A.1 Recordbelegung des EF_NOTEPAD mit einem HBCI-Parameterblock (Original B.1)

Ein HBCI-Recordeintrag hat folgenden prinzipiellen Aufbau:

Tag	Länge (Byte)	Wert	Format	Status	Erläuterung
'F0'	Var. max 'EC'				HBCI-Parameterblock
'C0'	'03'	'30' '30' '31'	3an	O	Version 1 des HBCI-Parameterblocks
'E1'	Var. max. '5B'			M	HBCI-Institutsparameterblock
'C1'	'01'-'14'	Kreditinstituts-bezeichnung	..20an	O	
'C2'	'03'	Länderkenn-zeichen	3an	M	ISO 3166 numerisch in 3 ASCII-Zeichen codiert
'C3'	'01'-'1E'	Kreditinstitutscode	..30an	M	in jeweils national bekannter Notation
'C4'	'1B'	Hashwert Instituts-schlüssel	27bin	O	
'C5'	'01'	Schlüsselstatus	1bin	M	8 Statusflags
'E2'	Var. max. '37'			M	HBCI-Kommunikations-parameterblock
'C6'	'01'	Kommunikations-dienst	1n	M	2 = TCP/IP
'C7'	'01'-'32'	Kommunikations-adresse	..50an	M	
			°		
'E2'	Var. max. '37'			O	2. HBCI-Kommunikations-parameterblock
'C6'	'01'	Kommunikations-dienst	1n	M	2 = TCP/IP
'C7'	'01'-'32'	Kommunikations-adresse	..50an	M	
'E3'	Var. max. '54'			O	HBCI-Kundenparameterblock
'C8'	'01'-'1E'	Benutzerkennung	..30an	M	
'C9'	'01'-'1E'	Kunden-ID	..30an	O	
'CA'	'0C' oder '12"	Info Inhaber-schlüssel	12an oder 18an	M	Schlüsselnummer und Schlüsselversion jeweils für den Signierschlüssel, den Chiffrierschlüssels und optional für den Signaturschlüssel des Karteninhabers

Die Längen der einzelnen Records werden wie folgt nach ASN.1 BER (Basic Encoding Rules) kodiert:

Längen 'XX', wobei 'XX' die hexadezimale Darstellung eines Wertes zwischen 0 und 127 ist, werden als 'XX' in ein Byte kodiert werden.

Längen 'XX', wobei 'XX' die hexadezimale Darstellung eines Wertes zwischen 128 und 255 ist, müssen als '81 XX' in zwei Byte kodiert werden

Ausnahme ist hier die Länge des TAG 'F0', dieser wird immer in der Form 'F0' '81 XX' kodiert.

Ist der Record länger als die tatsächliche ASN.1 Struktur so kann der überschüssige Speicherplatz im Record mit '00' belegt (z.B. ASN.1 Struktur 170 Byte, Recordlänge 239 Byte **à** Filler 69 Byte mit '00'). Das Kundenprodukt soll nur die Nutzdaten übertragen,

A.1.1 Tag ‘F0‘: HBCI-Parameterblock

Durch den Tag ‘F0‘ wird ein Record mit HBCI-Parameterblock für die HBCI-Anwendung gekennzeichnet. Für Belegungen der EF_NOTEPAD-Records durch andere Anwendungen stehen die Tags ‘F1‘ bis ‘FE‘ zur Verfügung.

Ein HBCI-Parameterblock enthält in der angegebenen Reihenfolge:

- **optional** ein Versionskennzeichen
- genau einen HBCI-Institutsparameterblock mit **Tag ‘E1‘**
- genau einen HBCI-Kommunikationsparameterblöcke mit **Tag ‘E2‘**
- **optional** einen weiteren HBCI-Kommunikationsparameterblöcke mit **Tag ‘E2**[1]
- **optional** einen HBCI-Kundenparameterblock mit **Tag ‘E3‘**

Die maximale Länge des HBCI-Parameterblocks wird beschränkt durch die maximale Recordlänge von 239 Byte[2].

A.1.2 Tag ‘C0‘: HBCI-Version

In jedem ‘F0‘ Record kann zur Kennzeichnung der Version des EF-NOTEPAD ein Sub-Record (z. B ’C0’ ’03’ ’30’ ’30’ ’30’) aufgenommen werden. Die Zählung der Version beginnt bei 1. Ist kein Sub-Record ’C0’ vorhanden, so bedeutet dieses, dass die Belegung des EF-NOTEPAD gemäß der Version 1 erfolgt.

---

1 Somit ist der erste HBCI-Kommunikationsparameterblock ist also verpflichtend, der zweite optional.

2 In einer konkreten Umsetzung ist es nicht möglich einen HBCI-Parameterblock mit allen Felder in der maximalen Länge zu nutzen. Dabei würde die maximale Recordlänge von 239 Byte überschritten.

A.1.3 Tag ‘E1‘: HBCI-Institutsparameterblock

Durch den Tag ‘E1‘ wird der Block der institutsspezifischen Parameter gekennzeichnet. Ein HBCI-Institutsparameterblock enthält in der angegebenen Reihenfolge:

- **optional** eine Kreditinstitutsbezeichnung mit **Tag ‘C1‘**, alphanumerisch mit bis zu 20 Zeichen
- genau ein Länderkennzeichen des kontoführenden Instituts mit **Tag ‘C2‘**. Verwendet wird der numerische ISO 3166-Code als 3-stellige alphanumerische Zeichenkette (z.B. Deutschland = “280“)
- genau eine Kreditinstitutskennung mit **Tag ‘C3‘**, in einer jeweils national bekannten Notation mit bis zu 30 Stellen. Für deutsche Kreditinstitute wird hier die 8-stellige Bankleitzahl verwendet.
- **optional** einen Hashwert des öffentlichen Signierschlüssels des Instituts mit **Tag ‘C4‘**, binär mit genau 27 Byte. Der Eintrag besteht aus

  [3 Byte Schlüsselnummer | 3 Byte Schlüsselversion | 1 Byte Kennzeichen Hashverfahren | 20 Byte Hashwert].

