Einführung der Gesundheitskarte

Spezifikation der elektronischen Gesundheitskarte

Teil 1: Spezifikation der elektrischen Schnittstelle

Version:	2.2.0
Stand:	20.03.2008
Status:	freigegeben

Dokumentinformationen

Änderungen zur Vorversion

Diese Version unterscheidet sich in folgenden Punkten von der Version 2.1.0:

- Kapitel 7.2.1, Tabelle: Referenzen zu T_2 korrigiert
- Kapitel 7.2.2, Tabelle: Beschreibung zu T_2 ergänzt
- In Kapitel 7.6.1.1 Hinweis (13): ergänzt
- Kapitel 9.4 §1 und (N162) modifiziert, ENABLE und DISABLE optional
- Kanalkontext modifiziert, 2 globale und 2 DF spezifische Benutzerverifikationen sind hinreichend, dies hat Auswirkungen auf:
  – Kapitel 9.8, 13.1
  – (N222)a.i,(N301)i, (N301)j, (N482)b, (N746), (N785), (N809), (N829)a.ii, (N829)c.i, (N1042)c
- (N395) präzisiert
- (N402) modifiziert, Vorgaben entfernt für den Fall, dass mittels LOAD APPLICATION kein File angelegt wurde
- Tabelle 45 Zeile mit Ne ergänzt
- (N482)b modifiziert, keyReferenceList wird beim SELECT stets gelöscht
- InternalAuth rsaSessionkey4SM (N869)f.ii korrigiert
- GAKP: (N977) korrigiert
- GAKP: (N978) neu aufgenommen, dadurch neue Fehlermeldung in Tabelle 151
- Abbildung 4: Antwort von ExternalAuthenticate korrigiert
- Abbildung 6 und Abbildung 7 editorisch ergänzt
- AlgID gegenseitige asymmetrische Authentisierung: (N1068) und (N1070) korrigiert
- Tabelle 172: Performancevorgaben verändert
- CAMS durch CMS ersetzt (Harmonisierung mit anderen gematik Dokumenten)

Referenzierung

Die Referenzierung in weiteren Dokumenten der gematik erfolgt unter:

[gemSpec_eGK_P1] gematik (20.03.2008): Einführung der Gesundheitskarte – Spezifikation elektronische Gesundheitskarte;
Teil 1 – Spezifikation der elektrischen Schnittstelle
Version 2.2.0, www.gematik.de

Dokumentenhistorie

Version	Stand	Kap./ Seite	Grund der Änderung, besondere Hinweise	Bearbeitung
1.3.0 … 1.5.9	04.04.07 … 02.11.07		Dokument erstellt Gegenüber der Version 1.2.0 wurde dieses Dokument editoriell komplett überarbeitet. Zudem wurden neue Anforderungen eingearbeitet. Dies betrifft im Wesentlichen kryptographische Operationen.	gematik, AFI
1.6.0	06.11.07		freigegeben zur Vorkommentierung	gematik
1.6.1	30.11.07		Kommentare eingearbeitet	gematik, AFI
2.0.0	13.12.07		freigegeben	gematik
2.0.1	11.01.08		Vorgaben zur Performance	SPE/DK, AFI
2.0.2	17.01.08		Prüfpunktbeschreibung ergänzt	SPE/DK, AFI
2.0.3 2.0.4	30.01.08		Kommentare eingearbeitet	SPE/DK, AFI
2.0.5	31.01.08		• Gewichte in Tabelle 172 angepasst • DISABLE, ENABLE VERIFICATION REQUIREMENT aus normativem Umfang entfernt • Kommentare eingearbeitet	SPE/DK, AFI
2.0.6	07.02.08		• editorische Ergänzungen	SPE/DK, AFI
2.1.0	20.02.08		freigegeben	gematik
2.1.1	10.03.08		• Benutzerverifikation: 2 globale und 2 DF spezifische sind hinreichend • editorische Ergänzungen und Korrekturen	SPE/DK, AFI
2.2.0	20.03.08		freigegeben	gematik

Inhaltsverzeichnis

1 Zusammenfassung

Die Dokumentation für die elektronische Gesundheitskarte besteht aus mehreren technischen Spezifikationen, ergänzenden Dokumenten und organisatorischen Festlegungen. Die Spezifikationen beschreiben den Aufbau und die Funktionsweise der eGK als solche. Die ergänzenden Dokumente definieren die in den Spezifikationen beschriebenen Verfahren sowie die Handhabung der Zertifikate.

1.1 Technische Spezifikationen zur eGK

- **Die Spezifikation der elektronischen Gesundheitskarte**
  Teil 1: Spezifikation der elektrischen Schnittstelle

  Im Teil 1 werden die Basiskommandos, die Grundfunktionen des Betriebssystems sowie die grundlegenden Sicherheitsfunktionen und -algorithmen (hard facts) detailliert beschrieben.

  Diese Spezifikation ist Grundlage der Entwicklung der Kommandostrukturen und Funktionen für eGK-konforme Chipkartenbetriebssysteme; sie ist somit die Grundarchitektur für die ROM-Maske des Halbleiters.

- **Die Spezifikation der elektronischen Gesundheitskarte**
  Teil 2: Anwendungen und anwendungsspezifische Strukturen

  Im Teil 2 werden die anwendungsspezifischen Strukturen der eGK beschrieben. Dieser Teil enthält die Spezifikationen für die Strukturen der Anwendungen, die bei der Initialisierung und Personalisierung in die eGK geladen werden. Außerdem werden in diesem Teil die Zugriffsrechte auf Elemente der eGK festgelegt.

- **Die Spezifikation der elektronischen Gesundheitskarte**
  Teil 3: Äußere Gestaltung

  Der Teil 3 beschreibt die äußere Gestaltung der eGK. Es werden die Bereiche auf der eGK festgelegt, in denen Lichtbild des Versicherten, Texte und Logos vorgesehen sind, und die dazugehörenden Formate definiert. Die Kartenrückseite wird entsprechend den Vorgaben für die europäische Krankenversicherungskarte (EHIC) bedruckt.

1.2 Ergänzende Dokumente zur eGK

- **Speicherstrukturen der eGK für Gesundheitsanwendungen**

  Das Dokument fasst die Daten und Datenstrukturen zusammen, die für die Realisierung der Fachanwendungen wie z.B. Versichertendatenmanagement, Notfalldatenmanagement, Verordnungsdatenmanagement, Verwaltung freiwilliger Anwendungen und Protokollierung maßgeblich sind.

- **Übergabeschnittstelle für die Produktion der eGK**

  In diesem Dokument werden die Daten beschrieben, die für die Herstellung der eGK im Rahmen der gesetzlichen Vorgaben notwendig sind. Die Frage, wer die Daten jeweils erzeugt und wem wie übergibt, muss zwischen Kartenherausgeber und Personalisierer bilateral vereinbart werden.

  Die Verteilung der Aufgaben zwischen den Kartenherausgebern, den Modulen des Kartensystems, den CA/ZDA und den Kartenproduzenten muss jeweils vertraglich festgelegt und dann über definierte Schnittstellen abgewickelt werden.

  Das Format der auf die eGK zu übertragenden Daten ist in XSD-Schemata festgelegt. Zu der Datenübergabeschnittstelle Personalisierung gehören XSD-Schema für

  - den Personalisierungsauftrag
  - die Rückmeldedaten zum Personalisierungsauftrag
  - die persönlichen Versichertendaten
  - die allgemeinen Versicherungsdaten
  - die geschützten Versichertendaten
  - die Typdefinitionen
  - und die Sammlung von Schlüsselausprägungen

- **Personalisierung kryptographischer Daten**

  In diesem Dokument wird für die Sicherheit der kryptographischen Daten durch alle an der Personalisierung einer eGK beteiligten Organisationen ein Mindestniveau festgelegt. Die zugehörigen Sicherheitsanforderungen beziehen sich dabei nicht nur auf die Verarbeitung der kryptographischen Daten durch eine Organisation, sondern auch auf den Transport dieser Daten zwischen den beteiligten Organisationen. Das definierte Mindestniveau für die Sicherheit ist verpflichtend für alle beteiligten Organisationen.

- **PKI für die X.509-Zertifikate Grobkonzept**

Zur Identifikation von Personen, Objekten, Organisationen, Geräten, Rechten und Rollen werden elektronische Zertifikate verwendet, bei denen die Identität durch eine übergeordnete „vertrauenswürdige" Instanz mittels einer elektronischen Signatur bestätigt wird.

Für die übergeordneten X.509-Zertifikate der ausstellenden Organisationen, der sog. Trust Service Provider (TSP), wird das Konzept der zentralisierten (Online-) Zertifikatsprüfung umgesetzt.

Das vorliegende Dokument trifft und erläutert die zum Vertrauensmodell der TSL notwendigen Festlegungen und verweist auf die jeweiligen weiterführenden Spezifikationen.

- **Verwendung von Zertifikaten in der Telematikinfrastruktur**

  Das Dokument beschreibt die unterschiedlichen Typen von Zertifikaten und deren Herausgabe und Nutzung in der Telematikinfrastruktur und stellt normative Vorgaben zur detaillierten Prüfung und Auswertung dieser Zertifikate auf, insbesondere bzgl. Prüfung des Vertrauensraums und des Zertifikatsstatus.

- **Festlegung einer einheitlichen X.509-Zertifikatsinfrastruktur für die Telematik im Gesundheitswesen**

  In dem Dokument werden die Vor- und Nachteile verschiedener Konzepte zur Verknüpfung von Public-Key-Infrastrukturen verglichen und ein konkreter Lösungsweg zur praktischen Umsetzung vorgeschlagen.

  Das Konzept zur flexiblen und vertrauenswürdigen Einbindung der verschiedenen Public-Key-Infrastrukturen durch die Schaffung einer „Trust Service List" wird beschrieben. Diese ermöglicht eine zentrale Sammlung und Verteilung der Root-Zertifikate unter Einhaltung eines einheitlichen Sicherheitsniveaus.

- **PKI für X.509-Zertifikate: Registrierung eines Trust Service Provider**

  Für die übergeordneten X.509-Zertifikate der ausstellenden Organisationen, der sog. Trust Service Provider (TSP), wird das Konzept der zentralisierten (Online-) Zertifikatsprüfung umgesetzt.

  Ein TSP muss in der gematik Trust-service Status List (gematik-TSL) eingetragen sein. Um dies beantragen zu können, muss sich der TSP vorher bei der gematik registrieren lassen.

  Das Dokument beschreibt den Prozess der Registrierung eines TSP durch die gematik.

- **Festlegungen zu den X.509-Zertifikaten der Versicherten**

  Die Inhalte aller personenbezogenen X.509-Zertifikate zur Authentifizierung (AUT und AUTN), Verschlüsselung (ENC und ENCV) und qualifizierten Signatur (QES) werden detailliert dargestellt. Das Dokument trifft die erforderlichen Festlegungen zur Versichertenidentität, zur Pseudonymisierung von AUTN und ENCV, sowie zur Schlüsselverwendung.

- **PKI für CV-Zertifikate: Grobkonzept**

  CV-Zertifikate dienen der C2C-Authentisierung von Mikroprozessorchipkarten, hier insbesondere der eGK und HBA, sowie SMC. Bei Anwendung der CV-Zertifikate erfolgt zwischen eGK und HBA (bzw. SMC) die vorgeschriebene gegenseitige Authentikation.

  Das Grobkonzept beschreibt

  - den grundsätzlichen Aufbau der PKI für CV-Zertifikate,
  - die technischen und organisatorischen Rahmenbedingungen für die Nutzung der CV-Zertifikate,
  - die zu realisierenden Sicherheitslevel und
  - die grundsätzlichen Vorgaben für die zu schaffenden Policies und die Umsetzung der Sicherheitskonzepte für die Herausgabe von CV-Zertifikaten.

- **PKI für CV-Zertifikate: Registrierung einer CVC-CA der zweiten Ebene**

  Das Dokument beschreibt den Prozess der Registrierung einer CVC-CA durch die gematik. Dabei werden die Mindestanforderungen an eine CVC-CA aus [gemPKI_Reg] konkretisiert. Zusätzlich wird der Prozess für das Beantragen und Ausstellen eines CV-Zertifikates für eine CVC-CA durch die Root-CVC-CA detailliert beschrieben.

2 Einführung

2.1 Zielsetzung und Einordnung des Dokuments

Diese Spezifikation definiert die Anforderungen an die Funktionalität einer Betriebssystemplattform für die elektronische Gesundheitskarte (eGK), die internationalen Standards entsprechen und die internationale und europäische Interoperabilität sicherstellen.

Im Einzelnen werden auf der Basis von [7816–4], [7816–8] und [7816–9] Kommandos und Optionen beschrieben, die von der eGK unterstützt werden müssen.

Der Aufbau dieses Dokumentes gliedert sich wie folgt:

- Kapitel 1 Zusammenfassung enthält die wichtigsten Aussagen dieses Dokumentes.
- Kapitel 2 Einführung enthält Aussagen zum Umgang mit diesem Dokument.
- Kapitel 3 Anforderungen und Annahmen (informativ) wird in einer späteren Version Eingangsanforderungen enthalten, welche die Basis für die normativen Aussagen in späteren Kapiteln bilden werden.
- Kapitel 4 Systemüberblick (informativ) wird in einer späteren Version Grundlagen zu Betriebssystemen für Chipkarten enthalten, die auf der Normenreihe ISO/IEC 7816 basieren.
- Kapitel 5 Lebenszyklus von Karte und Applikation (informativ) grenzt den Gültigkeitsbereich der eGK Spezifikation aus zeitlicher Sicht ab.
- Kapitel 6 Datentypen und Datenkonvertierung definiert einige grundlegende Datentypen, welche die normativen Beschreibungen in späteren Kapiteln vereinfachen.
- Kapitel 7 Kryptographische Algorithmen (normativ) definiert einige grundlegende kryptographischen Funktionen, welche die normativen Beschreibungen in späteren Kapiteln vereinfachen.
- Kapitel 8 CV–Zertifikat (normativ) beschreibt Zertifikate, welche zu nutzen sind um öffentliche Schlüssel in die eGK zu importieren.
- Kapitel 9 Objekte (normativ) enthält eine Art Klassendiagramm in textueller Form. Die dort definierten Objekte und Attribute vereinfachen die normativen Beschreibungen in nachfolgenden Kapiteln.
- Kapitel 10 Objektsystem (normativ) beschreibt, wie sich Informationen persistent auf einer eGK hierarchisch speichern lassen.
- Kapitel 11 Zugriffskontrolle (normativ) beschreibt, wie den Schutz von Informationen auf einer eGK vor unberechtigtem Zugriff.
- Kapitel 12 Kommunikation (normativ) beschreibt, wie die eGK Informationen mit anderen Systemen austauscht.

- Kapitel 13 Kanalkontext (normativ) beschreibt die Informationen, welche volatil und kanalspezifisch (zu logischen Kanälen siehe auch [7816–4]) von der eGK gespeichert werden.
- Kapitel 14 Gesicherte Kommunikation (normativ) beschreibt, wie die eGK Informationen kryptographisch geschützt mit anderen Systemen austauscht.
- Kapitel 15 Kommandos (normativ) enthält normative Aussagen zu Kommandos, welche an eine eGK geschickt werden und normative Aussagen, wie diese Kommandos von der eGK zu bearbeiten sind. Im Wesentlichen ist dies das wichtigste Kapitel des Dokumentes, weil es die äußere Sichtweise auf das Verhalten der eGK an der elektrischen Schnittstelle am umfassendsten beschreibt.
- Kapitel 16 Authentisierungsprotokolle (normativ) beschreibt Sequenzen, die aus mehr als einem Kommando bestehen.
- Kapitel 17 Verschiedenes (normativ) spezifiziert konkrete Werte für eine Reihe von Platzhaltern.

Das Dokument ist „bottom up“ aufgebaut, das bedeutet Artefakte werden zunächst beschrieben und definiert bevor sie verwendet werden. Für eine „top down“ Herangehensweise empfiehlt es sich mit Kapitel 15 zu beginnen. Dort werden, wenn möglich, Verweise auf andere Kapitel gesetzt, wenn Dinge dort ausführlicher beschrieben werden. Wegen der besonderen Bedeutung des Kapitels 15 wird dessen Aufbau im Folgenden näher beleuchtet:

Kapitel 15 enthält alle in der Normenreihe ISO/IEC 7816 standardisierten Kommandos. Der besseren Übersichtlichkeit halber ist Kapitel 15 unterteilt in die Abschnitte, Management des Objektsystems, Zugriff auf Daten in transparenten EF, Zugriff auf strukturierte Daten, Benutzerverifikation, Komponentenauthentisierung, Kryptoboxkommandos und Verschiedenes. Jeder Abschnitt enthält eine Reihe von Unterabschnitten mit Kommandos in alphabetischer Reihenfolge.

Am Beispiel des verhältnismäßig einfachen Kommandos READ BINARY wird hier die Spezifikation eines Kommandos erläutert: Am Anfang steht eine allgemeine Beschreibung des Kommandos. Es folgen in den Unterkapiteln 15.3.2.1 und 15.3.2.2 Vorgaben für Kommando APDUs. In diesem Fall wird anhand von (N517) deutlich, dass die Vorgaben für Kommando APDUs sich an nutzende Systeme und nicht an das COS richten. Die Vorgaben an das COS stehen in einem eigenen Unterabschnitt „Kommandoabarbeitung innerhalb der Karte“.

2.2 Zielgruppe

Das Dokument richtet sich an Hersteller von Chipkartenbetriebssystemen und an Anwendungsprogrammierer, die unmittelbar mit der Chipkarte kommunizieren.

2.3 Geltungsbereich

Der Inhalt des Dokumentes ist verbindlich für die Erstellung elektronischer Gesundheitskarten.

2.4 Arbeitsgrundlagen

Die Ausarbeitung steht in engem Zusammenhang mit der Spezifikation des Heilberufsausweises.

2.5 Abgrenzung des Dokuments

Dieses Dokument spezifiziert das Verhalten an der elektrischen Schnittstelle zu einem Chipkartenbetriebssystem (COS). Dieses Dokument spezifiziert NICHT die Architektur des COS. Der einfacheren Darstellung wegen wird in diesem Dokument von einer modularen Aufteilung des COS ausgegangen. Die hier beschriebene Aufteilung ist nicht verpflichtend. Es wird aber empfohlen sich an dieser Aufteilung zu orientieren, weil bei künftigen Ergänzungen und Erweiterungen die hier beschriebene Aufteilung zu Grunde gelegt wird.

Die Konfiguration einer Chipkarte, also die Festlegung welche Applikationen, Ordner, Dateien, Schlüssel und Passwörter auf einer Versichertenkarte zu finden sind, ist nicht Gegenstand dieses Dokumentes.

In Absprache mit den Verantwortlichen des Dokumentes [gemSpec_Krypt] werden in diesem Dokument bewusst Redundanzen zum vorgenannten Dokument akzeptiert. Trotzdem ist [gemSpec_Krypt] relevant für eine konkrete eGK, da dort, anders als in diesem Dokument, normative Vorgaben für die Nutzungsdauer gewisser kryptographischer Verfahren getroffen werden, die hier beschrieben werden.

2.6 Methodik

2.6.1 Nomenklatur

Tabelle 1: Präfixe, die auf Vielfachen von Zehnerpotenzen beruhen:

Name	Symbol	Wert gemäß SI	nächstliegende Zweierpotenz
kilo	k	10^{3} = 1.000	2^{10} = 1.024
mega	M	10^{6} = 1.000.000	2^{20} = 1.048.576
giga	G	10^{9} = 1.000.000.000	2^{30} = 1.073.741.824
tera	T	10^{12} = 1.000.000.000.000	2^{40} = 1.099.511.627.776
peta	P	10^{15} = 1.000.000.000.000.000	2^{50} = 1.125.899.906.842.624
exa	E	10^{18}	2^{60}
zetta	Z	10^{21}	2^{70}
yotta	Y	10^{24}	2^{80}

Die folgende Tabelle basiert auf [BinPrefix].

Tabelle 2: Präfixe, die auf Vielfachen von Zweierpotenzen beruhen:

Name	Symbol	Wert
kibi	Ki	$2^{10} = 1024^1 = 1.024$
mebi	Mi	$2^{20} = 1024^2 = 1.048.576$
gibi	Gi	$2^{30} = 1024^3 = 1.073.741.824$
tebi	Ti	$2^{40} = 1024^4 = 1.099.511.627.776$
pebi	Pi	$2^{50} = 1024^5 = 1.125.899.906.842.624$
exbi	Ei	$2^{60} = 1024^6 = 1.152.921.504.606.846.976$
zebi	Zi	$2^{70} = 1024^7 = 1.180.591.620.717.411.303.424$
yobi	Yi	$2^{80} = 1024^8 = 1.208.925.819.614.629.174.706.176$

Hinweis (1): Beispiel: 300 GB ≈ 279,4 GiB, sprich 300 giga Byte sind ungefähr 279,4 gibi Byte

In diesem Dokument wird eine objektorientierte Sichtweise verfolgt. Dazu werden etwa die Artefakte Datei (EF in der Nomenklatur nach [7816–4]) oder Schlüssel als Objekte aufgefasst und die Eigenschaften als Attribute des Objektes. Wenn Attribute eines Objektes gemeint werden, dann wird die Notation *obj.attribute* verwendet. Falls das Attribut wieder ein Objekt mit weiteren Attributen ist, dann sind auch längere Bezeichnungen möglich.

G1	Abkürzung für Generation 1, bezeichnet diese Version des Dokumentes, in der Regel ergänzt um den Zusatz „normativ“
G2	Abkürzung für Generation 2, bezeichnet eine spätere Version dieses Dokumentes, in der Regel ergänzt um den Zusatz „informativ“
´1D´	Hexadezimale Zahlen und Oktettstrings werden in Hochkomma eingeschlossen
x || y	Das Symbol || steht für die Konkatenierung von Oktettstrings oder Bitstrings ´1234´ || ´5678´ = ´12345678´
y = x	Der Variablen y wird der Wert von x zugewiesen (Standardnotation in gängigen Programmiersprachen).
y **ß** x	Der Variablen y wird der Wert von x zugewiesen (Notation aus [TR–03111]).

An der Benutzerschnittstelle werden für Benutzergeheimnisse andere Bezeichnungen verwendet, als in technischen Dokumenten. Tabelle 3 listet die Zuordnung.

Tabelle 3: Zuordnung der Bezeichnungen für PINs

Bezeichnung Benutzerschnittstelle	Bezeichnung in technischen Dokumenten
Praxis PIN	PIN.CH
Privat PIN	PIN.home
Signatur PIN	PIN.QES

2.6.2 Verwendung von Schüsselworten

Für die genauere Unterscheidung zwischen normativen und informativen Inhalten werden die dem [RFC2119] entsprechenden in Großbuchstaben geschriebenen, deutschen Schlüsselworte verwendet:

- **MUSS** bedeutet, dass es sich um eine absolutgültige und normative Festlegung bzw. Anforderung handelt.
- **DARF NICHT** bezeichnet den absolutgültigen und normativen Ausschluss einer Eigenschaft.
- **SOLL** beschreibt eine dringende Empfehlung. Abweichungen zu diesen Festlegungen sind in begründeten Fällen möglich. Wird die Anforderung nicht umgesetzt, müssen die Folgen analysiert und abgewogen werden.
- **SOLL NICHT** kennzeichnet die dringende Empfehlung, eine Eigenschaft auszuschließen. Abweichungen sind in begründeten Fällen möglich. Wird die Anforderung nicht umgesetzt, müssen die Folgen analysiert und abgewogen werden.
- **KANN** bedeutet, dass die Eigenschaften fakultativ oder optional sind. Diese Festlegungen haben keinen Normierungs- und keinen allgemeingültigen Empfehlungscharakter.

Abwandlungen von „**MUSS**“ zu „**MÜSSEN**“ etc. sind der Grammatik geschuldet.

Da im Beispielsatz „*Falls Bedingung X eintritt, DARF NICHT Aktion Y ausgeführt werden*.“ die Phrase „DARF NICHT“ semantisch irreführend ist (wenn nicht Aktion Y, welche denn dann?), wird in diesem Dokument stattdessen „*Falls Bedingung X eintritt, DARF Aktion Y NICHT ausgeführt werden*.“ verwendet.

Da im Beispielsatz „*Eine leere Liste DARF NICHT ein Element besitzen*.“ die Phrase „DARF NICHT“ semantisch irreführend wäre (wenn nicht ein, dann vielleicht zwei?), wird in diesem Dokument stattdessen „*Eine leere Liste DARF KEIN Element besitzen*.“ verwendet.

In diesem Dokument werden Aussagen mit dem Schlüsselwort KANN generell sowohl positiv als auch negativ formuliert. (N22) ist im Zusammenhang mit (N21)a dafür ein gutes Beispiel. In (N21)a wird eine normative Forderung erhoben, die offen lässt, ob zusätzliche Werte fakultativ verboten sind oder nicht. Diese Lücke wird durch (N22) geschlossen.

2.6.3 Normative und informative Abschnitte

Abschnitte mit normativen Inhalten tragen hinter der Kapitelüberschrift den Hinweis:

(normativ)

Generell gilt, dass lediglich die Gliederungen, welche durch eine Nummer (N 4711) gekennzeichnet sind, zulassungsrelevante Eigenschaften enthalten SOLLEN und somit im Rahmen der Zulassung getestet werden SOLLEN. Falls dies in einem speziellen Fall nicht so ist, handelt es sich höchstwahrscheinlich um einen editorischen Fehler.

Es sei angemerkt, dass die informativen Abschnitte zur Generation 2 lediglich einen Ausblick darstellen. Einerseits bedeutet dies, dass bis zur Generation 2 an diesen Abschnitten noch Änderungen möglich sind. Es besteht also keine Garantie, dass diese Abschnitte unverändert in Generation 2 gelten. Zum anderen bedeutet dies, dass derartige Abschnitte in keiner Art und Weise zulassungsrelevant sind.

3 Anforderungen und Annahmen (informativ)

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.

4 Systemüberblick (informativ)

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.

5 Lebenszyklus von Karte und Applikation (informativ)

In der Literatur finden sich verschiedene Beschreibungen für den Lebenszyklus. Dieses Kapitel stellt eine vereinfachte Sicht dar und legt dabei einen Gültigkeitsbereich für diese Spezifikation fest. Grob lässt sich der Lebenszyklus einer Karte in drei Phasen einteilen:

1. Vorbereitungsphase: Diese Phase umfasst aus Sicht der Produktion alle Schritte, die erforderlich sind, um eine Karte für die Nutzungsphase vorzubereiten. Dazu zählen im Wesentlichen die Entwicklung des Betriebssystems, dessen Test, Abnahme und gegebenenfalls auch Evaluierung. Entsprechende Chips werden anschließend produziert, initialisiert und personalisiert. Die Chips werden in einen Kartenkörper implantiert und an einen Kartennutzer ausgeliefert. Die Reihenfolge der Produktionsschritte weicht unter bestimmten Umständen von der genannten Reihenfolge ab und ist hier lediglich beispielhaft skizziert.
   Diese Spezifikation gilt nicht für die Vorbereitungsphase der Karte.
   Die Vorbereitungsphase der Karte endet mit der Übergabe der Karte an einen Kartennutzer. Dann beginnt die Nutzungsphase der Karte.

2. Nutzungsphase: Diese Phase umfasst den elektrischen Gebrauch der Karte.
   Diese Spezifikation gilt für die Nutzungsphase der elektrischen Kartenschnittstelle.
   Die Nutzungsphase der Karte endet, wenn sämtliche Business Use Cases irreversibel gesperrt sind, mithin also auch, wenn die Karte physikalisch zerstört wird.

3. Terminierungsphase: Befindet sich die Karte in der Terminierungsphase, dann sind alle intendierten Nutzungen der Karte irreversibel gesperrt. Es lassen sich also weder Daten auslesen noch speichern. Es ist keine Benutzerverifikation und auch keine Komponentenauthentisierung mehr möglich. Dies ist erreichbar durch eine physikalische Zerstörung des Chips, oder etwa auch durch die Unterstützung des Kommandos TERMINATE CARD USAGE (siehe Kapitel 15.2.7). Da nach Ausführung eines solchen Kommandos herstellerspezifisch noch gewisse Kommandos möglich sind (SELECT, GET CHALLENGE, …) oder die Übertragungsschicht T=1 möglicherweise noch aktiv ist, ist es nicht möglich hier von einer Karte zu sprechen, die völlig inaktiv ist. Diese Spezifikation gilt nicht für die Terminierungsphase der Karte.

Analog zu den Phasen einer Karte ist es möglich auch für Applikationen oder deren Bestandteile (EF, Passwörter, Schlüssel, …) die Phasen Vorbereitung, Nutzung und Terminierung zu definieren. Die Aussagen zur physikalischen Zerstörung der Karte gehen dann über in ein Löschen der Applikation oder deren Bestandteile.

Diese Spezifikation gilt nicht für die Vorbereitungsphase von Applikationen oder deren Bestandteile. Sie beschreibt lediglich den Zustand des Objektsystems in der Nutzungsphase.

Die Nutzungsphase einer Applikation oder eines Applikationsbestandteils beginnt, sobald sich ein derartiges Objekt, wie in der Spezifikation der Anwendung definiert, verwenden lässt. Die Nutzungsphase einer Applikation oder eines Applikationsbestandteils endet, wenn das entsprechende Objekt gelöscht oder terminiert wird.

6 Datentypen und Datenkonvertierung

Dieses Dokument verwendet die folgenden Datentypen äquivalent zu [TR–03111] Kapitel 3:

1. Oktettstring (OS),
2. Bitstring (BS),
3. Integer (I),
4. Gruppenelement (Field Element FE) und
5. elliptischen Kurvenpunkt (ECP)

Definition: Das höchstwertige Bit (most significant bit, MSBit) eines Bitstrings ist das am weitesten links stehende.

Definition: Das niedrigwertige Bit (least significant bit, LSBit) eines Bitstrings ist das am weitesten rechts stehende.

Definition: Das höchstwertige Oktett (most significant byte, MSByte) eines Oktettstrings ist das am weitesten links stehende.

Definition: Das niedrigwertige Oktett (least significant byte, LSByte) eines Oktettstrings ist das am weitesten rechts stehende.

Dieses Dokument verwendet die folgenden Konvertierungsfunktionen äquivalent zum Dokument [TR–03111] Kapitel 3.2:

1. Bitstring nach Oktettstring BS2OS
2. Oktettstring nach Bitstring OS2BS
3. Gruppenelement nach Oktettstring FE2OS
4. Oktettstring nach Gruppenelement OS2FE

6.1 BitLength Anzahl Bit in einem Bitstring

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input:	in	Bitstring mit beliebigem Inhalt und beliebiger Länge
Output:	out	Integer, Anzahl der Bits aus denen *in* besteht.
Errors:	–	keine
Notation:		*out* = BitLength(*in*)

6.2 OctetLenght Anzahl Oktett in einem Oktettstring

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input:	in	Oktettstring mit beliebigem Inhalt und beliebiger Länge, oder nicht negative ganze Zahl
Output:	out	Integer, Anzahl der Oktette aus denen *in* besteht, oder Anzahl Oktett, die mindestens nötig sind um eine nicht negative ganze Zahl zu kodieren.
Errors:	–	keine
Notation:		*out* = OctetLength(*in*)

Hinweis (2): Beispiele:
OctetLength(´´) = 0
OctetLength(´1234´) = 2
OctetLength(0) = 1, weil die Zahl null in einem Oktett zu kodieren ist, 0 = ´00´.
OctetLength(127) = 1, weil 127 = ´7F`
OctetLength(255) = 1, weil 255 = ´FF´
OctetLength(256) = 2, weil 256 = ´0100´.

