zustand der Einreicherkarte bezüglich :

- Schreiben aller Datensätze einer BZAHL-Datei durch ein Händlersystem
- Lesen aller Datensätze einer BZAHL-Datei durch ein Einreichungsterminal

2.3.1. FCP

Für das EF_VERWALT des DF_EINREICH sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'00 0B'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 0B'	Datei-Deskriptor für lineares EF
'83'	'02'	'02 05'	Datei-ID des EF_VERWALT
'86'	'06'	'00 00 00 00 00 00'	ACs für das EF_VERWALT

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Das EF_VERWALT enthält maximal 1 Record der Länge 11 Byte, so daß 11 Byte für die Nutzdaten

des EF_VERWALT zu allokieren sind.

Tag ’82’

Das EF_VERWALT ist ein lineares EF, dessen Recordlänge auf 11 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_VERWALT ist '02 05'.

Tag ’86’

Für die Kommandos APPEND RECORD, READ RECORD und UPDATE RECORD wird die AC '00 00' (ALW) festgelegt. Der Record des EF ist also frei lesbar und schreibbar.

2.3.2. Daten

Der Record mit der Recordnummer '01' des EF_VERWALT enthält Informationen über das letzte Schreiben (Lesen) der Datensätze einer BZAHL-Datei im EF_DATA durch ein Händlersystem (Einreicherterminal):

Byte	Länge (in Byte)	Wert	Erläuterung
1	1	’XX’	Statusbyte
2-5	4	JJJJ MM TT	Datum des letzten Schreibens/Lesens einer BZAHL-Datei
6-8	3	HH MM SS	Uhrzeit des letzten Schreibens/Lesens einer BZAHL-Datei
9-11	3	’nn nn nn’	Anzahl Datensätze der BZAHL-Datei

Im folgenden werden die Komponenten des Records näher erläutert:

Byte 1

Das Statusbyte zeigt an, welcher Schritt der Übertragung einer BZAHL-Datei von einem Händlersystem an ein Einreichungsterminal zuletzt ausgeführt wurde. Das Statusbyte kann die folgenden Werte annehmen:

Statusbyte	Bedeutung
’00’	Alle Datensätze einer BZAHL-Datei wurden in das EF_DATA geschrieben
'01'	Alle Datensätze der BZAHL-Datei wurden aus dem EF_DATA gelesen

Nur wenn das Statusbyte den Wert '01' hat, wird durch ein Händlersystem eine neue BZAHL-Datei in das EF_DATA der Einreicherkarte geschrieben. Erst wenn die Datei komplett geschrieben

wurde, wird das Statusbyte mittels UPDATE RECORD zu ’00’ gesetzt.

Eine BZAHL-Datei wird nur dann durch ein Einreichungsterminal gelesen, wenn das Statusbyte den Wert ’00’ hat. Wenn die gesamte BZAHL-Datei gelesen ist, setzt das Einreichungsterminal das Statusbyte mittels UPDATE RECORD zu ’01’.

Das Statusbyte muß mit dem Wert '01' initialisiert werden.

Byte 2-8

Byte 2-8 enthalten Datum und Uhrzeit des letzten Schreibens/Lesens einer BZAHL-Datei. Datum und Uhrzeit können mit 0 belegt werden, wenn das schreibende/lesende Terminal über keine Uhr verfügt. Datum und Uhrzeit werden zusammen mit dem jeweiligen Statusbyte in das EF_VERWALT mittels UPDATE RECORD eingetragen, wenn das Lesen/Schreiben erfolgreich durchgeführt wurde.

Byte 9-11

In Byte 9-11 wird die Gesamtzahl der Datensätze beim Schreiben der BZAHL-Datei eingetragen. Diese Zahl bleibt bei dem UPDATE des Records nach einem erfolgreichen Lesen unverändert.

Byte 2-11 des EF_VERWALT werden mit ’00..00’ initialisiert.

2.4. EF_DATA

In die Records des EF_DATA werden die Datensätze einer kompletten BZAHL-Datei eingetragen. Eine BZAHL-Datei besteht aus jeweils 80 Byte langen Datensätzen:

- Vor-Satz,
- einem oder mehreren S- und/oder M-Sätzen mit zugehörigen Z- und/oder F-Sätzen,
- Ende-Satz.

Das Format der Datensätze einer BZAHL-Datei ist in Kapitel 2.4 von [LIT 4E] beschrieben.

2.4.1. FCP

Für das EF_DATA des DF_EINREICH sind bei der Initialisierung die folgenden FCP festzulegen:

Tag	Länge (in Byte)	Wert	Erläuterung
’62’	’15’		Tag und Länge für FCP
'81'	'02'	'XX XX'	allokierter Speicherplatz in Byte
'82'	'03'	'02 41 50'	Datei-Deskriptor für lineares EF
'83'	'02'	'02 06'	Datei-ID des EF_DATA
'86'	'06'	'00 00 00 00 00 00'	ACs für das EF_DATA

Im folgenden wird die Belegung einzelner Datenobjekte näher erläutert:

Tag ’81’

Für das EF_DATA wird der gesamte nach Anlegen aller anderen Dateien verbleibende Speicherplatz im EEPROM allokiert. Von der Größe dieses Speicherplatzes ist es abhängig, wieviele Records im EF_DATA angelegt werden können bzw. wieviele Datensätze eine mit der Einreicherkarte zu transportierende BZAHL-Datei maximal haben darf.

Tag ’82’

Das EF_DATA ist ein lineares EF, dessen Recordlänge auf 80 Byte festgelegt ist.

Tag ’83’

Die Datei-ID des EF_DATA ist '02 06'.

Tag ’86’

Für die Kommandos APPEND RECORD, READ RECORD und UPDATE RECORD wird die AC '00 00' (ALW) festgelegt. Die Records des EF sind also frei lesbar und schreibbar.

2.4.2. Daten

Das Format der Daten in den einzelnen Records ist identisch mit dem Format der Datensätze der BZAHL-Datei gemäß [LIT 4E]. Der Vor-Satz kann die in [LIT 4E] beschriebene vereinfachte Form haben, wenn die BZAHL-Datei nur Daten einer Händlerkarte enthält.

Alle Records des EF_DATA müssen mit '00..00' initialisiert werden, da andernfalls ein Schreiben mit UPDATE RECORD nicht möglich ist.

3. Abläufe

3.1. Kurzbeschreibung

Auf die Einreicherkarte wird nur eine vollständige BZAHL-Dateien im Format gemäß [LIT 4E] geschrieben. Wenn eine Einreicherkarte Platz für n Datensätze bietet, muß daher spätestens nach n-3 Transaktionen ein Kassenschnitt erfolgen, durch den als (n-2)-ter Datensatz der zugehörige Summensatz erzeugt wird. Als (n-1)-ter und n-ter Datensatz sind der Vor-Satz und der Ende-Satz durch das Händlersystem zu erzeugen.

Auch wenn nicht nach jeder (n-3)-ten Transaktion eine Einreicherkarte beschrieben werden soll, muß ein Summensatz erzeugt werden und mit den Einzeltransaktionen gespeichert werden, so daß zu einem späteren Zeitpunkt eine BZAHL-Datei entsprechender Größe erzeugt und auf eine Einreicherkarte geschrieben werden kann.

Das Verteilen von BZAHL-Dateien auf mehrere Einreicherkarten (Folgekarten) ist nicht vorgesehen.

An einem Händlersystem, das die Einreichung von BZAHL-Dateien mittels Einreicherkarten unterstützt, muß die maximale Anzahl der in der Einreicherkarte speicherbaren Datensätze über Parameter einstellbar sein, so daß auf eine Veränderung der Aufnahmekapazität der verwendeten Einreicherkarten reagiert werden kann.

In Abhängigkeit von dem eingestellten Wert muß das Händlersystem nach dem oben beschriebenen Prinzip Kassenschnitte durchführen.

3.2. Beschreiben der Einreicherkarte

Nachdem eine Einreicherkarte in das Händlersystem eingegeben wurde und ein Reset der Karte erfolgt ist, wird das DF_EINREICH durch das Händlersystem selektiert.

Das Händlersystem liest mittels READ RECORD den Record des EF_VERWALT (SFI '15'). Wenn das Statusbyte nicht den Wert '01' hat, bricht das Händlersystem den Vorgang mit einer Warnung ab.

Wenn das Statusbyte den Wert '01' hat, wird wie folgt verfahren:

- Das Händlersystem kann die auf der Einreicherkarte gespeicherte BZAHL-Datei analysieren, um festzustellen, ob eine noch im Händlersystem gespeicherte BZAHL-Datei durch ein Einreicherterminal quittiert wurde.

- Sofern dies nicht bereits bei der Erzeugung des Summensatzes erfolgt ist, werden Vor- und Ende-Satz zu dem ersten seit der letzten Ausgabe von BZAHL-Dateien erzeugten Summensatz und den zugehörigen Einzeltransaktionen generiert. Die komplette BZAHL-Datei wird auf die Einreicherkarte geschrieben. Hierzu wird der Vor-Satz der BZAHL-Datei mittels UPDATE RECORD als Record '01' des EF_DATA (SFI '14') geschrieben. Es folgt der erste Summensatz in Record '02' des EF_DATA. Sukzessive werden die Datensätze der BZAHL-Datei in die Records des EF_DATA eingebracht. Hierbei ist die korrekte Reihenfolge der Datensätze einer BZAHL-Datei zu berücksichtigen. Als letzter Record wird der Ende-Satz geschrieben.

- Wenn alle Datensätze der BZAHL-Datei erfolgreich geschrieben wurden, wird der Record

des EF_VERWALT aktualisiert. Hierbei wird das Statusbyte zu ’00’ gesetzt. Datum und Uhrzeit erhalten die aktuellen Werte. Datum und Uhrzeit können mit 0 belegt werden, wenn das schreibende/lesende Terminal über keine Uhr verfügt. Die Gesamtzahl der Datensätze (und neu beschriebenen Records des EF_DATA) wird eingetragen.

Wenn seit der letzten Ausgabe von BZAHL-Dateien mehrere Summensätze bzw. BZAHL-Dateien erzeugt wurden und nun ausgelesen werden müssen, muß das Händlersystem zur Eingabe weiterer Einreicherkarten auffordern und wie oben beschrieben fortfahren.