  Als Kennzeichen für das Hashverfahren werden festgelegt:[3]

  - ‘02‘ = RIPEMD-160

  Die Parameter Schlüsselnummer und Schlüsselversion des Institutsschlüssels werden in je 3 Byte rechtsbündig mit führenden Nullen codiert (z.B. Schlüsselnummer 1 **à** die Bytefolge '30' '30' '31'.

[3] Aus folgenden Gründen wurde SHA-1 gestrichen:
Ursprünglich gab es in FinTS 3.0 beim INI-Brief-Verfahren die beiden Hashverfahren SHA-1 und RIPEMD-160. Gemäß ZKA-CR 205 wurde das Hashverfahren SHA-1 beim INI-Brief gestrichen, da hier ein eindeutiges Hashverfahren verwendet werden muss. Nur bei einer vorpersonalisierten SECCOS-Chipkarte wären im Grundsatz im Tag C4 beide Hashverfahren denkbar. Sollte aber die automatische Hashwertprüfung fehlschlagen (z.B. weil das Institut zwischenzeitlich die Schlüssel geändert hat), so wird clientseitig auf das INI-Briefverfahren (und damit auf das Hashverfahren RIPEMD-160) gewechselt. Auch beim Aufbringen neuer zusätzlicher Bankverbindungen auf die Chipkarte wird das INI-Briefverfahren (und damit RIPEMD-160) verwendet. Bei unpersonalisierten SECCOS-Chipkarten wird der Eintrag neuer Bankverbindungen immer über das INI-Brief-Verfahren (und damit über RIPEMD-160) abgesichert.

- genau ein Schlüsselstatus mit **Tag 'C5'**, binär von genau 1 Byte Länge. Der Schlüsselstatus enthält acht Flags mit folgender Bedeutung:

Bit1	Erstmalige Übermittlung der Kundenschlüssel notwendig	'1'b - Ja '0'b - Nein
Bit2	Institutsrechner erwartet Signaturen nach ISO9796 mit AnnexA	'1'b - Ja '0'b - Nein
Bit3	Institutsschlüssel validiert	'1'b - Ja '0'b - Nein
Bit4	Ausstehende Übermittlung des neuen öffentlichen Chiffrierschlüssels des Kunden bei Schlüsseländerung[4]	'1'b - Ja '0'b - Nein
Bit5	Ausstehende Übermittlung des neuen öffentlichen Signierschlüssels des Kunden bei Schlüsseländerung[5]	'1'b - Ja '0'b - Nein
Bit6	Schlüsselsperre mit Erfolg durchgeführt (Info, da terminierte Sperrung erst in der Zukunft wirksam werden kann)	'1'b - Ja '0'b - Nein
Bit7	Leitungsprobleme bei Übermittlung neuer Schlüssel	'1'b - Ja '0'b - Nein
Bit8	Reserviert	'0'b

Bei der Personalisierung muss als Initialisierungswert '01' aufgebracht werden.

Ein HBCI-Institutsparameterblock belegt inklusive der Tag- und Längenbytes somit maximal 93 Byte.

A.1.4 Tag ‘E2‘: HBCI-Kommunikationsparameterblock

Durch den Tag ‘E2‘ wird der Block der generellen Kommunikations-Parameter gekennzeichnet. Ein HBCI-Kommunikationsparameterblock enthält in der angegebenen Reihenfolge:

- genau einen Kommunikationsdienst mit **Tag ‘C6‘**, 1 Stelle numerisch. Zurzeit definiert ist der numerische Wert 2 (TCP/IP)
- genau eine Kommunikationsadresse mit **Tag ‘C7‘**, alphanumerisch mit bis zu 50 Zeichen

Ein HBCI-Kommunikationsparameterblock belegt inklusive der Tag- und Längenbytes somit maximal 57 Byte.

A.1.5 Tag ‘E3‘: HBCI-Kundenparameterblock

Durch den Tag ‘E3‘ wird der **optional** vorhandene Block der kundenspezifischen Parameter gekennzeichnet. Ist der Block nicht vorhanden, so handelt es sich um

---

[4] Nicht zu belegen, da die ZKA-Bankensignaturkarte keinen Wechsel der Kundenschlüssel unterstützt.

[5] Nicht zu belegen, da die ZKA-Bankensignaturkarte keinen Wechsel der Kundenschlüssel unterstützt.

eine im Rahmen der HBCI-Anwendung unpersonalisierte Karte. Ein HBCI-Kundenparameterblock enthält in der angegebenen Reihenfolge:

- genau eine Benutzerkennung mit **Tag 'C8'**, alphanumerisch mit bis zu 30 Zeichen
- **optional** eine Kunden-ID mit **Tag 'C9'**, alphanumerisch mit bis zu 30 Zeichen
- genau ein Info Inhaberschlüssel mit Tag 'CA', von genau 12 oder 18 numerischen Zeichen**.**

  Bei 12 Byte Länge des Blocks ist der Inhalt wie folgt definiert:

Schlüsselnummer Signierschlüssel [3n]
Schlüsselversion Signierschlüssel [3n]
Schlüsselnummer Chiffrierschlüssel [3n]
Schlüsselversion Chiffrierschlüssel [3n]

  Bei 18 Byte Lange ist der Inhalt wie folgt definiert:

Schlüsselnummer Signierschlüssel [3n]
Schlüsselversion Signierschlüssel [3n]
Schlüsselnummer Chiffrierschlüssel [3n]
Schlüsselversion Chiffrierschlüssel [3n]
Schlüsselnummer Signaturschlüssel [3n]
Schlüsselversion Signaturschlüssel [3n]

Die Parameter Schlüsselnummer und Schlüsselversion werden in je 3 Byte numerisch rechtsbündig mit führenden Nullen angegeben. (z.B. Schlüsselnummer 1 **à** "001" **à** die Bytefolge '30' '30' '31'.