6.3 I2OS Integer nach Oktettstring

Dieser Abschnitt beschreibt die Konvertierung einer nicht negativen ganzen Zahl in einen Oktettstring. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird diese Konvertierung als Funktion wie folgt verwendet:

Input:	x	Integer, nicht negative ganze Zahl
	n	Integer, Anzahl der Oktette in *out*, *n* darf null sein, dann ist *out* leer
Output:	out	Oktettstring der Länge *n* Oktett
Errors:	–	keine, ACHTUNG: Im Unterschied zu [TR–03111] wird hier keine Fehlermeldung erzeugt, falls *x* größer gleich 256^n ist.
Notation:		*out* = I2OS(*x*, *n*)

Hinweis (3): In gewissen Grenzen ist dies die Umkehrfunktion zu (N2).

(N1) Das Prinzip ist, die nicht negative ganze Zahl *x* als Ziffernfolge zur Basis 256 zu notieren und dann nur die niedrigwertigsten Ziffern für die Ausgabe zu verwenden:

a. Schritt 1: $x = 256^0 x_0 + 256^1 x_1 + 256^2 x_2 + \ldots + 256^i x_i + \ldots$

b. Schritt 2: $M_i = x_i$.
Anmerkung: Jede Ziffer wird vorzeichenlos in einem Oktett kodiert.

c. Schritt 3: $out = M_{n-1} \;||\; M_{n-2} \;||\; \dots \;||\; M_2 \;||\; M_1 \;||\; M_0$.

Hinweis (4): Beispiele: I2OS(30010, 1) = ´3A´, I2OS(30010, 2) = ´753A´, I2OS(30010, 3) = ´00753A´.

6.4 OS2I Oktettstring nach Integer

Dieser Abschnitt beschreibt die Konvertierung eines Oktetstrings in eine nicht negative ganze Zahl. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird diese Konvertierung als Funktion wie folgt verwendet:

Input:	in	Oktettstring beliebiger Länge und beliebigen Inhalts
Output:	out	Integer, nicht negative ganze Zahl
Errors:	–	keine
Notation:		*out* = OS2I(*x*)

Hinweis (5): In gewissen Grenzen ist dies die Umkehrfunktion zu (N1).

(N2) Das Prinzip ist, jedes Oktett als Ziffer zur Basis 256 einer nicht negativen ganzen Zahl im „big endian“ Format aufzufassen.

a. Schritt 1: n = OctetLength(*in*)

b. Falls *n*

i. gleich null ist, dann ist *out* = 0.

ii. ungleich null ist, dann wähle *out* so, dass gilt I2OS(*out*, *n*) = *in*.

6.5 OS2P Oktettstring nach Punkt

Dieser Abschnitt beschreibt die Konvertierung eines Oktettstrings in einen Punkt auf einer elliptischen Kurve. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird diese Konvertierung als Funktion wie folgt verwendet:

Input:	PO	Oktettstring, kodiert einen Punkt auf einer elliptischen Kurve
	dP	Domainparameter gemäß (N86)
Output:	P	Punkt auf einer elliptischen Kurve mit den Koordinaten $P = (x, y)$
Errors:	ERROR	falls *PO* nicht im Format „uncompressed encoding“ vorliegt
Notation:		*P* = OS2P(*PO*, *dP*)

(N3) Die Dekodierung von *PO* erfolgt gemäß [TR–03111] Kapitel 3.1.1:

a. Schritt 1: Falls OctetLength(*PO*) ungleich (2 *dP.L* + 1) ist, dann gebe den Fehler „*ERROR*“ zurück und beende diesen Algorithmus.

b. Schritt 2: Teile *PO* auf gemäß:
   $PO = PC \parallel X \parallel Y$,
   $1 = \text{OctetLenght}(PC)$,
   $L = \text{OctetLength}(X) = \text{OctetLength}(Y)$.

c. Schritt 3: Falls *PC* ungleich ´04´ ist, dann gebe den Fehler „*ERROR*“ zurück und beende diesen Algorithmus.

d. Schritt 4: $PE_A = (x, y) = (\text{OS2I}(X) \bmod dP.p, \text{OS2I}(Y) \bmod dP.p)$.

e. Schritt 5: Falls PE_A nicht auf der durch *dP* definierten Kurve liegt, dann gebe den Fehler „*ERROR*“ zurück und beende diesen Algorithmus.

6.6 P2OS Punkt nach Oktettstring

Dieser Abschnitt beschreibt die Konvertierung eines Punktes auf einer elliptischen Kurve in einen Oktettstring. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird diese Konvertierung als Funktion wie folgt verwendet:

Input:	P	Punkt auf einer elliptischen Kurve mit den Koordinaten $P = (x, y)$
	n	Integer, Anzahl Oktette pro Koordinate
Output:	PO	Oktettstring, kodiert einen Punkt auf einer elliptischen Kurve
Errors:	–	
Notation:		$PO = \text{P2OS}(P, n)$

(N4) Die Kodierung von *P* erfolgt gemäß [TR–03111] Kapitel 3.1.1:
$PO =$ ´04´ $\parallel \text{I2OS}(x, n) \parallel \text{I2OS}(y, n)$.

6.7 Extrahiere führende Elemente

6.7.1 Extrahiere führende Bits

Dieser Abschnitt beschreibt, wie aus einem Bitstring führende Bits extrahiert werden.

Input:	in	• entweder Bitstring der Länge *s* Bit • oder Oktettstring der Länge *s* Bit
	n	Integer, Anzahl der zu extrahierenden Bit
Output:	out	Bitstring der Länge *n* Bit
Errors:	–	keine
Notation:		$out = \text{Extract\_MSBit}(in, n)$

(N5) Es gilt die Vorbedingung *n* kleiner gleich *s*.

(N6) Der Bitstring *out* enthält die *n* MSBit von *in*.

6.7.2 Extrahiere führende Oktette

Dieser Abschnitt beschreibt, wie aus einem Oktettstring führende Oktette extrahiert werden.

Input:	in	• entweder Bitstring der Länge *s* Oktett • oder Oktettstring der Länge *s* Oktett
	n	Integer, Anzahl der zu extrahierenden Elemente
Output:	out	Oktettstring der Länge *n* Oktett
Errors:	–	keine
Notation:		*out* = Extract_MSByte(*in*, *n*)

(N7) Es gilt die Vorbedingung *n* kleiner gleich *s*.

(N8) Der Oktettstring *out* enthält die *n* MSByte von *in*.

6.8 PaddingIso

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Es wird gemäß [7816–4] Kapitel 6.2.3.1 Abschnitt „Sequential stage" Spiegelstrich 2 gepadded. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input:	*in*	Oktettstring mit beliebigem Inhalt und beliebiger Länge
	n	Integer, gibt die Blocklänge von *out* in Oktett an
Output:	*out*	Oktettstring mit einer Anzahl Oktette, die Vielfaches von *n* ist
Errors:	–	keine
Notation:		*out* = PaddingIso(*in*, *n*)

Hinweis (6): Dies ist die Umkehrfunktion zu (N10).

(N9) Führe folgende Aktion durch:

a. Schritt 1: *out* **ß** *in* || ´80´.
b. Schritt 2: Falls 0 = OctetLength(*out*) mod *n*, dann gebe *out* zurück und beende den Algorithmus, sonst fahre mit Schritt 3 fort.
c. Schritt 3: *out* **ß** *out* || ´00´.
d. Schritt 4: Fahre mit Schritt 2 fort.

6.9 TruncateIso

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input:	*in*	Oktettstring mit beliebigem Inhalt und einer Anzahl Oktette, die Vielfaches von *n* ist
	n	Integer, gibt die Blocklänge in Oktett an, auf die gepadded wurde.
Output:	*out*	Oktettstring mit beliebigem Inhalt und beliebiger Länge
Errors:	*paddingError*	• Die Länge von *in* ist kein Vielfaches von *n*. • Es sind zu viele Paddingbits vorhanden.

Notation: *out* = TruncateIso(*in*, *n*)

Hinweis (7): Dies ist die Umkehrfunktion (N9).

(N10) Führe folgende Aktion durch:

a. Schritt 1: *len* ß OctetLength(*in*)
Falls *len* kein Vielfaches von *n* ist, dann breche diesen Algorithmus mit der Fehlermeldung *paddingError* ab.

b. Schritt 2: Falls OctetLength(*in*) gleich null ist, dann breche diesen Algorithmus mit der Fehlermeldung *paddingError* ab.

c. Schritt 3: Falls LSByte von *in* den Wert ´00´ hat, dann setze
in ß Extract_MSByte(*in*, OctetLength(*in*) – 1)
und fahre mit Schritt 2 fort.

d. Schritt 3: Falls LSByte von *in* nicht den Wert ´80´ hat, dann breche diesen Algorithmus mit der Fehlermeldung *paddingError* ab.

e. Schritt 4: *out* ß Extract_MSByte(*in*, OctetLength(*in*) – 1)

f. Schritt 5: Falls (*len* – OctetLength(*out*)) größer als *n* ist, dann breche diesen Algorithmus mit der Fehlermeldung *paddingError* ab.

6.10 Mask Generation Function

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input:	Z	Bitstring mit beliebigem Inhalt und beliebiger Länge
	L_N	Integer, gibt die Bitlänge von *N* an
	i	Integer, Startwert der Iteration

Output: N Bitstring, Ergebnis der Mask Generation Function

Errors: error Gemäß [9796–2] Annex B.3.2 Punkt 1 ist ein Fehler zu werfen, falls Z zu lang, oder L_N zu groß ist.

Hinweis (8): Diese Fehler sind für Chipkarten derzeit nicht praxisrelevant, da sie erst dann auftreten, wenn Z oder N oberhalb mehrerer Gigabyte liegen.

Notation: N = MGF(Z, L_N , i)

Hinweis (9): Wenn diese Funktion mit i = 0 aufgerufen wird, dann ist das Ergebnis konform zu
– [9796–2] Annex B,
– [PKCS#1] Annex B.2.1.

Hinweis (10): Wenn diese Funktion mit i = 1 aufgerufen wird, dann ist das Ergebnis konform zu
– [ANSI X9.63] Kapitel 5.6.3, falls dort der optionale Parameter SharedInfo leer ist.
– [TR–03111] Kapitel 4.4.2, allerdings wird dort das Schlüsselmaterial anders aus N extrahiert.

(N11) Führe folgende Aktionen durch:

a. Schritt 1: Setze *n* **ß** ´´ (leerer Oktettstring).
b. Schritt 2: Setze *n* **ß** *n* || SHA_256(BS2OS(*Z*) || I2OS(*i*, 4)).
c. Schritt 3: Falls die Hashoperation mit einem Fehler terminierte, dann breche diesen Algorithmus mit der Fehlermeldung „*error*“ ab.
d. Schritt 4: Setze *i* **ß** *i* + 1.
e. Schritt 5: Falls i größer gleich 2^{32} = ´1 0000 0000´ ist, dann breche diesen Algorithmus mit der Fehlermeldung „*error*“ ab.
f. Schritt 6: Falls das achtfache von OctetLength(*n*) kleiner als L_N ist, dann setze diesen Algorithmus mit Schritt 2 fort.
g. Schritt 7: Setze *N* **ß** Extract_MSBit(*n*, L_N).

6.11 Zufälliger Oktettstring

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird die hier beschriebene Funktion wie folgt verwendet:

Input: n positive ganze Zahl, welche die Anzahl Oktette in *out* angibt

Output: out zufälliger Oktettstring der Länge *n* Oktett

Errors: – keine

Notation: *out* = RAND(*n*)

(N12) Führe folgende Aktionen durch: Erzeuge einen Oktettstring *out* der Länge *n*, wobei jedes Bit von *out* zufällig gewürfelt wird. Dieser Zufallszahlengenerator MUSS den Anforderungen aus [gemSpec_Krypt]#5.2 genügen.

7 Kryptographische Algorithmen (normativ)

7.1 Hashalgorithmen

Ein Hashalgorithmus errechnet zu einer beliebigen Folge von Bits von (beinahe unbegrenzter Länge) einen Bitstring fester Länge. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird ein Hashalgorithmus als Funktion wie in den folgenden Unterkapiteln gezeigt verwendet.

7.1.1 SHA–256, G1 und G2 (normativ)

Input:	M	beliebiger Oktettstring, der zu hashen ist
Output:	H	Oktettstring, Hashwert der Länge 32 Oktette = 256 Bit
Errors:	„M too long“	Zu hashende Nachricht *M* enthält zu viele Bits. Dies ist der Fall wenn die *M* länger als 2^{64} Bit = 2^{61} Byte = 2 EiB *Hinweis (11): Dieser Fehler ist für Chipkarten nicht praxisrelevant. Um eine Nachricht mit einer Länge von 2^{64} Bit innerhalb von 10 Jahren zu hashen wäre der Hashalgorithmus mit einer Geschwindigkeit von 6,8 GiB pro Sekunde auszuführen.*
Notation:		*H* = SHA_256(*M*)

(N13) Das COS MUSS *H* gemäß [SHA–256] aus *M* berechnen.

7.2 Schlüsselvereinbarung

In diesem Abschnitt wird eine Funktion beschrieben, die aus einem Eingabewert Schlüsselmaterial für symmetrische Algorithmen berechnet. Derartiges Schlüsselmaterial wird vorwiegend im Rahmen von sicherer Transportverschlüsselung eingesetzt. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine Schlüsselvereinbarung als Funktion wie in den folgenden Unterkapiteln gezeigt verwendet.

7.2.1 Verhalten G1 (normativ)

Input:	*KD*	Oktettstring, Key Derivation Data, Ausgangsmaterial zur Schlüsselvereinbarung, prinzipiell handelt es sich um einen Oktettstring beliebiger Länge und beliebigen Inhalts.

Output: K_{enc} Oktettstring der Länge 24 Oktette, der als 3TDES Schlüssel für Ver- und Entschlüsselung verwendet wird

T_1 nicht negative ganze Zahl, wird im Zusammenhang mit K_{enc} als Send Sequence Counter verwendet, siehe Funktionen in Kapitel 7.7.

K_{mac} Oktettstring der Länge 24 Oktette, der als 3TDES Schlüssel für MAC Generierung und MAC Prüfung verwendet wird

T_2 nicht negative ganze Zahl, wird im Zusammenhang mit K_{mac} als Send Sequence Counter verwendet, siehe (N328) und (N341). Kom 3002

Errors: – keine

Notation: (K_{enc}, T_1, K_{mac}, T_2) = KeyDerivation_3TDES(KD)

(N14) Das COS MUSS (K_{enc}, T_1, K_{mac}, T_2) gemäß [ANSI X9.63] Kapitel 5.6.3 wie folgt aus KD berechnen.

a. Schritt 1: km = BS2OS(MGF(OS2BS(KD), 512, 1)).

b. Schritt 2: Teile km so auf K_{enc}, t_1, K_{mac} und t_2 auf, dass gilt:

  i. OctetLength(K_{enc}) = OctetLength(K_{mac}) = 24

  ii. OctetLength(t_1) = OctetLength(t_2) = 8

  iii. $km = K_{enc} \;||\; t_1 \;||\; K_{mac} \;||\; t_2$

c. Schritt 3: T_1 = OS2I(t_1) und T_2 = OS2I(t_2)

7.2.2 Verhalten G2 (informativ)

Input: KD Oktettstring, Key Derivation Data, Ausgangsmaterial zur Schlüsselvereinbarung, prinzipiell handelt es sich um einen Oktettstring beliebiger Länge und beliebigen Inhalts.

Output: K_{enc} Oktettstring der Länge 16 Oktette, der als AES–128 Schlüssel für Ver- und Entschlüsselung verwendet wird

T_1 Oktettstring der Länge 16 Oktette, wird im Zusammenhang mit K_{enc} als Startwert für den Counter verwendet, siehe Funktionen in Kapitel 7.7.

K_{mac} Oktettstring der Länge 16 Oktette, der als AES–128 Schlüssel für MAC Generierung und MAC Prüfung verwendet wird

T_2 nicht-negative ganze Zahl, wird im Zusammenhang mit K_{mac} als Send Sequence Counter verwendet, siehe (N328) und (N341). Kom rektu

Errors: – keine

Notation: $(K_{enc}, T_1, K_{mac}, T_2)$ = KeyDerivation_AES128(KD)

(N15) Das COS MUSS (K_{enc}, T_1, K_{mac}, T_2) analog zu [TR–03111] Kapitel 4.4.2 wie folgt aus KD berechnen.

a. Schritt 1: km = BS2OS(MGF(OS2BS(KD), 512, 1)).

b. Teile km so auf K_{enc}, T_1, K_{mac} und SSC auf, dass gilt:

   i. OctetLength(K_{enc}) = OctetLength(K_{mac}) = 16

   ii. OctetLength(T_1) = OctetLength(T_2) = 16

   iii. $km = K_{enc} \;||\; T_1 \;||\; K_{mac} \;||\; T_2$

7.3 Symmetrischer Basisalgorithmus für Vertraulichkeit

Ein Verschlüsselungsalgorithmus berechnet zu einem beliebigen Oktettstring (plaintext) mit Hilfe eines geheimen Schlüssels ein Chiffrat (ciphertext). Ein entsprechender Entschlüsselungsalgorithmus berechnet zu einem Chiffrat (ciphertext) mit Hilfe desselben symmetrischen Schlüssels den ursprünglichen Oktettstring (plaintext). In diesem Kapitel wird lediglich die Verschlüsselung eines Blocks mittels eines Blockverschlüsselungsalgorithmus behandelt. Die Behandlung von Inputdaten mit beliebiger Länge wird in Kapitel 7.7 behandelt.

7.3.1 Symmetrische Verschlüsselung eines Datenblocks, G1 (normativ)

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine Verschlüsselung als Funktion wie folgt verwendet:

7.3.1.1 Verschlüsselung mittels DES

Input:	P_j	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit
	K	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit, der als Schlüssel verwendet wird
Output:	C_j	Oktettstring, verschlüsselte Daten der Länge 8 Oktett
Errors:	–	keine
Notation:		C_j = DES_ENC(K, P_j)

(N16) Das COS MUSS C_j mittels K gemäß [ANSI X3.92] aus P_j berechnen.

7.3.1.2 Verschlüsselung mittels 3TDES

Input:	P_j	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit
	K	beliebiger Oktettstring der Länge 24 Oktett = 192 Bit, der als Schlüssel verwendet wird. *K* setzt sich zusammen aus den drei Teilschlüsseln *Ka*, *Kb*, *Kc* und es gilt. $K = Ka \;\|\; Kb \;\|\; Kc$.
Output:	C_j	Oktettstring, verschlüsselte Daten der Länge 8 Oktett
Errors:	–	keine
Notation:		C_j = 3TDES_ENC(K, P_j)

(N17) Das COS MUSS C_j mittels *K* wie folgt aus P_j berechnen:
C_j = DES_ENC(*Kc*, DES_DEC(*Kb*, DES_ENC(*Ka*, P_j))).

7.3.2 Symmetrische Entschlüsselung eines Datenblocks, G1 (normativ)

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine Entschlüsselung als Funktion wie folgt verwendet:

7.3.2.1 Entschlüsselung mittels DES

Input:	C_j	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit
	K	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit, der als Schlüssel verwendet wird.
Output:	P_j	Oktettstring, entschlüsselte Daten der Länge 8 Oktett
Errors:	–	keine
Notation:		P_j = DES_DEC(K, C_j)

(N18) Das COS MUSS P_j mittels *K* gemäß [ANSI X3.92] aus C_j berechnen.

7.3.2.2 Entschlüsselung mittels 3TDES

Input:	C_j	beliebiger Oktettstring der Länge 8 Oktett = 64 Bit
	K	beliebiger Oktettstring der Länge 24 Oktett = 192 Bit, der als Schlüssel verwendet wird. *K* setzt sich zusammen aus den drei Teilschlüsseln *Ka*, *Kb*, *Kc* und es gilt: $K = Ka \;\|\; Kb \;\|\; Kc$.

Output: P_j Oktettstring, entschlüsselte Daten der Länge 8 Oktett

Errors: – keine

Notation: P_j = 3TDES_DEC(K, C_j)

(N19) Das COS MUSS P_j mittels K wie folgt aus C_j berechnen:
P_i = DES_DEC(Ka, DES_ENC(Kb, DES_DEC(Kc, C_i))).

7.3.3 Symmetrische Verschlüsselung eines Datenblocks, G2 (informativ)

Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine Verschlüsselung als Funktion wie folgt verwendet:

Input: P_j beliebiger Oktettstring der Länge 16 Oktett = 128 Bit

K beliebiger Oktettstring der Länge 16 Oktett = 128 Bit, der als Schlüssel verwendet wird

Output: C_j Oktettstring, verschlüsselte Daten der Länge 16 Oktet

Errors: – keine

Notation: C_j = AES_ENC(K, P_j)

(N20) Das COS MUSS C_j mittels K gemäß [AES–128] Abbildung 5 aus P_j berechnen.

7.3.4 Symmetrische Entschlüsselung eines Datenblocks, G2 (informativ)

Dieser Abschnitt enthält absichtlich keine normativen Forderungen, da eine entsprechende Funktionalität wegen des verwendeten CTR–Modes (siehe [AES–CTR]) nicht erforderlich ist.

7.4 Asymmetrischer Basisalgorithmus, G1 (normativ)

Als asymmetrischer Basisalgorithmus wird RSA verwendet. Bezüglich des mathematischen Hintergrundes wird an dieser Stelle lediglich auf [PKCS#1] verwiesen.

(N21) Das COS MUSS RSA Schlüssel mit

a. einer Moduluslänge n aus der Menge {2048} Bit unterstützen.

b. öffentlichen Exponenten *e* aus folgendem Intervall unterstützen:
$[2^{16}+1, 2^{32}–1]$ = [65.537, 4.294.967.295] = [´0001 0001´, ´FFFF FFFF´].

(N22) Das COS KANN weitere Moduluslängen unterstützen.
Das COS KANN weitere Moduluslängen ablehnen.

(N23) Das COS KANN öffentliche Exponenten aus weiteren Intervallen unterstützen.
Das COS KANN öffentliche Exponenten aus weiteren Intervallen ablehnen.

(N24) Private RSA Schlüssel, welche die ε_2 Schranke aus [TR–03116] Kapitel 3.5.2 verletzen, für die mithin NICHT gilt $|log_2(p) – log_2 (q)| < 30$,

a. KÖNNEN vom COS akzeptiert werden.

b. KÖNNEN vom COS abgelehnt werden.

Konform zu [PKCS#1] Kapitel 2 werden die RSA–Schlüsselparameter in diesem Dokument wie folgt dargestellt:

Tabelle 4: Liste der Schlüsselparameter eines RSA Schlüssels

Parameter	Bedeutung
n	RSA Modulus
e	RSA öffentlicher Exponent
d	RSA privater Exponent

Hinweis (12): In diesem Dokument wird der Einfachheit halber nicht mit „chinese remainder theorem“ Parametern gearbeitet. Wegen der größeren Performance wird aber empfohlen innerhalb des COS für Operationen mit dem privaten Schlüssel „chinese remainder theorem“ Parameter zu verwenden.

7.5 Asymmetrischer Basisalgorithmus, G2 (informativ)

Als asymmetrischer Basisalgorithmus werden elliptische Kurven verwendet. Bezüglich des mathematischen Hintergrundes wird an dieser Stelle lediglich auf [TR–03111] verwiesen.

(N25) Das COS MUSS elliptische Kurven unterstützen, deren Domainparamter den Anforderungen aus [BrainPool] Kapitel 3 entsprechen und eine Länge von 256 Bit aufweisen.

(N26) Das COS KANN weitere elliptische Kurven unterstützen.
Das COS KANN weitere elliptische Kurven ablehnen.

Konform zu [TR–03111] Tabelle 2.1 werden die Domainparameter in diesem Dokument wie folgt dargestellt:

Tabelle 5: Liste der Domainparameter einer elliptischen Kurve

Parameter	Bedeutung
p	Primzahl, welche die zugrunde liegende Gruppe F_p beschreibt
a	erster Koeffizient der Weierstraß'schen Gleichung
b	zweiter Koeffizient der Weierstraß'schen Gleichung
G	ein Basispunkt auf der Kurve $E(F_p)$
n	Ordnung des Basispunktes G in $E(F_p)$
h	Cofaktor von G in $E(F_p)$ Wegen (N25) gilt h = 1 für alle Kurven, die ein COS unterstützen MUSS.

7.6 Datenauthentisierung

Im Rahmen einer Datenauthentisierung werden beliebigen Daten Informationen derart hinzugefügt, dass die Integrität und Authentizität der Daten überprüfbar ist.

7.6.1 MAC Generierung

Die MAC Generierung ist eine Datenauthentisierung, welche auf einem symmetrischen Basisalgorithmus basiert. Dabei wird zu einem Oktettstring beliebigen Inhalts und Länge ein MAC berechnet, dessen Länge lediglich vom Algorithmus abhängt, nicht aber vom Oktettstring. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine MAC Generierung als Funktion wie folgt verwendet:

7.6.1.1 Generierung Retail–MAC, G1 (normativ)

Input: *M* beliebiger Oktettstring beliebiger Länge für den ein MAC berechnet wird

K beliebiger Oktettstring, der als Schlüssel verwendet wird. Die Länge von *K* beträgt 24 Oktett. *K* setzt sich zusammen aus den drei Teilschlüsseln *Ka*, *Kb*, *Kc* und es gilt: $K = Ka \ || \ Kb \ || \ Kc$.

Output: *T* Oktettstring, der zur Prüfung der Integrität und Authentizität von *M* verwendbar ist

Errors: – keine

Notation: *T* = CALCULATE_Retail_MAC(*K*, *M*)

(N27) Das COS MUSS *T* mittels *K* aus *M* berechnen. Dabei werden folgende Schritte durchgeführt:

a. Schritt 1: Berechne X = PaddingIso(M, 8)
b. Schritt 2: Teile X auf in Blöcke mit jeweils 64 Bit $X = X_1 \ || \ X_2 \ || \ \ldots \ || \ X_n$.
c. Schritt 3: Setze Y_0 = ´0000 0000 0000 0000´.
d. Schritt 4: Berechne Y_i = DES_ENC(Ka, Y_{i-1} XOR X_i) für $i = 1, \ldots, n - 1$.

e. Schritt 5: Berechne T = 3TDES_ENC(K, Y_{n-1} XOR X_n)

Hinweis (13): Im Rahmen von Secure Messaging wird ein von „null“ verschiedener Initialisierungsvektor Y_0 bei der Konstruktion der Nachricht M berücksichtigt, siehe dazu (N328) und (N341).

Kom 3002

7.6.1.2 Generierung CMAC, G2 (informativ)

Input: M beliebiger Oktettstring für den ein MAC berechnet wird

K beliebiger Oktettstring, der als Schlüssel verwendet wird. Die Länge von K beträgt 16 Oktett, falls K ein AES–128 Schlüssel ist.

Output: T Oktettstring, der zur Prüfung der Integrität und Authentizität von M verwendbar ist

Errors: – keine

Notation: T = CALCULATE_CMAC(K, M)

(N28) Das COS MUSS T mittels K gemäß [CMAC] Kapitel 6.2 aus M berechnen. Dabei werden folgende Schritte durchgeführt:

a. Schritt 0: $Mlen$ = BitLength(M).
CIPH = Block Cipher Algorithmus (siehe (N20))
b = Blocklänge von CIPH = 128 Bit

b. Schritt 1: Generiere Unterschlüssel $K1$ und $K2$ gemäß [CMAC] Kapitel 6.1.

c. Schritt 2: Falls $Mlen$ gleich null ist, dann setze n = 1,
sonst r = $Mlen$ mod b und
falls r = 0, dann setze n = $Mlen$ / b
sonst setze n = ($Mlen$ – r) / b + 1.

d. Schritt 3: Teile M wie folgt in Blöcke auf:
$M = M_1 \,||\, M_2 \,||\, \dots \,||\, M_{n-1} \,||\, M^*_n$.
Die Blöcke M_1 bis M_{n-1} besitzen die Länge b. Der Block M^*_n besitzt eine Länge kleiner gleich b.

e. Schritt 4: Falls r = 0 ist, dann setze $M_n = K1$ XOR M^*_n,
sonst setze $M_n = K2$ XOR PaddingIso(M^*_n, $b / 8$).

f. Schritt 5: C_0 = I2OS(0, b / 8)

g. Schritt 6: C_i = AES_ENC(K, C_{i-1} XOR M_i).

h. Schritt 7: T = C_n.

7.6.2 MAC Prüfung

Die MAC Prüfung prüft die Konsistenz zwischen Daten und den hinzugefügten Informationen der Datenauthentisierung. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine MAC Prüfung als Funktion wie folgt verwendet:

7.6.2.1 Prüfung Retail–MAC, G1 (normativ)

Input:	*M*	beliebiger Oktettstring der durch einen MAC geschützt ist
	T´	Oktettstring, der *M* zwecks Datenauthentisierung hinzugefügt wurde
	K	beliebiger Oktettstring, der als Schlüssel verwendet wird. Die Länge von *K* beträgt 24 Oktett. *K* setzt sich zusammen aus den drei Teilschlüsseln *Ka*, *Kb*, *Kc* und es gilt: *K* = *Ka* || *Kb* || *Kc*.
Output:	*out*	Ergebnis der MAC Prüfung, entweder VALID oder INVALID
Errors:	–	keine
Notation:		*out* = VERIFY_Retail_MAC(*K*, *T´*, *M*)

(N29) Das COS MUSS die Integrität der Daten *M* mittels *T´* und *K* prüfen. Dabei werden folgende Schritte durchgeführt:

a. Schritt 1: *T* = CALCULATE_Retail_MAC(*K*, *M*).

b. Schritt 2: Falls *T* identisch zu *T´* ist, dann gebe VALID zurück
sonst gebe INVALID zurück.