Für die Einzeltransaktionen, zu denen noch kein Summensatz existiert, kann ebenfalls ein Summensatz und eine BZAHL-Datei erzeugt werden, um diese in einer Einreicherkarte zu speichern.

3.3. Auslesen der Einreicherkarte

Nachdem eine Einreicherkarte in ein Einreicherterminal eingegeben wurde und ein Reset der Karte durchgeführt wurde, selektiert das Einreicherterminal das DF_EINREICH.

Das Einreicherterminal liest mittels READ RECORD den Record des EF_VERWALT (SFI '15'). Wenn das Statusbyte nicht den Wert '00' hat, bricht das Einreicherterminal den Vorgang ab.

Wenn das Statusbyte den Wert '00' hat, wird wie folgt verfahren:

- Sukzessive werden die Datensätze der gespeicherten BZAHL-Datei aus den Records des EF_DATA (SFI '14') mittels READ RECORD gelesen. Die Anzahl der zu lesenden Records ist dem EF_VERWALT zu entnehmen.

  Der letzte gelesene Satz muß der Ende-Satz sein.

- Wenn alle Datensätze der BZAHL-Datei erfolgreich gelesen wurden, wird der Record des EF_VERWALT aktualisiert. Hierbei wird das Statusbyte zu '01' gesetzt. Datum und Uhrzeit erhalten die aktuellen Werte. Datum und Uhrzeit können mit 0 belegt werden, wenn das schreibende/lesende Terminal über keine Uhr verfügt. Die Gesamtzahl der Datensätze bleibt unverändert.

Key-Management

Version 2.2
22.01.1997, Ausgabedatum: 27.10.95

Inhalt

1. Änderungshistorie

Version 1.1	Ausgabe am 15.12.1994
	Beschreibung der Schlüssel für die ec-Karte mit Chip
Version 2.0	Ausgabe am 06.03.1995

Abschnitt 2.: Ladezentrale bei Begriffsbestimmungen aufgenommen,

Abschnitt 4.2.: Im Falle von Online-Verbindungen wird der KGK_{Card} im Sonderfunktionsterminal nicht gespeichert.

Abschnitt 4.3.: Änderung in der Zeile Key User: K_{PIN} wird nicht an Terminalhersteller ausgeliefert. Terminals berechnen K_{PIN} aus $KGK_{PIN.}$

Abschnitt 4.8.: Überschrift geändert.

Abschnitt 6.: Schlüssel für die elektronische Geldbörse.

Version 2.1 Ausgabe am 17.05.1995

Änderung der Begriffsbestimmung für 'Kartenherausgeber'

Abschnitt 3.1.: Änderung der Bezeichnung des Initialwertes von K0 in I, Präzisierung der Triple-DES Entschlüsselung bei der Schlüsselerzeugung, Festlegung der Reihenfolge der Bits bei der Verschlüsselung, neue Nomenklatur

Abschnitt 3.2.: Korrektur der schwachen Schlüssel, Abschnitt 3.2. als optional gekennzeichnet

Abschnitt 7.: neu eingefügt, um allgemeine Schlüssel zu behandeln; der K_{PS} wurde aus Abschnitt 4.9 in dieses Kapitel übernommen, da es sich nicht um einen kartenindividuellen Schlüssel handelt; K_{SIG} und K_{Pers} sind neu hinzugekommen;

Version 2.1.1 Ausgabe am 18.08.1995

Der Algorithmus für die Schlüsselerzeugung wurde überarbeitet.

Wichtige Änderungen:

d*KGK bezeichnet die Triple-DES-Entschlüsselung im CBC-Mode mit ICV = 0

Die Transformationen Ad01 und Ad10 wurden um das Parity Adjustment ergänzt.

Neue Schlüssel K_{CD}, KGK_{CD}, K_{LT}, KGK_{LT}.

Die Verwendung von Hash-Werten als kryptographische Checksummen.

Version 2.1.2 Ausgabe am 08.09.1995

Abschnitt 3.1.: Der Algorithmus für die Schlüsselerzeugung wurde

überarbeitet: d*KGK bezeichnet die Triple-DES-Entschlüsselung im **ECB**-Mode. Die Ergänzung der Transformationen Ad01 und Ad10 um das Parity Adjustment ist **optional**.

Abschnitt 3.3.: Die kryptographische Checksumme wird über den **Klartext**-Schlüssel berechnet.

Abschnitt 4.: Die verschiedenen Typen von Chipkarten und Terminals werden berücksichtigt.

Abschnitt 4.5.: Der K_{INFO} dient auch der Authentikation von Chipkartendaten.

Abschnitt 5.: Die K_{AK} sind im EF_AUT der Anwendung ec cash gespeichert. Jeder Terminalhersteller erhält genau einen der 10 Key Generating Keys KGK_{RK} und KGK_{AK} (Beschluß 33).

Abschnitt 5.7.: Ein K_{LK} wird aus einem **der beiden** KGK_{LK} abgeleitet.

Abschnitt 6.: Die Erläuterung der Schlüssel K_{CD}, KGK_{CD}, K_{CAM} und KGK_{CAM} wurde aus dem Kapitel 6.1. ausgegliedert und in das neue Kapitel 6.3. übernommen, da diese Schlüssel sowohl in der Börsen- als auch in der Händlerkarte vorhanden sind.

Abschnitt 6.1.2.: Ein K_{LD}wird aus einem **der beiden** KGK_{LD} abgeleitet.

Abschnitt 6.1.5.: Die Erläuterung des KGK_{LT} wurde überarbeitet. Insbesondere erhält ein Ladeterminalhersteller genau einen der 10 KGK_{LT} (Beschluß 33).

Abschnitt 6.2.1.: Ein K_{ZD}wird aus einem **der beiden** KGK_{ZD} abgeleitet.

Abschnitt 6.3.: Sowohl K_{CD} als auch K_{CAM} sind in Börsen- und Händlerkarte vorhanden.

Abschnitt 6.3.2.: Die Erläuterung des KGK_{CD} wurde überarbeitet. Insbesondere ist ein KGK_{CD} ausschließlich in Bankensonderfunktionsterminals vorhanden.

Version 2.1.3 — Ausgabe am 27.10.1995

Abschnitt 3.4.: Verfahren zum Transport und zur Ausgabe von Masterkeys wurden festgelegt.

Version 2.2. — Ausgabe am 22.01.1997

Anpassung des Versionsstands an die anderen Spezifikationsdokumente. Keine inhaltlichen Änderungen.

2. Begriffsbestimmungen

Generierungsstelle: Eine Instanz, in deren Verantwortung ein Masterkey (KGK) erzeugt wird und die aus Masterkeys und kartenindividuellen Identifikationsdaten kartenspezifische Schlüssel und die Personalisierungsdaten erzeugt. Für das deutsche Kreditgewerbe sind dies die Verlage.

Personalisierung: Vorgang, bei dem kartenspezifische Daten in die vorgefertigten Karten eingebracht werden. Insbesondere werden auch die kartenindividuellen Schlüssel bei der Personalisierung in die Karten geladen.

Personalisierungsstelle : Eine Instanz, die die Personalisierung vornimmt.

Initialisierung: Vorgang, bei dem die benutzerunabhängigen Daten in die Karten eingebracht werden.

Terminalhersteller: Instanz, die Masterkeys aus dem Verantwortungsbereich einer Generierungsstelle auf sicherem Wege in die Sicherheitsmodule von Endgeräten einbringt.

Autorisierungsstelle: Instanz, die aus Masterkeys und von der Karte in einer Autorisierungsanfrage übermittelten Kartenidentifikationsdaten kartenspezifische Schlüssel berechnet, mit deren Hilfe Antwortnachrichten kryptographisch abgesichert werden.

Ladezentrale: Instanz, die nach einer Online-Autorisierung, durch die die Verfügungsmöglichkeit des Karteninhabers geprüft wird, das Laden einer Börsenkarte vornimmt. Ladezentralen sind die Autorisierungssysteme des Kreditgewerbes.

Kartenherausgeber: Verlage = die vom Kartenherausgeber und / oder ZKA bzw. den jeweiligen Verbänden beauftragte Institution

3. Verfahren zur Schlüsselerzeugung

Die kartenindividuellen Schlüssel für die ec-Karte mit Chip werden aus Kartenidentifikationsdaten (CID) und Masterkeys (Key Generating Keys KGK) erzeugt. Die Masterkeys werden ihrerseits durch eine Generierungsstelle des Kreditgewerbes erzeugt und an die verantwortlichen Stellen

übergeben. Die Übergabe aller Schlüssel erfolgt gesichert, indem diese mit Transportschlüsseln (Key Encryption Keys) verschlüsselt werden und die Transportschlüssel jeweils in 2 Hälften getrennt verwaltet werden.

3.1. Algorithmus für die Schlüsselerzeugung:

Ein kartenindividueller Schlüssel KK von 16 Byte Länge wird aus

- KGK (16 Byte),
- CID (vollständiger Inhalt von EF_ID des MF, mit '00' auf das nächste Vielfache von 8 Byte Länge gepaddet) und
- dem öffentlich bekannten Initialwert

  I = '52 52 52 52 52 52 52 52 25 25 25 25 25 25 25 25' (16 Byte)

zu

KK = P(d*KGK(H(I,CID))).

berechnet. Hierbei bezeichnen

- **P** die Funktion "Parity Adjustment", die wie folgt definiert ist:

  Sei b_1,..,b_8 die Darstellung eines Byte als Folge von 8 Bit. Dann setzt P das niedrigstwertige Bit b_8 jedes Byte auf ungerade Parität, d. h. b_8 wird in jedem Byte so gesetzt, daß es eine ungerade Anzahl von 1 enthält.

- **d*KGK** die Triple-DES Entschlüsselung im ECB-Mode mit dem Schlüssel KGK,
- **H** die in ISO 10118-2 ([ISO 10]) definierte Hash-Funktion, die Werte X mit einer Länge, die ein Vielfaches von 8 Byte ist, mittels des Startwertes I (gemäß Anhang A von ISO 10118-2) auf einen Wert von 16 Byte Länge abbildet. Es werden die um das Parity Adjustment P erweiterten Transformationen u (Ad10) und u' (Ad 01) aus Anhang A von ISO 10118-2 verwendet. H ist rekursiv definiert:
  - Sei $X = x_1|...|x_n$ die Zerlegung des Wertes X in 8 Byte lange Blöcke und $L_0|R_0$ die Zerlegung des vorgegebenen Startwerts I in zwei 8 Byte Blöcke.
  - eK(X) ist die DES-Verschlüsselung eines 8 Byte Wertes mit einem 8 Byte Schlüssel.
  - $\oplus$ sei die bitweise Addition modulo 2 (XOR).
  - Die Transformationen Ad10 und Ad01 transformieren 8 Byte Werte K wie folgt:

    Sei $K = k_1,...,k_{64}$ die Darstellung von K als Folge von 64 Bit. Dann ist

$Ad10(K) = [P](k_1,1,0,k_4,...,k_{64})$

$Ad01(K) = [P](k_1,0,1,k_4,...,k_{64})$.