Fehlen die Angaben für den Signaturschlüssel (CA Record der Länge 12 Byte) so werden als Schlüsselnummer und Schlüsselversion des Signaturschlüssels die Schlüsselnummer und Schlüsselversion des Signierschlüssels übernommen.

Fehlt der Teilrecord mit dem Tag 'CA' (nicht vorhandener Record E3 oder Record CA oder fehlendes EF_NOTEPAD) und liegen somit weder für den Signierschlüssel und den Chiffrierschlüssel noch für den Signaturschlüssel Schlüsselnummer und Schlüsselversion vor so sind vom FinTS-Client die Schlüsselnummern und Schlüsselversionen aller Schlüssel nach folgenden Mechanismen vorzubesetzen.

Die Schlüsselnummer wird gemäß dem genutzten RDH Verfahren besetzt. Die Schlüsselversion wird mit "001" vorbesetzt.

RDH Verfahren	Schlüsselnummer	Schlüsselversion
RDH3	"003" **à** '30' '30' 33'	"001" **à** '30' '30' '31'

RDH4	"004" **à** '30' '30' 34'	"001" **à** '30' '30' '31'
RDH5	"005" **à** '30' '30' 35'	"001" **à** '30' '30' '31'

Ein HBCI-Kundenparameterblock belegt inklusive der Tag- und Längenbytes somit maximal 86 Byte.

A.1.6 Beispiel

Beispiel für eine Recordbelegung (Tags und Längenbytes sind fett markiert

Inhalt	Erläuterung
F0 81 76	HBCI-Parameterblock
E1 3D	Institutsparameterblock
C1 0C 54 45 53 54 49 4E 53 54 49 54 55 54	Institutsbezeichnung “TESTINSTITUT“
C2 03 32 38 30	Länderkennzeichen “280“
C3 08 31 32 33 34 35 36 37 38	BLZ 12345678
C4 1B 30 30 31 30 30 31 02 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14	Schlüsselnummer 1, Schlüssel-version 1, Hashverfahren RIPEMD-160, Hashwert
C5 01 01	Schlüsselstatus '01'
E2 12	Kommunikationsparameterblock
C5 01 02	Kommunikationsdienst TCP/IP
C6 0D 31 39 32 2E 31 36 38 2E 31 31 2E 32 32	Kommunikationsadresse 192.168.11.22
E3 21	Kundenparameterblock
C8 0A 31 32 33 34 35 36 37 38 39 30	Benutzerkennung "1234567890"
C9 05 31 32 33 34 35	Kunden-ID "12345"
CA 0C 30 30 31 30 30 31 30 30 31 30 30 31	Info Inhaberschlüssel Schlüsselnummer SIG 1, Schlüsselversion SIG 1 Schlüsselnummer CHIF 1, Schlüsselversion CHIF 1

A.1.7 Erreichen der maximalen Recordlänge

Bei Ausnutzung aller Maximallängen und Aufnahme aller optionalen Felder und Angabe zweier Kommunikationsparameterblöcke und eines Kundenparameterblocks ergibt sich ein maximaler Platzbedarf von 297 Byte. Dieser Platzbedarf ist aber in einem Record nicht abbildbar. Normalerweise wird aber nur ein Kommunikationsparameterblock verwendet sowie selten alle Maximallängen ausgereizt, so dass meistens die maximale Recordlänge von 239 Byte genügt. Bei älteren bereits ausgegebenen Bankensignaturkarten ist nur eine maximale Recordlänge von 200 Byte vorgesehen.

B. TERMINALABLÄUFE

Dieses Kapitel spezifiziert die Terminalabläufe im Umgang mit dem RDH-Verfahren auf SECCOS-Chipkarten [SECCOS]. Ein Homebanking-Kundenprodukt nutzt

- zur Verschlüsselung und Signierung von HBCI-Nachrichten die auf der Chipkarte zur Verfügung stehende Signatur-Anwendung (DF_SIG, [ZKASIG]) und die durch das Betriebssystem bereitgestellten Signatur-Funktionen,
- als Sequenzzähler (Signatur-ID) interne Bedienungszähler der Signatur-Anwendung (siehe Kap. B.1),
- als Datenspeicher für die Zugangsdaten ein auf der Chipkarte optional vorhandenes DF_NOTEPAD ([DF_NOTEPAD], vgl. [NOTEPAD].

B.1 Verfahren zur Ermittlung der Sequenzzähler

Auf der Bankensignaturkarte wird kein eigenständiger Sequenzzähler (wie das EF_SEQ im HBCI DDV-Verfahren) verwaltet, sondern es werden jeweils chipkarteninterne Bedienungszähler der beiden zur Signatur verwendeten Schlüssel S_K.CH.DS und S_K.CH.$AUT_{C/S}$ herangezogen.

Für jedes Signaturschlüsselpaar wird ein separater Sequenzzähler verwaltet.