7.6.2.2 Prüfung CMAC, G2 (informativ)

Input:	*M*	beliebiger Oktettstring der durch einen MAC geschützt ist
	T´	Oktettstring, der *M* zwecks Datenauthentisierung hinzugefügt wurde
	K	beliebiger Oktettstring, der als Schlüssel verwendet wird. Die Länge von *K* beträgt 16 Oktett, falls *K* ein AES–128 Schlüssel ist.
Output:	*out*	Ergebnis der MAC Prüfung, entweder VALID oder INVALID
Errors:	–	keine
Notation:		*out* = VERIFY_CMAC(*K*, *T´*, *M*)

(N30) Das COS MUSS die Integrität der Daten *M* mittels *T´* und *K* gemäß [CMAC] Kapitel 6.3 prüfen. Dabei werden folgende Schritte durchgeführt:

a. Schritt 1: *T* = CALCULATE_CMAC(*K*, *M*).

b. Schritt 2: Falls *T* identisch zu *T´* ist, dann gebe VALID zurück
sonst gebe INVALID zurück.

7.6.3 Signatur Berechnung

Unter der Berechnung einer Signatur wird in diesem Dokument lediglich die mit dem privaten Schlüssel durchgeführte Operation verstanden.

7.6.3.1 Signatur Berechnung mittels RSA, G1 (normativ)

7.6.3.1.1 RSA, ISO9796–2, DS1, SIGN

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos INTERNAL AUTHENTICATE sichtbar (siehe (N869)e und (N869)f). Sie wird wie folgt verwendet:

Input:	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
	M	beliebiger Oktettstring der die zu signierende Nachricht repräsentiert
Output:	sig	Oktettstring, welcher die Signatur repräsentiert
	M_2	Oktettstring, welcher den „non recoverable part“ der Nachricht *M* repräsentiert
Errors:	–	–
Notation:		(*sig*, M_2) = RSA_ISO9796_2_DS1_SIGN(*PrK*, *M*)

(N31) Gemäß [9796–2] Kapitel 7, 8 MUSS das COS folgende Aktionen durchführen, wobei folgende Definitionen gelten: n = *PrK.n*, d = *PrK.d*

a. Schritt 1: Message allocation: Aufspalten der Nachricht *M* in M_1 und M_2 gemäß [9796–2] Kapitel 7.1.2.

b. Schritt 2: Message representative production: Berechnen des Repräsentanten *F* der Nachricht *M* gemäß [9796–2] Kapitel 8.2, wobei $t = 1$ gesetzt werden MUSS und als Hashfunktion SHA–256 (siehe (N13)) verwendet werden MUSS.

c. Schritt 3: Signature production: Gemäß [9796–2] Kapitel 7.1.4 und [9796–2] Annex A.4 werden folgende Operationen ausgeführt:

  i. Schritt 3.1: J = OS2I(F)

  ii. Schritt 3.2: $a = J^d \bmod n$

  iii. Schritt 3.3: $b = n - a$

  iv. Schritt 3.4: $c = \min\{ a, b \}$

  v. Schritt 3.5: sig = I2OS(c, OctetLength(n))

7.6.3.1.2 RSA, SSA, PKCS1–V1_5

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Compute Digital Signature sichtbar (siehe (N886)b). Sie wird wie folgt verwendet:

Input:	digestInfo	beliebiger Oktettstring der als Digest Info verwendet wird, siehe [PKCS#1] Annex A.2.4
	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
Output:	S	Oktettstring, welcher die Signatur repräsentiert
Errors:	–	DigestInfoTooLong
Notation:		*S* = RSASSA_PCKS1_V1_5_SIGN(*PrK, digestInfo*)

(N32) Gemäß [PKCS#1] Kapitel 8.2.1, 9.2 MUSS das COS folgende Aktion durchführen, wobei folgende Definitionen gelten: *n* = *PrK.n*, *d* = *PrK.d*.

a. Schritt 0: Falls OctetLength(*digestInfo*) > 0,4 OctetLength(*n*) ist, dann breche diesen Algorithmus mit der Fehlermeldung DigestInfoTooLong ab.
b. Schritt 1: Setze *EM* ß ´00´ || *digestInfo*.
c. Schritt 2: Setze *EM* ß ´FF´ || *EM*.
d. Schritt 3: Falls OctetLength(*EM*) kleiner als OctetLength(*n*) – 2 ist, dann fahre mit Schritt 2 fort.
e. Schritt 4: Setze *EM* ß ´01´ || *EM*.
f. Schritt 5: Setze *m* ß OS2I(*EM*).
g. Schritt 6: Setze *s* ß m^d mod *n*.
h. Schritt 7: Setze *S* ß I2OS(*s*, OctetLength(*n*)).

7.6.3.1.3 RSA, SSA, PSS

Diese Funktionalität ist an der physikalischen Schnittstelle nicht sichtbar. Sie wird im Rahmen interner Funktionen verwendet.

Input:	M_1	beliebiger Oktettstring der den „recoverable part“ der zu signierenden Nachricht *M* repräsentiert
	$h(M_2)$	beliebiger Oktettstring der den Hashwert über den „non recoverable part“ der zu signierenden Nachricht *M* berechnet wird
	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
Output:	sig	Oktettstring, welcher die Signatur repräsentiert
Errors:	–	–
Notation:		*sig* = RSA_PSS_SIGN(*PrK*, M_1, $h(M_2)$)

(N33) Gemäß [9796–2] Kapitel 7, 9 MUSS das COS folgende Aktionen durchführen, wobei folgende Definitionen gelten: *n* = *PrK.n*, *d* = *PrK.d*

a. Schritt 1: Message representative production: Berechnen des Repräsentanten F der Nachricht M gemäß [9796–2] Kapitel 9.2, wobei $t = 1$ und $L_s = L_h$ gesetzt werden MUSS und als Hashfunktion SHA–256 verwendet werden MUSS. Im einzelnen:
   i. Setze C = I2OS(BitLength(OS2BS(M_1)), 8)
   ii. Setze S = RAND(32)
   iii. Berechne H = SHA_256(C || M_1 || $h(M_2)$ || S)
   iv. Berechne F gemäß [9796–2] Kapitel 9.2.2.

b. Schritt 2: Signature production: Gemäß [9796–2] Kapitel 7.1.4 und [9796–2] Annex A.6 werden folgende Operationen ausgeführt:
   i. Schritt 2.1: J = OS2I(F)
   ii. Schritt 2.2: a = J^d mod n
   iii. Schritt 2.3: sig = I2OS(a, OctetLength(n))

7.6.3.1.4 RSA, ISO9796–2, DS2, SIGN

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Compute Digital Signature sichtbar (siehe (N886)a). Sie wird wie folgt verwendet:

Input:	M_1	beliebiger Oktettstring der den „recoverable part“ der zu signierenden Nachricht M repräsentiert
	h(M_2)	beliebiger Oktettstring der den Hashwert über den „non recoverable part“ der zu signierenden Nachricht M berechnet wird
	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
Output:	sig	Oktettstring, welcher die Signatur repräsentiert
Errors:	–	–
Notation:		sig = RSA_ISO9796_2_DS2_SIGN(PrK, M_1, $h(M_2)$)

(N34) Gemäß [9796–2] Kapitel 7, 9 MUSS das COS folgende Aktionen durchführen, wobei folgende Definition gilt: $n = PrK.n$

a. Schritt 1: a = OS2I(RSA_PSS_SIGN(PrK, M_1, $h(M_2)$)).
b. Schritt 2: $b = n – a$
c. Schritt 3: c = min{ a, b }
d. Schritt 4: sig = I2OS(c, OctetLength(n))

7.6.3.1.5 RSASSA–PSS–SIGN

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen der Kommandos INTERNAL AUTHENTICATE (siehe (N869)d) und PSO Compute Digital Signature sichtbar (siehe (N886)c). Sie wird wie folgt verwendet:

Input:	mHash	beliebiger Oktettstring der den Hashwert über die zu signierende Nachricht *M* repräsentiert
	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
Output:	S	Oktettstring, welcher die Signatur repräsentiert
Errors:	–	–
Notation:		*S* = RSASSA_PSS_SIGN(*PrK, mHash*)

(N35) Gemäß [PKCS#1] Kapitel 8.1.1, 9.1.1 MUSS das COS folgende Aktion durchführen:

a. Schritt 1: M_1 = ´´ (M_1 ist ein leerer Oktettstring)

b. Schritt 2: S = RSA_PSS_SIGN(*PrK*, M_1, *mHash*).

7.6.3.2 Signatur Berechnung mittels ELC, G2 (informativ)

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Compute Digital Signature sichtbar (siehe (N886)d). Sie wird wie folgt verwendet:

Input:	H	beliebiger Oktettstring der einen Hashwert repräsentiert
	PrK	ein privater ELC Schlüssel gemäß Kapitel 9.2.3
Output:	R	Oktettstring, erster Teil der ECDSA Signatur
	S	Oktettstring, zweiter Teil der ECDSA Signatur
Errors:	–	–
Notation:		(*R*, *S*) = ELC_SIG(*PrK, H*)

(N36) Gemäß [TR–03111] Kapitel 4.2.1 MUSS das COS folgende Aktionen durchführen, wobei folgende Definitionen gelten:

n = *PrK.domainParameter.n*, *G* = *PrK.domainParameter.G*,
d_A = *PrK.domainParameter.d*, *L* = *PrK.domainParameter.L*

a. Schritt 1: k ß RNG({1,2, …, n – 1}).

b. Schritt 2: Q ß [k] G mit $Q = (x_Q, y_Q)$.

c. Schritt 3: r ß OS2I(FE2OS(x_Q)) mod n.
Falls r gleich null ist, dann gehe zu Schritt 1.

d. Schritt 4: k_{inv} ß k^{-1} mod n.

e. Schritt 5: s ß k_{inv} ($r\, d_A$ + OS2I(H)) mod n.
Falls s gleich null ist, dann gehe zu Schritt 1.

f. Schritt 6: R ß I2OS(r, L).
S ß I2OS(s, L).

7.6.4 Signatur Prüfung

Unter der Prüfung einer Signatur wird in diesem Dokument lediglich die mit dem öffentlichen Schlüssel durchgeführte Operation verstanden. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine Signatur Prüfung als Funktion wie folgt verwendet:

7.6.4.1 RSA, ISO9796–2, DS1, VERIFY, G1 (normativ)

Input:	PuK	ein öffentlicher RSA Schlüssel gemäß Kapitel 9.2.4
	sig	beliebiger Oktettstring der eine Signatur repräsentiert
	M_2	beliebiger Oktettstring, der den „non recoverable part“ der Nachricht *M* repräsentiert. M_2 KANN leer sein.
Output:	out	Boolean, *True*, falls die Signatur gültig ist, andernfalls *False*
	M	Oktettstring, die rekonstruierte Nachricht
Errors:	–	–
Notation:		(*out*, *M*) = RSA_ISO9796_2_DS1_VERIFY(*PuK*, *sig*, M_2)

(N37) Gemäß [9796–2] Kapitel 7.2 MUSS das COS folgende Aktionen durchführen, wobei folgende Definitionen gelten: n = *PuK.n*, e = *PuK.e.*

a. Schritt 1: Signature opening: Gemäß [9796–2] Annex A.5:

  i. Schritt 1.1: Falls die Bitlänge von *sig* ungleich der Bitlänge von n ist, dann gebe *False* zurück und breche diesen Algorithmus ab.

  ii. Schritt 1.2: Falls das höchstwertige Bit von *sig* den Wert 1 hat, dann gebe *False* zurück und breche diesen Algorithmus ab.

  iii. Schritt 1.3: s = OS2I(*sig*).

  iv. Schritt 1.4: J^* = s^e mod n.

  v. Schritt 1.5: Falls J^* gerade ist, dann setze $I^* = J^*$,
  sonst setze $I^* = n - J^*$.

  vi. Schritt 1.6: Falls I^* mod 256 ungleich ´BC´ ist, dann gebe *False* zurück und breche diesen Algorithmus ab.

  vii. Schritt 1.7: F^* = I2OS(I^*, OctetLength(n)).

  viii. Schritt 1.8: Falls das höchstwertige Bit von F^* den Wert 1 hat, dann gebe *False* zurück und breche diesen Algorithmus ab.

b. Schritt 2: Message recovery: Gemäß [9796–2] Kapitel 8.3:

i. Schritt 2.1: Falls das zweithöchste Bit von F^* den Wert 0 hat, dann gebe *False* zurück und breche diesen Algorithmus ab.

ii. Schritt 2.2: Berechne H^* und M_1^* gemäß [9796–2] Kapitel 8.3.

iii. Schritt 2.3: Berechne $M = M_1 \;||\; M_2$.

iv. Schritt 2.4: Berechne H = SHA_256(M).

v. Schritt 2.5: Falls H identisch ist zu H^*, dann gebe *True* und M zurück, sonst gebe *False* ohne M zurück.

Hinweis (14): ACHTUNG: In (N37)b.i ist F ein Oktettstring. Aus diesem F* geht durch Abschneiden des „most significant bit" ein Bitstring hervor, der F* aus [9796–2] Kapitel 8.3 entspricht. Deshalb wird in (N37)b.i das zweithöchste Bit geprüft und in [9796–2] Kapitel 8.3 das „leftmost" Bit.*

7.6.4.2 Signatur Prüfung mittels elliptischer Kurven, G2 (informativ)

Input:	PuK	ein öffentlicher ELC Schlüssel gemäß Kapitel 9.2.4
	H	beliebiger Oktettstring der einen Hashwert repräsentiert
	R	Oktettstring, erster Teil der ECDSA Signatur
	S	Oktettstring, zweiter Teil der ECDSA Signatur
Output:	out	Boolean, *True*, falls die Signatur gültig ist, andernfalls *False*
Errors:	–	–
Notation:		out = ELC_VER_SIG(PuK, R, S, H)

(N38) Gemäß [TR–03111] Kapitel 4.2.2 MUSS das COS folgende Aktionen durchführen, wobei folgende Definitionen gelten:
n = *PuK.domainParameter.n*, G = *PuK.domainParameter.G*,
P_a = *PuK.P*

a. Schritt 0: r ß OS2I(R).
s ß OS2I(S).

b. Schritt 1: Überprüfe, ob r und s Element der Menge {1, 2, …, n – 1} sind. Falls nicht gebe *False* zurück und breche diesen Algorithmus ab.

c. Schritt 2: s_{inv} ß s^{-1} mod n.

d. Schritt 3: u_1 ß s_{inv} OS2I (H) mod n.
u_2 ß s_{inv} r mod n.

e. Schritt 4: Q ß $[u_1]\, G + [u_2]\, P_a$.

f. Schritt 5: v ß OS2I (FE2OS(x_Q)) mod n.

g. Schritt 6: Gebe *True* zurück, falls v gleich r ist, andernfalls *False*.

7.7 Vertraulichkeit von Daten, symmetrischer Fall

7.7.1 Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung überführt eine beliebige Nachricht *plaintext* (Oktettstring beliebigen Inhalts und Länge) in ein Chiffrat *ciphertext*. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine symmetrische Verschlüsselung als Funktion wie folgt verwendet:

7.7.1.1 Verschlüsselung 3TDES, G1 (normativ)

Input:	P	Oktettstring, Klartext (*plaintext*), beliebiger Oktettstring beliebiger Länge, der verschlüsselt wird.
	K	beliebiger Oktettstring der Länge 24 Oktett, der als Schlüssel verwendet wird
	T_1	beliebige nicht negative Zahl, die als Startwert verwendet wird
Output:	C	Oktettstring, Chiffrat (*ciphertext*)
Errors:	*lengthError*	die Länge von P ist kein ganzzahliges Vielfaches der Blocklänge
Notation:		C = 3TDES_CBC_ENC(K, T_1, P)

(N39) Das COS MUSS C mittels K und T_1 wie folgt aus P berechnen. Dabei sind folgende Aktionen durchzuführen:

a. Schritt 1: Falls OctetLength(P) mod 8 ungleich 0 ist, dann gebe den Fehler *lengthError* zurück und breche diesen Algorithmus ab.

b. Schritt 2: Teile P auf in Blöcke mit jeweils 64 Bit $P = P_1 \;||\; P_2 \;||\; \ldots \;||\; P_n$.

c. Schritt 3: Setze C_0 = I2OS(T_1, 8)

d. Schritt 4: Berechne C_i = 3TDES_ENC(K, C_{i-1} XOR P_i) für $i = 1,\ldots, n$.

e. Schritt 5: Berechne $C = C_1 \;||\; C_2 \;||\; \ldots \;||\; C_{n-1} \;||\; C_n$.

7.7.1.2 Verschlüsselung AES–128, G2 (informativ)

Input:	P	Oktettstring, Klartext (*plaintext*), beliebiger Oktettstring beliebiger Länge, der verschlüsselt wird.
	K	beliebiger Oktettstring der Länge 16 Oktett, der als Schlüssel verwendet wird

	T_1	beliebige nicht negative Zahl, die als Startwert für die Zähler (Counter) verwendet wird
Output:	C	Oktettstring, Chiffrat (*ciphertext*), das dieselbe Länge wie *P* besitzt.
	T_{n+1}	positive Zahl, die bei der nächsten Operation mit *K* als T_1 verwendet wird.
Errors:	–	keine
Notation:		$(C, T_{n+1}) = \text{AES\_CTR\_ENC}(K, T_1, P)$

(N40) Das COS MUSS *C* und T_{n+1} mittels *K* gemäß [AES–CTR] Kapitel 6.5 aus *P* berechnen. Dabei sind folgende Aktionen durchzuführen:

a. Teile *P* auf in die Blöcke $P = P_1 \ || \ P_2 \ || \ \dots \ || \ P_{n-1} \ || \ P^*_n$.
Die Blöcke P_1 bis P_{n-1} besitzen die Länge 128 Bit. Der Block P^*_n besitzt eine Länge kleiner gleich 128 Bit.

b. Setze $O_j = \text{AES\_ENC}(K, \text{I2OS}(T_1 + j - 1, 16))$ für $j = 1, 2, \dots, n$.

c. Setze $C_j = P_j \text{ XOR } O_j$ für $j = 1, 2, \dots, n - 1$.

d. Setze $C^*_n = P^*_n \text{ XOR } \text{Extract\_MSByte}(O_n, \text{OctetLength}(P^*_n))$.

e. Setze $C = C_1 \ || \ C_2 \ || \ \dots \ || \ C_{n-1} \ || \ C^*_n$

f. Setze $T_{n+1} = T_1 + n$

7.7.2 Symmetrische Entschlüsselung

Die symmetrische Entschlüsselung überführt ein Chiffrat *ciphertext* (Oktettstring beliebigen Inhalts und Länge) in einen Klartext *plaintext*. Diese Funktionalität wird an der physikalischen Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine symmetrische Verschlüsselung als Funktion wie folgt verwendet:

7.7.2.1 Entschlüsselung 3TDES, G1 (normativ)

Input:	C	beliebiger Oktettstring, Chiffrat (*ciphertext*), der entschlüsselt wird, die Länge ist ein ganzzahliges Vielfaches der Blocklänge
	K	beliebiger Oktettstring der Länge 192 Bit, der als Schlüssel verwendet wird
	T_1	beliebige nicht negative Zahl, die als Startwert verwendet wird
Output:	P	Oktettstring, Klartext (*plaintext*)

Errors:	InvalidLength	Die Länge von C ist kein ganzzahliges Vielfaches der Blocklänge
Notation:		P = 3TDES_CBC_DEC(K, T_1, C)

(N41) Das COS MUSS P mittels K und T_1 wie folgt aus C berechnen. Dabei sind folgende Aktionen durchzuführen:

a. Schritt 1: Teile C auf in Blöcke mit jeweils 64 Bit $C = C_1 \,||\, C_2 \,||\, \dots \,||\, C_n$.
b. Schritt 2: Setze P_0 = I2OS(T_1, 8)
c. Schritt 3: Berechne P_i = 3TDES_DEC(K, C_i) XOR C_{i-1} für $i = 1,\dots,n$.
d. Schritt 4: Berechne $P = P_1 \,||\, P_2 \,||\, \dots \,||\, P_{n-1} \,||\, P_n$.

7.7.2.2 Entschlüsselung AES–128, G2 (informativ)

Input:	C	beliebiger Oktettstring, Chiffrat (*ciphertext*), beliebiger Länge, der entschlüsselt wird.
	K	beliebiger Oktettstring der Länge 128 Bit, der als Schlüssel verwendet wird
	T_1	beliebige nicht negative Zahl, die als Startwert für die Zähler (Counter) verwendet wird
Output:	P	Oktettstring, Klartext (*plaintext*), der dieselbe Länge wie C besitzt.
	T_{n+1}	positive Zahl, die bei der nächsten Operation mit K als T_1 verwendet wird.
Errors:	–	keine
Notation:		(P, T_{n+1}) = AES_CTR_DEC(K, T_1, C)

(N42) Es gilt: (P, T_{n+1}) = AES_CTR_DEC(K, T_1, C) = AES_CTR_ENC(K, T_1, C).

7.8 Vertraulichkeit von Daten, asymmetrischer Fall

7.8.1 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung überführt eine Nachricht M (Oktettstring beliebigen Inhalts und Länge) in ein Chiffrat C. Diese Funktionalität wird an der physikalischen

Schnittstelle nicht unmittelbar sichtbar. Im Rahmen diverser interner Operationen im Betriebssystem wird eine asymmetrische Verschlüsselung als Funktion wie folgt verwendet:

7.8.1.1 RSA, ES, PKCS1 V1.5, G1 (normativ)

Input:	PuK	ein öffentlicher RSA Schlüssel gemäß Kapitel 9.2.4
	M	beliebiger Oktettstring, zu verschlüsselnde Nachricht
Output:	C	Oktettstring, Chiffrat zur Nachricht *M*
Errors:	ERROR	falls einer der folgenden Fälle eintritt

- „message too long“, falls *M* ist zu lang

Hinweis (15): Solange diese Funktion im Rahmen von PSO Transcipher eingesetzt wird, tritt diese Fehlersituation nicht ein.

Notation: *C* = RSAES_PKCS1_V1_5_ENCRYPT(*PuK*, *M*)

(N43) Das COS MUSS *C* mittels *PuK* und *M* gemäß [PKCS#1] Kapitel 7.2.1 berechnen. Es gelten folgende Definitionen: *n* = *PuK.n*, *e* = *PuK.e*.

a. Schritt 1: Setze *Plen* OctetLength(*n*) – OctetLength(*M*) – 3.
b. Falls *Plen* kleiner als acht ist, dann breche diesen Algorithmus mit der Fehlermeldung „message too long“ ab.
c. Schritt 1: Setze *EM* ß ´00´ || *M*.
d. Schritt 2: Setze *PS* ß RAND(1).
e. Schritt 3: Falls *PS* gleich ´00´ ist, fahre mit Schritt 2 fort.
f. Schritt 4: Setze *EM* ß *PS* || *EM*.
g. Schritt 5: Falls OctetLength(*EM*) kleiner als OctetLength(*n*) – 2 ist, dann fahre mit Schritt 2 fort.
h. Schritt 6: Setze *EM* ß ´02´ || *EM*.
i. Schritt 7: Setze *m* ß OS2I(*EM*).
j. Schritt 6: Setze *c* ß m^e mod *n*.
k. Schritt 7: Setze *C* ß I2OS(*c*, OctetLength(*n*)).

Kom
tenz
gen c

7.8.1.2 RSA, OAEP, Verschlüsselung, G1 (normativ)

Input:	PuK	ein öffentlicher RSA Schlüssel gemäß Kapitel 9.2.4
	M	beliebiger Oktettstring, zu verschlüsselnde Nachricht
Output:	C	Oktettstring, Chiffrat zur Nachricht *M*
Errors:	ERROR	falls einer der folgenden Fälle eintritt

- „message too long“, falls *M* ist zu lang

Hinweis (16): Solange diese Funktion im Rahmen von PSO Transcipher eingesetzt wird, tritt diese Fehlersituation nicht ein.

Notation: *C* = RSAES_OAEP_ENCRYPT(*PuK*, *M*)

(N44) Das COS MUSS *C* mittels *PuK* und *M* gemäß [PKCS#1] Kapitel 7.1.1 berechnen. Es gelten folgende Definitionen: *n* = *PuK.n*, *e* = *PuK.e*.

a. Schritt 1: Falls Octetlength(*M*) größer als OcetLength(*n*) – 66 ist, dann breche diesen Algorithmus mit dem Fehler „message too long“ ab.
b. Schritt 2: Setze *L* = ´´, (leerer Oktettstring).
c. Schritt 3: Setze *lhash* = SHA_256(*L*).
d. Schritt 4: Setze *Plen* = OctetLength(*n*) – OctetLength(*M*) – 66.
e. Schritt 5: Setze *PS* = I2OS(0, *Plen*).
f. Schritt 6: Setze DB = lHash || PS || ´01´ || *M*.
g. Schritt 7: Setze *seed* = RAND(32).
h. Schritt 8: Setze *dbMask* = MGF(seed, Octetlength(*n*) – 33).
i. Schritt 9: Setze *maskedDB* = *DB* XOR *dbMasked*.
j. Schritt 10: Setze *seedMask* = MGF(*maskedDB*, 32).
k. Schritt 11: Setze *maskedSeed* = *seed* XOR *seedMask*.
l. Schritt 12: Setze *EM* = ´00´ || *maskedSeed* || *maskedDB*.
m. Schritt 13: Setze *m* = OS2I(*EM*).
n. Schritt 14: Setze *c* = m^e mod *n*.
o. Schritt 15: Setze *C* = I2OS(*c*, OctetLength(*n*)).

7.8.1.3 ELC Verschlüsselung, G2 (informativ)

Input:	M	Oktettstring, zu verschickende Nachricht mit beliebigem Inhalt und beliebiger Länge, die verschlüsselt wird
	PO_B	Oktettstring, öffentlicher Punkt P_B des Empfängers
	dP	Domainparameter gemäß (N86)
Output:	PO_A	Oktettstring, ephemer Punkt PE_A des Senders
	C	Oktettstring, Chiffrat der Nachricht *M*
	T	Oktettstring, MAC über das Chiffrat *C*
Errors:	ERROR	falls das Ergebnis in Schritt 5 der unendlich ferne Punkt ist

Notation: (PO_A, C, T) = ELC_ENC(M, PO_B, dP)

(N45) Das COS MUSS PO_A, C und T mittels M und PO_B berechnen, wobei die Schritte 1 bis 6 dem Kapitel 4.3.1 aus [TR–03111] entsprechen und folgende Definitionen gelten: $n = dP.n$, $h = dP.h$, $G = dP.G$, $L = dP.L$

a. Schritt 0: P_B ß OS2P(PO_B, dP).
Falls diese Funktion mit einem Fehler terminiert, dann gebe "*ERROR*" zurück und beende diesen Algorithmus.

b. Schritt 1: r ß RNG({1, 2, …, n – 1}).

c. Schritt 2: PE_A ß [r] G.

d. Schritt 3: l ß h^{-1} mod n.

e. Schritt 4: Q ß [h] P_B.

f. Schritt 5: S ß [r l mod n] Q.
Falls S gleich dem unendlich fernen Punk O der Kurve ist, dann gebe den Fehler „*ERROR*" zurück und beende diesen Algorithmus.

g. Schritt 6: K_{AB} ß I2OS(x_S, L).

h. Schritt 7: Berechne abgeleitete Schlüssel gemäß (N15)
(K_{enc}, T_1, K_{mac}, SSC) ß KeyDerivation_AES128(K_{AB}).

i. Schritt 8: PO_A ß P2OS(PE_A, L).

j. Schritt 9: (C, x) ß AES_CTR_ENC(K_{enc}, T_1, M).

k. Schritt 10: T ß AES_CMAC(K_{mac}, C).

Hinweis (17): Der Rückgabewert SSC in Schritt 7 wird hier nie verwendet.

Hinweis (18): Der Rückgabewert x in Schritt 9 wird hier nie verwendet.

7.8.2 Asymmetrische Entschlüsselung

Die asymmetrische Entschlüsselung überführt ein Chiffrat C in eine Nachricht M.

7.8.2.1 RSA, ES, PKCS1 V1.5, Decrypt, G1 (normativ)

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Decipher sichtbar (siehe (N903)a).

Input:	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
	C	beliebiger Oktettstring, Chiffrat der Nachricht *M*
Output:	M	Oktettstring, Klartextnachricht zum Chiffrat
Errors:	ERROR	„decryption error", falls einer der folgenden Fälle eintritt
		• *C* ist numerisch größer gleich dem Modulus

- *EM* fängt nicht mit ´0002´an
- In *EM* gibt es kein ´00´ Oktett, welches PS von M trennt
- *PS* hat weniger als acht Oktett

Notation: *M* = RSAES_PKCS1_V1_5_DECRYPT(*PrK*, *C*)

(N46) Das COS MUSS *M* mittels *PrK* und *C* gemäß [PKCS#1] Kapitel 7.2.2 berechnen. Es gelten folgende Definitionen: *n* = *PrK.n*, *d* = *PrK.d*.

a. Schritt 1: Falls OctetLength(C) ungleich OctetLength(n) ist, breche diesen Algorithmus mit „decryption error" ab.
b. Schritt 2: Setze *c* = OS2I(*C*).
c. Schritt 3: Falls *c* größer oder gleich *n* ist, breche diesen Algorithmus mit „decryption error" ab.
d. Schritt 4: Setze *EM* = I2OS(c^d mod *n*, OctetLength(*n*)).
e. Falls das erste Oktett in *EM* ungleich ´00´ist, oder das zweite Oktett in *EM* ungleich ´02´ ist, breche diesen Algorithmus mit „decryption error" ab.
f. Schritt 5: Teile EM wie folgt auf: *EM* = ´00 02 || *PS* || 00 || *M*´. Dabei MUSS jedes Oktett in *PS* ungleich ´00´ sein.
g. Schritt 6: Breche diesen Algorithmus mit „decryption error" ab, falls
   i. *PS* kürzer als 8 Oktett ist, oder
   ii. kein Oktett mit dem Wert ´00´existiert, welches *PS* von *M* trennt.
h. Schritt 7: Gebe *M* zurück.

7.8.2.2 RSA, OAEP, Decrypt, G1 (normativ)

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Decipher sichtbar (siehe (N903)b).