[P]: Das Parity Adjustment in Ad10 und Ad01 ist optional. Wenn vor der DES-Verschlüsselung eK(X) keine Paritätsprüfung des Schlüssels K erfolgt, kann P entfallen.

- Dann errechnet sich $L_i|R_i$ aus $L_{i-1}|R_{i-1}$ und x_i wie folgt:

  L_{i-1} bestehe aus den Bits $l_1,...,l_{64}$ und R_{i-1} bestehe aus den Bits $r_1,...,r_{64}$,

  Schritt 1: $L'_i := Ad10(L_{i-1}) = [P](l_1,1,0,l_4,...,l_{64})$

  $R'_i := Ad01(R_{i-1}) = [P](r_1,0,1,r_4,...,r_{64})$

  Schritt 2: $A_i = A_{i[links]}|A_{i[rechts]} = eL'_i(x_i) \oplus x_i$.

  $B_i = B_{i[links]}|B_{i[rechts]} = eR'_i(x_i) \oplus x_i$.

  Schritt 3: $L_i = A_{i[links]}|B_{i[rechts]}$

  $R_i = B_{i[links]}|A_{i[rechts]}$

- $L_n|R_n$ ist dann der Hash-Wert von X unter H:

  $H(I,X) = L_n|R_n$

Soll ein kartenindividueller Schlüssel K von 8 Byte Länge aus KGK, CID und I berechnet werden, wird zuerst der 16 Byte lange Schlüssel KK bestimmt und die linke Hälfte von KK als K verwendet.

3.2. Vermeidung schwacher DES-Schlüssel

Wird bei diesem Verfahren für eine Karte und einen KGK ein 16-Byte-Schlüssel erzeugt, der in der linken 8-Byte-Hälfte einen schwachen oder semi-schwachen Schlüssel enthält, so wird der in den letzten beiden Bytes des EF_ID dieser Karte enthaltene Wert um eins inkrementiert, und die Berechnung der Schlüssel für diese Karte wird von vorne begonnen. Für die Personalisierung wird der tatsächlich verwendete Wert in die beiden letzten Bytes des EF_ID eingestellt.

Die Anwendung dieses Verfahrens ist optional.

Die folgenden DES- Schlüssel sind schwach (Angabe aller Schlüssel mit ungerader Parität):

```
'01 01 01 01 01 01 01 01'
'FE FE FE FE FE FE FE FE'
'1F 1F 1F 1F 0E 0E 0E 0E'
'E0 E0 E0 E0 F1 F1 F1 F1'
```

Die folgenden DES- Schlüssel sind semi-schwach:

```
'01 FE 01 FE 01 FE 01 FE'
'FE 01 FE 01 FE 01 FE 01'
'1F E0 1F E0 0E F1 0E F1'
'E0 1F E0 1F F1 0E F1 0E'
'01 E0 01 E0 01 F1 01 F1'
'E0 01 E0 01 F1 01 F1 01'
'1F FE 1F FE 0E FE 0E FE'
'FE 1F FE 1F FE 0E FE 0E'
'01 1F 01 1F 01 0E 01 0E'
'1F 01 1F 01 0E 01 0E 01'
'E0 FE E0 FE F1 FE F1 FE'
'FE E0 FE E0 FE F1 FE F1'
```

3.3. Die Verwendung von Hash-Werten als kryptographische Checksummen

Die in 3.1. definierte ISO-Hash-Funktion H wird ebenfalls verwendet, um beim Transport von Schlüsseln auftretende Fehler erkennen zu können. Dabei wird der folgende Algorithmus verwendet:

1. Zu einem Klartext-Schlüssel K wird bei der Generierungsstelle der Wert $H_K = H(I,K)$ mit dem Initialwert I = '52 52 52 52 52 52 52 52 25 25 25 25 25 25 25 25' (16 Byte) berechnet und abgespeichert.

2. Der Schlüssel K wird transportverschlüsselt zusammen mit dem Wert H_K an den Empfänger gesendet.

3. Der Empfänger berechnet mit dem zurückgewonnenen Klartext-Schlüssel K' den Wert $H_K' = H(I,K')$ und prüft, ob $H_K = H_K'$ ist. Falls das nicht der Fall ist, liegt ein Übertragungsfehler vor. Im anderen Fall wird die Übertragung als fehlerfrei angesehen.

3.4. Transport und Ausgabe von Schlüsseln

3.4.1. Transport von Schlüsseln

Zum Transport eines Masterkeys wird dieser mit einem Transportschlüssel unter Verwendung des Triple-DES-Algorithmus verschlüsselt. Der Transportschlüssel wird mit Hilfe eines (Pseudo-)

Zufallszahlengenerators von der Sicherheitsbox im Trust Center (TC) erzeugt.

Für jede verschlüsselte Ausgabe wird von der Sicherheitsbox ein neuer Transportschlüssel erzeugt. Es ist jedoch möglich, mehrere Schlüssel für einen Empfänger unter einem Transportschlüssel zu sichern.

Es bezeichne Ti einen 16 Byte langen Transportschlüssel. Jeder Transportschlüssel Ti wird in zwei 16 Byte große *Hälften* Ti_1 und Ti_2 zerlegt, so daß

$$Ti = Ti_1 \oplus Ti_2$$

gilt. Diese Zerlegung wird von der Sicherheitsbox im TC erzeugt. Die Schlüsselhälften Ti_1 und Ti_2 werden auf Papier (zum Format vgl. Kapitel 3.4.2.) ausgegeben und an je einen **Sicherheitsbeauftragten** persönlich übergeben.

Die Sicherheitsbeauftragten sollten sinnvollerweise bei der Ausgabe der Schlüsselhälften zugegen sein, damit die Transportschlüssel auf keinen Fall einem Dritten zur Kenntnis gelangen. Das TC ist für die korrekte Einhaltung der Abläufe verantwortlich; insbesondere sollte die Sicherheitsbox so einfach zu bedienen sein, daß ein Sicherheitsbeauftragter den Schlüssel alleine auslesen kann, und die Sicherheitsbox ist so zu programmieren, daß jede Schlüsselhälfte des Transportschlüssels nur **einmal** ausgegeben wird und nach der Ausgabe der zweiten Hälfte ausschließlich der verschlüsselte Masterkey ausgegeben wird.

Sei e*Ti(.) die Triple-DES-Verschlüsselung mit dem 16 Byte langen Schlüssel Ti im ECB-Mode. Die Ausgabe eines verschlüsselten 16 Byte langen Masterkeys MK

$$e*\mathbf{Ti}(MK)$$

erfolgt ebenfalls auf Papier (zum Format vgl. Kapitel 3.4.2.). Ein Empfangsberechtigter nimmt den verschlüsselten MK persönlich entgegen.

Die Transportschlüsselhälften sollten ausschließlich in Tresoren (Tresorfächer) aufbewahrt werden, zu denen nur der jeweilige Sicherheitsbeauftragte bzw. dessen Vertreter Zugang hat.

Alle Empfangsberechtigten werden auf den korrekten Umgang mit kryptographischen Schlüsseln hingewiesen.

3.4.2. Ausgabe von Schlüsseln

Jede der beiden Transportschlüsselhälften Ti_j (j = 1,2) wird zusammen mit mit dem gemäß Kapitel 3.1. gebildeten Hash-Wert $H(I,Ti_j)$ als Checksumme ausgegeben.

Schlüsselhälften und Checksummen werden hexadezimal dargestellt, wobei die Notierung vom höherwertigen zum niederwertigen (Halb-)Byte von links nach rechts erfolgt.

(höchstwertiges Byte,...,niedrigstwertiges Byte)

Zusätzlich zu den Ausführungen in Kapitel 3.3. wird beim Export von verschlüsselten Schlüsseln eine weitere Absicherung zum Schutz vor Fehleingaben auf Empfängerseite bzw. gegen beabsichtigte Manipulation von Schlüsseln vorgenommen.

Jedem zu verschlüsselnden Schlüssel werden ein ***Schlüsselname*** und eine ***Sequenznummer*** zugeordnet.

Der Schlüsselname wird ASCII-codiert (7 Bit) mit maximal 20 Byte Länge angegeben. Nicht benötigte Stellen werden mit 'Blank' (= 20h) aufgefüllt.

Beispiel: TESTKGKPIN wird codiert als

54 45 53 54 4B 47 4B 50 49 4E 20 20 20 20 20 20 20 20 20 20 (hex)

Die Sequenznummer hat 4 Dezimalstellen, kann also Werte zwischen 0000 und 9999 annehmen. Sie wird mit einer festen Länge von 4 Byte angegeben, wobei jede Dezimalstelle ASCII in ein Byte kodiert wird. Die Beschreibung erfolgt rechtsbündig mit vorangestellten Nullen.

Beispiel: Sequenznummer 9 wird codiert als

30 30 30 39 (hex).

Für Schlüssel, die mit mehreren logischen Schlüsselnummern oder mit mehreren Generationsnummern verwendet werden, gibt die Sequenznummer die Reihenfolge der jeweiligen Nummern an.

Beispiele: Der Schlüssel TESTKGKRD mit der Sequenznummer 0000 erhält die logische Schlüsselnummer '05'. Der Schlüssel TESTKGKRD mit der Sequenznummer 0001 erhält die logische Schlüsselnummer '06', etc.

Der Schlüssel TESTKGKZD mit der Sequenznummer 0000 erhält die Generationsnummer '00'. Der Schlüssel TESTKGKZD mit der Sequenznummer 0001 erhält die Generationsnummer '01'.

Der zu verschlüsselnde Schlüssel MK muß immer 16 Byte lang sein. 8 Byte lange Schlüssel werden durch Konkatenation mit sich selbst auf die erforderliche Länge gebracht.