Da die Bedienungszähler auf der Chipkarte dekrementiert werden, als Sequenzzähler (Signatur-ID) aber ein streng monoton aufsteigender Zähler gefordert ist, wird der konkrete Sequenzzähler nach folgendem Algorithmus ermittelt:

1. Auslesen der 2 Byte langen Bedienungszähler BZ_{DS} des Schlüssels S_K.CH.DS bzw. BZ_{AUT} des Schlüssels S_K.CH.$AUT_{C/S}$.
2. Sei **neg**(BZ) die bitweise logische Negation von BZ. Dann ist der Sequenzzähler

$$SZ_{DS} = \mathbf{neg}(BZ_{DS})$$

$$SZ_{AUT} = \mathbf{neg}(BZ_{AUT})$$

Da ein einzelner Bedienungszähler einen Wertebereich von 0 bis 65535 (2 Byte) hat, hat der Sequenzzähler SZ auch einen Wertebereich von 0 bis 65535 und benötigt zur Darstellung mindestens 2 Byte. Ein Wrap-Around bei Erreichen des Maximalwerts findet nicht statt, da das Erreichen eines Bedienungszählers 0 den Schlüssel der Chipkarte für die weitere Verwendung sperrt.

Beispiel:

BZ_{DS} = ’00 0A‘ (dezimal 10) $\Rightarrow$ SZ_{DS} = **neg**(BZ_{DS}) = ‘FF F5‘ (dezimal 65525)

BZ_{AUT} = ‘FA 1D‘ (dezimal 64029) $\Rightarrow$ SZ_{AUT} = **neg**(BZ_{AUT}) = ’05 E2‘(dezimal 1506)

Dieser Algorithmus ist in der jeweiligen Anwendungssoftware zu realisieren.

B.2 Beschreibung der Terminalabläufe

Nachfolgend werden die Anwendungsabläufe aus Endgerätesicht an einem privaten Signaturterminal [KT-KONZEPT] spezifiziert. Hierbei werden ausschließlich die chipkartenbezogenen Aspekte berücksichtigt. Anwendungsbezogene Details sind nicht Bestandteil dieser Spezifikation.

Um die Abläufe möglichst einfach beschreiben zu können, werden in der nachfolgenden Beschreibung Befehle der ZKA-SIG-API [KT-SIG] verwendet. Hiermit ist jedoch die Verwendung der ZKA-SIG-API für technische Implementierungen nicht zwingend vorgeschrieben. Wird die ZKA-SIG-API nicht verwendet, so sind die in [KT-SIG] angegebenen Abläufe zum Aufruf der KT-Kommandos zu berücksichtigen.

Die Anwendungsabläufe lassen sich auch auf öffentliche Signaturterminals (Geschäftsterminals) erweitern. Zu beachten ist dabei insbesondere, dass in diesem Fall zusätzlich eine

- Komponenten-Authentikation zwischen Chipkarte und Geschäftsterminal mit Aushandlung eines Sessionkey-Paares (SK1, SK2) stattfindet;
- alle Befehle an die Chipkarte im Secure Messaging mit einem SK2-MAC durchgeführt werden müssen.

Falls bei der Ausführung der Kommandos ein Fehler auftritt, bricht das Terminal den Vorgang ab, es sei denn, es ist ein abweichendes Verhalten spezifiziert.

In den hier beschriebenen Abläufen ist das Kundenterminal durch ein *zka_sig_open* (zu Beginn des Ablaufs „Signatur einleiten“) und ein *zka_sig_close* (Am Ende des Ablaufs „Signatur beenden“) für die gesamte Zeitdauer exklusiv für die Kundenanwendung reserviert.

Um zwischenzeitlich anderen Anwendungen die Möglichkeit zu geben, die Signaturdienste der Karte zu nutzen (z.B. für die Zeitdauer der Nachrichtengenerierung), können die im Folgenden beschriebenen Teilabläufe jeweils auch durch ein *zka_sig_open* und ein *zka_sig_close* gekapselt werden. Dadurch wird die exklusive Reservierung des Kundenterminals aufgehoben, die internen Zwischenwerte der ZKA-SIG-API (insbes. der Chipdaten) bleiben jedoch erhalten. Erst durch Aufruf des *zka_sig_fini_signature_application* im Ablauf „Signatur beenden“ werden die internen Zwischenwerte der ZKA-SIG-API gelöscht.

Zur Administration der Signaturkarten (z.B. Freischalten eines Zertifikates, Rücksetzen des Fehlbedienungszählers) werden von den Kreditinstituten bzw. den Kartenemittenten Softwarekomponenten zur Verfügung gestellt werden, die in der privaten Kundenumgebung zum Einsatz kommen sollen. In Kundenprodukten, die nicht von den Kartenemittenten herausgegeben werden, sollen diese Administrationsfunktionen nicht realisiert werden.

> Für die kreditinstitutsseitige Realisierung dieser Softwarekomponenten hat der ZKA Anforderungen und Festlegungen formuliert, die bei Bedarf über die jeweiligen Ansprechpartner der Standards erhältlich sind.