Input:	PrK	ein privater RSA Schlüssel gemäß Kapitel 9.2.3
	C	beliebiger Oktettstring, Chiffrat der Nachricht *M*
Output:	M	Oktettstring, Klartextnachricht zum Chiffrat
Errors:	ERROR	„decryption error", falls einer der folgenden Fälle eintritt • *C* ist numerisch größer gleich dem Modulus

Notation: *M* = RSAES_OAEP_DECRYPT(*PrK*, *C*)

(N47) Das COS MUSS *M* mittels *PrK* und *C* gemäß [PKCS#1] Kapitel 7.1.2 berechnen. Es gelten folgende Definitionen: *n* = *PrK.n*, *d* = *PrK.d*.

a. Schritt 1: Setze *L* = ´´, (leerer Oktettstring).
b. Schritt 2: Setze *c* = OS2I(*C*).
c. Schritt 3: Falls *c* größer gleich *n* ist, dann breche diesen Algorithmus mit dem Fehler „decryption error“ ab.
d. Schritt 4: Setze *m* = c^d mod *n*.
e. Schritt 5: Setze *EM* = I2OS(*m*, OctetLength(*n*)).
f. Schritt 6: Setze *lHash* = SHA_256(*L*).
g. Schritt 7: Teile *EM* wie folgt auf:
    i. Schritt 7.1: *EM* = *Y* || *maskedSeed* || *maskedDB*.
    ii. Schritt 7.2: 1 = OctetLenght(*Y*).
    iii. Schritt 7.3: 32 = OctetLength(*maskedSeed*).
h. Schritt 8: Setze *seedMask* = MGF(OS2BS(*maskedDB*), 256, 0).
i. Schritt 9: Setze *seed* = OS2BS(*maskedSeed*) XOR *seedMask*.
j. Schritt 10: Setze *dbMask* = MGF(*seed*, 8 OctetLength(*maskedDB*),0).
k. Schritt 11: Setze *DB* = *maskedDB* XOR BS2OS(*dbMask*).
l. Schritt 12: Teile *DB* wie folgt auf:
    i. Schritt 12.1: *DB* = *lHash*’ || *PS* || ´01´ || *M*.
    ii. Schritt 12.2: 32 = OctetLength(*lhash*’).
    iii. Schritt 12.3: Der möglicherweise leere Oktettstring *PS* darf nur Oktette mit dem Wert ´00´ enthalten.
m. Schritt 13: Breche diesen Algorithmus mit „decryption error“ ab, wenn
    i. *lHash*’ ungleich *lHash* ist, oder
    ii. *Y* nicht den Wert ´00´ besitzt, oder
    iii. kein Oktett mit dem Wert ´01´ existiert, welches *PS* von *M* trennt.
n. Gebe *M* zurück.

7.8.2.3 Asymmetrische Entschlüsselung mittels ELC, G2 (informativ)

Diese Funktionalität wird an der physikalischen Schnittstelle im Rahmen des Kommandos PSO Decipher sichtbar (siehe (N903)c).

Input:	PrK	ein privater ELC Schlüssel gemäß Kapitel 9.2.3
	PO	Oktettstring, ephemer Punkt PE_A des Senders
	C	Oktettstring, Chiffrat der Nachricht *M*
	T´	Oktettstring, MAC über das Chiffrat *C*
Output:	M	Oktettstring, Klartextnachricht zum Chiffrat

Errors:	ERROR	• falls *PO* nicht im Format „uncompressed encoding“ vorliegt • falls *PO* einen Punkt bezeichnet, der nicht auf derselben Kurve liegt, wie *PrK* • falls das Ergebnis in Schritt 3 der unendlich ferne Punkt ist • falls die MAC Prüfung fehlschlägt
Notation:		M = ELC_DEC(PO, PrK, C, T')

(N48) Das COS MUSS M mittels PE_A, PrK, C und T' berechnen, wobei die Schritte 1 bis 4 dem Kapitel 4.3.2 aus [TR–03111] entsprechen. Es gelten folgende Definitionen:

n = *PrK.domainParameter.n*, h = *PrK.domainParameter.h*,
d_B = *PrK.domainParameter.d*, L = *PrK.domainParameter.L*

a. Schritt 0: PE_A ß OS2P(*PO*, *PrK.domainParameter*).
Falls diese Funktion mit einem Fehler terminiert, dann gebe “*ERROR*” zurück und beende diesen Algorithmus.

b. Schritt 1: l ß h^{-1} mod n.

c. Schritt 2: Q ß [h] PE_A mit $Q = (x_Q, y_Q)$.

d. Schritt 3: S ß [d_B l mod n] Q. mit $S = (x_S, y_S)$.
Falls S gleich dem unendlich fernen Punk O der Kurve ist, dann gebe den Fehler „*ERROR*“ zurück und beende diesen Algorithmus.

e. Schritt 4: K_{AB} ß I2OS(x_S, L).

f. Schritt 5: Berechne abgeleitete Schlüssel gemäß (N15)
(K_{enc}, T_1, K_{mac}, *SSC*) ß Key_Derivation(K_{AB}).

g. Schritt 6: *out* = AES_VER_CMAC(K_{mac}, T', C).
Falls *out* den Wert *INVALID* besitzt, dann gebe den Fehler „*ERROR*“ zurück und beende diesen Algorithmus.

h. Schritt 7: (M, x) ß AES_CTR_DEC(K_{enc}, T_1, C)

Hinweis (19): Der Rückgabewert SSC in Schritt 5 wird hier nie verwendet.

Hinweis (20): Der Rückgabewert x in Schritt 7 wird hier nie verwendet.

8 CV–Zertifikat (normativ)

Nach [7816–8] ist ein Card Verifiable Certificate (CVC) ein Zertifikat, welches von Chipkarten prüfbar ist. Typischerweise enthalten CV–Zertifikate nur die Informationen, welche eine Chipkarte notwendigerweise für einen bestimmten Use Case benötigt. Sie sind deshalb kompakter als andere Zertifikatsformate, was sich positiv auf die Performance auswirkt.

8.1 CV–Zertifikat für RSA Schlüssel, G1 (normativ)

In diesem Dokument werden nur so genannte „nicht selbstbeschreibende“ CV–Zertifikate betrachtet, welche eine Signatur mit „Message Recovery“ gemäß [9796–2] DS1 enthalten.

Technisch betrachtet ist ein derartiges Zertifikat eine mittels [9796–2] DS1 signierte Nachricht *M*. Die Nachricht *M* enthält unterschiedliche Daten, die ohne Strukturinformationen oder Trennzeichen konkateniert ist, daher die Bezeichnung „nicht selbstbeschreibend“. Damit die enthaltenen Informationen eindeutig extrahierbar sind, wird das höchstwertige Oktett von *M* ausgewertet.

8.1.1 Bestandteile eines CV–Zertifikats

Alle im Kapitel 8.1 betrachteten signierten Nachrichten *M* enthalten die in den folgenden Unterkapiteln beschriebenen Informationen.

8.1.1.1 Certificate Profile Identifier (CPI)

Der Certificate Profile Identifier (CPI) hat den Zweck, die genaue Struktur eines CV–Zertifikates anzuzeigen.

8.1.1.2 Certification Authority Reference (CAR)

Die Certification Authority Reference (CAR) referenziert den Schlüssel der CA, welche das Zertifikat ausstellte. Typischerweise besitzt die CAR eine innere Struktur, welche sicherstellt, dass die CAR weltweit eineindeutig ist. Festlegungen dazu sind in Dokument [gemPKI_Reg] enthalten.

Typischerweise besitzt die ausstellende CA wiederum ein CV–Zertifikat. Auf diese Weise wird eine baumartige PKI aufgespannt mit der Root–CA an der Wurzel.

Der öffentliche Schlüssel der Root–CA wird oft als Sicherheitsanker bezeichnet und typischerweise in der Vorbereitungsphase (siehe Kapitel 5) in einer Chipkarte gespeichert. Deshalb ist es nicht notwendig, den Sicherheitsanker per Zertifikat zu importieren (Abbruch der Rekursion).

8.1.1.3 Certificate Holder Reference (CHR)

Die Certificate Holder Reference (CHR) wird dazu verwendet, dem im Zertifikat enthaltenen öffentlichen Schlüssel einen eindeutigen Identifier zuzuordnen. Typischerweise wird die ebenfalls eindeutige Seriennummer einer Chipkarte (ICCSN) als Teil der CHR verwendet. Im Rahmen dieses Dokumentes ist es irrelevant, wie die CHR gebildet wird. Festlegungen dazu sind in Dokument [gemPKI_Reg] enthalten.

8.1.1.4 Certificate Holder Autorisation (CHA)

Die Certificate Holder Autorisation (CHA) zeigt eine Rolle des Zertifikatsinhabers an. Typischerweise fordern Sicherheitskonzepte, dass zwei verschiedene Rollen unterhalb einer Root–CA nicht dieselbe CHA haben dürfen. Deshalb wird häufiger auch die Forderung erhoben, dass die CHA weltweit eineindeutig ist. Dann wird typischerweise ein Teil des eindeutigen Application Identifiers als Teil der CHA verwendet. Im Rahmen dieses Dokumentes ist es irrelevant, wie konkrete CHA–Werte festgelegt werden. Festlegungen dazu sind in Dokument [gemPKI_Reg] enthalten.

8.1.1.5 Object Identifier (OID)

Der Object Identifier (OID) in einem CVC beschreibt den Algorithmus des Schlüssels des Zertifikatsinhabers. Im Rahmen dieser Spezifikation werden verschiedene Algorithmen für verschiedene Verwendungszwecke genutzt, so dass implizit durch den Algorithmus–OID auch der Verwendungszweck des im Zertifikat enthaltenen öffentlichen Schlüssels fest. OIDs sind weltweit eineindeutig.

Im Rahmen dieses Dokumentes werden nur folgende OID–Werte verwendet:

Tabelle 6: Object Identifier der Registration Authority TeleTrusT (www.teletrust.de)

OID–Name	OID–Wert	OID–Kodierung
sigS_ISO9796-2Withrsa_sha256 signature scheme with RSA signature and DSI according to [9796–2] and [SHA–256]	{1 3 36 3 4 2 2 4}	`2B24 0304 0202 04`
authS_ISO9796-2Withrsa_sha256_mutual authentication scheme with RSA signature and DSI according to [9796–2] and [SHA–256] for mutual authentication with or without establishment of a Trusted Channel	{1 3 36 3 5 2 4}	`2B24 0305 0204`

8.1.1.6 Öffentlicher Schlüssel

Der öffentliche RSA Schlüssel besteht aus den in den folgenden Unterkapiteln beschriebenen Teilen.

8.1.1.6.1 Modulus

Der Modulus wird hexadezimal, vorzeichenlos im „big endian“ Format kodiert.

8.1.1.6.2 Öffentlicher Exponent

Der öffentliche Exponent wird hexadezimal, vorzeichenlos im „big endian“ Format kodiert.

8.1.2 Zertifikatsprofile

Dieses Kapitel listet die zu unterstützenden Zertifikatsprofile auf, die anhand des CPI identifiziert werden.

8.1.2.1 CV–Zertifikat für CA–Schlüssel

In diesem Unterkapitel wird der Inhalt eines CV–Zertifikates beschrieben, welches den öffentlichen Schlüssel einer CA enthält. Dieser öffentliche Schlüssel ist wiederum geeignet im Rahmen eines PSO Verify Certificate Kommandos (siehe 15.8.6.1) weitere öffentliche Schlüssel per CV–Zertifikat zu importieren. Für derartige CV–Zertifikate gilt:

(N49) Der CPI MUSS den Wert ´21´ haben.

(N50) Die Kodierung des OID MUSS sieben Byte lang sein. Im Rahmen dieses Dokumentes wird dann lediglich der Wert ´2B24 0304 0202 04´ für den OID {1.3.36.3.4.2.2.4} verwendet (siehe Tabelle 6).

(N51) Die Länge des Modulus MUSS 28 Oktette kleiner als die Nachricht *M* sein.

(N52) Der öffentliche Exponent MUSS vier Byte lang sein.

(N53) Die CHR MUSS acht Byte lang sein.

(N54) Die CAR MUSS acht Byte lang sein.

(N55) Für die zu signierende Nachricht *M* gilt:
M = CPI || Modulus || öffentlicherExponent || OID || CHR || CAR.

Wenn (N21)a berücksichtigt wird und der Hashwert gemäß Kapitel 7.1 gebildet wird, dann gehören CHR und CAR stets zum „non recoverable part“ der signierten Nachricht und liegen somit stets im Klartext vor.

8.1.2.2 CV–Zertifikat für Authentisierungsschlüssel

In diesem Unterkapitel wird der Aufbau eines CV–Zertifikats beschrieben, welches den öffentlichen Schlüssel einer Instanz enthält, welche diesen Schlüssel zu Authentisierungszwecken verwendet. Für derartige CV–Zertifikate gilt:

(N56) Der CPI MUSS den Wert ´22´ haben.

(N57) Die CHA MUSS sieben Byte lang sein.

(N58) Die Kodierung des OID MUSS sechs Byte lang sein. Im Rahmen dieses Dokumentes wird dann lediglich der Wert ´2B24 0305 0204´ für den OID {1 3 36 3 5 2 4} verwendet (siehe Tabelle 6).

(N59) Die Länge des Modulus MUSS 38 Oktette kleiner als die Nachricht *M* sein.

(N60) Der öffentliche Exponent MUSS vier Byte lang sein.

(N61) Die CHR MUSS zwölf Byte lang sein.

(N62) Die CAR MUSS acht Byte lang sein.

(N63) Für die zu signierende Nachricht *M* gilt:
M = CPI || Modulus || öffentlicherExponent || OID || CHA || CHR || CAR.

Wenn (N21)a berücksichtigt wird und der Hashwert gemäß Kapitel 7.1 gebildet wird, dann gehören CHR und CAR stets zum „non recoverable part“ der signierten Nachricht und liegen damit stets im Klartext vor.

8.1.3 Struktur und Inhalt eines CV–Zertifikats

Die Inhalte eines Zertifikats–Datenobjektes werden wie folgt berechnet:

(N64) Schritt 1: Die Nachricht *M* MUSS gemäß Kapitel 8.1.2 erzeugt werden.

(N65) Schritt 2: Die Nachricht *M* MUSS mit einem privaten RSA Schlüssel *PrK* signiert werden. Als Signaturverfahren MUSS (N31) verwendet werden, so dass gilt:
(*SIG.CA*, M_2) = RSA_ISO9796_2_DS1_SIGN(*PrK*, *M*).

(N66) Ein CV–Zertifikats–EF MUSS ein zusammengesetztes Zertifikat–Datenobjekt mit Tag = ´7F21´ (RSA–Zertifikat mit message recovery) enthalten. Das Zertifikats–Datenobjekt MUSS zwei primitive Datenobjekte enthalten:

a. Datenelement *SIG.CA* als Wertfeld in einem Datenobjekt mit Tag = ´5F37´.

b. Non–recoverable part M_2 als Wertfeld in einem Datenobjekt mit Tag = ´5F38´.

CV–Zertifikate für RSA Schlüssel mit einer Moduluslänge von 2048 Bit = 256 Oktett:

Tabelle 7: CV–Zertifikat einer CA mit CPI = ´21´, SHA–256

Tag	L	Wert		
´7F21´	´820146´	CV–Zertifikat (´0146´ = 326 Oktett)		
		Tag	L	Wert
		´5F37´	´820100´	Signatur SIG.CA (´0100´ = 256 Oktett)
		´5F38´	´3E´	non recoverable part M2 (´003E´ = 62 Oktett)

Tabelle 8: CV–Zertifikat zur Authentisierung mit CPI = ´22´, SHA–256

Tag	L	Wert		
´7F21´	´820150´	CV–Zertifikat (´0150´ = 336 Oktett)		
		Tag	L	Wert
		´5F37´	´820100´	Signatur SIG.CA (´0100´ = 256 Oktett)
		´5F38´	´48´	non recoverable part M2 (´0048´ = 72 Oktett)

8.2 CV–Zertifikat für ELC Schlüssel, G2 (informativ)

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.

9 Objekte (normativ)

9.1 Diverse Attribute (normativ)

Dieses Unterkapitel beschreibt einige Attribute, die für mehrere Objekttypen gleichermaßen relevant sind.

9.1.1 File Identifier

Der Attributstyp *fileIdentifier* wird von den Objekttypen DF und EF verwendet.

Aus der Norm [7816–4] leiten sich folgende Regeln ab, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N67) Der Wert von *fileIdentifier* MUSS eine ganze Zahl im Intervall [´1000´, ´FEFF´] sein.

(N68) Ein Objekt, das nicht *root* (siehe Kapitel 10.1) ist, DARF KEINEN *fileIdentifier* mit dem Wert ´3F00´ besitzen.

(N69) KEIN Objekt DARF einen *fileIdentifier* mit dem Wert ´3FFF´ besitzen.

Hinweis (21): Die Forderung (N67) ist strenger als [7816–4]. Dies lässt Raum für herstellerspezifische Ordner (DF) und Dateien (EF).

9.1.2 Short File Identifier

Es ist möglich, dass der Attributstyp *shortFileIdentifier* von den Objekttypen Datei verwendet wird (siehe Kapitel 9.3.2).

Aus der Norm [7816–4] leiten sich folgende Regeln ab, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N70) Der Wert von *shortFileIdentifier* MUSS eine ganze Zahl im Intervall [1, 30] sein.

9.1.3 Life Cycle Status

Der Attributstyp *lifeCycleStatus* wird von den Objekttypen Ordner, Datei und Rekord verwendet zur Speicherung eines „physikalischen" Life Cycle Status (siehe (N205)). Die Objekttypen Passwort und Schlüssel kennen lediglich einen „logischen" Life Cycle Status (siehe (N206)).

Aus den Normen [7816–4] Tabelle 13 und [7816–9] Abbildung 1 leiten sich folgende Regeln ab, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N71) Der Wert von *lifeCycleStatus* MUSS ein Element der Menge {
„Operational state (activated)“,
„Operational state (deactivated)”
} sein.
Ein COS KANN weitere Werte für *lifeCycleStatus* unterstützen.
Ein COS KANN weitere Werte für *lifeCycleStatus* ablehnen.

9.1.4 Zugriffsregelliste

Der Attributstyp *accessRuleList* wird von den Objekttypen verwendet, welche die Ausführung von Kommandos von der Erfüllung von Zugriffsregeln abhängig machen. Dabei handelt es sich um eine Ansammlung von Zugriffsregeln. Typischerweise werden einem Objekt mehr als eine Zugriffsregel zugeordnet, weil es dem COS dadurch möglich ist, situationsbezogen zu reagieren. So sind gewisse Leseoperationen typischerweise nur über eine kontaktbehaftete Schnittstelle zulässig, während sie für eine kontaktlose Schnittstelle verboten sind. Andererseits ist der Zugriff auf deaktivierte Objekte typischerweise stark eingeschränkt. Darüber hinaus werden Security Environments genutzt um Abhängigkeiten der Zugriffsregel von der Einsatzumgebung abbilden zu können.

Akademisch betrachtet wäre eine dreidimensionale Matrix mit Zugriffsregeln angebracht, wo je nach Interface (kontaktbehaftet/kontaktlos = erste Dimension) Lebenszyklus (aktiviert/deaktiviert = zweite Dimension) und Einsatzumgebung (SE–Identifier = dritte Dimension) eine Zugriffsregel ausgewählt wird.

In diesem Dokument fällt die erste Dimension mangels kontaktloser Schnittstelle weg und für die zweite Dimension ist es im deaktivierten Fall hinreichend, nur eine Einsatzumgebung vorzusehen. Deshalb wird in dieser Version des Dokumentes statt einer (mehrdimensionalen) Matrix eine schlichte Liste verwendet.

(N72) Der Wert von *accessRuleList* MUSS eine Liste mit mindestens zwei und maximal fünf Elementen sein.
Ein COS KANN Listen mit mehr Elementen unterstützen.
Ein COS KANN Listen mit mehr Elementen ablehnen.

(N73) Ein Listenelement MUSS für den Wert der Variable *lifeCycleStatus* = „Operational state (deactivated)” (siehe 9.1.3) verwendet werden.

(N74) Die übrigen Listenelement MÜSSEN genau einem *seIdentifier* Wert gemäß (N79) zugeordnet werden.

(N75) Jedes Listenelement MUSS genau eine Zugriffsregel gemäß Kapitel 11.3 enthalten.

9.1.5 Rekord

Der Objekttyp *record* wird von den Objekttypen linear variables, linear fixes und zyklisches Elementary File (EF) verwendet.

(N76) Ein *record* MUSS eine Rekordnummer *number* besitzen. Die Rekordnummer MUSS eine ganze Zahl im Intervall [1, 254] sein.
Ein COS KANN Rekordnummern mit weiteren Werten unterstützen.
Ein COS KANN Rekordnummern mit weiteren Werten ablehnen.

(N77) Ein *record* MUSS einen Oktettstring *data* besitzen, dessen Anzahl enthaltener Oktette im dem Intervall [1, 255] liegen MUSS.
Ein COS KANN weitere Längen für *record* unterstützen.
Ein COS KANN weitere Längen für *record* ablehnen.

(N78) Ein *record* MUSS eine Liste mit Attributen vom Typ *lifeCycleStatus* (siehe Kapitel 9.1.3) unterstützen. Diese Liste ist entweder

a. leer (der *record* hat keinen *lifeCycleStatus* und befindet sich implizit im Zustand „Operational state (activated)“) oder
b. enthält genau ein Element (der *record* hat genau einen *lifeCycleStatus*).

9.1.6 SE–Identifier

Der Attributstyp *seIdentifier* ist eng mit dem Begriff Security Environment verknüpft (siehe Kapitel 9.7). Aus dem gemäß [7816–4] Kapitel 6.3.3 erlaubten Wertebereich wird hier folgende Untermenge ausgewählt:

(N79) Der Wert von *seIdentifier* MUSS eine ganze Zahl im Intervall [1, 4] sein.
Das COS KANN weitere Werte für *seIdentifier* akzeptieren.
Das COS KANN weitere Werte für *seIdentifier* ablehnen.

9.1.7 PIN

Der Attributstyp *pin* wird im Zusammenhang mit der Benutzerverifikation verwendet. Es gilt:

(N80) Ein *pin* MUSS eine Folge von Ziffern sein, die aus dem Wertebereich {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} stammen MÜSSEN. Die Anzahl der Ziffern MUSS im Intervall [4, 12] liegen.
Das COS KANN weitere Ziffern oder weitere Längen für *pin* unterstützen.
Das COS KANN weitere Ziffern oder weitere Längen für *pin* ablehnen.

(N81) Für die Umwandlung des Attributstyps *pin* nach Format–2–PIN Block gilt:

a. Der Format–2–PIN Block ist ein Oktettstring mit acht Oktett = 16 Nibble.
b. Das erste Nibble MUSS den Wert ´2´ haben.
c. Das zweite Nibble MUSS hexadezimal die Anzahl Ziffern in *pin* kodieren.
d. Das *i* + 2-te Nibble MUSS hexadezimal die *i*-te Ziffer von *pin* kodieren.
e. Alle anderen Nibble MÜSSEN den Wert ´F´ besitzen.

9.2 Schlüsselmaterial

9.2.1 Symmetrische Schlüssel

9.2.1.1 3TDES Schlüssel, G1 (normativ)

Der Attributstyp *3TDES_Key* dient der Speicherung von Schlüsselmaterial für einen 3TDES Schlüssel, welcher aus drei Teilschlüsseln *Ka*, *Kb* und *Kc* besteht.

(N82) Der Wert von *3TDES_Key* MUSS ein Oktettstring mit 24 Oktette sein.

(N83) Der Wert von *3TDES_Key* MUSS beliebig wählbar sein.

Hinweis (22): Aus Sicherheitsgründen sind die Teilschlüssel Ka, Kb und Kc paarweise verschieden zu wählen. Diese Anforderung richtet sich nicht an das COS, sondern an die externe Welt, welche derartige Schlüssel etwa im Rahmen der Personalisierung in die Chipkarte einbringt.

Hinweis (23): Aus Sicherheitsgründen darf keiner der Teilschlüssel Ka, Kb oder Kc einen Wert haben, welcher einem schwachen oder halbschwachen DES Schlüssel entspricht. Diese Anforderung richtet sich nicht an das COS, sondern an die externe Welt, welche derartige Schlüssel etwa im Rahmen der Personalisierung in die Chipkarte einbringt.

9.2.1.2 AES Schlüssel, G2 (informativ)

Der Attributstyp *aesKey* dient der Speicherung von Schlüsselmaterial für einen AES–128 Schlüssel.

(N84) Der Wert von *aesKey* MUSS ein Oktettstring mit sechzehn Oktette sein.

(N85) Der Wert von *aesKey* MUSS beliebig wählbar sein.

Das Kapitel könnte in einer späteren Version geändert werden. Da Generation 2 Karten möglicherweise bis 2020 im Einsatz sind, ist es denkbar in Generation 2 statt AES–128 gleich AES–256 einzusetzen.
Das muss mal in Ruhe überlegt und diskutiert werden.

9.2.2 Domainparameter für elliptische Kurven, G2 (informativ)

Der Attributstyp *domainParameter* dient der Speicherung von Parametern, welche eine elliptische Kurve charakterisieren. Bei der Spezifikation von Anwendungen sind folgende Regeln zu beachten, vergleiche Tabelle 5:

(N86) domainParameter enthält

a. eine Primzahl *p*,
b. eine Zahl *a*,
c. eine Zahl *b*,
d. einen Punkt *G* auf der elliptischen Kurve, eine Zahl *n*,
e. eine Zahl *h*,
f. eine Zahl *L*, welche die minimale Anzahl Oktette angibt, die nötig sind um *p* als vorzeichenlose Zahl zu kodieren. Dieser Parameter wurde der „offiziellen" Liste der Domainparameter hinzugefügt, da dieser Parameter in diesem Do-

kument vielfach verwendet wird.
Wegen (N25) gilt L = 32 für alle Kurven, die ein COS unterstützen MUSS.

g. einen Oktettstring *OID*, durch den die Domainparameter (p, a, b, G, n, h) weltweit eindeutig bestimmt werden. Dieser Parameter wurde der „offiziellen“ Liste der Domainparameter hinzugefügt, da in diesem Dokument Domainparameter vielfach per *OID* referenziert werden.

9.2.3 Privater Schlüssel

Der Attributstyp *privateKey* dient als Oberbegriff für *privateRsaKey* und *privateElcKey*.

9.2.3.1 Privater RSA Schlüssel, G1 (normativ)

Der Attributstyp *privateRsaKey* dient der Speicherung des privaten Teils eines asymmetrischen RSA Schlüsselpaares. Bei der Spezifikation von Anwendungen sind folgende Regeln zu beachten:

(N87) Ein *privateRsaKey* MUSS ein Attribut n gemäß (N21)a besitzen.

(N88) Ein *privateRsaKey* MUSS ein Attribut d besitzen, dessen Wert eine ganze Zahl im Intervall [1, n – 1] ist.

9.2.3.2 Privater ELC Schlüssel, G2 (informativ)

Der Attributstyp *privateElcKey* dient der Speicherung des privaten Teils eines asymmetrischen ELC Schlüsselpaares. Bei der Spezifikation von Anwendungen sind folgende Regeln zu beachten:

(N89) Ein *privateElcKey* MUSS ein Attribut *domainParameter* gemäß (N86) besitzen, die gemäß (N86)g per *OID* referenziert werden KÖNNEN.

(N90) Ein *privateElcKey* MUSS ein Attribut d besitzen, dessen Wert eine ganze Zahl im Intervall [1, n – 1] sein MUSS.

9.2.4 Öffentlicher Schlüssel

9.2.4.1 Öffentlicher RSA Schlüssel, G1 (normativ)

Der Attributstyp *publicRsaKey* dient der Speicherung des öffentlichen Teils eines asymmetrischen RSA Schlüsselpaares. Bei der Spezifikation von Anwendungen sind folgende Regeln zu beachten:

(N91) Ein *publicRsaKey* MUSS ein Attribut n gemäß (N21)a besitzen.

(N92) Ein *publicRsaKey* MUSS ein Attribut e gemäß (N21)b besitzen.

9.2.4.2 Öffentlicher ELC Schlüssel, G2 (informativ)

Der Attributstyp *publicElcKey* dient der Speicherung des öffentlichen Teils eines asymmetrischen ELC Schlüsselpaares. Bei der Spezifikation von Anwendungen sind folgende Regeln zu beachten:

(N93) Ein *publicElcKey* MUSS ein Attribut *domainParameter* gemäß (N86) besitzen, die gemäß (N86)g per *OID* referenziert werden MÜSSEN.

(N94) Ein *publicElcKey* MUSS ein Attribut *P* besitzen, das einen Punkt auf der durch die Domainparameter vorgegebenen Kurve bezeichnet.

9.2.5 Transportschutz für ein Passwort (normativ)

Der Attributstyp *transportStatus* gibt an, ob ein Passwort mit einem Transportschutz versehen ist. Der Wert dieses Attributes gibt also an, ob ein VERIFY Kommando möglich ist (siehe (N828)) und welche Variante von CHANGE REFERENCE DATA anzuwenden ist. An der externen Schnittstelle des COS wird dieser Attributstyp im Rahmen des Kommandos GET PIN STATUS verwendet (siehe Tabelle 106).

(N95) Für *transportStatus* gilt folgender Wertebereich:

a. regularPassword
b. Leer–PIN_1
c. Leer–PIN_2
d. Transport–PIN_0000
e. Transport–PIN_Zufallszahl
f. Transport–PIN_abgeleitet
g. Transport–PIN_festerWert

(N96) Das COS MUSS den Wert regularPassword unterstützen.
Das COS MUSS mindestens einen Wert aus der Menge {Leer–PIN_1, Leer–PIN_2, Transport–PIN_0000} unterstützen.
Ein COS KANN weitere Werte für *transportStatus* unterstützen, auch solche, die nicht im Wertebereich von (N95) liegen.
Ein COS KANN weitere Werte für *transportStatus* ablehnen.

(N97) Wenn zur Aufhebung des Transportschutzes das Kommando CHANGE REFERENCE DATA verwendet wird (siehe (N747)c.iii), dann MUSS in Abhängigkeit des Wertes von *transportStatus* an der physikalischen Schnittstelle eine Variante gemäß Tabelle 9 verwendet werden (wenn *oldSecret* oder *newSecret* für eine Variante nicht erwähnt werden, so MUSS deren Inhalt im Rahmen von (N80) und (N81) beliebig wählbar sein):

Hinweis (24): Auch das Kommando RESET RETRY COUNTER ist in der Lage den Transportschutz aufzuheben, siehe Use Cases in Kapitel 15.6.5.1 und 15.6.5.3, sowie (N812)b.

Tabelle 9: Transportschutzkodierung

Transportschutz	Parameter für CHANGE REFERENCE DATA
regularPassword	P1=´00´
Leer–PIN_1	P1=´01´
Leer–PIN_2	P1=´00´, *oldSecret* = ´20FF FFFF FFFF FFFF´
Transport–PIN_0000	P1=´00´, *oldSecret* = ´2400 00FF FFFF FFFF´
Transport–PIN_Zufallszahl	P1=´00´
Transport–PIN_abgeleitet	P1=´00´
Transport–PIN_festerWert	P1=´00´

9.3 File (normativ)

Dieses Unterkapitel beschreibt fileorientierte Objekttypen. File wird in diesem Zusammenhang als Oberbegriff für Ordner (siehe Kapitel 9.3.1) und Datei (siehe Kapitel 9.3.2) verwendet.

9.3.1 Ordner

Ordner dienen der hierarchischen Anordnung von Objekten in einem Objektsystem. In diesem Dokument wird Ordner als Oberbegriff für
– Applikationen (Kapitel 9.3.1.1),
– Dedicated Files (Kapitel 9.3.1.2) und
– Application Dedicated Files (Kapitel 9.3.1.3)
verwendet. Gemäß der Norm [7816–4] gelten für einen Ordner folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N98) Ein Ordner MUSS genau ein Attribut vom Typ *lifeCycleStatus* (siehe Kapitel 9.1.3) besitzen.