Beispiel: Der 8 Byte große K_{PS} wird durch Konkatenation ($K_{PS}|K_{PS}$) auf 16 Byte vergrößert.

Die Werte werden in der Reihenfolge

Schlüsselname|Sequenznummer|Klartextschlüssel

konkateniert, so daß eine 40 Byte lange Binärfolge entsteht.

Über diese 40 Byte wird mit dem 16 Byte langen Transportschlüssel Ti der Retail-CBC-MAC berechnet.

Die Schlüsselausgabe umfaßt nunmehr die 40 Byte lange Konkatenation der folgenden Werte:

1. 16 Byte Triple-DES im ECB-Mode verschlüsselter Schlüssel MK:
   e*Ti(MK)

2. 16 Byte Hash-Wert von MK gemäß Kapitel 3.3.:
   H(I,MK)

3. 8 Byte Retail-CBC-MAC, wie oben beschrieben:
   m*Ti(Schlüsselname|Sequenznummer|MK)

Alle Werte werden hexadezimal dargestellt, wobei die Notierung vom höherwertigen zum niederwertigen (Halb-)Byte von links nach rechts erfolgt.

(höchstwertiges Byte,...,niedrigstwertiges Byte)

4. Übergeordnete kartenindividuelle Schlüssel

Im EF_KEY des MF jeder Chipkarte ist der übergeordnete kartenindividuelle Schlüssel

- K_{Card}

gespeichert. Im EF_KEY des MF jeder ec-Karte und kontobezogenen Börsenkarte sind außerdem die übergeordneten kartenindividuellen Schlüssel

- K_{PIN} und
- K_{INFO}

abgelegt.

Der Schlüssel

- K_{RAND}

wird im EF_RAND des MF jeder Chipkarte gespeichert.

Diese Schlüssel und die zugehörigen Key Generating Keys werden im folgenden beschrieben.

4.1. K_{Card}

Schlüssellänge	16 Byte.
Anzahl	Es wird genau ein K_{Card} in jeder Karte verwendet. Dieser ist für jede Karte unterschiedlich.
Verwendungszweck	K_{Card} wird für die Absicherung von Administrationsfunktionen (z.B. Nachladen und Personalisieren von Applikationen, Zurücksetzen des PIN Fehlbedienungszählers, etc.) verwendet. Er berechtigt zum CREATE und UPDATE von EFs und DFs der Chipkarte.
Generierung	K_{Card} wird bei den Generierungsstellen dynamisch aus KGK_{Card} und Kartenidentifikationsdaten CID generiert.
Verteilung	Die Generierungsstelle übergibt den K_{Card} den Personalisierungsstellen transportverschlüsselt. Bei der Personalisierung der Chipkarte wird K_{Card} in die Karte geladen.
Logische Schlüsselnummer	K_{Card} wird unter der logischen Schlüsselnummer '00' im EF_KEY des MF jeder Chipkarte gespeichert.
Key User	Kartenherausgeber, ec-Karte, Börsenkarte, Händlerkarte.
Wechsel	Ein Wechsel des K_{Card} in einer Karte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.2. K_{Card} Generating Key (KGK_{Card})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein KGK_{Card} pro Kartenherausgeber verwendet.
Verwendungszweck	KGK_{Card} wird für die dynamische Generierung des K_{Card} verwendet.
Generierung	KGK_{Card} wird von der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	Die Generierungsstelle übergibt den KGK_{Card} transportverschlüsselt an Bankensonderfunktionsterminalhersteller bzw. an das kartenherausgebende Institut oder dessen Beauftragte.
Key User	Generierungsstelle, Sicherheitsmodul des Bankensonderfunktionsterminals.
Wechsel	Ein Wechsel des Card Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.3. PIN Protecting Key (K_{PIN})

Schlüssellänge	8 Byte.
Anzahl	Es wird ein PIN Protecting Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{PIN} wird für die Absicherung der PIN beim Transport vom Terminal zur Karte verwendet, wobei das Kommando VERIFY durch das Secure Messaging der ec-Karte abgesichert ist.
Generierung	K_{PIN} wird von der Generierungsstelle dynamisch aus KGK_{PIN} und CID generier
Verteilung	Die Generierungsstelle übergibt den PIN Protecting Key den Personalisierungsstellen transportverschlüsselt zusammen mit den übrigen Personalisierungsdaten. K_{PIN} wird bei der Personalisierung der Chipkarte in die ec-Karte geladen.
Logische Schlüsselnummer	K_{PIN} wird unter der logischen Schlüsselnummer '01' im EF_KEY des MF der ec-Karte bzw. kontobezogenen Börsenkarte gespeichert.
Key User	ec-Terminal, Ladeterminal, ec-Karte, kontobezogene Börsenkarte.
Wechsel	Ein Wechsel des PIN Protecting Keys in der ec-Karte bzw. kontobezogenen Börsenkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.4. PIN Protecting Key Generating Key (KGK_{PIN})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein PIN Protecting Key Generating Key verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim deutschen Kreditgewerbe, d.h. KGK_{PIN} ist notwendig ZKA-weit derselbe.
Verwendungszweck	KGK_{PIN} wird für die dynamische Generierung des K_{PIN} verwendet.
Generierung	KGK_{PIN} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	KGK_{PIN} wird von den Generierungsstellen an die Terminalhersteller ausgeliefer Der Transport von der Generierungsstelle zum Terminalhersteller geschieht transportverschlüsselt. Der Terminalhersteller speichert den KGK_{PIN} sicher und lädt ihn in das Sicherheitsmodul des Terminals.
Key User	Generierungsstelle, ec-Terminal, Ladeterminal.
Wechsel	Ein Wechsel des PIN Protecting Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.5. Info Protecting Key (K_{INFO})

Schlüssellänge	8 Byte.
Anzahl	Es wird ein Info Protecting Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{INFO} wird für eine externe Authentikation zur Absicherung von Karten- und Applikationsinformation vor unberechtigtem Lesen und zur Authentikation von Chipkartendaten verwendet.
Generierung	K_{INFO} wird von der Generierungsstelle dynamisch aus KGK_{INFO} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Info Protecting Key den Personalisierungsstellen transportverschlüsselt zusammen mit den übrigen Personalisierungsdaten. K_{INFO} wird bei der Personalisierung der Chipkarte in di ec-Karte geladen.
Logische Schlüsselnummer	K_{INFO} wird unter der logischen Schlüsselnummer '02' im EF_KEY des MF der ec-Karte bzw. kontobezogenen Börsenkarte gespeichert.
Key User	ec-Terminal, Ladeterminal, ec-Karte, kontobezogene Börsenkarte.
Wechsel	Ein Wechsel des Info Protecting Keys in der ec-Karte bzw. kontobezogenen Börsenkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.6. Info Protecting Key Generating Key (KGK_{INFO})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Info Protecting Key Generating Key verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim deutschen Kreditgewerbe, d.h. KGK_{INFO} ist ZKA-weit derselbe.
Verwendungszweck	KGK_{INFO} wird für die dynamische Generierung des K_{INFO} verwendet.
Generierung	KGK_{INFO} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	KGK_{INFO} wird von den Generierungsstellen an die Terminalhersteller ausgeliefert. Der Transport von der Generierungsstelle zum Terminalhersteller geschieht transportverschlüsselt. Der Terminalhersteller speichert den KGK_{INFO} sicher und lädt ihn in das Sicherheitsmodul des Terminals.
Key User	Generierungsstelle, ec-Terminal, Ladeterminal.
Wechsel	Ein Wechsel des Info Protecting Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.7. Random Number Key (K_{RAND})

Schlüssellänge	8 Byte.
Anzahl	Es wird ein Random Number Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{RAND} wird für die karteninterne Erzeugung von Pseudozufallszahlen verwendet.
Generierung	K_{RAND} wird von der Generierungsstelle dynamisch aus KGK_{RAND} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Random Number Key den Personalisierungsstellen transportverschlüsselt zusammen mit den übrigen Personalisierungsdaten. K_{RAND} wird bei der Personalisierung der Chipkarte in die ec-Karte geladen.
Logische Schlüsselnummer	K_{RAND} wird im EF_RAND des MF gespeichert.
Key User	ec-Karte, Börsenkarte, Händlerkarte
Wechsel	Ein Wechsel des Random Number Keys in einer Chipkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

4.8. Random Number Key Generating Key (KGK_{RAND})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Random Number Key Generating Key verwendet.
Verwendungszweck	KGK_{RAND} wird für die dynamische Generierung von K_{RAND} verwendet. Jede Generierungsstelle verwendet einen eigenen KGK_{RAND}.
Generierung	KGK_{RAND} wird von der Generierungsstelle zufällig mit ungerader Parität bestimmt und sicher gespeichert.
Verteilung	KGK_{RAND} bleibt bei der Generierungsstelle.
Key User	Generierungsstelle.
Wechsel	Ein Wechsel des Random Number Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5. Schlüssel für electronic cash

Die im EF_KEY der Anwendung ec cash gespeicherten kartenindividuellen Schlüssel sind

- K_{Zert}
- K_{RK}
- $K_{LK.}$

Im EF_AUT der Anwendung ec cash sind die kartenindividuellen Authentikations-Schlüssel

- K_{AK}

gespeichert.

Diese Schlüssel und die zugehörigen Key Generating Keys werden im folgenden beschrieben.