B.2.1 Signatur einleiten

Chipkarte			Endgerät	
			M1	Aufruf der ZKA-SIG-API-Funktion *zka_sig_open*
R2	OK	ß à	M2	Aufruf der ZKA-SIG-API-Funktion *zka_sig_init_signature_application*
R3	OK	ß à	M3	Aufruf der ZKA-SIG-API-Funktion *zka_sig_verify_CSA_password*
R4	OK / „File not found“	ß à	C4	SELECT FILE DF_NOTEPAD
R5	Bankverbindung	ß à	C5	ggf. READ RECORD EF_NOTEPAD
			A5	Daten prüfen und speichern

♦ **Erläuterung**

1. Die ZKA-SIG-API-Funktion *zka_sig_open* wird ausgeführt. Diese Funktion stellt eine exklusive Verbindung zum Kundenterminal her.
2. Die ZKA-SIG-API-Funktion *zka_sig_init_signature_application* wird ausgeführt. Diese sorgt insbesondere für ein Reset der Karte und das Auslesen der relevanten Basisinformationen der Karte.
3. Die ZKA-SIG-API-Funktion *zka_sig_verify_CSA_password* wird ausgeführt. Diese Funktion liest das CSA-Passwort ein und führt eine Verifikation gegenüber der Chipkarte durch.
4. Die Applikation „Notepad“ wird geöffnet, indem das ADF der Applikation, DF_NOTEPAD durch das Terminal mittels des Kommandos SELECT FILE ausgewählt wird.

♦ **Command APDU**

Byte	Wert	Erläuterung
1-2	'00 A4'	CLA, INS
3	'04'	P1, Selektion mit DF-Name
4	'0C'	P2, Keine Antwortdaten
5	'09'	L_C
6-14	'D2 76 00 00 25 4E 50 01 00'	AID der Notepad-Applikation

Wenn die Notepad-Applikation auf der Karte nicht vorhanden ist, wird der folgende Schritt übersprungen. In diesem Fall müssen die Zugangsdaten von einer anderen Stelle gelesen oder vom Benutzer eingegeben werden.

4. Das Terminal liest mittels READ RECORD sukzessive die Bankverbindungsdaten in den Records des EF_NOTEPAD (SFI '1A'), bis der oder die ”passenden” Einträge gefunden wurden. Das Lesen von Einträgen ist erst nach erfolgreicher CSA-Passwort-Verifikation (Schritt 2) möglich.

♦ **Command APDU**

Byte	Wert	Erläuterung
1-2	'00 B2'	CLA, INS
3	'0X'	P1, Recordnummer X
4	'D4'	P2, Reference Control Byte
5	'00'	L_e

Wenn das READ RECORD erfolgreich ausgeführt wird, gibt die Chipkarte eine Antwortnachricht mit der folgenden Struktur zurück:

Byte	Länge	Wert	Erläuterung
1-2	2	'XX LL'	Kennung und Länge
3-LL	LL	'XX..XX'	Nutzdaten
(LL+1)-(LL+2)	2	'XX XX'	Positiver Returncode SW1 SW2

Ist die Kennung ungleich ’00‘, so sind Parameterdaten gemäß Kap. A.1 enthalten. Es werden alle weiteren Records gelesen, bis die Chipkarte das Ende der Datei (keine weiteren Records) signalisiert.

Anstatt alle Records auszulesen und auf Übereinstimmung mit der Kennung zu überprüfen, kann alternativ auch das Kommando SEARCH RECORD verwendet werden, um mittels eines übergebenen Suchmusters vorab die ”passenden” Recordnummern in einem Schritt zu finden. Anschließend müssen dann nur diese Recordnummern mittels READ RECORD ausgelesen werden.

♦ **Command APDU**

Byte	Wert	Erläuterung
1-2	'00 A2'	CLA, INS für SEARCH RECORD
3	'01'	P1, Start mit Recordnummer 1
4	'D7'	P2, spezifische Suche im SFI ‘1A‘
5	'04'	L_C
6	'04'	CTRLB
7	'00'	Offset Indicator Byte
8	'02'	Konfigurationsbyte
9	’F0‘	Suchmuster
10	'00'	L_e

Wenn das SEARCH RECORD erfolgreich ausgeführt wird, gibt die Chipkarte eine Antwortnachricht mit der folgenden Struktur zurück:

Byte	Länge	Wert	Erläuterung
1-n	n	'XX XX'	Recordnummer(n)
n+1	1	'XX'	Statusbyte SW1
n+2	1	'XX'	Statusbyte SW2

Es können nun gezielt nur die in der Antwortnachricht angegebenen Records ausgelesen werden.

B.2.2 Nachrichten generieren

Dieser Teil des Gesamtablaufs ist nur insofern chipkartenrelevant, als (optional) Bankverbindungsdaten, die für die Auftragsgenerierung benötigt werden, aus der Chipkarte entnommen werden. Dies ist bereits im Schritt „Signatur einleiten“ (Kap. B.2.1) geschehen. Für die folgende Ablaufbeschreibung wird angenommen, dass die Anwendung bereits Auftrags-Nachrichten generiert hat. Diese Nachrichten müssen jetzt ggf. noch kryptographisch gesichert werden, d.h. es werden Segmente für die elektronische(n) Signatur(en) und für die Verschlüsselung entsprechend den jeweiligen Spezifikationen eingefügt.

B.2.3 Nachrichten signieren

B.2.3.1 Nachrichten signieren bei HBCI

Die folgenden Abläufe können im Falle von HBCI offline, d.h. außerhalb des Übertragungsdialogs vollzogen werden. Dies gilt für alle Nachrichten mit Ausnahme der Dialoginitialisierung. Der Grund besteht darin, dass für die Absicherung aller Kreditinstitutsnachrichten der Schlüssel des Senders der Dialoginitialisierungsnachricht erforderlich ist. Daher muss auch die Chipkarte des Senders während des gesamten Dialogs im Endgerät stecken.

Die Abläufe für die Signatur der Dialoginitialisierungsnachricht sind grundsätzlich identisch mit den im Folgenden beschriebenen Abläufen für die Signatur von Auftragsnachrichten. Da aber für die Dialoginitialisierung anwendungsseitig noch weitere Chipkartendaten (Benutzerkennung, Dialog-ID, Kommunikationszugang etc.) benötigt werden, wird der komplette Ablauf einschließlich der Signatur der Dialoginitialisierung im Kap. B.2.5 "Übertragungsdialog" noch einmal beschrieben.