(N99) Ein Ordner MUSS genau ein Attribut vom Typ *accessRuleList* (siehe Kapitel 9.1.4) besitzen.

(N100) Ein Ordner MUSS eine (möglicherweise leere) Liste *children* mit Kindobjekten besitzen.

a. Das COS MUSS unter Berücksichtigung der maximalen Schachtelungstiefe (siehe (N204)) Listenelemente der Objekttypen
– Dedicated File (siehe Kapitel 9.3.1.2)
– Application Dedicated File (siehe Kapitel 9.3.1.3)
– Datei (siehe Kapitel 9.3.2)
– Passwort (siehe Kapitel 9.4)
– Symmetrisches Authentisierungsobjekt (siehe Kapitel 9.5.1)
– Privates Schlüsselobjekt (siehe Kapitel 9.5.2
unterstützen.

b. Das COS KANN weitere Objekttypen als Listenelemente unterstützen.
Das COS KANN weitere Objekttypen als Listenelemente ablehnen.

Hinweis (25): In (N100)a sind Applikationen (siehe Kapitel 9.3.1.1) absichtlich nicht enthalten, vergleiche auch (N202).

Einem Ordner sind weitere, kanalspezifische Attribute zugeordnet, die typischerweise einem flüchtig (etwa im RAM) gespeicherten Kanalkontext (siehe (N299)i) zugerechnet werden. Sie werden typischerweise im Rahmen einer Selektion des Ordners verändert, aber auch durch Benutzerverifikation oder Komponentenauthentisierung.

9.3.1.1 Applikation

Eine Applikation ist ein Ordner, der nur mittels AID selektierbar ist. Dies ist der Hauptunterschied zu einem DF (siehe Kapitel 9.3.1.2).

Gemäß der Norm [7816–4] gelten für eine Applikation folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N101) Eine Applikation ist eine Erweiterung von Ordner und MUSS deshalb den Anforderungen aus Kapitel 9.3.1 genügen.

(N102) Das Attribut *applicationIdentifier* (AID) MUSS ein Oktettstring sein.

a. Die Länge von *applicationIdentifier* MUSS im Intervall [5, 16] liegen.

b. Die Oktette in *applicationIdentifier* MÜSSEN beliebig wählbar sein.

c. Ein COS KANN *applicationIdentifier* akzeptieren, die gegen diese Regeln verstoßen.
Ein COS KANN *applicationIdentifier* ablehnen, die gegen diese Regeln verstoßen.

(N103) Eine Applikation MUSS mindestens einen *applicationIdentifier* besitzen.

(N104) Eine Applikation DARF NICHT mehr als zwei *applicationIdentifier* besitzen.
Ein COS KANN für Applikationen mehr als zwei *applicationIdentifier* erlauben.

9.3.1.2 Dedicated File

Ein Dedicated File (DF) ist ein Ordner, der nur mittels File Identifier selektierbar ist. Dies ist der Hauptunterschied zu einer Applikation (siehe Kapitel 9.3.1.1).

Gemäß der Norm [7816–4] gelten für ein DF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N105) Ein DF ist eine Erweiterung von Ordner und MUSS deshalb den Anforderungen aus Kapitel 9.3.1 genügen.

(N106) Ein DF MUSS genau ein Attribut vom Typ *fileIdentifier* (siehe Kapitel 9.1.1) besitzen.

9.3.1.3 Application Dedicated File

Ein Application Dedicated File (ADF) ist ein Ordner, der sowohl mittels *applicationIdentifier* als auch *fileIdentifier* selektierbar ist. ADF wird in diesem Dokument also als „Vereinigungsmenge“ von Applikation (Kapitel 9.3.1.1) und DF (Kapitel 9.3.1.2) gesehen. Dementsprechend gelten für ein ADF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N107) Ein ADF MUSS die Eigenschaften von Applikation und von DF in sich vereinen. Deshalb MUSS es sowohl die Anforderungen aus Kapitel 9.3.1.1 als auch aus Kapitel 9.3.1.2 erfüllen.

9.3.2 Datei

Eine Datei dient in diesem Dokument als Oberbegriff für transparente EF (siehe Kapitel 9.3.2.1) und strukturierte EF (siehe Kapitel 9.3.2.2).

Gemäß der Norm [7816–4] gelten für eine Datei folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N108) Eine Datei MUSS genau ein Attribut vom Typ *fileIdentifier* (siehe Kapitel 9.1.1) besitzen.

(N109) Eine Datei MUSS eine Liste mit Elementen vom Typ *shortFileIdentifier* (siehe Kapitel 9.1.2) unterstützen. Diese Liste ist entweder

a. leer (die Datei hat keinen *shortFileIdentifier*) oder

b. enthält genau ein Element (die Datei hat genau einen *shortFileIdentifier*).

(N110) Eine Datei MUSS genau ein Attribut vom Typ *lifeCycleStatus* (siehe Kapitel 9.1.3) besitzen.

(N111) Eine Datei MUSS genau ein Attribut vom Typ *accessRuleList* (siehe Kapitel 9.1.4) besitzen.

(N112) Eine Datei MUSS ein Attribut *flagTransactionMode* vom Typ Boolean besitzen.
Ein COS KANN pro Datei ein solches Attribut individuell verwalten.
Ein COS KANN dieses Attribut für alle Dateien implizit auf True setzen.

(N113) Ein Datei MUSS ein Attribut *flagChecksum* vom Typ Boolean besitzen.
Ein COS KANN pro Datei ein solches Attribut individuell verwalten.
Ein COS KANN dieses Attribut für alle Dateien implizit auf True setzen.

Hinweis (26): Das Setzen des Flags flagTransactionMode verringert die Performance bei Schreibzugriffen.

Hinweis (27): Das Setzen des Flags flagChecksum verringert die Performance bei Schreib- und Lesezugriffen.

9.3.2.1 Transparentes Elementary File

Ein transparentes Elementary File (transparentes EF) dient der Speicherung eines Oktettstrings, wobei beliebige Teile des Oktettstrings zugreifbar sind. Der Inhalt von Oktettstrings in transparenten EF, welche im Rahmen einer Anwendungsspezifikation definiert werden, wird von einem COS lediglich gespeichert, niemals aber interpretiert oder für karteninterne Prozesse verwendet. Der Oktettstrings wird in „data units“ unterteilt (siehe [7816–4]). Die „data units“ werden über einen Offset referenziert.

Gemäß der Norm [7816–4] gelten für ein transparentes EF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N114) Ein transparentes EF ist eine Erweiterung von Datei und MUSS deshalb den Anforderungen aus Kapitel 9.3.2 genügen.

(N115) Ein transparentes EF MUSS genau ein Attribut *numberOfBytes* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [1, 32768] sein MUSS.
Ein COS KANN *numberOfBytes* ablehnen, wenn es außerhalb des Intervalls liegt.
Ein COS KANN *numberOfBytes* akzeptieren, wenn es außerhalb des Intervalls liegt.

(N116) Ein transparentes EF MUSS ein Attribut *body* vom Typ Oktettstring besitzen. Die Anzahl der Oktette in *body* ist gleich *numberOfBytes*.

(N117) Die Größe einer „data unit“ in *body* MUSS 1 Byte betragen.

(N118) Der Offset des ersten Bytes in *body* ist null. Ist *i* der Offset des *i*-ten Bytes, dann ist (*i* + 1) der Offset des nächsten Bytes.

(N119) Ein transparentes EF MUSS die folgenden Kommandos unterstützen:

a. ERASE BINARY (siehe Kapitel 15.3.1),
READ BINARY (siehe Kapitel 15.3.2),
UPDATE BINARY (siehe Kapitel 15.3.4).
Bevor diese Kommandos in der Lage sind mit dem Oktettstring zu arbeiten, ist das transparente EF zu selektieren. Dies geschieht entweder mittels SELECT Kommando (siehe Kapitel 15.2.6.13 und 15.2.6.14), oder mittels Short File Identifier, der als Parameter dem zugreifenden Kommando mitgegeben wird.

b. ACTIVATE (siehe Kapitel 15.2.1),
DEACTIVATE (siehe Kapitel 15.2.3),
DELETE (siehe Kapitel 15.2.4).
Bevor diese Kommandos in der Lage sind mit dem Oktettstring zu arbeiten, ist das transparente EF zu selektieren. Dies geschieht mittels SELECT Kommando (siehe Kapitel 15.2.6.13 und 15.2.6.14).

(N120) Ein transparentes EF KANN weitere Kommandos unterstützen.
Ein transparentes EF KANN weitere Kommandos ablehnen.

9.3.2.2 Strukturiertes Elementary File

Ein strukturiertes Elementary File (strukturiertes EF) dient der Speicherung von einer Liste von Rekords (siehe Kapitel 9.1.5). Ein Zugriff auf beliebige Listenelemente ist möglich. Der Inhalt des Oktettstrings eines Rekords in strukturierten EF, welche im Rahmen einer Anwendungsspezifikation definiert werden, wird von einem COS lediglich gespeichert, niemals aber interpretiert oder für karteninterne Prozesse verwendet.

Gemäß der Norm [7816–4] gelten für ein strukturiertes EF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N121) Ein strukturiertes EF ist eine Erweiterung von Datei und MUSS deshalb den Anforderungen aus Kapitel 9.3.2 genügen.

(N122) Ein strukturiertes EF MUSS ein Attribute *recordList* vom Typ Liste mit Elementen vom Typ Rekord besitzen. Die Anzahl der Listenelemente MUSS im Intervall [0, 254] liegen.
Ein COS KANN eine größere Anzahl von Listenelemente zulassen.
Ein COS KANN eine größere Anzahl von Listenelemente ablehnen.

(N123) Das *i*-te Element in *recordList* MUSS die Rekordnummer *i* besitzen.

(N124) Ein strukturiertes EF MUSS genau ein Attribut *maximumNumberOfRecords* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [1, 254] sein MUSS.
Ein COS KANN *maximumNumberOfRecords* akzeptieren, wenn dessen Wert außerhalb dieses Intervalls liegt.
Ein COS KANN *maximumNumberOfRecords* ablehnen, wenn dessen Wert außerhalb dieses Intervalls liegt.

(N125) Ein strukturiertes EF MUSS genau ein Attribut *maximumRecordLength* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [1, 255] sein MUSS.
Ein COS KANN *maximumRecordLength* akzeptieren, wenn dessen Wert außerhalb des Intervalls liegt.
Ein COS KANN *maximumRecordLength* ablehnen, wenn dessen Wert außerhalb des Intervalls liegt.

(N126) Ein strukturiertes EF MUSS genau ein Attribute *flagRecordLifeCycleStatus* vom Typ Boolean besitzen. Falls dieses Attribut

a. True ist, dann MÜSSEN alle Elemente in *recordList* einen *lifeCycleStatus* besitzen.
b. False ist, dann DARF KEIN Element in *recordList* einen *lifeCycleStatus* besitzen.

(N127) Ein strukturiertes EF MUSS die folgenden Kommandos unterstützen:

a. ACTIVATE RECORD (siehe Kapitel 15.4.1),
APPEND RECORD (siehe Kapitel 15.4.2),
DEACTIVATE RECORD (siehe Kapitel 15.4.3),
ERASE RECORD (siehe Kapitel 15.4.4),
READ RECORD (siehe Kapitel 15.4.5),
SEARCH RECORD (siehe Kapitel 15.4.6),
UPDATE RECORD (siehe Kapitel 15.4.7).
Bevor diese Kommandos in der Lage sind spezifikationsgemäß zu arbeiten, ist das strukturierte EF zu selektieren. Dies geschieht entweder mittels SELECT Kommando (siehe Kapitel 15.2.6.13 und 15.2.6.14), oder mittels Short File Identifier, der als Parameter dem zugreifenden Kommando mitgegeben wird.

b. ACTIVATE (siehe Kapitel 15.2.1),
DEACTIVATE (siehe Kapitel 15.2.3),
DELETE (siehe Kapitel 15.2.4).
Bevor diese Kommandos in der Lage sind mit dem Oktettstring zu arbeiten, ist das strukturierte EF zu selektieren. Dies geschieht mittels SELECT Kommando (siehe Kapitel 15.2.6.13 und 15.2.6.14).

(N128) Ein strukturiertes EF KANN weitere Kommandos unterstützen.
Ein strukturiertes EF KANN weitere Kommandos ablehnen.

9.3.2.2.1 Linear variables Elementary File

Ein linear variables Elementary File (linear variables EF) dient der Speicherung einer Liste von Elementen des Typs *record* (siehe Kapitel 9.1.5), wobei es möglich ist, dass der Oktettstring eines jeden Listenelementes eine andere Anzahl von Oktette enthält.

Gemäß der Norm [7816–4] gelten für ein linear variables EF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N129) Ein linear variables EF ist eine Erweiterung des strukturierten EF und MUSS deshalb den Anforderungen aus Kapitel 9.3.2.2 genügen.

(N130) Ein linear variables EF MUSS genau ein Attribut *numberOfBytes* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [1, 64770] sein MUSS
(64770 = 254 Rekords x 255 Oktett pro Rekord).
Ein COS KANN *numberOfBytes* akzeptieren, wenn es außerhalb des Intervalls liegt.
Ein COS KANN *numberOfBytes* ablehnen, wenn es außerhalb des Intervalls liegt.

(N131) Wird mittels APPEND RECORD der Liste ein neuer *record* hinzugefügt, so MUSS der neue *record* am Ende der Liste eingefügt werden.

9.3.2.2.2 Linear fixes Elementary File

Ein linear fixes Elementary File (linear fixes EF) dient der Speicherung einer Liste von Elementen des Typs *record* (siehe Kapitel 9.1.5), wobei der Oktettstring eines jeden Listenelementes dieselbe Anzahl von Oktette enthält.

Gemäß der Norm [7816–4] gelten für ein linear fixes EF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N132) Ein linear fixes EF ist eine Erweiterung des strukturierten EF und MUSS deshalb den Anforderungen aus Kapitel 9.3.2.2 genügen.

(N133) Jeder record in recordList MUSS maximumRecordLength Oktette besitzen.

(N134) Wird mittels APPEND RECORD der Liste ein neuer *record* hinzugefügt, so MUSS der neue *record* am Ende der Liste eingefügt werden.

9.3.2.2.3 Zyklisches Elementary File

Ein zyklisches Elementary File (zyklisches EF) dient der Speicherung einer Liste von Elementen des Typs *record* (siehe Kapitel 9.1.5), wobei der Oktettstring eines jeden Listenelementes dieselbe Anzahl von Oktette enthält.

Gemäß der Norm [7816–4] gelten für ein zyklisches EF folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N135) Ein zyklisches EF ist eine Erweiterung des strukturierten EF und MUSS deshalb den Anforderungen aus Kapitel 9.3.2.2 genügen.

(N136) Jeder record in recordList MUSS maximumRecordLength Oktette besitzen.

(N137) Wird mittels APPEND RECORD der Liste ein neuer *record* hinzugefügt, so MUSS der neue *record* am Anfang der Liste eingefügt werden.

(N138) Zyklische Elementary Files MÜSSEN für das Attribute flagRecordLifeCycleStatus den Wert False verwenden.

9.3.3 File Control Parameter

Die File Control Parameter enthalten Attribute eines Files. Im Rahmen dieses Dokumentes werden sie lediglich an der Schnittstelle Interpreter (siehe Abbildung 1) in den Antwortdaten eines SELECT Kommandos (siehe (N483)a) sichtbar. Für die Kodierung der File Control Parameter gilt:

(N139) Die File Control Parameter (FCP) MÜSSEN gemäß DER–TLV in einem DO_FCP kodiert werden, welches ein Tag = ´62´ besitzen MUSS.

(N140) ´80´: Falls das File vom Typ transparentes EF (siehe 9.3.2.1) oder linear variables EF (siehe 9.3.2.2.1) ist, genau dann MUSS DO_FCP ein DO_Size enthalten. Es gilt:

a. DO_Size MUSS ein Tag = ´80´ besitzen.
b. Das Wertfeld KANN eine beliebige Länge besitzen.
c. Wird der Oktettstring des Wertfeldes mittels OS2I(…) konvertiert, dann MUSS diese Zahl gleich *numberOfBytes* (siehe (N115), bzw. (N130)) sein.

(N141) ´82´: DO_FCP MUSS ein DO_FileDescriptor enthalten. Falls File vom Typ

a. Ordner ist, dann gilt:
   i. DO_FileDescriptor MUSS ein Tag = ´82´ besitzen
   ii. Das Wertfeld MUSS aus einem Oktett bestehen.
   iii. Das Wertfeldes MUSS in den sechs LSBit den Wert ´38´ enthalten.
b. Transparentes EF ist, dann gilt:
   i. DO_FileDescriptor MUSS ein Tag = ´82´ besitzen
   ii. Das Wertfeld MUSS aus einem Oktett bestehen.
   iii. Das Wertfeldes MUSS in den drei LSBit den Wert ´01´ enthalten.
c. Strukturiertes EF ist, dann gilt:
   i. DO_FileDescriptor MUSS ein Tag = ´82´ besitzen.
   ii. Die Länge des Wertfeldes von DO_FileDescriptor MUSS aus der Menge {5, 6} sein.
   iii. Die drei LSBit im ersten Oktett des Wertfeld MÜSSEN
      1. ´02´ sein, falls File vom Typ linear fixes EF ist.
      2. ´04´ sein, falls File vom Typ linear variables EF ist.
      3. ´06´ sein, falls File vom Typ zyklisches EF ist.
   iv. Das zweite Oktett im Wertfeld MUSS den Wert ´41´ besitzen.
   v. Das 3. und 4. Oktett MUSS gleich I2OS(*maximumRecordLength*, 2) sein.
   vi. Das fünfte Oktett MUSS bzw. das fünfte und sechste Oktett MÜSSEN so gewählt werden, dass nach eine OS2I Konvertierung das Attribut *maximumNumberOfRecords* liefert.

(N142) ´83´: Falls das File ein Attribut vom Typ *fileIdentifier* gemäß Kapitel 9.1.1 besitzt, genau dann MUSS DO_FCP ein DO_FID enthalten. Es gilt:
DO_FID = ´83 02 I2OS(*fileIdentifier*, 2)´.

(N143) ´84´: Falls das File Attribute vom Typ *applicationIdentifier* gemäß (N102) besitzt, dann MUSS DO_FCP jedes dieser Attribute enthalten. Jedes dieser Attribute MUSS als Wertfeld in einem DO_AID kodiert werden. Es gilt:
DO_AID = ´84 || I2OS(OctetLength(*applicationIdentifier*, 1) || *applicationIdentifier*´.

(N144) ´88´: Falls das File vom Typ Datei ist, genau dann MUSS DO_FCP ein DO_SFI enthalten. Falls die Datei

a. ein Attribut *shortFileIdentifier* gemäß Kapitel 9.1.2 besitzt, gilt:
DO_SFI = ´88 01 I2OS(8 *shortFileIdentifier*, 1)´.

b. kein Attribut *shortFileIdentifier* besitzt, gilt:
DO_SFI = ´88 00´.

(N145) ´8A´: Das Attribut *lifeCycleStatus* (siehe Kapitel 9.1.3) MUSS in DO_FCP enthalten sein. Es MUSS als Wertfeld in einem DO_LCS mit Tag = ´8A´ kodiert werden. Der Wert von *lifeCycleStatus* MUSS in einem Oktett gemäß [7816–4] Tabelle 13 kodiert werden. Das bedeutet

a. „Operational state (activated)“ MUSS aus der Menge {´05´, ´07´} stammen.

b. „Operational state (deactivated)” MUSS aus der Menge {´04´, ´06´} stammen.

(N146) ´8F´: Falls das File vom Typ strukturiertes EF ist und das Attribut *flagRecordLifecycleStatus* hat den Wert

a. False, dann KANN DO_FCP

i. ein DO_ProfileIdentifier = ´8F 01 00´ enthalten.

ii. kein Do mit Tag = ´8F´ enthalten.

b. True, genau dann MUSS DO_FCP ein DO_ProfileIdentifier = ´8F 01 XX´ enthalten. OS2I(´XX´) MUSS eine ungerade Zahl kleiner als 128 sein.

(N147) ´C5´: Falls das File vom Typ transparentes EF (siehe Kapitel 9.3.2.1) ist und das Konzept „logical End Of File“ unterstützt, genau dann MUSS DO_FCP ein DO_ReadSize enthalten. Es gilt:

a. DO_ReadSize MUSS ein Tag = ´C5´ besitzen.

b. Das Wertfeld KANN eine beliebige Länge besitzen.

c. Wird der Oktettstring des Wertfeldes mittels OS2I(…) konvertiert, dann MUSS diese Zahl gleich der maximal mittels READ BINARY auslesbaren Anzahl Oktette sein.

(N148) In DO_FCP KÖNNEN weitere Datenobjekte enthalten sein. Deren Kodierung und Bedeutung KANN herstellerspezifische sein.

(N149) Für die Reihenfolge der Datenobjekte in DO_FCP gilt:

a. Die Reihenfolge aller Datenobjekte ist herstellerspezifisch.

b. Falls in DO_FCP vorhanden, dann MÜSSEN die Datenobjekte DO_Size, DO_FileDescriptor, DO_FID, DO_AID, DO_SFI, DO_LCS, DO_ProfileIdentifier und DO_ReadSize vollständig in den ersten 256 Oktetten des Oktettstrings DO_FCP enthalten sein.

Hinweis (28): Das Konzept „logical End Of File“ unterscheidet zwischen einer Dateigröße, die mittels UPDATE BINARY beschreibbar ist und einer „logischen“ Dateigröße, die mittels READ BINARY auslesbar ist.

9.4 Passwort (normativ)

Ein Passwort dient der Speicherung eines Geheimnisses, das in der Regel nur einem Karteninhaber bekannt ist. Das COS wird bestimmte Dienste erst dann zulassen, wenn dieses Geheimnis im Rahmen einer Benutzerverifikation erfolgreich präsentiert wurde. Die Notwendigkeit der Benutzerverifikation lässt sich einschalten (enable) oder abschalten (disable), sofern dass COS die Kommandos ENABLE VERFICATION REQUIREMENT und DISABLE VERFICATION REQUIREMENT unterstützt.

Kom
3002

Typischerweise ist das Geheimnis aus Sicherheitsgründen änderbar. Ebenfalls aus Sicherheitsgründen wird der Dienst Benutzerverifikation mit einem bestimmten Passwort vom COS gesperrt, wenn die Benutzerverifikation mit diesem Passwort zu oft fehlschlug. Typischerweise ist es möglich, diese Sperrung aufzuheben.

Hinweis (29): In anderen Dokumenten wird in der Regel der Begriff PIN verwendet und bezeichnet dann teilweise das einem Karteninhaber bekannte 6 stellige Geheimnis („Bitte geben Sie Ihre PIN ein.“) und teilweise das Objekt in seiner Gesamtheit („Die PIN hat ist blockiert.“). In diesem Dokument wird der sprachlichen Klarheit wegen weitgehend auf den Begriff PIN verzichtet und zur Bezeichnung des Objekttyps stets der Begriff Passwort verwendet.

Bei der Spezifikation von Applikationen sind folgende Regeln einzuhalten:

(N150) Ein Passwort MUSS genau ein Attribut *pwdIdentifier* besitzen, dessen Wert eine ganz Zahl aus dem Intervall [0, 3] sein MUSS.
Ein COS KANN weitere Werte für *pwdIdentifier* unterstützen.
Ein COS KANN weitere Werte für *pwdIdentifier* ablehnen.

(N151) Ein Passwort MUSS genau ein Attribut vom Typ *accessRuleList* (siehe Kapitel 9.1.4) besitzen.

(N152) Ein Passwort MUSS genau ein Attribut *secret* vom Typ *pin* (siehe 9.1.7) besitzen.

(N153) Das Passwort MUSS genau ein Attribut *minimumLength* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [4, 12] sein MUSS.
Ein COS KANN weitere Werte für die *minimumLength* unterstützen.
Ein COS KANN weitere Werte für die *minimumLength* ablehnen.

(N154) Ein Passwort MUSS genau ein Attribut *startRetryCounter* besitzen, dessen Wert eine ganze Zahl im Intervall [1, 15] sein MUSS.
Ein COS KANN weitere Werte für *startRetryCounter* unterstützen.
Ein COS KANN weitere Werte für *startRetryCounter* ablehnen.

(N155) Ein Passwort MUSS genau ein Attribut *retryCounter* besitzen, dessen Wert eine ganze Zahl im Intervall [0, 15] sein MUSS.
Ein COS KANN weitere Werte für *retryCounter* unterstützen.
Ein COS KANN weitere Werte für *retryCounter* ablehnen.

(N156) Ein Passwort MUSS genau ein Attribut *transportStatus* gemäß Kapitel 9.2.5 besitzen.

(N157) Ein Passwort MUSS genau ein Attribut *flagEnabled* vom Typ Boolean besitzen, wenn das COS die Kommandos DISABLE und ENABLE VERIFCATION REQUI-

REMENT unterstützt. Es wird im Rahmen der Zugriffsregelauswertung verwendet (siehe (N222)a.ii).

(N158) Ein Passwort MUSS genau ein Attribut *startSsecList* besitzen

a. Die Liste MUSS mindestens ein Element und DARF NICHT mehr als vier Elemente enthalten.
Ein COS KANN Listen mit mehr Elementen unterstützen.
Ein COS KANN Listen mit mehr Elementen ablehnen.

b. Jedes Listenelement MUSS

i. genau einen *seIdentifier* gemäß (N79) und

ii. genau eine ganze Zahl *startSsec* enthalten. *startSsec* MUSS aus dem Intervall [1, 250] sein oder den Wert "unendlich" repräsentieren.
Ein COS KANN weitere Werte für *startSsec* unterstützen.
Ein COS KANN weitere Werte für *startSsec* ablehnen.

(N159) Ein Passwort MUSS genau ein Attribut *PUK* vom Typ *pin* (siehe 9.1.7) besitzen, welches ein Geheimnis repräsentiert, welches das Zurücksetzten eines abgelaufenen *retryCounter* ermöglicht.

(N160) Ein Passwort MUSS ein Attribut *pukUsage* besitzen, dessen Wert eine ganze Zahl aus dem Intervall [0, 15] sein MUSS.
Ein COS KANN weitere Werte für *pukUsage* unterstützen.
Ein COS KANN weitere Werte für *pukUsage* ablehnen.

(N161) Die Ziffernfolgen *secret* und *PUK* MÜSSEN an der Schnittstelle Interpreter (siehe Abbildung 1) stets als Format–2–PIN Block (siehe (N81)) übertragen werden.
Ein COS KANN weitere Übertragungsformate unterstützen.
Ein COS KANN weitere Übertragungsformate ablehnen.

(N162) Ein Passwort MUSS die folgenden Kommandos unterstützen:
– CHANGE REFERENCE DATA (siehe Kapitel 15.6.1),
– GET PIN STATUS (siehe Kapitel 15.6.4),
– RESET RETRY COUNTER (siehe Kapitel 15.6.5),
– VERIFY (siehe Kapitel 15.6.6).
Bevor diese Kommandos in der Lage sind mit dem Passwort zu arbeiten, ist das Passwort zu selektieren. Dies geschieht mittels eines Passwort Identifiers, der als Parameter dem zugreifenden Kommando mitgegeben wird.

(N163) Ein Passwort KANN weitere Kommandos unterstützen.
Ein Passwort KANN weitere Kommandos ablehnen.

Einem Passwort ist ein weiteres, kanalspezifisches Attribut *securityStatusEvaluationCounter* (siehe (N299)k) zugeordnet, das typischerweise einem flüchtig (etwa im RAM) gespeicherten Sicherheitszustand (siehe Kapitel 9.8) zugerechnet wird. Es wird im Rahmen von Kommandos zur Benutzerverifikation (siehe Kapitel 15.6.6) geändert.

Kom
3002

Kom
3002

9.5 Schlüsselobjekt (normativ)

Dieses Unterkapitel beschreibt Schlüsselobjekte, die im Rahmen kryptographischer Operationen zum Einsatz kommen. Der Terminus Schlüsselobjekt dient in diesem Dokument als Oberbegriff für symmetrische, private und öffentliche Schlüsselobjekte.

Symmetrische Schlüssel werden in diesem Dokument zu folgenden Zwecken eingesetzt:

1. Mit persistent gespeichertem Geheimnis (Schlüssel) zur einseitigen Authentisierung (siehe Kapitel 16.1.1).
2. Mit persistent gespeichertem Geheimnis (Schlüssel) zur gegenseitigen Authentisierung bei gleichzeitiger Aushandlung von Sessionkeys (siehe Kapitel 16.4.1).
3. Als Sessionkeys zur Sicherstellung einer vertraulichen Kommunikation.
4. Als Sessionkeys zur Sicherstellung einer integren und authentischen Kommunikation.

Private Schlüssel werden in diesem Dokument zu folgenden Zwecken eingesetzt:

5. Berechnung elektronischer Signaturen (siehe Kapitel 15.8.1)
6. Entschlüsselung von Daten (siehe Kapitel 15.8.2)
7. Nachweis der Authentizität dieser Karte (siehe Kapitel 16.2)
8. Transportsicherung von Sessionkey Material (siehe (N844)g)

Öffentliche Schlüssel werden in diesem Dokument zu folgenden Zwecken eingesetzt:

9. Prüfen elektronischer Signaturen beim Import von Zertifikaten (siehe (N959))
10. Prüfen von Signaturen im Rahmen von Rollenauthentisierungen (siehe (N844))
11. Transportsicherung von Sessionkey Material (siehe (N869)f)

9.5.1 Symmetrisches Authentisierungsobjekt

Ein symmetrisches Authentisierungsobjekt wird im Rahmen von Authentisierungen gemäß Kapitel 16.1.1 und 16.4.1 eingesetzt. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N164) Ein symmetrisches Authentisierungsobjekt MUSS ein Attribut *keyIdentifier* besitzen, dessen Wert eine ganz Zahl aus dem Intervall [2, 28] sein MUSS.
Ein COS KANN weitere Werte für *keyIdentifier* unterstützen.
Ein COS KANN weitere Werte für *keyIdentifier* ablehnen.

(N165) Ein symmetrisches Authentisierungsobjekt MUSS genau ein Attribut vom Typ *accessRuleList* (siehe Kapitel 9.1.4) besitzen.

(N166) Ein symmetrisches Authentisierungsobjekt MUSS genau ein Attribut *encKey* gemäß Kapitel 9.2.1 besitzen.

(N167) Ein symmetrisches Authentisierungsobjekt MUSS genau ein Attribut *macKey* gemäß Kapitel 9.2.1 besitzen.

(N168) Ein symmetrisches Authentisierungsobjekt MUSS genau ein Attribut vom Typ *algorithmIdentifier* besitzen, welches angibt, für welchen Zweck es verwendbar ist.

a. Für symmetrische Authentisierungsobjekte in Generation 1 MUSS der Wert von *algorithmIdentifier* Element der Menge {
desRoleAuthentication, (siehe Kapitel 15.7.4.1 und (N869)b)
desRoleCheck, (siehe Kapitel 15.7.1.1 und (N844)c)
desSessionkey4SM (siehe Kapitel 15.7.1.2 und (N844)d)
} sein (siehe Tabelle 166).

b. Ein COS KANN weitere Werte für *algorithmIdentifier* unterstützen.
Ein COS KANN weitere Werte für *algorithmIdentifier* ablehnen.