5.1. Zertifizierungs Key (K_{Zert})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Zertifizierungs Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{Zert} ist ein kartenindividueller Key, der für die Absicherung der Buchungsdaten der ec cash-Anwendung verwendet wird.
Generierung	K_{Zert} wird bei der Generierungsstelle bzw. beim Kartenherausgeber dynamisch aus KGK_{Zert} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Zertifizierungs Key transportverschlüsselt an die Personalisierungsstellen. Bei der Personalisierung der Chipkarte wird K_{Zert} in die ec-Karte geladen.
Logische Schlüsselnummer	K_{Zert} wird unter der logischen Schlüsselnummer '01' im EF_KEY der Applikation electronic cash gespeichert.
Key User	Kartenherausgeber, ec-Karte.
Wechsel	Ein Wechsel des Zertifizierungs Keys in der ec-Karte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.2. Zertifizierungs Key Generating Key (KGK_{Zert})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Zertifizierungs Key Generating Key pro Kartenherausgeber verwendet.
Verwendungszweck	KGK_{Zert} wird für die dynamische Generierung des K_{Zert} verwendet.
Generierung	KGK_{Zert} wird von der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	Der Transport von der Generierungsstelle zum Kartenherausgeber geschieht transportverschlüsselt.
Key User	Generierungsstelle, Kartenherausgeber.
Wechsel	Ein Wechsel des Zertifizierungs Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.3. Reduce Key (K_{RK})

Schlüssellänge	8 Byte.
Anzahl	Es werden 10 Reduce Keys verwendet. Alle Schlüssel sind kartenindividuell.
Verwendungszweck	K_{RK} wird beim Abbuchen aus dem Verfügungsrahmen für die Absicherung des Kommandos REDUCE_EC verwendet.
Generierung	K_{RK} wird bei der Generierungsstelle bzw. im Terminal dynamisch aus KGK_{RK} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Reduce Key transportverschlüsselt der Personalisierungsstelle. K_{RK} wird bei der Personalisierung der Chipkarte in die ec-Karte geladen.
Logische Schlüsselnummer	Die K_{RK} werden unter den logischen Schlüsselnummern '05' bis '0E' im EF_KEY der Applikation ec cash gespeichert.
Key User	ec-Terminal, ec-Karte.
Wechsel	Ein Wechsel der Reduce Keys in der ec-Karte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.4. Reduce Key Generating Key (KGK_{RK})

Schlüssellänge	16 Byte.
Anzahl	Es werden 10 Reduce Key Generating Keys verwendet. Der Verantwortungsbereich für diese Schlüssel liegt beim deutschen Kreditgewerbe. Im electronic cash Verfahren muß in einem ec-Terminal ein Schlüssel aus dem Satz von 10 Schlüsseln vorhanden sein.
Verwendungszweck	KGK_{RK} wird für die dynamische Generierung des K_{RK} verwendet.
Generierung	KGK_{RK} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	Die Generierungsstelle übergibt den Reduce Key Generating Key transportverschlüsselt den Terminalherstellern. Jeder Terminalhersteller erhält genau einen KGK_{RK}. Der Terminalhersteller speichert den KGK_{RK} sicher und lädt ihn in das Sicherheitsmodul des Terminals.
Key User	Generierungsstelle, ec-Terminal.
Wechsel	Ein Wechsel der Reduce Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.5. Authentication Key (K_{AK})

Schlüssellänge	8 Byte.
Anzahl	Es werden 10 Authentication Keys für das electronic cash Verfahren verwendet. Alle Schlüssel sind kartenindividuell.
Verwendungszweck	K_{AK} wird für die interne Authentikation der ec-Karte gegenüber dem ec-Terminal verwendet.
Generierung	K_{AK} wird bei der Generierungsstelle bzw. im Terminal dynamisch aus KGK_{AK} und CID erzeugt.
Verteilung	Die Generierungsstelle übergibt den Authentication Key transportverschlüsselt den Personalisierungsstellen. K_{AK} wird bei der Personalisierung der Chipkarte in die ec-Karte geladen.
Logische Schlüsselnummer	Die K_{AK} werden unter den logischen Schlüsselnummern '05' bis '0E' im EF_AUT der Applikation ec cash gespeichert.
Key User	ec-Terminal, ec-Karte.
Wechsel	Ein Wechsel der Authentication Keys in der ec-Karte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.6. Authentication Key Generating Key (KGK_{AK})

Schlüssellänge	16 Byte.
Anzahl	Es werden 10 Authentication Key Generating Keys für die Applikation ec cash verwendet. Der Verantwortungsbereich für diese Schlüssel liegt beim deutschen Kreditgewerbe. Im ec-Terminal muß ein Schlüssel aus diesem Satz von 10 Schlüsseln vorhanden sein.
Verwendungszweck	KGK_{AK} wird für die dynamische Generierung des K_{AK} verwendet.
Generierung	KGK_{AK} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	Der Transport von der Generierungsstelle zum Terminalhersteller geschieht transportverschlüsselt. Jeder Terminalhersteller erhält genau einen KGK_{AK}. Der Terminalhersteller speichert den KGK_{AK} sicher und lädt ihn in das Sicherheitsmodul des Terminals.
Key User	Generierungsstelle, ec-Terminal.
Wechsel	Ein Wechsel der Authentication Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.7. Load Key (K_{LK})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Load Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{LK} wird für das gesicherte Laden des Verfügungsrahmens der Applikation ec cash verwendet.
Generierung	K_{LK} wird bei der Generierungsstelle bzw. bei der Autorisierungsstelle dynamisch aus einem der beiden KGK_{LK} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Load Key den Personalisierungsstellen transportverschlüsselt. Bei der Personalisierung der Chipkarte wird der K_{LK} in die ec-Karte geladen. Dort wird er im EF_KEY der Applikation ec cash in klarer Form gespeichert.
Logische Schlüsselnummer	Der K_{LK} wird unter der logischen Schlüsselnummer '02' im EF_KEY der Applikation ec cash gespeichert.
Key User	Autorisierungsstellen, ec-Karte.
Wechsel	Ein Wechsel des Load Keys in der ec-Karte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

5.8. Load Key Generating Key (KGK_{LK})

Schlüssellänge	16 Byte.
Anzahl	Es werden zwei Load Key Generating Keys verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt im Verbandsbereich des jeweiligen Kartenherausgebers. Daher wird verbandsweit der gleiche KGK_{LK} verwendet.
Verwendungszweck	KGK_{LK} wird für die dynamische Generierung des K_{LK} verwendet.
Generierung	KGK_{LK} wird von der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	Der KGK_{LK} wird von der Generierungsstelle an die Autorisierungsstellen übergeben. Der Transport von der Generierungsstelle zur Autorisierungsstelle geschieht transportverschlüsselt.
Key User	Generierungsstelle, Autorisierungsstellen.
Wechsel	Ein Wechsel des Load Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6. Schlüssel für die elektronische Geldbörse

6.1. Schlüssel der Börsenkarte

Neben den globalen Schlüsseln K_{Card}, eventuell K_{PIN} und K_{INFO} im EF_KEY und K_{RAND} im EF_RAND des MF sind im EF_KEY des Applikationsverzeichnisses DF_BÖRSE einer Börsenkarte die folgenden Schlüssel gespeichert

- K_{RD},
- K_{LD},
- K_{LT} und
- K_{CD}.

Zusätzlich ist in EF_AUT des Applikationsverzeichnisses DF_BÖRSE der Schlüssel K_{CAM} gespeichert.

Die Schlüssel K_{RD}, K_{LD}, K_{LT} und die zugehörigen Key Generating Keys werden im folgenden beschrieben.

Die Schlüssel K_{CD} und K_{CAM} sowie die zugehörigen Key Generating Keys werden im Kapitel 6.3.

beschrieben.

6.1.1. Reduce Key (K_{RD})

Schlüssellänge	8 Byte.
Anzahl	Es werden 10 Reduce Keys verwendet. Alle Schlüssel sind kartenindividuell.
Verwendungszweck	K_{RD} wird beim Abbuchen aus der elektronischen Geldbörse für die Absicherung der Abbuchungs- und Rückbuchungstransaktionen der elektronischen Geldbörse verwendet.
Generierung	K_{RD} wird bei der Generierungsstelle bzw. in der Händlerkarte (vgl. 6.2.) dynamisch aus KGK_{RD} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Reduce Key transportverschlüsselt der Personalisierungsstelle. K_{RD} wird bei der Personalisierung in die Börsenkarte geladen.
Logische Schlüsselnummer	Die K_{RD} werden unter den logischen Schlüsselnummern '05' bis '0E' im EF_KEY der Applikation DF_BÖRSE gespeichert.
Key User	Händlerkarte, Börsenkarte.
Wechsel	Ein Wechsel der Reduce Keys der elektronischen Börse ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.1.2. Load Key (K_{LD})

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Load Key verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{LD} wird zur Absicherung des Ladens der elektronischen Geldbörse und des Entladens der kontobezogenen elektronischen Geldbörse verwendet.
Generierung	K_{LD} wird bei der Generierungsstelle bzw. bei der Ladezentrale/Autorisierungsstelle dynamisch aus einem der beiden KGK_{LD} und CID generiert.
Verteilung	Die Generierungsstelle übergibt den Load Key den Personalisierungsstellen transportverschlüsselt. Bei der Personalisierung wird K_{LD} in die Börsenkarte geladen.
Logische Schlüsselnummer	Der K_{LD} wird unter der logischen Schlüsselnummer '02' im EF_KEY des Applikationsverzeichnisses DF_BÖRSE gespeichert.
Key User	Ladezentrale/Autorisierungsstelle, Börsenkarte.
Wechsel	Ein Wechsel des Load Keys in der Börsenkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.1.3. Load Key Generating Key (KGK_{LD})

Schlüssellänge	16 Byte.
Anzahl	Es werden zwei Load Key Generating Keys verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt im Verbandsbereich des jeweiligen Kartenherausgebers.
Verwendungszweck	KGK_{LD} wird für die dynamische Generierung des K_{LD} verwendet.
Generierung	KGK_{LD} wird von der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	Der KGK_{LD} wird von der Generierungsstelle an die Ladezentrale/Autorisierungsstellen übergeben. Der Transport von der Generierungsstelle zur Autorisierungsstelle geschieht transportverschlüsselt.
Key User	Generierungsstelle, Ladezentrale/Autorisierungsstelle.
Wechsel	Ein Wechsel des Load Key Generating Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.1.4. K_{LT}

Schlüssellänge	8 Byte.
Anzahl	Es werden 10 Schlüssel K_{LT} verwendet. Alle Schlüssel sind kartenindividuell.
Verwendungszweck	K_{LT} dient zur Absicherung der Kommunikation zwischen Börsenkarte und Ladeterminals beim Laden gegen andere Zahlungsmittel.
Generierung	Die 10 K_{LT} werden bei der Generierungsstelle aus den 10 KGK_{LT} und im Ladeterminal aus jeweils einem von diesen mittels CID abgeleitet.
Verteilung	Die Generierungsstelle übergibt die K_{LT} transportverschlüsselt der Personalisierungsstelle. K_{LT} werden bei der Personalisierung in die Börsenkarte geladen.
Logische Schlüsselnummer	Die K_{LT} werden unter den logischen Schlüsselnummern '0F' bis ‘18’ im EF_KEY der Applikation DF_BÖRSE gespeichert.
Key User	Börsenkarte, Ladeterminal.
Wechsel	Ein Wechsel der K_{LT} der elektronischen Börse ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.1.5. Key Generating Key KGK_{LT}