Chipkarte			Endgerät	
R1	BZ	à ß	M1	Sequenzzähler (Signatur-ID) ermitteln durch Aufruf der ZKA-SIG-API-Funktion *zka_sig_read_key_usage_counter* und anschließende Invertierung des Rückgabewerts gemäß Abschnitt B.1)
			A2	Signaturkopf aufbauen und in HBCI-Nachricht einfügen
			A3	Daten (Signaturkopf, HBCI-Nutzdaten) für Signatur bereitstellen
		à ß	M4	Signaturerstellung (siehe Kap. B.3.1)
			A5	Signaturabschluss aufbauen und in HBCI-Nachricht einfügen
			A6	ggf. M1 bis A5 für weitere Nachrichten wiederholen
			A7	signierte HBCI-Nachrichten zur Weiterverarbeitung speichern

♦ **Erläuterung**

1. Der Sequenzzähler (Signatur-ID) wird durch Auslesen der Bedienungszähler der Signaturanwendung und anschließende Berechnung ermittelt. Das Auslesen erfolgt durch Aufruf der ZKA-SIG-API-Funktion *zka_sig_read_key_usage_counter* mit der Parameterbelegung

   - counter_type = ’00‘ bei Verwendung des S_K.CH.DS, bzw.
   - counter_type = ‘02‘ bei Verwendung des S_K.CH.$AUT_{C/S}$

Das Ergebnis BZ wird gemäß Kap. B.1 zu SZ = **neg**(BZ) invertiert und als Sequenzzähler gespeichert.

2. Der Signaturkopf wird aufgebaut und in die HBCI-Nachricht eingefügt.
3. Die Daten (Signaturkopf, HBCI-Nutzdaten) für die Signaturerstellung werden bereitgestellt.
4. Die Signatur wird berechnet (siehe hierzu Kap. B.3).
5. Der Signaturabschluss wird aufgebaut und in die HBCI-Nachricht eingefügt.
6. Ggf. können die Schritte 1 bis 5 für weitere Nachrichten wiederholt werden.
7. Die signierten HBCI-Nachrichten können zur Weiterverarbeitung gespeichert werden.

Anmerkung: Für Mehrfachsignaturen wird jeweils die Abfolge „Signatur einleiten“ – „Nachrichten signieren“ – „Signatur beenden“ wiederholt. Dies kann auch zu einem späteren Zeitpunkt geschehen. Mehrfachsignaturen müssen jedoch abgeschlossen sein, bevor die Verschlüsselung der Nachricht (Kap. B.2.4) durchgeführt wird.

B.2.4 Nachrichten verschlüsseln

B.2.4.1 Nachrichten verschlüsseln bei HBCI

Die Chipkarte ist bei der eigentlichen Nachrichtenverschlüsselung nicht involviert. Die Software berechnet einen Sessionkey, verschlüsselt das Dokument und verschlüsselt den Sessionkey zur Übertragung mit dem öffentlichen Key-Encryption-Schlüssel $P_K.RECV_{INST}.KE$ des empfangenden Instituts, welches der übermittelten Kreditinstitutsnachricht entnommen wurde[6].

Allerdings wird die Chipkarte zur Berechnung von Zufallszahlen herangezogen, welche den Session Key bilden.

Chipkarte			Endgerät	
			A1	Daten (HBCI-Nutzdaten und ggf. Signaturkopf/-abschluss) für die Verschlüsselung bereitstellen
R2	RND	ß à	C2	Aufruf der ZKA-SIG-API-Funktion *zka_sig_get_challenge*
			A2	RND als Nachrichtenschlüssel-Hälfte KS_L speichern
R3	RND	ß à	C3	Aufruf der ZKA-SIG-API-Funktion *zka_sig_get_challenge*
			A3	RND als Nachrichtenschlüssel-Hälfte KS_R speichern
			A4	KS_L mit KS_R zu KS konkatenieren und speichern
			A5	KS auf Eigenschaft „(halb-)schwacher Schlüssel“ überprüfen und ggfs. Schritte 2-4 wiederholen.
			A6	Herstellung der Parität für KS (Parity Adjustment)
			A7	Daten mit KS (symmetrisch) verschlüsseln
			A8	KS mit $P_K.RECV_{INST}.KE$ (asymmetrisch) verschlüsseln
			A9	Verschlüsselungskopf aufbauen und in HBCI-Nachricht einfügen
			A10	Verschlüsselte Daten als Binärdaten in HBCI-Nachricht einfügen
			A11	ggf. A1 bis A10 für weitere Nachrichten wiederholen
			A12	Verschlüsselte und signierte HBCI-Meldungen zur weiteren Bearbeitung speichern

♦ **Erläuterung**

1. Die Daten (HBCI-Nutzdaten und ggf. Signaturkopf/-abschluss) für die Verschlüsselung werden bereitgestellt.
2. Mit dem Aufruf der ZKA-SIG-API-Funktion *zka_sig_get_challenge* lässt sich das Terminal eine Zufallszahl von der HBCI-Karte geben.

   Wenn das Kommando erfolgreich ausgeführt wurde, gibt die HBCI-Karte eine 8 Byte lange Zufallszahl als Antwortdatum aus, die als Nachrichtenschlüssel-Hälfte KS_L gespeichert wird.
3. Mit dem Aufruf der ZKA-SIG-API-Funktion *zka_sig_get_challenge* lässt sich das Terminal eine weitere Zufallszahl von der HBCI-Karte geben, die als Nachrichtenschlüssel-Hälfte KS_R gespeichert wird.