(N169) Ein symmetrisches Authentisierungsobjekt MUSS die folgenden Kommandos unterstützen:
– EXTERNAL AUTHENTICATE (siehe Kapitel 15.7.1.1 und 15.9.6.4),
– MUTUAL AUTHENTICATE (siehe Kapitel 15.7.1.2 und 15.9.6.6),
Bevor dieses Kommando in der Lage ist mit dem symmetrisches Authentisierungsobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels der in Kapitel 15.9.6 beschriebenen Use Cases.

(N170) Ein symmetrisches Authentisierungsobjekt KANN weitere Kommandos unterstützen.
Ein symmetrisches Authentisierungsobjekt KANN weitere Kommandos ablehnen.

9.5.2 Privates Schlüsselobjekt

Ein privates Schlüsselobjekt wird als Oberbegriff für
– private Authentisierungsschlüssel (Kapitel 9.5.2.1)
– private Entschlüsselungsobjekte (Kapitel 9.5.2.2)
– private Signierobjekte (Kapitel 9.5.2.3)
verwendet. Für private Schlüsselobjekte gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N171) Ein privates Schlüsselobjekt MUSS ein Attribut *keyIdentifier* besitzen, dessen Wert eine ganz Zahl aus dem Intervall [2, 28] sein MUSS.
Ein COS KANN weitere Werte für *keyIdentifier* unterstützen.
Ein COS KANN weitere Werte für *keyIdentifier* ablehnen.

(N172) Ein privates Schlüsselobjekt MUSS genau ein Attribut vom Typ *accessRuleList* (siehe Kapitel 9.1.4) besitzen.

(N173) Ein privates Schlüsselobjekt MUSS genau ein Attribut *privateKey* gemäß Kapitel 9.2.3 besitzen.

(N174) Ein privates Schlüsselobjekt MUSS genau ein Attribut *listAlgorithmIdentifier* mit Elementen vom Typ *algorithmIdentifier* besitzen, welche angeben, für welche Zweck es verwendbar ist. Für die Länge der Liste gilt:

a. Die Liste MUSS mindest ein Element enthalten.

b. Die Liste MUSS fähig sein sechs Elemente aufzunehmen.

c. Das COS KANN Listen mit mehr als drei Elementen unterstützen.
Das COS KANN Listen mit mehr als drei Elementen ablehnen.

9.5.2.1 Privates Authentisierungsobjekt

Ein privates Authentisierungsobjekt wird zur Authentisierung einer Karte gegenüber einer anderen technischen Komponente eingesetzt. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N175) Ein privates Authentisierungsobjekt ist eine Erweiterung von privatem Schlüsselobjekt und MUSS deshalb den Anforderungen aus Kapitel 9.5.2 genügen.

(N176) Für private Authentisierungsobjekte in Generation 1 MUSS der Wert von *algorithmIdentifier* Element der Menge {
rsaClientAuthentication, (siehe Kapitel 15.7.4.1 und (N869)d)
rsaRoleAuthentication, (siehe Kapitel 15.7.4.1 und (N869)e)
rsaSessionkey4SM (siehe Kapitel 15.7.4.1 und (N869)f)
} sein (siehe Tabelle 166).
Ein COS KANN weitere Werte für *algorithmIdentifier* unterstützen.
Ein COS KANN weitere Werte für *algorithmIdentifier* ablehnen.

(N177) Ein privates Authentisierungsobjekt MUSS die folgenden Kommandos unterstützen:
– EXTERNAL AUTHENTICATE (siehe (N844)g und Kapitel 15.9.6.3)
– INTERNAL AUTHENTICATE (siehe Kapitel 15.7.4.1 und 15.9.6.3),
– PSO Compute Digital Signature (siehe (N886)c und Kapitel 15.9.6.7)
Bevor dieses Kommando in der Lage ist mit dem privaten Authentisierungsobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels der in Kapitel 15.9.6 beschriebenen Use Cases.
Ein privates Authentisierungsobjekt KANN weitere Kommandos unterstützen.
Ein privates Authentisierungsobjekt KANN weitere Kommandos ablehnen.

9.5.2.2 Privates Entschlüsselungsobjekt

Ein privates Entschlüsselungsobjekt wird eingesetzt zur Entschlüsselung von Daten. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N178) Ein privates Entschlüsselungsobjekt ist eine Erweiterung von privatem Schlüsselobjekt und MUSS deshalb den Anforderungen aus Kapitel 9.5.2 genügen.

(N179) Für private Entschlüsselungsobjekte in Generation 1 MUSS der Wert von *algorithmIdentifier* Element der Menge {
rsaDecipherPKCS1_V1_5, (15.8.2.1, (N903)a und 15.8.5.1, (N942)a)
rsaDecipherOaep, (15.8.2.1, (N903)b und 15.8.5.1, (N942)b)
} sein (siehe Tabelle 168).
Ein COS KANN weitere Werte für *algorithmIdentifier* unterstützen.
Ein COS KANN weitere Werte für *algorithmIdentifier* ablehnen.

(N180) Ein privates Entschlüsselungsobjekt MUSS die folgenden Kommandos unterstützen:
– PSO Decipher (siehe Kapitel 15.8.2),
– PSO Transcipher (siehe Kapitel 15.8.5)
Bevor dieses Kommando in der Lage ist mit dem privaten Entschlüsselungsobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels des in Kapitel 15.9.6.9 beschriebenen Use Cases.
Ein privates Entschlüsselungsobjekt KANN weitere Kommandos unterstützen.
Ein privates Entschlüsselungsobjekt KANN weitere Kommandos ablehnen.

9.5.2.3 Privates Signierobjekt

Ein privates Signierobjekt wird eingesetzt zur Berechnung einer elektronischen Signatur. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N181) Ein privates Signierobjekt ist eine Erweiterung von privatem Schlüsselobjekt und MUSS deshalb den Anforderungen aus Kapitel 9.5.2 genügen.

(N182) Ein privates Signierobjekt MUSS ein Attribut *keyAvailable* vom Typ Boolean unterstützen. Auf dieses Attribut wird im Rahmen der Kommandos
– GENERATE ASYMMETRIC KEY PAIR (siehe 15.9.2.5) und
– PSO Compute Digital Signature (siehe (N885))
zugegriffen.
Der Wert True zeigt an, dass das Attribut *privateKey* verwendbar ist.
Der Wert False zeigt an, dass das Attribut *privateKey* nicht verwendbar ist.

(N183) Für private Signierobjekte in Generation 1 MUSS der Wert von *algorithmIdentifier* Element der Menge {
sign9796_2_DS2, (siehe Kapitel 15.8.1.2 und (N886)a)
signPKCS_V1_5, (siehe Kapitel 15.8.1.1 und (N886)b)
signPSS, (siehe Kapitel 15.8.1.1 und (N886)c)
} sein (siehe Tabelle 169).
Ein COS KANN weitere Werte für *algorithmIdentifier* unterstützen.
Ein COS KANN weitere Werte für *algorithmIdentifier* ablehnen.

(N184) Ein privates Signierobjekt MUSS die folgenden Kommandos unterstützen:
– GENERATE ASYMMETRIC KEY PAIR (siehe Kapitel 15.9.2)
– PSO Compute Digital Signature (siehe Kapitel 15.8.1),
Bevor dieses Kommando in der Lage ist mit dem privaten Signierobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels des in Kapitel 15.9.6.7 beschriebenen Use Cases.
Ein privates Signierobjekt KANN weitere Kommandos unterstützen.
Ein privates Signierobjekt KANN weitere Kommandos ablehnen.

9.5.3 Öffentliches Schlüsselobjekt

Ein öffentliches Schlüsselobjekt wird als Oberbegriff für
– öffentliche Signaturprüfobjekte (Kapitel 9.5.3.1)
– öffentliche Authentisierungsobjekte (Kapitel 9.5.3.2)
verwendet. Für öffentliche Schlüsselobjekte gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N185) Ein öffentliches Schlüsselobjekt MUSS ein Attribut *keyIdentifier* besitzen.

(N186) Ein öffentliches Schlüsselobjekt MUSS genau ein Attribut *publicKey* gemäß Kapitel 9.2.4 besitzen.

(N187) Ein öffentliches Schlüsselobjekt MUSS genau ein Attribut vom Typ *oid* besitzen, welches angibt, für welche Zwecke es verwendet werden darf.

(N188) Ein COS KANN Zugriffsregeln für öffentliche Schlüsselobjekte unterstützen. Diese MÜSSEN für alle Zugriffsarten, welche das Schlüsselobjekt unterstützt als Zugriffsbedingung ALWAYS verwenden.

(N189) Ein COS KANN Zugriffsregeln für öffentliche Schlüsselobjekte ablehnen.

9.5.3.1 Öffentliches Signaturprüfobjekt

Ein öffentliches Signaturprüfobjekt wird eingesetzt zur Prüfung von Signaturen in einem CV–Zertifikat. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N190) Ein öffentliches Signaturprüfobjekt ist eine Erweiterung von öffentlichem Schlüsselobjekt und MUSS deshalb den Anforderungen aus Kapitel 9.5.3 genügen.

(N191) Als Wert des Attributes *keyIdentifier* MUSS ein beliebiger Oktettstring der Länge acht Oktett möglich sein.
Ein COS KANN weitere Längen für *keyIdentifier* unterstützen.
Ein COS KANN weitere Längen für *keyIdentifier* ablehnen.

(N192) Für öffentliche Signaturprüfobjekte in Generation 1 MUSS der Wert von *oid* Element der Menge {
sigS_ISO9796–2Withrsa_sha256, (siehe Tabelle 6)
} sein.
Ein COS KANN weitere Werte für *oid* unterstützen.
Ein COS KANN weitere Werte für *oid* ablehnen.

(N193) Ein öffentliches Signaturprüfobjekt MUSS die folgenden Kommandos unterstützen:
– PSO Verify Certificate (siehe Kapitel 15.8.6),
Bevor dieses Kommando in der Lage ist mit dem öffentlichen Signaturprüfobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels des in Kapitel 15.9.6.8 beschriebenen Use Cases.
Ein öffentliches Signaturprüfobjekt KANN weitere Kommandos unterstützen.
Ein öffentliches Signaturprüfobjekt KANN weitere Kommandos ablehnen.

9.5.3.2 Öffentliches Authentisierungsobjekt

Ein öffentliches Authentisierungsobjekt wird zur Authentisierung einer anderen technischen Komponente gegenüber dieser Komponente eingesetzt. Für dieses Schlüsselobjekt gelten folgende Regeln, die bei der Spezifikation einer Anwendung einzuhalten sind:

(N194) Ein öffentliches Authentisierungsobjekt ist eine Erweiterung von öffentlichem Schlüsselobjekt und MUSS deshalb den Anforderungen aus Kapitel 9.5.3 genügen.

(N195) Als Wert des Attributes *keyIdentifier* MUSS ein beliebiger Oktettstring der Länge zwölf möglich sein.
Ein COS KANN weitere Werte für *keyIdentifier* unterstützen.
Ein COS KANN weitere Werte für *keyIdentifier* ablehnen.

(N196) Für öffentliche Authentisierungsobjekte in Generation 1 MUSS der Wert von *oid* Element der Menge {
authS_ISO9796–2Withrsa_sha256_mutual, (siehe Tabelle 6)
} sein.
Ein COS KANN weitere Werte für *oid* unterstützen.
Ein COS KANN weitere Werte für *oid* ablehnen.

(N197) Falls *publicKey* vom Typ

a. *publicRsaKey* ist, dann MUSS ein öffentliches Authentisierungsobjekt genau ein Attribut vom Typ *CHA* gemäß Kapitel 8.1.1.4 besitzen.

b. *publicElcKey* ist, dann … (ab Generation 2 relevant).

(N198) Ein öffentliches Authentisierungsobjekt MUSS die folgenden Kommandos unterstützen:
– EXTERNAL AUTHENTICATE (siehe Kapitel 15.7.1),
– INTERNAL AUTHENTICATE (siehe (N869)f)
Bevor dieses Kommando in der Lage ist mit dem öffentlichen Authentisierungsobjekt zu arbeiten, ist es zu selektieren. Dies geschieht mittels des in Kapitel 15.9.6.5 beschriebenen Use Cases.
Ein öffentliches Authentisierungsobjekt KANN weitere Kommandos unterstützen.

Ein öffentliches Authentisierungsobjekt KANN weitere Kommandos ablehnen.

9.6 Datenobjekte (informativ)

Auf Datenobjekte wird im Rahmen der [7816–4] Kommandos GET DATA und PUT DATA zugegriffen. Da diese Kommandos für diese Version der Spezifikation nicht verpflichtend sind, werden sie hier nicht weiter behandelt.

9.7 Security Environment (informativ)

Zum Begriff „Security Environment“ gemäß [7816–4] Kapitel 6.3.3 existiert eine große Bandbereite an Interpretationen. Deshalb wird versucht auf diesen problematischen Begriff im normativen Bereich dieses Dokumentes möglichst zu verzichten. Aus demselben Grund wird in diesem informativen, das heißt nicht–normativen, Kapitel nur das weitgehend unstrittige Prinzip der Security Environments kurz skizziert.

Security Environments eröffnen die Möglichkeit gewisse Attribute von Objekten in Abhängigkeit von der Einsatzumgebung mit konkreten Werten zu versehen. In diesem Dokument wird davon etwa beim Attributstyp *accessRuleList* (Kapitel 9.1.4) Gebrauch gemacht. Damit ergeben sich beim Design von Anwendungen Gestaltungsmöglichkeiten.

Beispiel: Eine Karte werde in zwei verschiedenen Umgebungen eingesetzt und in beiden sind sowohl vertrauliche Daten (EF.Verordnung) als auch allgemein bekannte Daten (EF.X509_Zertifikat) auslesbar (READ BINARY).

1. In der Umgebung_1 werden Lauschangriffe auf die physikalische Schnittstelle (siehe Abbildung 1) befürchtet. Zur Abwehr derartiger Angriffe wird die Zugriffsregel so gewählt, dass sensitive Daten nur vertraulich und integer übertragbar sind (siehe (N226)c). Für allgemein bekannte Daten wird ein derartiger Schutz nicht verlangt.

2. In der Umgebung_2 existieren keine Angriffe auf die physikalische Schnittstelle (siehe Abbildung 1). Deshalb wird aus diversen Gründen die Zugriffsregel so gewählt, dass alle Daten ungeschützt übertragen werden.

Zu jeder Einsatzumgebung existiert also ein Satz Zugriffsregeln:

1. Umgebung_1:

a. EF.Verordnung — READ BINARY in einer Apotheke mit Secure Messaging

b. EF.X509_Zertifikat — READ BINARY immer erlaubt

2. Umgebung_2:

a. EF.Verordnung — READ BINARY in einer Apotheke ohne Verschlüsselung

b. EF.X509_Zertifikat — READ BINARY immer erlaubt

Welcher Satz von Zugriffsregeln gilt, wird also durch die Einsatzumgebung bestimmt. Eine konkrete Einsatzumgebung (ein konkretes Security Environment) wird durch den Use Case in Kapitel 15.9.6.1 ausgewählt.

Auf der anderen Seite werden Security Environments verwendet, um Schlüssel und andere Parameter für kryptographische Operationen auszuwählen. Die Vorgehensweise ist hier anders als bei Passwörtern. Welches Passwort von einem Kommando aus Kapitel 15.6 betroffen ist, bestimmt ein Parameter dieses Kommandos.

Die Vorgehensweise ist für Schlüssel eher vergleichbar mit Dateien. Welche Datei von einem Kommando betroffen ist, bestimmt das Attribut *currentEF*, das mittels SELECT Kommando vor Ausführung dieses Kommandos passend einzustellen ist. Weil die dateiorientierten Kommandos lediglich in der Lage sind stets nur mit einer Datei pro Kommando zu arbeiten, ist ein einziges Attribut *currentEF* pro Ordner völlig ausreichend (siehe (N300)b).

Bei einigen kryptographischen Kommandos hingegen ist es möglich, dass bei der Bearbeitung mehrere kryptographische Objekte beteiligt sind wie etwa

- Sessionkeys für Vertraulichkeit
- Sessionkeys für Authentizität
- Use Case spezifischer Schlüssel, etwa PSO Compute Digital Signature

Statt nun jedem derartigen Kommando analog zu den Passwort Kommandos alle Schlüsselparameter mitzuliefern, ist in [7816–4] entschieden worden, diese Parameter zuvor mittels MSE Kommando (siehe Kapitel 15.9.6) auszuwählen. Derartige Selektionen wirken sich im Datenmodell dieses Dokumentes auf das Attribut *keyReferenceList* (siehe (N299)c) aus.

Hinweis (30): Die normativen Regeln dieses Dokumentes sind so gewählt, dass es für eine performante Implementierung zulässig ist, die kanalspezifischen Attribute (siehe Kapitel 13) im RAM zu speichern. Die genannten Attribute wurden lediglich der einfacheren Darstellung wie gezeigt zugeordnet.

Hinweis (31): Die normativen Regeln dieses Dokumentes sind so gewählt, dass es für eine speicherplatzoptimale Implementierung zulässig ist, die kanalspezifischen Ordnerattribute in (N299)i nur für currentFolder und allen seinen Vorfahren inklusive root zu speichern, anstatt für alle Ordner innerhalb des Objektsystems. Die genannten Attribute wurden lediglich der einfacheren Darstellung wegen jedem Ordner zugeordnet.

9.8 Sicherheitsstatus (informativ)

Zum Begriff „Sicherheitsstatus“ gemäß [7816–4] Kapitel 5.4.1 existiert eine große Bandbereite an Interpretationen. Deshalb wird versucht auf diesen problematischen Begriff im normativen Bereich dieses Dokumentes möglichst zu verzichten. Aus demselben Grund wird in diesem informativen, das heißt nicht–normativen, Kapitel nur das weitgehend unstrittige Prinzip des Sicherheitsstatus kurz skizziert.

Gemäß [7816–4] zählen zum Sicherheitsstatus

1. Globale Sicherheitsstatus, welche durch Benutzerverifikation oder Komponentenauthentikation mit Objekten in Root modifizierbar sind.
2. Applikationsspezifischer Sicherheitsstatus, welche mit Benutzerverifikationen oder Komponentenauthentisierungen mit Objekten in beliebigen Ordnern modifizierbar sind.
3. File spezifischer Status, welcher in diesem Dokument unberücksichtigt bleibt.
4. Kommando spezifischer Status, welchem bei der Auswertung von Zugriffsregeln Rechnung getragen wird.

In diesem Dokument wird bezüglich der globalen und applikationsspezifischen Status für alle Ordner ein Sicherheitsstatus angenommen und für die

a. Benutzerverifikation durch (N299)i, (N299)j und (N299)k Rechnung getragen, wobei dort auch die kleinste Anzahl der zu unterstützenden Sicherheitsstatus für Benutzerverifikation festgelegt wird.
b. Komponentenauthentisierung durch (N299)e, (N299)f und (N299)h Rechnung getragen, wobei dort auch die kleinste Anzahl der zu unterstützenden Sicherheitsstatus für Komponentenauthentisierung festgelegt wird.

Kom
3002

Hinweis (32): Die normativen Regeln dieses Dokumentes sind so gewählt, dass es für eine performante Implementierung zulässig ist, die Attribute globalSecurityList, dfSpecificSecurityList, globalPasswordList und dfSpecificPasswordList im RAM zu speichern. Die genannten Attribute wurden lediglich der einfacheren Darstellung wegen wie dargestellt zugeordnet.

10 Objektsystem (normativ)

Eine Chipkarte wird in diesem Dokument als *sicherer Datenspeicher* betrachtet, wobei die Betonung auf beiden Wörtern liegt.

- Datenspeicher: Eine Chipkarte speichert beliebige Informationen ganz analog zur Festplatte eines Computers in Dateien (siehe Kapitel 9.3.2).
- Sicherheit: Der Zugriff auf die in Dateien gespeicherten Informationen wird durch Regeln festgelegt. Die Einhaltung der Zugriffsregeln wird durch das Betriebssystem gewährleistet. Typischerweise enthalten Regeln Zugriffsbeschränkungen dergestalt, dass erst nach erfolgreicher Benutzerverifikation oder Komponentenauthentisierung ein Zugriff gestattet ist. Darüber hinaus wird vielfach zusätzlich ein vertraulicher und authentischer Datenaustausch erzwungen.

Typischerweise werden Dateien und damit Informationen hierarchisch strukturiert.

Die Regeln in diesem Dokument sind so aufgebaut, dass sich ein hierarchisches System mit mindestens vier Ordnerebenen aufbauen lässt (*root* enthält DF2, DF2 enthält DF3, DF3 enthält DF4, DF4 enthält keinen weiteren Ordner).

Private und symmetrische Schlüssel sowie Passwörter lassen sich per „backtracking“ suchen. Das bedeutet, sie werden zunächst im aktuellen Verzeichnis gesucht. Falls die Suche dort erfolglos ist, wird rekursiv in der nächsthöheren Ebene.

Eine DF–spezifische Suche bezieht *root* nie mit ein.

Öffentliche Schlüssel werden als zentral gespeicherte Objekte betrachtet. In der Regel gehören solche Schlüssel einer externen Komponente und werden mittels eines Zertifikates importiert (siehe Kapitel 15.8.6).

10.1 Aufbau und Strukturtiefe

Dieses Kapitel legt die hierarchische Struktur fest, so wie sie an der Schnittstelle gesehen wird. Wie die Information und die Struktur chipkartenintern gespeichert werden, ist nicht Gegenstand dieses Dokumentes. Zudem wird hier nicht der sonst übliche Terminus „Filesystem“ verwendet sondern „Objektsystem“, weil neben Files auch andere Objekttypen, wie Passwörter und Schlüssel, als eigenständige Artefakte betrachtet werden.

Bei der Spezifikation von Anwendungen sind folgende Regeln einzuhalten:

(N199) Das COS MUSS ein hierarchisches Objektsystem mit mehreren Ebenen unterstützen.

a. Das Objektsystem MUSS ein Attribut *root* mit folgenden Eigenschaften besitzen:

i. Das Attribut *root* MUSS vom Typ Applikation (siehe Kapitel 9.3.1.1) sein.

ii. Die Zugriffbedingung für die Zugriffsart DELETE (siehe Kapitel 15.2.4) MUSS NEVER (siehe (N221)) sein.

b. Das Objektsystem MUSS ein Attribut *answerToReset* vom Typ Oktettstring besitzen. Die zulässigen Werte für dieses Attribut sind in Kapitel 12.2.1 beschrieben.

c. Das Objektsystem MUSS ein Attribut *iccsn8* vom Typ Oktettstring besitzen. Für *iccsn8* MÜSSEN beliebige Werte mit acht Oktette möglich sein.

d. Das Objektsystem MUSS ein Attribut *persistentPublicKeyList* unterstützen.

   i. Das COS MUSS eine beliebige Anzahl Listenelemente unterstützen. Eine Beschränkung ist lediglich durch den verfügbaren persistenten Speicherplatz gegeben.

   ii. Als Listenelemente MUSS der Objekttyp „Öffentliches Signaturprüfobjekt" (siehe Kapitel 9.5.3.1) unterstützt werden in Verbindung mit einer Referenz zu einem Ordner, dem dieses Schlüsselobjekt zugeordnet ist (siehe (N216) und (N959)a.ix).
   Ein COS KANN weitere öffentliche Schlüsselobjekttypen unterstützen.
   Ein COS KANN weitere öffentliche Schlüsselobjekttypen ablehnen.

   iii. Elemente der Liste MÜSSEN persistent gespeichert werden.

e. Das Objektsystem MUSS ein Attribut *publicKeyList* unterstützen.

   i. Das COS MUSS eine Listenlänge von eins unterstützen.
   Das COS KANN eine größere Listenlänge unterstützen.
   Das COS KANN eine größere Listenlänge ablehnen.

   ii. Als Listenelemente MUSS der Objekttyp „Öffentliches Schlüsselobjekt" (siehe Kapitel 9.5.3) unterstützt werden in Verbindung mit einer Referenz zu einem Ordner, dem dieses Schlüsselobjekt zugeordnet ist (siehe (N216) und (N959)a.ix).

   iii. Elemente der Liste KÖNNEN persistent gespeichert werden.
   Elemente der Liste KÖNNEN volatil gespeichert werden.

f. Ab Generation 2 gilt: Das Objektsystem MUSS ein Attribut *domainParamaterList* unterstützen.

   i. Das COS MUSS eine beliebige Anzahl Listenelemente unterstützen. Eine Beschränkung ist lediglich durch den verfügbaren persistenten Speicherplatz gegeben.

   ii. Als Listenelemente MUSS … (für Generation 2 vervollständigen)

   iii. Änderungen an der Liste, Nachladen, Löschen…, das Ganze dann mit AC gesichert, … (für Generation 2 vervollständigen)

(N200) Der Ordner *root* wird der Ebene_0 zugeordnet. Ebene_0 wird als „höchste Ebene im Objektsystem" bezeichnet.

(N201) Wenn ein Ordner der Ebene_i zugeordnet ist, dann werden alle Elemente der Liste *children* dieses Ordners der Ebene_(i + 1) zugeordnet.
Ebene_i ist die nächsthöhere Ebene zu Ebene_(i + 1).
Ebene_(i + 1) ist die nächsttiefere Ebene zu Ebene_i.

(N202) Gemäß Kapitel 9.3.1.1 besitzen Applikationen kein Attribut *fileIdentifier*. Damit lassen sie sich nicht mittels SELECT Kommando und Parameter *selectionMode* = ´01´ selektieren. Eine Applikation, die nicht *root* ist,

a. MUSS der Ebene_1 zugeordnet werden,

b. KANN als Elemente der Liste *children* gemäß (N100) von *root* geführt werden,

c. KANN anderweitig in der Karte gespeichert sein.

(N203) Ordner, die der Ebene_0, Ebene_1, Ebene_2 oder Ebene_3 zugeordnet sind, MÜSSEN im Attribut *children* als Listenelement die Objekttypen Dedicated File (Kapitel 9.3.1.2) und Application Dedicated File (Kapitel 9.3.1.3) zulassen.

(N204) Ordner, die der Ebene_4 zugeordnet sind,

a. KÖNNEN im Attribut *children* Ordner zulassen.

b. KÖNNEN im Attribut *children* Ordner ablehnen.

Dem Objektsystem sind weitere, kanalspezifische Attribute zugeordnet, die typischerweise einem flüchtig (im RAM) gespeicherten Kanalkontext (siehe (N299)) zugerechnet werden. Sie werden typischerweise im Rahmen einer Selektion des Ordners verändert.

Gemäß Kapitel 9.3.1, 9.3.2 und 9.1.5 besitzen die Objekttypen Ordner, Datei und Rekord ein Attribut *lifeCycleStatus*. Gemäß (N200) und (N201) lassen sich die Instanzen aller Objekttypen Ebenen zuordnen, wobei Rekords hier als eine nächsttiefere Ebene zu der Datei aufgefasst werden, in der sie enthalten sind. Es wird nun unterschieden zwischen dem physikalischen und dem logischen Wert des *lifeCycleStatus*:

(N205) Als physikalischer Wert des *lifeCycleStatus* wird der Wert bezeichnet, den das entsprechende Attribut eines Objektes besitzt. Im Rahmen dieses Dokumentes besitzen nur Ordner, Dateien und Rekords ein derartiges Attribut.

(N206) Als logischer Wert des *lifeCycleStatus* wird der Wert bezeichnet, der sich aus der Betrachtung des physikalischen Wertes des *lifeCycleStatus* eines Objektes sowie der logischen Werte des *lifeCycleStatus* in allen höheren Ebenen ergibt. Dabei gilt folgende Rekursion:

a. Für das Objekt *root* MUSS der logische Wert und der physikalische Wert des *lifeCycleStatus* identisch sein.

b. Für ein Objekt mit einem Attribut *lifeCycleStatus*, welches von *root* verschieden ist, MUSS gelten: Wenn der logische Wert des Objektes in der nächsthöheren Ebene

i. „Operational state (activated)“ ist, genau dann MUSS für dieses Objekt der logische Wert und der physikalische Wert des *lifeCycleStatus* identische sein.

ii. „Operational state (deactivated)” ist, genau dann MUSS für dieses Objektes der logische Wert des *lifeCycleStatus* identisch sein zum logischen Wert der nächsthöheren Ebene.

c. Für ein Objekt, welches selbst kein Attribute *lifeCycleStatus* besitzt (im Rahmen dieses Dokumentes sind das Passwortobjekte und Schlüsselobjekte), MUSS der logische Wert von *lifeCycleStatus* identisch sein zum logischen Wert der nächsthöheren Ebene.

10.2 Objektsuche

Gemäß Kapitel 10.1 ist an der Schnittstelle zur Karte ein hierarchisches Objektsystem sichtbar. Dieses Kapitel legt fest, wie ein Objekt gesucht und nach welchen Regeln es gefunden wird.

10.2.1 Filesuche

Eine Suche nach Files, das heißt Ordnern und Dateien, findet lediglich im Rahmen einer Selektion statt. Deshalb ist die explizite Suche im Kapitel 15.2.6 beschrieben. Daneben unterstützen die Kommandos, welche sich auf Dateien beziehen, Varianten mit Parameter *shortFileIdentifier*. Bei den jeweiligen Kommandos ist beschrieben, wie in diesem Fall eine Datei gesucht wird.

10.2.2 Passwortsuche

Passwörter werden im Rahmen der Kommandos in Kapitel 15.6 verwendet, aber auch bei der Auswertung von Zugriffsregeln (siehe (N222)). Bei der Passwortsuche handelt es sich um eine karteninterne Funktionalität, bei der viele Implementierungsdetails eine Rolle spielen. An der Schnittstelle „Interpreter“ (siehe Abbildung 1) wird das im Folgenden festgelegte Verhalten deutlich:

Input:	passwordReference	gemäß (N728)
	startFolder	Ordner, bei dem die Suche nach einem Passwortobjekt startet
Output:	password	Enthält das gefundene Passwortobjekt
Errors:	pwdNotFound	Zu den gegebenen Inputparametern wurde kein passendes Passwortobjekt gefunden
Notation:		*password* = searchPwd(*startFolder*, *passwordReference*)

(N207) Der Input–Parameter *passwordReference* wird gemäß (N728) wie folgt zerlegt:

a. identifier = passwordReference mod 128

b. location = passwordReference – identifier

(N208) Wenn *location* den Wert ´00´ = 0 besitzt und damit ein globales Passwort adressiert, dann wird im Wurzelverzeichnis *root* des Objektsystems nach einem Passwort gesucht, dessen Attribut *pwdIdentifier* identisch zu *identifier* ist. Wenn ein solches Passwortobjekt

a. existiert, dann wird es als Output–Parameter *password* verwendet.

b. nicht existiert, wird der Fehler „pwdNotFound“ zurückgemeldet.