Schlüssellänge	16 Byte.
Anzahl	Es werden 10 KGK_{LT} verwendet, davon jeweils einer in einem Ladeterminal, das das Laden gegen andere Zahlungsmittel unterstützt. Der Verantwortungsbereich für diese Schlüssel liegt beim deutschen Kreditgewerbe, d.h. KGK_{LT} ist notwendig ZKA-weit in allen Ladeterminals, die das Laden gegen andere Zahlungsmittel unterstützen, derselbe.
Verwendungszweck	KGK_{LT} wird für die dynamische Generierung des K_{LT} verwendet.
Generierung	KGK_{LT} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	Der KGK_{LT} wird von der Generierungsstelle an die Hersteller von Ladeterminals übergeben. Jeder Hersteller erhält genau einen KGK_{LT}. Der Transport von der Generierungsstelle zu den Herstellern geschieht transportverschlüsselt.
Key User	Generierungsstelle, Ladeterminals.
Wechsel	Ein Wechsel des KGK_{LT} ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.2. Schlüssel der Händlerkarte

Neben den globalen Schlüsseln K_{Card} im EF_KEY und K_{RAND} im EF_RAND des MF sind im EF_KEY des Applikationsverzeichnisses DF_BSAM einer Händlerkarte die folgenden Schüssel gespeichert

- $K_{ZD,}$
- KGK_{RD} und
- K_{CD}.

Zusätzlich ist in EF_AUT des Applikationsverzeichnisses DF_BSAM der Schlüssel K_{CAM} gespeichert.

Die Schlüssel K_{ZD} und KGK_{RD} und der Key Generating Key KGK_{ZD} werden im folgenden beschrieben.

Die Schlüssel K_{CD} und K_{CAM} sowie die zugehörigen Key Generating Keys werden in Kapitel 6.3. beschrieben.

6.2.1. Zertifizierungsschlüssel K_{ZD}

Schlüssellänge	16 Byte.
Anzahl	Es wird ein händlerkartenindividueller Schlüssel K_{ZD} verwendet.
Verwendungszweck	K_{ZD} wird zur Erzeugung von Einzeltransaktions- und Summenzertifikaten verwendet.
Generierung	K_{ZD} wird bei der Generierungsstelle dynamisch aus einem der beiden KGK_{ZD} und CID erzeugt.
Verteilung	Die Generierungsstelle übergibt den Zertifizierungsschlüssel den Personalisierungsstellen transportverschlüsselt. Bei der Personalisierung wird K_{ZD} in die Händlerkarte geladen.
Logische Schlüsselnummer	'01' im EF_KEY des DF_BSAM .
Key User	EZ, Händlerkarte.
Wechsel	Ein Wechsel des Zertifizierungsschlüssels der Händlerkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.2.2. Zertifizierungs-Key Generating Key (KGK_{ZD})

Schlüssellänge	16 Byte.
Anzahl	Es werden zwei KGK_{ZD} verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim deutschen Kreditgewerbe, d. h. jede EZ besitzt beide KGK_{ZD}.
Verwendungszweck	KGK_{ZD} wird für die dynamische Generierung des K_{ZD} verwendet.
Generierung	KGK_{ZD} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	Der KGK_{ZD} wird von der Generierungsstelle an die EZ übergeben. Der Transport von der Generierungsstelle zur EZ geschieht transportverschlüsselt.
Key User	EZ, Generierungsstelle.
Wechsel	Ein Wechsel des KGK_{ZD} ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.2.3. Reduce Key Generating Key (KGK_{RD})

Schlüssellänge 16 Byte.

Anzahl Es werden 10 Reduce Key Generating Keys verwendet. Der Verantwortungsbereich für diese Schlüssel liegt beim deutschen Kreditgewerbe. In einer Händlerkarte muß ein zufällig ausgewählter Schlüssel aus dem Satz von 10 Schlüsseln vorhanden sein.

Verwendungszweck KGK_{RD} wird für die dynamische Generierung des K_{RD} (vgl. 6.1.) verwendet.

Generierung KGK_{RD} wird von der Generierungsstelle bestimmt und sicher gespeichert.

Verteilung Die Generierungsstelle übergibt den Reduce Key Generating Key transportverschlüsselt der Personalisierungsstelle für Händlerkarten.

Logische Schlüsselnummer **Ein** KGK_{RD} wird unter einer logischen Schlüsselnummer zwischen '05' und '0E' im EF_KEY des DF_BSAM der Händlerkarte abgelegt.

Key User Generierungsstelle, Händlerkarte.

Wechsel Ein Wechsel der Reduce Key Generating Keys ist grundsätzlich möglich,

erfordert aber entsprechenden organisatorischen Aufwand.

6.3. Schlüssel der Börsen- und Händlerkarte

6.3.1. K_{CD}

Schlüssellänge	16 Byte.
Anzahl	Es wird 1 Schlüssel K_{CD} verwendet. Dieser ist für jede Chipkarte unterschiedlich.
Verwendungszweck	K_{CD} wird zur Absicherung von Daten der Elektronischen Geldbörse bzw. des Geldbörsen-SAM verwendet, deren Echtheit vom Bankensonderfunktionsterminal überprüft werden soll.
Generierung	K_{CD} wird bei der Generierungsstelle bzw. im Bankensonderfunktionsterminal dynamisch aus KGK_{CD} und CID der jeweiligen Karte abgeleitet.
Verteilung	Die Generierungsstelle übergibt den K_{CD} transportverschlüsselt der Personalisierungsstelle. K_{CD} wird bei der Personalisierung in die Börsenkarte bzw. Händlerkarte geladen.
Logische Schlüsselnummer	Der K_{CD} wird unter der logischen Schlüsselnummer '03' im EF_KEY der Applikation DF_BÖRSE bzw. der Applikation DF_BSAM gespeichert.
Key User	Börsenkarte, Händlerkarte, Bankensonderfunktionsterminal.
Wechsel	Ein Wechsel des K_{CD} der elektronischen Geldbörse bzw. des Geldbörsen-SAM ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.3.2. Key Generating Key KGK_{CD}

Schlüssellänge	16 Byte.
Anzahl	Es wird ein KGK_{CD} verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim deutschen Kreditgewerbe, d.h. KGK_{CD} ist notwendig ZKA-weit in allen Bankensonderfunktionsterminals derselbe .
Verwendungszweck	KGK_{CD} wird für die dynamische Generierung des K_{CD} verwendet.
Generierung	KGK_{CD} wird von der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	Der KGK_{CD} wird von der Generierungsstelle transportverschlüsselt an die Hersteller von Bankensonderfunktionsterminals bzw. an ein kartenausgebendes Institut oder dessen Beauftragte übergeben.
Key User	Generierungsstelle, Bankensonderfunktionsterminals.
Wechsel	Ein Wechsel des KGK_{CD} ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.3.3. Der Authentikations-Schlüssel K_{CAM}

Zur Authentikation der elektronischen Geldbörse bzw. des Geldbörsen-SAM gegenüber der externen Welt mit dem Kommando INTERNAL AUTHENTICATE wird der Schlüssel K_{CAM} eingesetzt. Er ist im EF_AUT der Applikation DF_BÖRSE der Börsenkarte bzw. DF_BSAM der Händlerkarte gespeichert.

Schlüssellänge	16 Byte.
Anzahl	Es wird ein kartenindividueller K_{CAM} eingesetzt.
Verwendungszweck	K_{CAM} wird zur Authentikation mit der externen Welt benötigt. So kann sich ein Bankensonderfunktionsterminal von der Echtheit der elektronischen Geldbörse bzw. des Geldbörsen-SAM überzeugen.
Generierung	K_{CAM} wird bei der Generierungsstelle und im Bankensonderfunktionsterminal aus KGK_{CAM} mittels CID der jeweiligen Karte abgeleitet.
Verteilung	Die Generierungsstelle übergibt den K_{CAM} den Personalisierungsstellen transportverschlüsselt. Bei der Personalisierung wird K_{CAM} in die Börsenkarte bzw. Händlerkarte geladen.
Logische Schlüsselnummer	Logische Schlüsselnummer ‘00’ im EF_AUT des DF_BÖRSE bzw. des DF_BSAM.
Key User	Bankensonderfunktionsterminal, Börsenkarte, Händlerkarte.
Wechsel	Ein Wechsel des K_{CAM} in der Börsenkarte bzw. Händlerkarte ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

6.3.4. KGK_{CAM}

Schlüssellänge	16 Byte.
Anzahl	Es wird ein KGK_{CAM} pro Kartenherausgeber eingesetzt.
Verwendungszweck	KGK_{CAM} wird zur dynamischen Generierung von K_{CAM} verwendet.
Generierung	KGK_{CAM} wird von der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	Die Generierungsstelle übergibt den KGK_{CAM} transportverschlüsselt den zugelassenen Herstellern von Bankensonderfunktionsterminals bzw. dem kartenherausgebenden Institut oder dessen Beauftragten.
Key User	Bankensonderfunktionsterminal, Generierungsstelle.
Wechsel	Ein Wechsel des KGK_{CAM} ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

7. Allgemeingültige Schlüssel

7.1. PIN Storage Key K_{PS}

Der PIN Storage Key wird zur verschlüsselten Speicherung der PIN auf der ec-Karte mit Chip verwendet.

Schlüssellänge	8 Byte.
Anzahl	Es wird ein PIN Storage Key verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim deutschen Kreditgewerbe, d.h. K_{PS} ist ZKA-weit derselbe.
Verwendungszweck	K_{PS} wird für die Verschlüsselung des Format 0 PIN Blocks verwendet. Die so verschlüsselte PIN wird im EF_PWD0 des MF der Chipkarte gespeichert.
Generierung	K_{PS} wird bei der Generierungsstelle bestimmt und sicher gespeichert.
Verteilung	K_{PS} wird von den Generierungsstellen an die Terminalhersteller ausgeliefert. Der Transport zum Terminalhersteller geschieht transportverschlüsselt. Der Terminalhersteller speichert den PIN Storage Key sicher und lädt ihn in das Sicherheitsmodul des Terminals.
Key User	Generierungsstelle, ec-Terminal, Ladeterminal.
Wechsel	Ein Wechsel des PIN Storage Keys ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

7.2. Der Signaturschlüssel K_{SIG}

Das Nachladen von zusätzlichen Kommandos in die ec-Karte mit Chip erfolgt in Bankensonderfunktionsterminals. Diese überprüfen vor dem eigentlichen Ladevorgang die Authentizität der nachzuladenden Kommandos unter Verwendung des Schlüssels K_{SIG}. Die Signatur der nachzuladenden Kommandos wird durch die Verlage erstellt.