---

[6] [DIN-SIG4, Kapitel 6.10.1]: „If an enciphered document is sent, the card is not involved: the software computes the content encryption key, enciphers the document and finally enciphers the content encryption key by applying the receiver’s public key taken from the receiver’s KE certificate.“

4. KS_L wird mit KS_R zu KS konkateniert und gespeichert.
5. KS wird auf die Eigenschaft „(halb-)schwacher Schlüssel“ überprüft. Liegt ein (halb-)schwacher Schlüssel vor, so wird Schritt 2-4 wiederholt.

Schwache Schlüssel des DES:

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

Halbschwache Schlüssel des DES:

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

6. Für KS wird ein Parity Adjustment durchgeführt. Das Resultat ist der zu verwendende Session-Key.
7. Die zu übertragenden Daten werden mit KS symmetrisch verschlüsselt (Triple-DES CBC-Mode, IV=0, ANSI X9.23 Padding).
8. Der Session-Key KS wird linksbündig mit Nullbits auf 768 Bit aufgefüllt und anschließend mit dem öffentlichen Key-Encryption-Schlüssel $P_K.RECV_{INST}.KE$ des empfangenden Instituts, welches der übermittelten Kreditinstitutsnachricht entnommen wurde, verschlüsselt. Stimmt das Verschlüsselungsergebnis mit dem Ausgangswert überein, werden die Schritte 2 bis 8 wiederholt (Generierung eines neuen Schlüssels); ansonsten wird das Ergebnis mit führenden Nullbits auf 1024 Bit erweitert und es wird mit dem folgenden Schritt 9 fortgefahren.
9. Der Verschlüsselungskopf wird aufgebaut und in die HBCI-Nachricht eingefügt.
10. Die verschlüsselten Daten als Binärdaten in die HBCI-Nachricht eingefügt.
11. Ggf. werden die Schritte 1 bis 10 für weitere Nachrichten wiederholt.
12. Die verschlüsselten und signierten HBCI-Meldungen werden zur weiteren Bearbeitung gespeichert.

B.2.5 Übertragungsdialog

Chipkarte			Endgerät			Kreditinstitut	
			A1	Benutzerkennung aus der bereits gelesenen Bankverbindung extrahieren			
		à ß	M2	Nachricht signieren (s. Kap. B.2.3)			
			A3	Kommunikationszugang aus Bankverbindung herstellen			
			C4	Nachricht (beginnend mit Dialoginitialisierungsnachricht) senden	à ß	R4	Antwortnachricht
			A5	falls Antwortnachricht verschlüsselt: Daten (Binärdaten nach dem Verschlüsselungskopf) und verschlüsselten Session-Key enc(KS) aus dem Signaturkopf für die Entschlüsselung bereitstellen			
		à ß	M6	Ausführung der ZKA-SIG-API-Funktion *zka_sig_decrypt* zur Session-Key-Entschlüsselung, Resultat ist der Session-Key KS			
			A7	Daten mit Session-Key KS entschlüsseln.			
			A8	falls Kreditinstitutsnachricht signiert: Daten (Signaturkopf, Nutzdaten, Signatur) für Signatur-Prüfung bereitstellen			
		à ß	M9	Signatur-Prüfung (siehe KapB.3.2)			
			A10	C4 bis M9 für alle weiteren HBCI-Nachrichten wiederholen			

B.2.6 Signatur beenden

Chipkarte	

Endgerät	
M1	Aufruf der ZKA-SIG-API-Funktion *zka_sig_fini_signature_application*
M2	Aufruf der ZKA-SIG-API-Funktion *zka_sig_close*

- **Erläuterung**

1. Die ZKA-SIG-API-Funktion *zka_sig_fini_signature_application* wird ausgeführt. Diese Funktion setzt die ZKA-SIG-API in den Zustand „passiv“ und löscht die darin gespeicherten Werte.
2. Die ZKA-SIG-API-Funktion *zka_sig_close* gibt die Verbindung zum Kundenterminal wieder frei.

B.3 Makros

B.3.1 Signatur-Berechnung

Signaturen mit der Chipkarte werden im Rahmen der beiden Sicherheitsdienste „Authentication“ und „Non-Repudiation“ erzeugt.

- Sicherheitsdienst Authentication: Signatur mit Schlüssel S_K.CH.$AUT_{C/S}$ (Client-Server-Authentikations-Schlüssel)
- Sicherheitsdienst Non-Repudiation: Signatur mit Schlüssel S_K.CH.DS (Digitaler Signatur-Schlüssel)

Die tatsächliche Durchführung der Signatur durch die Chipkarte ist insbesondere an die Erfüllung von Zugriffsbedingungen geknüpft, hier sind dies insbesondere eine vorhergehende Benutzer-Authentikation in Form der Verifikation

- des CSA-Passworts für die Erlaubnis zur Signatur mit dem Schlüssel S_K.CH.$AUT_{C/S}$
- der Signatur-PIN für die Erlaubnis zur Signatur mit dem Schlüssel S_K.CH.DS

Durch einen in der Chipkarte personalisierten Parameter der Signatur-Anwendung [ZKASIG] wird dabei festgelegt, nach wie vielen elektronischen Signaturen spätestens die Benutzer-Authentikation zu wiederholen ist. Eine Benutzer-Authentikation wird bei Bedarf innerhalb der ZKA-SIG-API-Funktionen *zka_sig_digital_signature* bzw. *zka_sig_cs_authentication* durchgeführt.