(N209) Wenn *location* den Wert ´80´ = 128 besitzt und damit ein DF–spezifisches Passwort adressiert, dann wird die „lokale Variable“ *folder* auf *startFolder* gesetzt.

(N210) Wenn folder

a. auf das Wurzelverzeichnis *root* zeigt, dann wird der Fehler „pwdNotFound“ zurückgemeldet.

b. ansonsten wird in *folder* nach einem Passwort gesucht, dessen Attribut *pwdIdentifier* identisch zu *identifier* ist. Wenn ein solches Passwortobjekt

   i. existiert, dann wird es als Output–Parameter *password* verwendet.

   ii. nicht existiert, wird *folder* auf den nächsthöheren Ordner gesetzt und der Algorithmus mit Schritt (N210)a fortgesetzt.

10.2.3 Suche nach einem geheimen Schlüsselobjekt

In diesem Kapitel werden symmetrische Schlüsselobjekte gemäß Kapitel 9.5.1 und private Schlüsselobjekte gemäß Kapitel 9.5.2 gemeinsam behandelt, da nach ihnen auf gleiche Art und Weise gesucht wird. Sie werden im Rahmen diverser kryptographischer Operationen verwendet. Symmetrische Schlüsselobjekte werden zudem auch bei der Auswertung von Zugriffsregeln (siehe (N223)) verwendet. Bei dieser Art der Schlüsselsuche handelt es sich um eine karteninterne Funktionalität, bei der viele Implementierungsdetails eine Rolle spielen. An der Schnittstelle „Interpreter“ (siehe Abbildung 1) wird das im Folgenden festgelegte Verhalten deutlich:

Input:	keyReference	gemäß (N164), (N171)
	algID	Kryptographisches Verfahren, welches der zu suchende Schlüssel unterstützt, oder „Wildcard“
	startFolder	Ordner, bei dem die Suche nach einem Schlüsselobjekt startet
Output:	key	Enthält das gefundene Schlüsselobjekt
Errors:	keyNotFound	Zu den gegebenen Inputparametern wurde kein passendes Schlüsselobjekt gefunden.
	notSupported	Der Schlüssel unterstützt den von *algID* geforderten Algorithmus nicht.
Notation:		key = SearchSecretKey(startFolder, keyReference, algID)

(N211) Der Input–Parameter *keyReference* wird gemäß (N996) in die Bestandteile

a. identifier = keyReference mod 128

b. location = keyReference – identifier

(N212) Wenn *location* den Wert ´00´ = 0 besitzt und damit ein globales Schlüsselobjekt adressiert, dann wird im Wurzelverzeichnis *root* des Objektsystems nach einem Schlüsselobjekt gesucht, dessen Attribut *keyIdentifier* identisch zu *identifier* ist. Wenn ein solches Schlüsselobjekt

a. existiert, dann wird es als Output–Parameter *key* verwendet.

b. nicht existiert, wird der Fehler „keyNotFound“ zurückgemeldet.

(N213) Wenn *location* den Wert ´80´ = 128 besitzt und damit ein DF–spezifisches Passwort adressiert, dann wird die „lokale Variable" *folder* auf *startFolder* gesetzt.

(N214) Wenn folder

a. auf das Wurzelverzeichnis *root* zeigt, dann wird der Fehler „keyNotFound" zurückgemeldet.

b. ansonsten wird in *folder* nach einem Schlüsselobjekt gesucht, dessen Attribut *keyIdentifier* identisch zu *identifier* ist. Wenn ein solches Schlüsselobjekt

   i. existiert, dann wird es als Output–Parameter *key* verwendet.

   ii. nicht existiert, wird *folder* auf den nächsthöheren Ordner gesetzt und der Algorithmus mit Schritt (N214)a fortgesetzt.

(N215) Wenn der Output–Parameter *key* ein

a. symmetrischer Schlüssel ist und dessen Attribut *algorithmIdentifier* nicht zum Parameter *algID* passt, dann wird der Fehler „notSupported" zurückgemeldet.

b. privater Schlüssel ist und kein Listenelement im Attribut *listAlgorithmIdentifier* zum Parameter *algID* passt, dann wird der Fehler „notSupported" zurückgemeldet.

10.2.4 Suche nach einem öffentlichen Schlüsselobjekt

Öffentliche Schlüsselobjekte werden zum Importieren von Schlüsseln mittels Zertifikaten und im Rahmen von Authentisierungsprotokollen verwendet. Die Unterklasse „Öffentliche Authentisierungsobjekte" wird zudem auch bei der Auswertung von Zugriffsregeln (siehe (N224) und (N225)) verwendet. Bei dieser Art der Schlüsselsuche handelt es sich um eine karteninterne Funktionalität, bei der viele Implementierungsdetails eine Rolle spielen. An der Schnittstelle „Interpreter" (siehe Abbildung 1) wird das im Folgenden festgelegte Verhalten deutlich:

Input:	*identifier*	gemäß (N185), (N191), (N195)
	algID	Kryptographisches Verfahren, welches der zu suchende Schlüssel unterstützt
	startFolder	Ordner, aus dem heraus die Suche startet
Output:	*key*	Enthält das gefundene Schlüsselobjekt
Errors:	*keyNotFound*	Zu den gegebenen Inputparametern wurde kein passendes Schlüsselobjekt gefunden.
Notation		*key* = SearchPublicKey(*identifier*, *algID*)

(N216) Die „lokale Variable" *folder* wird auf *startFolder* gesetzt. Dann werden folgende Schritte ausgeführt:

a. Es wird in den Attributen *persistentPublicKeyList* und *publickeyList* des Objektsystems nach einem Schlüsselobjekt gesucht,

– dessen Attribut *keyIdentifier* identisch zu *identifier* ist und
– dessen Attribut *oid* zu *algID* passt und
– das dem Ordner *folder* zugeordnet ist.
Wenn ein solches Schlüsselobjekt

b. existiert, dann wird es als Output–Parameter *key* verwendet.

c. nicht existiert und folder ist

   i. gleich *root*, wird der Fehler „keyNotFound“ zurückgemeldet.

   ii. ungleich *root*, dann wird *folder* auf den nächsthöheren Ordner gesetzt und der Algorithmus mit Schritt (N216)a fortgesetzt.

11 Zugriffskontrolle (normativ)

Fast alle in Kapitel 15 beschriebenen Kommandos werden durch Zugriffsregeln geschützt. Das bedeutet, dass das Betriebssystem kontrolliert, ob der Sicherheitsstatus für die Ausführung der Operation ausreichend ist. Die Regeln in diesem Kapitel bilden eine Untermenge der in [7816–4] Kapitel 5.4.3.2 Expanded format definierten Regeln.

11.1 Zugriffsart

Die Zugriffsart (access mode) zeigt an, ob die der Zugriffsart zugeordnete Zugriffsbedingung im Rahmen einer Zugriffsregelprüfung auszuwerten ist. Bei der Spezifikation von Anwendungen sind folgende Regeln einzuhalten:

(N217) Eine Zugriffsart MUSS eine Liste von Kommandobeschreibungen sein.

(N218) Als Untermenge zu [7816–4] Tabelle 22 MUSS für jedes Listenelement gelten:

a. Ein Listenelement MUSS den Namen eines Kommandos enthalten, der gemäß Kapitel 15 äquivalent ist zu genau einer Kombination aus CLA Byte (siehe Kapitel 12.5.1) und INS Byte (siehe Kapitel 12.5.2)
b. Innerhalb eines Listenelementes ist der Parameter P1 (siehe Kapitel 12.5.3) optional. Das bedeutet: Es MUSS möglich sein, dass ein Listenelement
   i. genau einen Parameter P1 enthält.
   ii. keinen Parameter P1 enthält.
c. Innerhalb eines Listenelementes ist der Parameter P2 (siehe Kapitel 12.5.4) optional. Das bedeutet: Es MUSS möglich sein, dass ein Listenelement
   i. genau einen Parameter P2 enthält.
   ii. keinen Parameter P2 enthält.
d. Ein COS KANN weitere Kommandobeschreibungen unterstützen.
   Ein COS KANN weitere Kommandobeschreibungen ablehnen.

(N219) Definition: Bei der Auswertung der Zugriffsart MUSS diese genau dann zur aktuellen Kommando APDU passen, wenn die Bestandteile der Zugriffsart identisch sind zu den Bestandteilen der Kommando APDU an der Schnittstelle „Interpreter“ (siehe Abbildung 1).

11.2 Zugriffsbedingung

Eine Zugriffsbedingung ist ein boolescher Ausdruck. Bei der Spezifikation einer Anwendung sind folgende Regeln einzuhalten, wobei in der Beschreibung folgende Definitionen eingeführt werden:

startFolder Falls diese Zugriffsbedingung zu einem Objekt

- vom Typ Ordner gehört, dann ist *startFolder* gleich dieser Ordner.
- anderen Typs gehört, dann ist *startFolder* gleich dem Ordner, dessen Attribut *children* dieses Objekt enthält.

path(*folder*) Pfad eines Ordners mit Namen *folder*. Zum Pfad gehört nach dieser Definition sowohl der Ordner *folder*, als auch alle seine übergeordneten Ebenen gemäß Kapitel 10.1 einschließlich *root*.

(N220) Das boolesche Element ALWAYS MUSS stets den Wert True liefern.

(N221) Das boolesche Element NEVER MUSS stets den Wert False liefern.

(N222) Das boolesche Element PWD(*passwordReference*) MUSS

a. genau dann True liefern, wenn die Passwortsuche gemäß Kapitel 10.2.2 mit den Input–Parametern *passwordReference* und *startFolder* ein Passwort findet und wenigstens eine der beiden folgenden Bedingungen erfüllt ist:

   i. Dieses Passwortobjekt ist entweder in *globalPasswordList* (siehe (N299)i) oder in *dfSpecificPasswordList* (siehe (N299)j) enthalten und das Attribut *securityStatusEvaluationCounter* dieses Passwortes ist ungleich null. Dabei MUSS das Attribut *securityStatusEvaluationCounter* dieses Passwortes um eins dekrementiert werden. Falls durch das Dekrementieren der Wert auf Null gefallen ist,

      1. KANN der Eintrag in der Liste verbleiben, oder
      2. KANN der Eintrag aus der Liste entfernt werden.

   ii. Das Attribut *flagEnabled* dieses Passwortes besitzt den Wert False.

b. in allen anderen Fällen False liefern.

(N223) Das boolesche Element AUT(*keyReference*) MUSS

a. genau dann True liefern, wenn die Schlüsselsuche gemäß Kapitel 10.2.3 mit den Input–Parametern *keyReference*, „Wildcard“ und *startFolder* ein Schlüsselobjekt *key* findet und genau dieses Schlüsselobjekt entweder in *globalSecurityList* (siehe (N299)e) oder in *dfSpecificSecurityList* (siehe (N299)f) enthalten ist.

b. in allen anderen Fällen False liefern.

(N224) Ab Generation 2 gilt: Das boolesche Element AUT(*acBitList*) MUSS … (für Generation 2 vervollständigen).

(N225) Das boolesche Element AUT(*CHA*) MUSS

a. genau dann True liefern, wenn *CHA* in *globalSecurityList* (siehe (N299)e) oder in *dfSpecificSecurityList* (siehe (N299)f) enthalten ist.

b. in allen anderen Fällen False liefern.

(N226) Secure Messaging gemäß Kapitel 14 wird wie folgt in Zugriffsbedingungen verwendet:

a. Das boolesche Element SmMac MUSS den Wert *SessionkeyContext.flagSessionEnabled* liefern.

Kom
3002

b. Das boolesche Element SmCmdEnc MUSS den Wert von *SessionkeyContext.flagCmdEnc* liefern.

c. Das boolesche Element SmRspEnc MUSS stets den Wert True liefern und *SessionkeyContext.flagRspEnc* auf True setzen

(N227) Die zuvor definierten booleschen Elemente MÜSSEN sich zu einem booleschen Ausdruck verbinden lassen.
Der boolesche Ausdruck MUSS den AND–Oparator unterstützen.
Der boolesche Ausdruck MUSS den OR–Oparator unterstützen.
Der boolesche Ausdruck DARF KEINEN anderen Oparator enthalten.

(N228) Der boolesche Wert der Zugriffsbedingung ist das Ergebnis des booleschen Ausdrucks.

Hinweis (33): Die normativen Regeln dieses Dokumentes, insbesondere die
– zu Attributen eines Passwortes (siehe Kapitel 9.4)
– zu Statusänderungen bei erfolgreicher Benutzerverifikation (siehe (N746) und (N829))
– zum booleschen Element PWD (siehe (N222))
sind so aufgebaut, dass nach erfolgreicher Benutzerverifikation
– genau eine Operation (etwa Signatur) möglich ist (startSSEC = 1, siehe (N158)b.ii), oder
– genau n Operationen möglich sind (startSSEC = n > 1), oder
– beliebig viele Operationen möglich sind (startSSEC = „unendlich“).

11.3 Zugriffsregel

Eine Zugriffsregel kombiniert Zugriffsart und Zugriffsbedingung und liefert bei der Auswertung ein boolesches Ergebnis. Bei der Spezifikation einer Anwendung sind folgende Regeln einzuhalten:

(N229) Eine Zugriffsregel ist eine Liste mit mindestens einem Element.

(N230) Ein Listenelement MUSS genau eine Zugriffsart gemäß Kapitel 11.1 und eine Zugriffsbedingung gemäß Kapitel 11.2 enthalten.

(N231) Ein Listenelement MUSS

a. genau dann den booleschen Wert True liefern, wenn

i. die Zugriffsart gemäß (N219) passt und

ii. der boolesche Wert der Zugriffsbedingung den Wer True hat.

b. in allen anderen Fällen False liefern (implizites NEVER).

(N232) Eine Zugriffsregel MUSS

a. genau dann als erfüllt gelten, wenn mindestens ein Listenelement den Wert True liefert.

b. in allen anderen Fällen als nicht erfüllt gelten.

(N233) Eine Zugriffsregel, welche die übrigen Bedingungen des Kapitels 11 erfüllt, DARF bei Kodierung gemäß [7816–4] Kapitel 5.4.3.2 NICHT mehr als 240 Byte beanspruchen.

11.4 Zugriffsregelauswertung

Dieses Unterkapitel beschreibt wie das COS feststellt, ob ein bestimmtes Kommando erlaubt oder verboten ist.

Input:	obj	Instanz eines Objektes vom Typ Ordner, Datei, Passwortobjekt, symmetrisches Authentisierungsobjekt oder privates Schlüsselobjekt
	CLA	CLA Byte
	INS	INS Byte
	P1	Parameter P1
	P2	Parameter P2
Output:	b	Boolean, True bedeutet Kommando erlaubt, False bedeutet Kommando verboten
Errors:	–	–
Notation:		*b* = AccessRuleEvaluation(*obj*, *CLA* , *INS*, *P1*, *P2*)

(N234) Schritt 1: Handelt es sich bei *obj*

a. um einen Ordner, dann MUSS *se* = *seIdentifier* dieses Ordners sein (siehe (N300)a.

b. sonst MUSS *se* = *seIdentifier* des Ordners sein (siehe (N300)a, der *obj* in seinem Attribut *children* (siehe (N100)) enthält.

(N235) Schritt 2: Unter Berücksichtigung des logischen Wertes von *obj.lifeCycleStatus* (siehe (N206)) und *se* wird aus *obj.accessRuleList* die passende Zugriffsregel ausgewählt (siehe Kapitel 9.1.4).

(N236) Schritt 3: Anhand von *CLA*, *INS*, *P1* und *P2* wird ermittelt, ob die Zugriffsregel erfüllt ist (siehe Kapitel 11.3).

(N237) Schritt 4: Ist die ausgewählte Zugriffsregel

a. erfüllt, dann MUSS *b* = True gelten.

b. sonst MUSS *b* = False gelten.

Hinweis (34): Falls in folgender Zugriffsbedingung PWD1 OR PWD2 beide Passwortobjekte mit SSEC ungleich unendlich vorkommen, dann ist es herstellerspezifisch, ob beide SSEC oder nur einer und diesem Falle welcher dekrementiert wird.

12 Kommunikation (normativ)

12.1 Request – Response

Für Chipkarten entspricht es dem Stand der Technik, dass sie Nachrichten mit einem externen Kommunikationspartner über einen Kanal im Halbduplex Verfahren austauschen. Zudem arbeiten sie ähnlich wie ein Server. Das bedeutet, dass der externe Kommunikationspartner eine Nachricht (Kommando) über den Kanal schickt. Diese Nachricht (Kommando) wird von der Chipkarte verarbeitet. Anschließend sendet die Chipkarte über denselben Kanal eine Nachricht (Antwort) zurück. Erst nach dem vollständigen Empfang der Chipkarten–Nachricht hat der externe Kommunikationspartner die Möglichkeit eine weitere Nachricht zu schicken.

12.2 Elektrische Schnittstelle

12.2.1 Aktivierung

(N238) Der externe Kommunikationspartner MUSS die Chipkarte gemäß [7816–3] Kapitel 6.2.1 aktivieren.

(N239) Im Anschluss an die Aktivierung gemäß (N238) MUSS der externe Kommunikationspartner einen Cold Reset gemäß [7816–3] Kapitel 6.2.2 ausführen.

(N240) Die Chipkarte MUSS einen Warm Reset gemäß [7816–3] Kapitel 6.2.3 unterstützen.

(N241) Ein Cold Reset gemäß (N239) und ein Warm Reset gemäß (N240) lassen sich als spezielle „Nachrichten“ des externen Kommunikationspartners auffassen, die von der Chipkarte mit einem Answer–to–Reset (ATR) gemäß [7816–3] Kapitel 8.2 beantwortet wird. Der ATR MUSS gemäß [7816–3] Kapitel 8.1 versendet werden.

(N242) Die Kommunikation MUSS mittels „direct convention“ (siehe [7816–3] Kapitel 8.1) erfolgen. Das bedeutet „initial character“ TS im Answer–To–Reset MUSS den Wert ´3B´ haben.

(N243) Der ATR MUSS in TA1 einen Wert aus der Menge {´18´, ´95´, ´96´} verwenden.

(N244) Der ATR SOLL ein TC1 enthalten. Falls TC1 enthalten ist, dann MUSS der Wert ´FF´ sein.

(N245) Der ATR MUSS in TD1 das Protokoll T=1 anzeigen.

(N246) Der ATR DARF KEIN TA2 enthalten. Daraus folgt, dass die Chipkarte den „negotiable mode“ unterstützen MUSS (siehe Kapitel 12.2.4).

(N247) Der ATR MUSS einen „class indicator“ gemäß [7816–3] Tabelle 10 enthalten. Der „class indicator“ MUSS einen Wert aus der Menge {3, 7} sein.

12.2.2 Deaktivierung

(N248) Der externe Kommunikationspartner SOLL die Chipkarte gemäß [7816–3] Kapitel 6.4 deaktivieren.

(N249) Die Deaktivierung KANN auf andere Art erfolgen.

(N250) Geschieht die Deaktivierung zu einem Zeitpunkt, in welchem eine transaktionsgeschützte Schreiboperation durchgeführt wird (siehe Kapitel 15.1), so MUSS das COS dafür Sorge tragen, dass gemäß den Regeln in Kapitel 15.1.1 bzw. 15.1.2 verfahren wird, bevor die nächste Kommando APDU von der Komponente „Cmd Interpreter“ in Abbildung 1 bearbeitet wird.

12.2.3 Character Frame

(N251) Auf der physikalischen Schnittstelle (siehe Abbildung 1) MÜSSEN Character gemäß [7816–3] Kapitel 7 verwendet werden.

12.2.4 Protocol and Parameter Selection

(N252) Gemäß (N246) MUSS die Chipkarte den „negotiable mode“ anzeigen. Deshalb MUSS die Chipkarte „Protocol and Parameter Selection“ Verfahren gemäß [7816–3] Kapitel 9 unterstützen.

(N253) Falls TA1 im ATR den Wert ´18´ besitzt, dann MUSS die Chipkarte in PPS1 einen Wert aus der Menge {´12´, ´13´} akzeptieren.

(N254) Falls TA1 im ATR den Wert ´95´ besitzt, dann MUSS die Chipkarte in PPS1 einen Wert aus der Menge {´92´, ´93´, ´94´} akzeptieren.

(N255) Falls TA1 im ATR den Wert ´96 besitzt, dann MUSS die Chipkarte in PPS1 einen Wert aus der Menge {´92´, ´93´, ´94´, ´95´} akzeptieren.

12.3 OSI Referenzmodell (informativ)

In Anlehnung an das OSI–Referenzmodell werden hier verschiedene Layer definiert, die an der Bearbeitung einer Nachricht beteiligt sind, welche von einem externen Kommunikationspartner gesendet wurde. Es sei ausdrücklich darauf hingewiesen, dass die Inhalte dieses Abschnittes keine Implementierungsdetails festlegen. Das bedeutet, es ist zulässig, den internen Aufbau eines Chipkartenbetriebssystems oder dessen Kommunikationsaufbau anders zu gestalten. Die normativen Teile dieses Dokumentes lassen sich aber leichter beschreiben und verstehen, wenn man das Folgende zugrunde legt.

In Abbildung 1 wird exemplarisch gezeigt, wie eine Kommando APDU CmdApdu1 vom externen Kommunikationspartner zunächst entsprechend den Konventionen des Übertragungsprotokolls (siehe Kapitel 12.8) im physikalischen Layer und im Data Link Layer in einen oder möglicherweise mehrere TPDU zerlegt und im I/O der Chipkarte wieder zu einer CmdApdu1 gemäß Kapitel 12.5 zusammengesetzt wird.

Der nächste Layer verwaltet logische Kanäle (siehe [7816–4] Kapitel 5.1.1.2) und leitet das empfangene Kommando entsprechend der Kanalnummer im CLA Byte weiter an den entsprechenden logischen Kanal, hier Channel_0.

Im nächsten Layer „SecMes“ wird die optional per Secure Messaging gesicherte Kommando APDU ausgepackt und an den Kommandointerpreter weitergeleitet.

Der Kommandointerpreter verarbeitet die Kommando APDU und erstellt eine entsprechende Antwort APDU, welche optional per Secure Messaging gesichert wird. Die eventuell gesicherte Antwort wird im I/O in eine oder mehrere TPDU zerlegt, welche dann über die physikalische Schnittstelle an den externen Kommunikationspartner übermittelt werden.

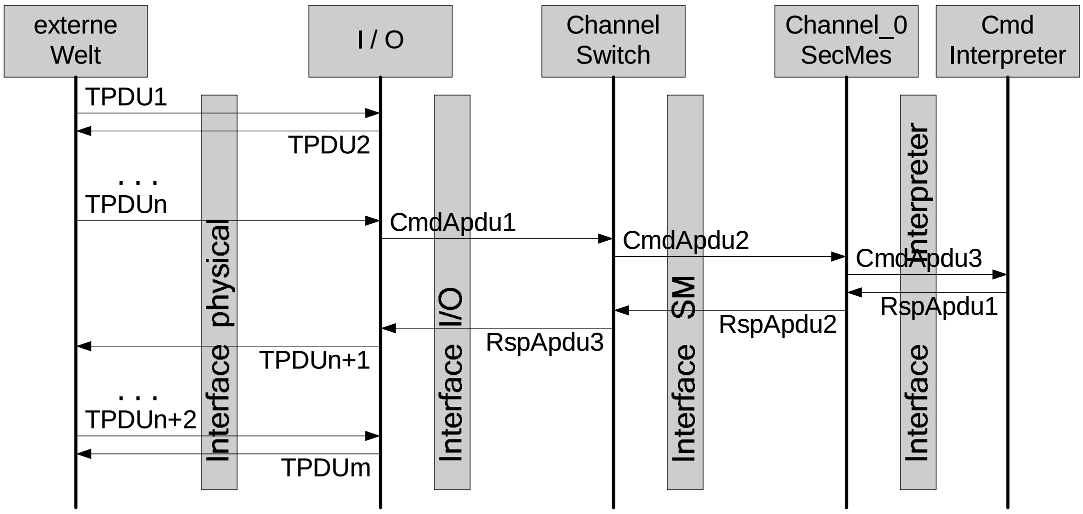

Abbildung 1: Message Sequence Chart für die Kommandobearbeitung

12.4 Kommandobearbeitung

Dieses Kapitel beschreibt die Bearbeitung eines Kommandos, welches von einem externen Kommunikationspartner an das COS gesendet wurde. Es werden die Begriffe aus Kapitel 12.3 verwendet. Es wird nochmals darauf verwiesen, dass die Inhalte von Kapitel 12.3 nicht normativ sind. Insofern ist die genaue Zuordnung der normativen Aussagen dieses Kapitels zu Teilen eines COS nicht festgelegt. Wichtig ist aber, dass das Verhalten des COS auf Ebene der physikalischen Schnittstelle den normativen Vorgaben entspricht.

Bei der Bearbeitung einer Kommando APDU und der Generierung einer Response APDU werden folgende Schritte durchlaufen:

(N256) Ein externer Kommunikationspartner sendet über die physikalische Schnittstelle eine Kommando APDU CmdApdu1, welche Kapitel 12.5 entsprechen MUSS. Dazu wird CmdApdu1 auf der physikalischen Schnittstelle entsprechend dem Übertragungsprotokoll als (eine oder mehrere) TPDU transportiert. Das I/O der Chipkarte setzt die TPDU zusammen und gewinnt CmdApdu1 zurück.

(N257) Die Fehlerbehandlung auf der physikalischen Schnittstelle MUSS [7816–3] entsprechen.

(N258) Anschließend MUSS die Kanalnummer aus dem CLA Byte von CmdApdu1 extrahiert werden.

(N259) Falls der Kanal mit der Kanalnummer aus dem CLA Byte nicht geöffnet ist, MUSS

a. die Bearbeitung des Kommandos terminieren.

b. RspApdu3 enthält in diesem Fall keine Antwortdaten und besteht lediglich aus dem Trailer ChannelClosed = ´68 81´.

(N260) Entsprechend der Kanalnummer wird CmdApdu1 unverändert als CmdApdu2 an den entsprechenden logischen Kanal weitergeleitet.

(N261) Weil das Kommando MANAGE CHANNEL gemäß Kapitel 15.9.5 nicht verpflichtend vorgeschrieben ist, MUSS die Kanalnummer im Rahmen dieser Spezifikation stets null sein.
Das COS KANN weitere Kanalnummern akzeptieren.
Das COS KANN weitere Kanalnummern ablehnen.

(N262) Innerhalb des logischen Kanals wird CmdApdu2 mit dem entsprechenden Kanalkontext an den Secure Messaging Layer („SecMes“ in Abbildung 1) weitergeleitet und dort gemäß Kapitel 14.1.2 bearbeitet. Das Ergebnis ist RspApdu2.

(N263) RspApdu2 MUSS unverändert als RspApdu3 an das I/O der Chipkarte geschickt werden.

(N264) Das I/O der Chipkarte MUSS RspApdu3 gemäß dem verwendeten Übertragungsprotokoll in eine oder mehrere TPDU umwandeln und über die physikalische Schnittstelle senden.

12.5 Kommando APDU

12.5.1 Class Byte

Das Class Byte (CLA Byte) enthält die Kommandoklasse. Zusammen mit dem Instruction Byte (siehe Kapitel 12.5.2) legt es eindeutig das auszuführende Kommando fest.

Welche Kombinationen aus CLA, INS, P1 und P2 zu unterstützen sind, wird in Kapitel 15 festgelegt.

(N265) Gemäß [7816–3] MUSS das CLA Byte in einem Oktett kodiert werden.

12.5.2 Instruction Byte

Das Instruction Byte (INS Byte) enthält die Kodierung für den auszuführenden Befehl. Zusammen mit dem Class Byte (siehe Kapitel 12.5.1) legt es eindeutig das auszuführende Kommando fest.

Welche Kombinationen aus CLA, INS, P1 und P2 zu unterstützen sind, wird in Kapitel 15 festgelegt.

(N266) Gemäß [7816–3] MUSS das INS Byte in einem Oktett kodiert werden.

12.5.3 Parameter P1

Der Parameter P1 enthält in der Regel eine Variable, die bei der Kommandoausführung benötigt wird. Die Bedeutung dieses Parameters hängt gewöhnlich von der Kombination aus CLA, INS und P2 ab.

Welche Kombinationen aus CLA, INS, P1 und P2 zu unterstützen sind, wird in Kapitel 15 festgelegt.

(N267) Gemäß [7816–3] MUSS der Parameter P1 in einem Oktett kodiert werden.

12.5.4 Parameter P2

Der Parameter P2 enthält in der Regel eine Variable, die bei der Kommandoausführung benötigt wird. Die Bedeutung dieses Parameters hängt von der Kombination aus CLA und INS ab.

Welche Kombinationen aus CLA, INS, P1 und P2 zu unterstützen sind, wird in Kapitel 15 festgelegt.

(N268) Gemäß [7816–3] MUSS der Parameter P2 in einem Oktett kodiert werden.

12.5.5 Datenfeld

Das Datenfeld einer Kommando APDU ist optional. Es ist möglich, dass es fehlt. Das Datenfeld ist ein Oktettstring der Länge Nc. Gemäß [7816–3] ist der Definitionsbereich von Nc gleich {1, …, 65535}.

(N269) An der Schnittstelle „Interpreter“ (siehe Abbildung 1) gilt für Nc:

a. Das COS MUSS für Nc alle Werte aus dem Intervall [1, 543] unterstützen.
b. Ein COS KANN weitere Werte für Nc unterstützen.
c. Ein COS KANN weitere Werte für Nc ablehnen.
d. Ein COS DARF eine APDU NICHT wegen eines Längenfehlers abweisen, wenn die APDU die Längenbeschränkung aus (N269)a oder die Längenbeschränkung aus EF.ATR erfüllt.
e. Der Sender einer Kommando APDU SOLL die Längenbeschränkung durch passende Wahl von Nc einhalten.

Hinweis (35): Falls ein COS für bestimmte CLA, INS, P1 und P2 Kombinationen weitere Werte für Nc unterstützt, dann ist es ratsam dies im EF.ATR anzuzeigen.

Hinweis (36): Die obere Grenze in (N269)a gilt für das COS einer Chipkarte. Für das IFD (Kartenleser) wird wegen möglicher, zukünftiger Erweiterungen, Entwicklungen und wegen Performancegewinns empfohlen, mindestens 2048 = ´0800´ zu unterstützen.