Schlüssellänge	16 Byte.
Anzahl	Es wird ein Signaturschlüssel pro Verband verwendet. Der Verantwortungsbereich für diesen Schlüssel liegt beim jeweiligen Verband, d.h. K_{SIG} ist notwendig verbandsweit und in allen Bankensonderfunktionsterminals dieses Verbandes derselbe .
Verwendungszweck	K_{SIG} wird zur Signatur von Kommandos verwendet, die während der Kartenlaufzeit nachgeladen werden soll. Die Signatur wird im Bankensonderfunktionsterminal geprüft.
Generierung	K_{SIG} wird bei der Generierungsstelle zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	K_{SIG} wird von den Generierungsstellen an die Bankensonderfunktionsterminalhersteller ausgeliefert. Der Transport zum Bankensonderfunktionsterminalhersteller geschieht transportverschlüsselt. Es sind besondere Sicherheitsanforderungen an die Personalisierung von Bankensonderfunktionsterminals zu stellen.
Key User	Generierungsstelle, Bankensonderfunktionsterminal.
Wechsel	Ein Wechsel des Signaturschlüssels ist grundsätzlich möglich, erfordert aber entsprechenden organisatorischen Aufwand.

7.3. Der Personalisierungsschlüssel K_{Pers}

Eine Personalisierungsstelle erhält von einem Verlag ein Auftragsband. Für eine zu personalisierende Chipkarte werden dem Auftragsband die zu der Karte gehörenden Personalisierungsdatensätze übernommen. Dieser Datensatz muß verschlüsselt sein. Hierzu wird K_{Pers} verwendet. K_{Pers} wird während der Initialisierungsphase in den Chip eingebracht.

Schlüssellänge	8 Byte.
Anzahl	Es wird ein Personalisierungsschlüssel für die Absicherung zwischen Verlag und Initialisierungsstelle verwendet.
Verwendungszweck	K_{Pers} wird zur Absicherung von Personalisierungsdaten verwendet.
Generierung	K_{Pers} wird vom Verlag zufällig mit ungerader Parität gewählt und sicher gespeichert.
Verteilung	K_{Pers} wird vom Verlag an die Initialisierungsstelle ausgeliefert. Der Transport geschieht transportverschlüsselt.
Key User	Verlag, Personalisierungsstelle.
Wechsel	Ein Wechsel des Personalisierungsschlüssels ist grundsätzlich mit jeder neuen Initialisierung möglich, erfordert aber entsprechenden organisatorischen Aufwand.

Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen

Version 2.2
22.01.1997

Inhalt

V Protokollierung

VI Schlüsselmanagement

VII Forderungen an die Hardware

VIII Organisatorische Maßnahmen bei der Herstellung und Personalisierung von Sicherheitsmodulen

IX Ablaufsicherung

X Verarbeitung anderer Anwendungen

XI Verschlüsselungsverfahren

XII Eindeutige Repräsentation

XIII Personelle Forderungen

Kriterien für die Bewertung und Konstruktion von chipkartengestützten Zahlungssystemen

Angesichts der geltenden Zahlungsgarantie der deutschen Kreditwirtschaft für ihre chipkartengestützten Zahlungssysteme müssen durch die Komponenten und Subsysteme des Zahlungssystems die Sicherheitsanforderungen der deutschen Kreditwirtschaft eingehalten werden.

Die Sicherheitsanforderungen werden durch Systemeigenschaften, die Art der Systemsoftware und -hardware und den Betrieb von Netzen praktisch umgesetzt. Ihre Einhaltung muß über alle Detailstufen der Architektur eines derartigen chipkartengestützten Zahlungssystems nachgewiesen werden. Daher muß für alle Sicherheitsfunktionen die Konsistenz aufeinanderfolgender Spezifikationsebenen von den System- und Schnittstellenspezifikationen des deutschen Kreditgewerbes bis hin zur Realisierung in Hard- und Software evaluiert werden.

Für Subsysteme eines chipkartengestützten Zahlungssystems, die durch Dritte betrieben werden (z. B. für electronic-cash-Netze Dritter), erfordert dies eine explizite Beschreibung eines Sicherheitsmodells für das betreffende Subsystem, welches mit den Sicherheitsanforderungen der deutschen Kreditwirtschaft verträglich ist. Dieses Modell muß die beteiligten Komponenten und Kommunikationsbeziehungen des Subsystems identifizieren.

Zur Erfüllung der Sicherheitsanforderungen müssen die folgenden Kriterien eingehalten werden:

I Komponenten-Authentikation

Alle aktiv an der Kommunikation teilnehmenden Komponenten in einem chipkartengestützten Zahlungssystem müssen sich partnerweise mit Hilfe kryptographischer

Verfahren authentisieren.

Erläuterung:

a. Die zulässigen kryptographischen Verfahren sind durch Forderung XI eingeschränkt.

b. Durch diese Forderung wird kein bestimmtes Authentikationsverfahren festgelegt. Im allgemeinen erfolgt die Komponenten-Authentikation durch den Nachweis des Besitzes einer geheimen Information.

c. Eine Komponente nimmt aktiv an der Kommunikation teil, wenn sie von ihrer Stellung im System her sicherheitsbezogene Informationen auswerten, verändern oder bearbeiten kann. Solche Informationen sind insbesondere die persönliche Geheimzahl (PIN), PIN-Fehlbedienungszähler, Message Authentication Codes (MACs), Transaktions- und Verfügungsbeträge. Beispiele solcher Komponenten sind:

   - Chipkarten, (z. B. ec-Karte mit Chip, Händlerkarte eines Akzeptanzterminals für die elektronische Geldbörse),
   - PIN-Pads,
   - Sicherheitsmodule, die Umschlüsselungen oder MAC-Bildungen durchführen,
   - Sicherheitsboxen, die zur Umschlüsselung von PINs oder MACs eingesetzt werden oder
   - Autorisierungssysteme.

d. Die Authentikation zwischen Chipkarte und externer Welt beinhaltet die Chipkartenechtheitsprüfung.

II Nachrichtenintegrität

1. Alle sicherheitsrelevanten Informationen in den Nachrichten sind vor und nach der Übertragung in den Komponenten des chipkartengestützten Zahlungssystems mit geeigneten Verfahren gegen unberechtigte Veränderung zu schützen.

2. Veränderungen an sicherheitsrelevanten Informationen während der Übertragung zwischen Komponenten des chipkartengestützten Zahlungssystems müssen erkannt werden. Entsprechende Reaktionen auf Integritätsverletzungen müssen erfolgen.

3. Das unautorisierte Einspielen von Nachrichten muß erkannt werden. Entsprechende Reaktionen müssen auch auf wiedereingespielte Nachrichten erfolgen.

Erläuterung:

a. Sicherheitsrelevante Informationen sind insbesondere Identifikations- und Authentikationsmerkmale, Sequenzzähler oder Transaktions- und Verfügungsbeträge.

b. Zu Forderung 1. beachte man Kriterium VII.

III Benutzer-Authentikation

Wenn das chipkartengestützte Zahlungssystem die Authentikation des Benutzers durch Verwendung seiner PIN erfordert, muß sichergestellt sein, daß bestimmte Funktionen nur dann ausgeführt werden können, wenn die richtige PIN bekannt ist.

Erläuterung:

a. Gegebenenfalls muß sich der Benutzer gegenüber seiner Chipkarte durch korrekte Angabe seiner PIN authentisieren.

b. Es sind Forderungen 1. und 3. von Kriterium IV zu beachten.

IV Geheimhaltung von PINs und kryptographischen Schlüsseln

1. Die PIN darf außerhalb von gesicherten Bereichen nie im Klartext übertragen werden. Wird sie in Systemkomponenten bearbeitet oder gespeichert, so muß sie gegen unautorisiertes Auslesen und Verändern geschützt sein.

2. Kryptographische Schlüssel dürfen auf elektronischen Übertragungswegen nie im Klartext übertragen werden. Werden sie in Systemkomponenten benutzt oder gespeichert, so müssen sie gegen unautorisiertes Auslesen und Verändern geschützt sein.

3. Keine Systemkomponente darf eine Möglichkeit zur Bestimmung einer PIN aufgrund einer erschöpfenden Suche bieten.

Erläuterung:

a. Forderung 1. ist von der Eingabe der PIN über die Tastatur an einzuhalten.

   Die Übertragung der PIN im Klartext von der Eingabetastatur an die Chipkarte des Benutzers ist dann zulässig, wenn sichergestellt ist, daß die eingegebene Klartext-PIN einen physikalisch gesicherten Bereich, der auch die Kontakte der Chipkarte umfaßt, niemals verläßt und innerhalb dieses Bereiches nicht aufgezeichnet werden kann.

b. Für Hardware-Komponenten (z. B. PIN-Pads und Chipkarten), in denen PINs oder Schlüssel gespeichert oder verarbeitet werden, ist ferner Kriterium VII zu beachten.

c. Kryptographische Schlüssel sind insbesondere diejenigen, die zur PIN-Verschlüsselung und zur MAC-Bildung benutzt werden.

V Protokollierung

1. Innerhalb des chipkartengestützten Zahlungssystems sind alle Daten zu protokollieren, die für die Rekonstruktion der betreffenden Abläufe benötigt werden.

2. Die Auswertung der Protokollinformationen muß geregelt erfolgen.

3. Gespeicherte Protokolldaten müssen gegen unberechtigte Veränderung geschützt sein und authentisch an eine auswertende Instanz übertragen werden können.

Erläuterung:

a. In electronic-cash-Netzen sind beispielsweise Sicherheitsverstöße und Unregelmäßigkeiten wie

   - fehlerhafte Übertragung,
   - mehrfache Initialisierung einzelner Systemkomponenten,
   - kein electronic-cash-Verkehr über einen längeren Zeitraum,
   - mehrfache falsche PIN-Eingabe,
   - wiedereingespielte Nachrichten

   zu protokollieren.

b. Es ist Kriterium XIII zu beachten.