Chipkarte			Endgerät	
R1	evtl. Hash-wert	ß à	M1	Hashwert HASH berechnen, optional unter Verwendung der ZKA-SIG-API-Funktion *zka_sig_hash*
R2a	Signatur	ß à	M2a	Sicherheitsdienst Non-Repudiation: Aufruf der ZKA-SIG-API-Funktion *zka_sig_digital_signature*
				oder:
R2b	Signatur	ß à	M2b	Sicherheitsdienst Authentication: Aufruf der ZKA-SIG-API-Funktion *zka_sig_cs_authentication*

♦ **Erläuterung**

1. Die Berechnung des Hashwertes erfolgt in der Regel außerhalb der Chipkarte (Hashalgorithmus gemäß Vorgabe für den Sicherheitsdienst bzw. vom Institut übermittelter BPD). Optional ist es auch möglich, den letzen Schritt oder alle Schritte der Hashwert-Berechnung durch die Chipkarte durchführen zu lassen. Diese Berechnung ist dann Bestandteil des Ablaufs der ZKA-SIG-API-Funktion *zka_sig_hash*. Der zu verwendende Hash-Algorithmus wird dabei in Form der zugehörigen OID übergeben:
   - OID = 1.3.14.3.2.26 für SHA-1
   - OID = 1.3.36.3.2.1 für RIPEMD-160

2a. Bei Verwendung des Schlüssels S_K.CH.DS (Sicherheitsdienst Non-Repudiation) wird die Signatur durch Aufruf der ZKA-SIG-API-Funktion *zka_sig_digital_signature* erzeugt. Die Auswahl des Signaturalgorithmus und Paddingverfahrens erfolgt gemäß Vorgabe für den Sicherheitsdienst bzw. vom Institut übermittelter BPD. Die Signaturanwendung der Chipkarte bietet die Verfahren „sha-1WithRSAEncryption“ (PKCS#1-Signaturverfahren, Standard-RSA, SHA-1) und „sigS_ISO9796-2rndWithripemd160“ (DIN-Signaturverfahren, Standard-RSA, RIPEMD-160) an.

Falls der Hashwert im vorangegangenen Schritt 1 durch die Chipkarte berechnet wurde, ist er noch in der Chipkarte gespeichert und braucht nicht erneut als Parameter des *zka_sig_digital_signature* übergeben zu werden.

2b. Bei Verwendung des Schlüssels S_K.CH.$AUT_{C/S}$ (Sicherheitsdienst Authentication) wird die Signatur durch Aufruf der ZKA-SIG-API-Funktion *zka_sig_cs_authentication* erzeugt. Die Chipkarte verwendet dabei intern ein Paddingformat gemäß PKCS#1 ([SECCOS, Kapitel 8.3.2.1][7]), wobei die Digest-Info nicht von der Chipkarte selbst erzeugt wird, sondern als aufbereiteter „Authentication-Input“ (= zu signierendes Datenfeld) übergeben werden muss.

Der Authentication-Input ist wie folgt aufgebaut ([SECCOS, Kapitel 8.1.8.3.1]):

Tag	Länge	Wert	Erläuterung
'30'	'21'		Tag und Länge von SEQUENCE
'30'	'09'		Tag und Länge von SEQUENCE
'06'	'05'	'2B 0E 03 02 1A' bzw. '2B 24 03 02 01'	OID des SHA-1 (1 3 14 3 2 26) bzw. OID des RIPEMD-160 (1 3 36 3 2 1)
'05'	'00'	-	TLV-Kodierung von NULL
'04'	'14'	'XX..XX'	Hash-Wert

Anmerkung: Die direkte Weiterverwendung eines eventuell im Chip berechneten und dort zwischengespeicherten Hashwerts ist bei der Signatur im Sicherheitsdienst „Authentication“ nicht möglich. Der Hashwert (als Ergebnis von Schritt 1) muss daher explizit als Aufrufparameter in der oben beschriebenen Form in Schritt 2 übergeben werden.

B.3.2 Signatur-Prüfung

Die ZKA-Chipkarte selbst unterstützt zurzeit keine Signatur-Prüfung[8]. Die Prüfung einer Signatur wird vom Kundenterminal-Makro “Überprüfen der Korrektheit der elektronischen Unterschrift” durchgeführt.

---

[7] Auszug aus [SECCOS, Kapitel 8.3.2.1]: Falls der Authentication Input nicht zu lang ist, wird er zu einer Folge von N-1 Byte wie folgt formatiert:

Bezeichnung	Byte-Länge	Wert
Blocktyp	1	'01'
Paddingfeld (PS)	N-3-L	'FF...FF'
Separator	1	'00'
Datenfeld	L	Authentication Input (AI)

[8] [ZKASIG, Kapitel 1.1]: „Die ZKA-Chipkarte unterstützt [die] Signaturprüfung zur Zeit aus dem folgenden Grund nicht: Die Prüfung digitaler Signaturen, die mit beliebigen privaten Schlüsseln und/oder Algorithmen berechnet sind, würde voraussetzen, dass die Chipkarte X.509-Zertifikate auswertet. Dies ist gemäß Kapitel 16.1 von [DINSIG] zur Zeit nicht möglich.“

Die (mathematische) Korrektheit einer elektronischen Unterschrift wird überprüft, in dem sie mit dem entsprechenden öffentlichen Schlüssel entschlüsselt wird und das Ergebnis mit dem Hashwert über die signierten Daten verglichen wird. Der für die Überprüfung der elektronischen Signatur eingesetzte öffentliche Schlüssel liegt in dem Kundenterminal authentisch vor, falls die zu ihm gehörende Zertifikatshierarchie vorher ebenfalls in dem Kundenterminal überprüft wurde [KT-KONZEPT].