VI Schlüsselmanagement

1. Zur Verteilung, Verwaltung und eventuell zum turnusmäßigen Wechsel und Ersetzen von Schlüsseln sind Regelungen zu treffen.

2. Schlüssel, für die mindestens der Verdacht auf Kompromittierung besteht, dürfen im gesamten chipkartengestützten Zahlungssystem nicht mehr verwendet werden.

3. Für symmetrische Verschlüsselungsverfahren sind zwischen allen Komponenten des chipkartengestützten Zahlungssystems Verfahren zur Dynamisierung von MAC-Bildung und Verschlüsselung, insbesondere PIN-Verschlüsselung, zu verwenden.

4. Für die Verschlüsselung der PIN und zur MAC-Bildung sind unterschiedliche Schlüssel zu verwenden.

Erläuterung:

Es ist Kriterium XIII zu beachten.

VII Forderungen an die Hardware

1. Alle Verschlüsselungen, Entschlüsselungen, Umschlüsselungen, die MAC-Bildung und kryptographische Prüfungsprozeduren werden in gegen unberechtigte Zugriffe besonders geschützten Komponenten (Sicherheitsmodulen) durchgeführt. Die zugehörigen Schlüssel sind ebenfalls in solchen Sicherheitsmodulen abgelegt.

2. In Sicherheitsmodulen müssen sicherheitsrelevante Daten und Abläufe (z. B. Schlüssel, Programme) gegen unberechtigte Veränderung und geheime Daten (z. B. Schlüssel, PINs) gegen unberechtigtes Auslesen geschützt sein. Dies muß durch folgende Maßnahmen gewährleistet werden:

   - Bauart des Sicherheitsmoduls, eventuell im Zusammenwirken mit Sicherheitsmechanismen der Software des Sicherheitsmoduls,

   - Laden von Programmen in Sicherheitsmodule nur bei der Herstellung oder kryptographische Absicherung des Ladevorgangs,

   - kryptographische Absicherung des Ladens von sicherheitsrelevanten Daten, insbesondere von kryptographischen Schlüsseln.

   Auch vor dem Auslesen mittels Angriffen, die die Zerstörung des Moduls in Kauf nehmen, müssen geheime Daten in Sicherheitsmodulen geschützt sein.

Erläuterung:

a. Der Schutz von Daten und Programmen gegen Veränderung bzw. Auslesen in Sicherheitsmodulen muß so hoch sein, daß während der Lebensdauer des Moduls Angriffe mit vertretbarem Aufwand nicht möglich sind, wobei der für einen erfolgreichen Angriff nötige Aufwand gegen den hieraus zu ziehenden Nutzen abzuwägen ist.

b. Zur Sicherung von Daten und Abläufen soll sowohl mechanischer als auch elektronischer Speicherschutz vorgesehen sein.

c. Die PIN-Tastatur muß gegen das Hinzufügen zusätzlicher Baugruppen geschützt sein.

d. Die Anzeige eines Sicherheitsmoduls muß gegen Verfälschung geschützt sein.

e. Unerwünschte Funktionen dürfen durch ein Sicherheitsmodul nicht ausführbar sein.

f. Die Chipkarten des jeweiligen Zahlungssystems müssen den Anforderungen dieses Kriteriums genügen. Insbesondere muß durch Chip-Hardware und -Software sichergestellt werden, daß die Chipkarte nur über die spezifizierte Schnittstelle in der festgelegten Weise kommunizieren kann.

VIII Organisatorische Maßnahmen bei der Herstellung und Personalisierung von Sicherheitsmodulen

1. Die Herstellung und Personalisierung (Ersteinbringung geheimer Schlüssel, eventuell benutzerspezifischer Daten) von Sicherheitsmodulen (insbesondere Chipkarten) muß in einer Produktionsumgebung stattfinden, die verhindert, daß

   - Schlüssel bei der Personalisierung kompromittiert werden,
   - der Personalisierungsvorgang mißbräuchlich oder unberechtigt durchgeführt wird,
   - unautorisierte Software oder Daten eingebracht werden können,
   - Sicherheitsmodule entwendet werden.

2. Es muß sichergestellt sein, daß in das chipkartengestützte Zahlungssystem keine unautorisierten Komponenten eingebracht werden können, die sicherheitsrelevante Funktionen ausführen.

3. Der Lebensweg aller Sicherheitsmodule muß kontinuierlich aufgezeichnet werden.

Erläuterung:

a. Zu Forderung 2. beachte man Kriterium XIII.

b. Die Aufzeichnung des Lebenswegs eines Sicherheitsmoduls umfaßt:

- Herstellungs- und Personalisierungsdaten,
- räumlicher/zeitlicher Verbleib,
- Reparatur und Wartung,
- Außerbetriebnahme,
- Verlust bzw. Diebstahl,

für Chipkarten

- Herstellungs- und Personalisierungsdaten,
- Einbringen neuer Anwendungen,
- Änderung von Anwendungen,
- Änderung von Schlüsseln.
- Außerbetriebnahme,
- Verlust bzw. Diebstahl.

IX Ablaufsicherung

Es muß sichergestellt sein, daß die Abläufe der spezifizierten Transaktionen und die hierbei verwendeten sicherheitsrelevanten Daten eines chipkartengestützten Zahlungssystems nicht manipuliert werden können.

Die beteiligten Instanzen, insbesondere der Benutzer, dürfen durch ein Sicherheitsmodul über die Abläufe der Transaktionen nicht getäuscht werden können.

Erläuterung:

Es ist Kriterium VII zu beachten.

X Verarbeitung anderer Anwendungen

Werden in Komponenten eines chipkartengestützten Zahlungssystems gleichzeitig andere Anwendungen verarbeitet, so darf dadurch die Sicherheit des betreffenden chipkartengestützten Zahlungssystems nicht beeinflußt werden.

Erläuterung:

a. Es muß zum Beispiel ausgeschlossen sein, daß die Funktionen, die in PIN-Pads die PIN-Verarbeitung von Kreditkarten durchführen, zur Kompromittierung der PINs im electronic-cash-System mißbraucht werden können.

b. Können in einer Chipkarte verschiedene Funktionen ausgeführt werden, so darf keine Funktion die Sicherheit einer anderen Funktion beeinflussen.

XI Verschlüsselungsverfahren

Es dürfen nur Verschlüsselungsverfahren verwendet werden, die einer Kryptoanalyse mit ausgewähltem Klartext widerstehen.

Erläuterung:

Die Sicherheit darf nicht in der Geheimhaltung der Verfahren beruhen, sie muß durch die Geheimhaltung der Schlüssel gewährleistet sein.

XII Eindeutige Repräsentation

Jede Sicherheitskomponente eines chipkartengestützten Zahlungssystems muß im System eindeutig identifiziert sein.

Erläuterung:

Die Identifikationsdaten sind zu verwenden, um sicherheitsrelevante Nachrichten dem jeweiligen Absender oder Empfänger eindeutig zuordnen zu können.

XIII Personelle Forderungen

1. Es sind vertrauenswürdige Personen zu benennen, die die Verantwortung für die Überprüfung von Protokolldaten tragen.

2. Es sind vertrauenswürdige Personen zu benennen, die für die Schlüsselerzeugung und -einbringung in Sicherheitsmodule gemäß Kriterium VI verantwortlich sind.

3. Es sind vertrauenswürdige Personen zu benennen, die bei Änderungen an zugelassenen Systemkomponenten dafür verantwortlich sind, daß entweder die sicherheitsrelevanten Eigenschaften der Komponenten erhalten bleiben oder diese Änderungen der deutschen Kreditwirtschaft mitgeteilt werden.

4. Es sind vertrauenswürdige Personen zu benennen, die für das Einbringen von Software und von sicherheitsrelevanten Daten in Sicherheitsmodule die Verantwortung tragen.

5. Bei den Forderungen 1. bis 4. ist das 4-Augen-Prinzip einzuhalten.

Anmerkung

1 (Popup)
1 Available on TeleGodzilla as part of YZMODEM.ZOO

2 (Popup)
2 The ZMODEM design allows encoded packets for less transparent media.

3 (Popup)
3 With XOFF and XON, or out of band flow control such as X.25 or CTS

4 (Popup)
4 Files that have been translated in such a way as to modify their length cannot be updated with the ZCRECOV Conversion Option.

5 (Popup)
5 This and other constants are defined in the zmodem.h include file. Please note that constants with a leading 0 are octal constants in C.

6 (Popup)
6 The frame types are cardinal numbers beginning with 0 to minimize state transition table memory requirements.
Future extensions to ZMODEM may use the high order bits of the type byte to indicate thread selection.

7 (Popup)
7 Strategies for adjusting the subpacket length for optimal results based on real time error rates are still evolving. Shorter subpackets speed error detection but increase protocol overhead slightly.

8 (Popup)
8 Special considerations apply when sending commands.

9 (Popup)
9 If the receiver specifies the same or higher level of escaping, the ZSINIT frame need not be sent unless an Attn sequence is needed.

10 (Popup)
10 See below, under ZFILE header type.

11 (Popup)
11 The crc is initialized to 0xFFFFFFFF.

12 (Popup)
12 This does not apply to files that have been translated.

13 (Popup)
13 If the ZMSPARS option is used, the receiver instead seeks to the position given in the ZDATA header.

14 (Popup)
14 The call to rdchk() in sz.c performs this function.

15 (Popup)
15 The obvious choice of ZCRCW packet, which would trigger an ZACK from the receiver, is not used because multiple in transit frames could result if the channel has a long propagation delay.

16 (Popup)

16 The call to getinsync() in sz.c performs this function.

17 (Popup)

17 If sampling is possible.

18 (Popup)

18 ZRPOS and other error packets are handled normally.

19 (Popup)

19 When used with modems or networks that simultaneously assert flow control with XON and XOFF characters and pass XON characters that violate flow control, the receiving program should have a revisiondate of May 9 or later.

20 (Popup)

20 One in 256 for escaping ZDLE, about two (four if 32 bit CRC is used) in 1024 for data subpacket CRC's

21 (Popup)

21 Filtering RUBOUT, NULL, Ctrl-Z, etc.

22 (Popup)

22 Or ZCOMPL in case of server mode.

23 (Popup)

23 Fields may not be skipped.