Hinweis (37): Die obere Grenze in (N269)a errechnet sich aus (N941)ff und (N21)a wie folgt:

cipherIn = cipher || plainDO
= ´A6 LA6 cipherDO$_{in}$ || A0 L$_{A0}$ [algDO || keyDO]´
= ´A6 L$_{A6}$ [86 L$_{86}$ (00 C$_{in}$)] || A0 L$_{A0}$ [80 01 XX || 7F49 L$_{7F49}$ (81 L$_{81}$ n || 82 04 e)]
à 1+ 3+ 1+ 3+ 257 + 1+ 3 + 1+1+ 1 + 2+ 3+ 1+ 3+256+1+1+4 = 543

12.5.6 Le Feld

Das Le Feld einer Kommando APDU ist optional. Es ist möglich, dass es fehlt. Das Le Feld enthält eine Zahl Ne, die angibt, wie viele Oktette im Datenfeld der Antwort APDU erwartet werden. In diesem Dokument gelten folgende Definitionen:

WildCardShort	Dieser Wert wird verwendet um anzuzeigen, dass innerhalb der Obergrenze von 256 Oktetten alle verfügbaren Oktette in das Datenfeld der Antwortnachricht einzustellen sind.
WildCardExtended	Dieser Wert wird verwendet um anzuzeigen, dass innerhalb der Obergrenze von 65536 Oktette alle verfügbaren Oktette in das Datenfeld der Antwortnachricht einzustellen sind.

Gemäß [7816–4] ist der Definitionsbereich für Ne: {1, …, 65535, WildCardShort, WildCardExtended}

(N270) An der Schnittstelle „Interpreter“ (siehe Abbildung 1) gilt für Ne:

a. Das COS MUSS für Ne alle Werte aus dem Intervall [1, 543] unterstützen.

b. Das COS MUSS für Ne den Wert WildCardShort unterstützen.

c. Ein COS KANN weitere Werte für Ne unterstützen.
Ein COS KANN weitere Werte für Ne ablehnen.

d. Ne MUSS so gewählt werden, dass die vom COS maximal unterstützte Länge Nr der Antwortdaten (entweder 543, oder angezeigt im EF.ATR) nicht überschritten wird.

e. Ein COS DARF eine APDU NICHT wegen eines Längenfehlers abweisen, wenn die APDU die Längenbeschränkung aus (N270)a oder die Längenbeschränkung aus EF.ATR erfüllt.

f. Der Sender einer Kommando APDU SOLL die Längenbeschränkung durch passende Wahl von Ne einhalten.

Hinweis (38): Falls ein COS für bestimmte CLA, INS, P1 und P2 Kombinationen weitere Werte für Ne unterstützt, dann ist es ratsam dies im EF.ATR anzuzeigen.

Hinweis (39): Die obere Grenze in (N270)a gilt für das COS einer Chipkarte und (N270)d gilt für die „externe Welt“. Für das IFD (Kartenleser) wird hier wegen möglicher, zukünftiger Erweiterungen, Entwicklungen und wegen Performancegewinns empfohlen, mindestens 2048 = ´0800´ zu unterstützen.

12.6 Response APDU

Mit den Definitionen aus den Kapiteln 12.6.1 und 12.6.2 ergibt sich für eine Response APDU:

(N271) Eine Response APDU MUSS ein Oktettstring gemäß ´*rspData* || Trailer´ sein, wobei *rspData* möglicherweise leer ist.

12.6.1 Datenfeld

Das Datenfeld *rspData* einer Antwort APDU ist optional. Es ist möglich, dass es fehlt. Das Datenfeld *rspData* ist ein Oktettstring der Länge Nr. Gemäß [7816–3] ist der Definitionsbereich von Nr {1, …, 65536}.

(N272) An der Schnittstelle „Interpreter“ (siehe Abbildung 1) gilt für Nr: Nr MUSS kleiner gleich Ne der zugehörigen Kommando APDU sein.

12.6.2 Trailer

Der Trailer (siehe [7816–4] Tabelle 1) besteht aus zwei Oktetten. Tabelle 170 zeigt die in diesem Dokument definierten Trailer. Der Trailer enthält Statusangaben über die Bearbeitung des Kommandos. Dazu zählen auch Fehlerindikationen.

12.7 Zulässige Kommando Antwort Paare

Wie in den Kapiteln 12.5.5 und 12.5.6 dargestellt sind zwei Bestandteile einer Kommando APDU optional. Daraus ergeben sich vier Kombinationsmöglichkeiten, die im Folgenden beschrieben werden.

12.7.1 Case 1 Kommando Antwort Paar

Im Fall einer Case 1 Kommando APDU fehlen Datenfeld und Le Feld. Die Kommando APDU enthält folgende Angaben:

Tabelle 10: Case 1 Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 1 Kommando APDU:

(N273) Eine Case 1 Kommando APDU MUSS aus vier Oktette bestehen.

(N274) Die vier Oktette einer Case 1 Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2.

Die zu einer Case 1 Kommando APDU gehörende Antwort APDU besteht nur aus dem Trailer.

Tabelle 11: Case 1 Response APDU

Inhalt	Beschreibung
Trailer	Statusbytes SW1 und SW2

12.7.2 Case 2 Kommando Antwort Paar

Im Fall einer Case 2 Kommando APDU fehlt das Datenfeld. Das Le Feld ist vorhanden. In Abhängigkeit vom Wert Ne, der im Le Feld transportiert wird, werden die Fälle „short" und „extended" unterschieden.

12.7.2.1 Case 2 Short Kommando

In diesem Fall enthält das Datenfeld der Antwortnachricht nie mehr als 256 Oktette. Die Kommando APDU enthält im Fall einer Case 2 Short folgende Angaben:

Tabelle 12: Case 2 Short Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Ne	´XX´	Ne aus der Menge {1, …, 255, WildCardShort}

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 2 Short Kommando APDU:

(N275) Eine Case 2 Short Kommando APDU MUSS aus fünf Oktette bestehen.

(N276) Der Wert von Ne wird wie folgt in einem Oktett kodiert, welches als LeFeld bezeichnet wird:

a. Wenn Ne im Intervall [1, 255] liegt, dann MUSS diese ganze Zahl in einem Oktett kodiert werden: LeFeld = I2OS(Ne, 1).

b. Wenn Ne den Wert WildCardShort besitzt, dann MUSS das LeFeld gleich ´00´ sein.

(N277) Die fünf Oktette einer Case 2 Short Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || LeFeld.

12.7.2.2 Case 2 Extended Kommando

In diesem Fall enthält das Datenfeld der Antwortnachricht möglicherweise mehr als 256 Oktette. Die Kommando APDU enthält im Fall einer Case 2 Extended folgende Angaben:

Tabelle 13: Case 2 Extended Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Ne	´XX´	Ne aus der Menge {256, …, 65535, WildCardExtended}

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 2 Extended Kommando APDU:

(N278) Eine Case 2 Extended Kommando APDU MUSS aus sieben Oktette bestehen.

(N279) Der Wert von Ne wird wie folgt in zwei Oktett kodiert, welche das LeFeld bilden:

a. Wenn Ne im Intervall [256, 65535] liegt, dann MUSS diese ganze Zahl in zwei Oktette kodiert werden: LeFeld = I2OS(Ne, 2).

b. Wenn Ne den Wert WildCardExtended besitzt, dann MUSS das LeFeld gleich ´0000´ sein.

(N280) Die sieben Oktette einer Case 2 Extended Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || ´00´ || LeFeld.

Hinweis (40): Das Oktett ´00´ nach dem Parameter P2 lässt sich als Indikator für „extended length" auffassen.

12.7.2.3 Case 2 Response

Die zu einer Case 2 Kommando APDU gehörende Antwort APDU besteht aus dem optionalen Datenfeld und aus dem Trailer.

Tabelle 14: Case 2 Response APDU

Inhalt	Beschreibung
Daten	optionaler Bestandteil der Response APDU, falls vorhanden, dann gelten die Bestimmungen aus Kapitel 12.6.1
Trailer	Statusbytes SW1 und SW2

12.7.3 Case 3 Kommando

Im Fall einer Case 3 Kommando APDU fehlt das Le Feld. Das Datenfeld ist vorhanden. In Abhängigkeit von der Anzahl der Oktette im Datenfeld, werden die Fälle „short" und „extended" unterschieden.

12.7.3.1 Case 3 Short Kommando

In diesem Fall enthält das Datenfeld der Kommandonachricht nie mehr als 255 Oktette. Die Kommando APDU enthält im Fall einer Case 3 Short folgende Angaben:

Tabelle 15: Case 3 Short Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Data	´XX…YY´	Datenfeld mit beliebigen Oktette, Anzahl Oktette aus der Menge [1, 255]

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 3 Short Kommando APDU:

(N281) Eine Case 3 Short Kommando APDU MUSS aus fünf Oktette plus den Oktette aus dem Datenfeld bestehen.

(N282) Die Anzahl Nc der Oktette des Datenfeldes MUSS in einem Oktett kodiert werden, welches als LcFeld bezeichnet wird: LcFeld = I2OS(Nc, 1).

(N283) Die fünf plus Nc Oktette einer Case 3 Short Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || LcFeld || Datenfeld.

12.7.3.2 Case 3 Extended Kommando

In diesem Fall enthält das Datenfeld der Kommandonachricht mehr als 255 Oktette. Die Kommando APDU enthält im Fall einer Case 3 Extended folgende Angaben:

Tabelle 16: Case 3 Extended Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Data	´XX…YY´	Datenfeld mit beliebigen Oktette, Anzahl Oktette aus der Menge [255, 65535]

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 3 Extended Kommando APDU:

(N284) Eine Case 3 Extended Kommando APDU MUSS aus sieben Oktette plus den Oktette aus dem Datenfeld bestehen.

(N285) Die Anzahl Nc der Oktette des Datenfeldes MUSS in zwei Oktett kodiert werden, welche als LcFeld bezeichnet werden: LcFeld = I2OS(Nc, 2).

(N286) Die sieben plus Nc Oktette einer Case 3 Extended Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || ´00´ || LcFeld || Datenfeld.

Hinweis (41): Das Oktett ´00´ nach dem Parameter P2 lässt sich als Indikator für „extended length“ auffassen.

12.7.3.3 Case 3 Response

Die zu einer Case 3 Kommando APDU gehörende Antwort APDU besteht nur aus dem Trailer.

Tabelle 17: Case 3 Response APDU

Inhalt	Beschreibung
Trailer	Statusbytes SW1 und SW2

12.7.4 Case 4 Kommando

Eine Case 4 Kommando APDU enthält alle optionalen Bestandteile. In Abhängigkeit von der Anzahl der Oktette im Datenfeld der Kommandonachricht und in Abhängigkeit vom Wert Ne, der im Le Feld transportiert wird, werden die Fälle „short“ und „extended“ unterschieden.

12.7.4.1 Case 4 Short Kommando

In diesem Fall enthält das Datenfeld der Kommandonachricht nie mehr als 255 Oktette und das Datenfeld der Antwortnachricht nie mehr als 256 Oktette. Die Kommando APDU enthält im Fall einer Case 4 Short folgende Angaben:

Tabelle 18: Case 4 Short Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Data	´XX…YY´	Datenfeld mit beliebigen Oktette, Anzahl Oktette aus der Menge [1, 255]
Ne	´XX´	Ne aus der Menge {1, …, 255, WildCardShort}
Notation		CmdApdu = Case4S(CLA, INS, P1, P2, Data, Ne)

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 4 Short Kommando APDU:

(N287) Eine Case 4 Short Kommando APDU MUSS aus sechs Oktette plus den Oktette aus dem Datenfeld bestehen.

(N288) Die Anzahl Nc der Oktette des Datenfeldes MUSS in einem Oktett kodiert werden, welches als LcFeld bezeichnet wird: LcFeld = I2OS(Nc, 1).

(N289) Der Wert von Ne wird wie folgt in einem Oktett kodiert, welches das Le Feld bildet:

a. Wenn Ne im Intervall [1, 255] liegt, dann MUSS diese ganze Zahl als ein Oktett als LeFeld verwendet werden: LeFeld = I2OS(Ne, 1).

b. Wenn Ne den Wert WildCardShort besitzt, dann MUSS das LeFeld gleich ´00´ sein.

(N290) Die sechs plus Nc Oktette einer Case 4 Short Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || LcFeld || Datenfeld || LeFeld.

12.7.4.2 Case 4 Extended Kommando

In diesem Fall enthält das Datenfeld der Kommandonachricht mehr als 255 Oktette oder das Datenfeld der Antwortnachricht möglicherweise mehr als 256 Oktette. Die Kommando APDU enthält im Fall einer Case 4 Extended folgende Angaben:

Tabelle 19: Case 4 Extended Kommando APDU

	Inhalt	Beschreibung
CLA	´XX´	CLA Byte gemäß [7816–4]
INS	´XX´	Instruction Byte gemäß [7816–4]
P1	´XX´	Erster Parameter
P2	´XX´	Zweiter Parameter
Data	´XX…YY´	Datenfeld mit beliebigen Oktette, Anzahl Oktette aus der Menge [1, 65535]
Ne	´XX´	Ne aus der Menge {1, …, 65535, WildCardShort, WildCardExtended}
Notation		CmdApdu = Case4E(CLA, INS, P1, P2, Data, Ne)

Gemäß den Regeln aus Kapitel 12.5 werden CLA, INS, P1 und P2 in jeweils einem Oktett kodiert. Damit gilt für die gesamte Case 4 Extended Kommando APDU:

(N291) Eine Case 4 Extended Kommando APDU MUSS aus neun Oktette plus den Oktette aus dem Datenfeld bestehen.

(N292) Die Anzahl Nc der Oktette des Datenfeldes MUSS in zwei Oktett kodiert werden, welche als LcFeld bezeichnet werden: LcFeld = I2OS(Nc, 2).

(N293) Der Wert von Ne wird wie folgt in zwei Oktett kodiert, welche das LeFeld bilden:

a. Wenn Ne im Intervall [256, 65535] liegt, dann MUSS diese ganze Zahl in zwei Oktette kodiert werden: LeFeld = I2OS(Ne, 2).

b. Wenn Ne den Wert WildCardExtended besitzt, dann MUSS das LeFeld gleich ´0000´ sein.

(N294) Die neun plus Nc Oktette einer Case 4 Extended Kommando APDU MÜSSEN wie folgt konkateniert werden:
CLA || INS || P1 || P2 || ´00´ || LcFeld || Datenfeld || LeFeld.

Hinweis (42): Das Oktett ´00´ nach dem Parameter P2 lässt sich als Indikator für „extended length“ auffassen.

12.7.4.3 Case 4 Response

Die zu einer Case 4 Kommando APDU gehörende Antwort APDU besteht aus dem optionalen Datenfeld und aus dem Trailer.

Tabelle 20: Case 2 Response APDU

Inhalt	Beschreibung
Daten	optionaler Bestandteil der Response APDU, falls vorhanden, dann gelten die Bestimmungen aus Kapitel 12.6.1
Trailer	Statusbytes SW1 und SW2

12.8 Übertragungsprotokoll

(N295) Als Übertragungsprotokoll MUSS an der physikalischen Schnittstelle (siehe Abbildung 1) T=1 gemäß [7816–3] Kapitel 11 verwendet werden.

(N296) Für die „Information Field Size“ IFSC und IFSD (siehe [7816–3] Kapitel 11.4.2) MÜSSEN die Werte ´FE´ = 254 verwendet werden.

(N297) Das NAD Oktett in TPDU an die Karte MUSS den Wert ´00´ haben.
Das COS KANN andere Werte akzeptieren.
Das COS KANN andere Werte ablehnen.

(N298) S–Block ABORT KANN vom COS verwendet werden, wenn die I/O Puffergröße nicht beachtet wurde.

13 Kanalkontext (normativ)

Während das Objektsystem gemäß Kapitel 10 die Information bündelt, die persistent in der Chipkarte zu speichern sind und in allen logischen Kanälen gleichermaßen zur Verfügung stehen, wird in diesem Kapitel auf die Information eingegangen, die lediglich zwischen Öffnen und Schließen eines logischen Kanals zur Verfügung steht und damit kanalspezifisch ist. Die kanalspezifischen Attribute werden im Objekt *channelContext* zusammengefasst.

Die im Folgenden genannten Attribute werden typischerweise einem flüchtig (im RAM) gespeicherten Security Environment (siehe Kapitel 9.7) oder Sicherheitsstatus (siehe Kapitel 9.8) zugerechnet. oder durch 15.9.6 MSE verändert.

13.1 Attribute eine logischen Kanals

An der physikalischen Schnittstelle (siehe Abbildung 1) verhält sich das COS so, als gäbe es pro logischen Kanal ein Objekt *channelContext* mit den folgenden Attributen:

(N299) Folgende Attribute von *channelContext* sind dem Objektsystem zugeordnet: Der *channelContext* MUSS

a. genau ein Attribut *currentFolder* enthalten, dass auf ein Objekt vom Typ Ordner zeigt.

b. genau ein Attribut *RND.ICC*, welches eine vom COS erzeugte Zufallszahl speichert (siehe (N993)).

c. genau eine Liste *keyReferenceList* besitzen. Jedes Listenelement MUSS einen Wert unterstützen, der anzeigt, dass es leer ist. Die Liste *keyReferenceList* MUSS sich aus folgenden Elementen zusammensetzen:

   i. externalAuthenticate, mit den Komponenten keyReference und algorithmIdentifier.

   ii. internalAuthenticate, mit den Komponenten keyReference und algorithmIdentifier.

   iii. verifyCertificate, mit der Komponente keyReference.

   iv. signatureCreation, mit den Komponenten keyReference und algorithmIdentifier.

   v. dataDecipher, mit den Komponenten keyReference und algorithmIdentifier.

d. genau ein Attribut *SessionkeyContext*, welches folgende Elemente enthält:

   i. *flagSessionEnabled* vom Typ Boolean. Der Wert

      1. True zeigt an, dass die übrigen Attribute gebrauchsfertiges kryptographisches Material enthalten.

      2. False zeigt an, dass keine Sessionkeys ausgehandelt wurden.

   ii. *Kenc* ist ein symmetrischer Schlüssel zur Ver- und Entschlüsselung.

iii. *SSCenc* ist eine nicht negative, ganze Zahl, die als Send Sequence Counter im Zusammenhang mit *Kenc* verwendet wird.

iv. *Kmac* ist ein symmetrischer Schlüssel zur MAC Berechnung und MAC Verifikation.

v. *SSCmac* ist eine nicht negative, ganze Zahl, die als Send Sequence Counter im Zusammenhang mit *Kmac* verwendet wird.

vi. *flagCmdEnc* ist eine boolesche Variable, welche anzeigt, ob die gesicherte Kommando APDU ein Datenobjekt mit verschlüsselten Kommandodaten enthält. Dieses Flag wird in (N317)b.i mit einem Wert versehen und in (N226)b ausgewertet.

vii. *flagRspEnc* ist eine boolesche Variable, welche anzeigt, ob die Daten einer Antwort APDU verschlüsselt übertragen werden. Das Flag wird in (N226)c mit einem Wert versehen und in (N322) ausgewertet.

e. genau eine Liste *globalSecurityList* besitzen.

   i. Das COS MUSS alle Werte des Intervalls [0, 3] für die Länge dieser Liste unterstützen.
   Ein COS KANN längere Listen unterstützen.

f. genau eine Liste *dfSpecificSecurityList* besitzen.

   i. Das COS MUSS alle Werte des Intervalls [0, 3] für die Länge dieser Liste unterstützen.
   Ein COS KANN längere Listen unterstützen.

g. Jedes Element der Liste *globalSecurityList* und jedes Element der Liste *dfSpecificSecurityList* MUSS entweder

   i. ein *CHA* gemäß (N197)a sein in Verbindung mit einer Referenz zu einem Ordner, wodurch angezeigt wird, dass eine erfolgreiche Komponentenauthentisierung gemäß (N844)f oder (N844)g stattgefunden hat mit einem Schlüsselobjekt aus vorgenanntem Ordner, dem dieses Schlüsselobjekt zugeordnet ist (siehe (N216) und (N959)a.ix)., oder

   ii. eine Referenz auf ein symmetrisches Authentisierungsobjekt (siehe Kapitel 9.5.1) sein, wodurch angezeigt wird, dass eine erfolgreiche Komponentenauthentisierung gemäß (N844)a, (N844)b, (N844)c oder (N844)d stattgefunden hat.

h. genau eine Liste *bitSecurityList* besitzen … (für Generation 2 vervollständigen)

i. genau eine Liste *globalPasswordList* besitzen.

Kom 3002

   i. Das COS MUSS alle Werte des Intervalls [0, 2] für die Länge dieser Liste unterstützen.
   Ein COS KANN längere Listen unterstützen.

j. genau eine Liste *dfSpecificPasswordList* besitzen.

   i. Das COS MUSS alle Werte des Intervalls [0, 2] für die Länge dieser Liste unterstützen.
   Ein COS KANN längere Listen unterstützen.

k. Jedes Element der Liste *globalPasswordList* und jedes Element der Liste *dfSpecificPasswordList* MUSS genau ein Attribute *securityStatusEvaluation-*

Counter sein in Verbindung mit einer Referenz auf ein Passwortobjekt, wodurch angezeigt wird, dass eine erfolgreiche Benutzerverifikation gemäß Kapitel 15.6.6.1 mit diesem Passwortobjekt stattgefunden hat. Der Wertebereich von *securityStatusEvaluationCounter* MUSS alle Werte von *startSsec* (siehe (N158)b.ii) umfassen und KANN den Wert null beinhalten.

(N300) Folgende Attribute von *channelContext* sind einem Ordner zugeordnet: Der *channelContext* MUSS für jeden Ordner im Objektsystem

a. genau ein Attribut *seIdentifier* gemäß (N79) enthalten.

b. genau ein Attribut *currentEF* enthalten, dass

   i. entweder unbestimmt ist,

   ii. oder auf ein Listenelement von *currentFolder.children* vom Typ Datei zeigt.

13.2 Reset Verhalten

Ein „neuer" Kanalkontext gemäß Kapitel 13.1 wird dann etabliert, wenn ein Reset durchgeführt wird, oder wenn ein neuer logischer Kanal geöffnet wird. dann gilt:

(N301) Wenn der logische Kanal mit der Nummer null (Basiskanal) durch (N239) oder (N240) geöffnet wird, oder ein anderer logischer Kanal durch Kapitel 15.9.5 geöffnet wird, dann MUSS für den neu geöffneten Kanal gelten:

a. Dem neu geöffneten logischen Kanal wird exklusiv ein Kanalkontext gemäß Kapitel 13.1 zugeordnet.

b. Das Attribut *currentFolder* MUSS auf *root* gesetzt werden.

c. Das Attribut *RND.ICC* KANN herstellerspezifisch behandelt werden.

d. Das Attribut *keyReferenceList* MUSS so gesetzt werden, dass alle Elemente leer sind.

e. Das Attribut *SessionkeyContext.flagSessionEnabled* MUSS auf False gesetzt werden.

f. Die Liste *globalSecurityList* MUSS leer sein.

g. Die Liste *dfSpecificSecurityList* MUSS leer sein.

h. Das Attribut *bitSecurityList* … (für Generation 2 vervollständigen)

i. Die Liste *globalPasswordList* MUSS leer sein.

j. Die Liste *dfSpecificPasswordList* MUSS leer sein.

k. Für alle Ordner gilt:

   i. *seIdentifier* MUSS auf den Wert 1 gesetzt werden.

   ii. *currentEF* MUSS auf den Wert „unbestimmt" gesetzt werden.

13.3 Setzen eines Sicherheitsstatus

Die hier beschriebene Routine setzt den Sicherheitsstatus des als Parameter übergebenen Authentisierungsschlüssels.

Input:	*obj*	ein Schlüsselobjekt
Output:	–	kein Rückgabewert
Notation:		setSecurityStatus(*obj*)

(N302) Falls *obj*

a. im Ordner *root* (siehe (N199)a) in der Liste *children* eingetragen ist, dann gilt *tmpList* = *globalSecurityList* (siehe (N299)e).

b. in einem anderen Ordner zugeordnet ist, dann gilt *tmpList* = *dfSpecificSecurityList* (siehe (N299)f).

(N303) Falls *obj*

a. ein symmetrisches Authentisierungsobjekt (siehe Kapitel 9.5.1) ist und *obj* in der Liste *tmpList*

i. bereits vorhanden ist, dann beende diesen Algorithmus.

ii. noch nicht vorhanden ist, dann MUSS *obj* am Anfang von *tmpList* eingetragen werden.

b. ein öffentliches Authentisierungsobjekt (siehe Kapitel 9.5.3.2) ist und *obj.publicKey* ein RSA Schlüssel ist und *obj.CHA* (siehe (N197)a) in der Liste *tmpList*

i. bereits vorhanden ist, dann beende diesen Algorithmus.

ii. noch nicht vorhanden ist, dann MUSS *obj.CHA* am Anfang von *tmpList* eingetragen werden.

c. Falls *tmpList* durch Eintragungen länger wurde, als vom COS unterstützt, dann MUSS das COS das letzte Listenelement entfernen (FIFO = first in first out).

(N304) Falls *obj* ein öffentliches Authentisierungsobjekt (siehe Kapitel 9.5.3.2) ist und *obj.publicKey* ein ELC Schlüssel ist, dann (N197)b … (für Generation 2 vervollständigen)

13.4 Löschen eines Sicherheitsstatus

Die hier beschriebene Routine löscht den Sicherheitsstatus des als Parameter übergebenen Authentisierungsschlüssels.

Input:	*obj*	ein Schlüsselobjekt

Output: – kein Rückgabewert

Notation: clearSecurityStatus(*obj*)

(N305) Falls *obj*

a. im Ordner *root* (siehe (N199)a) in der Liste *children* eingetragen ist, dann gilt *tmpList* = *globalSecurityList* (siehe (N299)e).

b. in einem anderen Ordner zugeordnet ist, dann gilt *tmpList* = *dfSpecificSecurityList* (siehe (N299)f).

(N306) Falls *obj*

a. ein symmetrisches Authentisierungsobjekt (siehe Kapitel 9.5.1) ist und *obj* in der Liste *tmpList*

i. nicht vorhanden ist, dann beende diesen Algorithmus.

ii. noch vorhanden ist, dann MUSS *obj* aus *tmpList* entfernt werden.

b. ein öffentliches Authentisierungsobjekt (siehe Kapitel 9.5.3.2) ist und *obj.publicKey* ein RSA Schlüssel ist und *obj.CHA* (siehe (N197)a) in der Liste *tmpList*

i. nicht vorhanden ist, dann beende diesen Algorithmus.

ii. noch vorhanden ist, dann MUSS *obj.CHA* aus *tmpList* entfernt werden.

c. Falls der Sicherheitszustand eines Schlüssels gelöscht wurde, der an der Aushandlung von Sessionkeys beteiligt war, dann MUSS *flagSessionEnabled* (siehe (N299)d.i) auf den Wert False gesetzt werden.

(N307) Falls *obj* ein öffentliches Authentisierungsobjekt (siehe Kapitel 9.5.3.2) ist und *obj.publicKey* ein ELC Schlüssel ist, dann (N197)b … (für Generation 2 vervollständigen).

Kom
3002

13.5 Setzen eines Passwortstatus

Die hier beschriebene Routine setzt den Sicherheitsstatus des als Parameter übergebenen Passwortes.

Input: *obj* ein Passowrtobjekt

Output: – kein Rückgabewert

Notation: setPasswordStatus(*obj*)

(N308) Falls *obj*

a. im Ordner *root* (siehe (N199)a) in der Liste *children* eingetragen ist, dann gilt *tmpList* = *globalPasswordList* (siehe (N299)i).

b. in einem anderen Ordner zugeordnet ist, dann gilt *tmpList* = *dfSpecificPasswordList* (siehe (N299)j).

(N309) Falls *obj* in der Liste *tmpList* noch nicht vorhanden ist, dann MUSS *obj* am Anfang von *tmpList* eingetragen werden.

(N310) Falls *tmpList* durch Eintragungen länger wurde, als vom COS unterstützt, dann MUSS das COS das letzte Listenelement entfernen (FIFO = first in first out).

(N311) Im Eintrag zu *obj* in *tmpList* MUSS das Attribute *securityStatusEvaluationCounter* auf den Wert *obj.startSsec* gesetzt werden.

Kom
3002

13.6 Löschen eines Sicherheitsstatus

Die hier beschriebene Routine löscht den Sicherheitsstatus des als Parameter übergebenen Passwortes.

Input:	*obj*	ein Passwortobjekt
Output:	–	kein Rückgabewert
Notation:		clearPasswordStatus(*obj*)

(N312) Falls *obj*

a. im Ordner *root* (siehe (N199)a) in der Liste *children* eingetragen ist, dann gilt *tmpList* = *globalSecurityList* (siehe (N299)i).

b. in einem anderen Ordner zugeordnet ist, dann gilt *tmpList* = *dfSpecificSecurityList* (siehe (N299)j).

(N313) Falls *obj* in der Liste *tmpList*

a. nicht vorhanden ist, dann beende diesen Algorithmus.

b. noch vorhanden ist, dann MUSS *obj* aus *tmpList* entfernt werden.

14 Gesicherte Kommunikation (normativ)

14.1 Secure Messaging Layer

Dieses Unterkapitel beschreibt die Funktionsweise des Layers „SecMes“ in Abbildung 1. Dieser Layer benutzt neben den Informationen aus (N299)d folgende weitere Attribute:

- KD.i ist ein Oktettstring, der eine vom COS generierte Zufallszahl speichert, die im Rahmen der Ableitung von Sessionkeys verwendet wird.
- KD.e ist ein Oktettstring, der eine extern generierte Zufallszahl speichert, die im Rahmen der Ableitung von Sessionkeys verwendet wird.

14.1.1 Ableitung von Sessionkeys

Input: – der benötigte Input wird dem *channelContext* entnommen

Output: – kein Output vorhanden

Errors: – keine

Notation: SessionkeyDerivation()

Im folgenden gelten die Definitionen:
algId = *channelContext.keyReferenceList.internalAuthenticate.algorithmIdentifier*

(N314) Falls *algId* die Aushandlung von 3TDES Schlüsseln anzeigt, gilt für die Attribute aus *SessionkeyContext*:

a. (*Kenc*,*SSCenc*,*Kmac*,*SSCmac*) = KeyDerivation_3TDES(KD.i XOR KD.e).

b. Setze SessionkeyContext.flagSessionEnabled auf den Wert True.

(N315) Falls *algId* die Aushandlung von AES–128 Schlüsseln anzeigt, gilt für die Attribute aus *SessionkeyContext*: … (für Generation 2 vervollständigen)

14.1.2 Bearbeitung einer Kommando APDU

(N316) Falls im CLA Byte kein Secure Messaging angezeigt wird (siehe [7816–4] Kapitel 5.1.1) und *SessionkeyContext.flagSessionEnabled* den Wert

a. True besitzt, dann

   i. MUSS *flagSessionEnabled* auf den Wert False gesetzt werden.

   ii. MUSS der Sicherheitszustand des zugehörigen Aushandlungsschlüssels zurückgesetzt werden.

   iii. MUSS CmdApdu3 = CmdApdu2 gesetzt werden.

b. False besitzt, dann MUSS CmdApdu3 = CmdApdu2 gesetzt werden.

(N317) Falls im CLA Byte Secure Messaging angezeigt wird und *flagSessionEnabled* besitzt den Wert

a. False, dann

i. MUSS die Bearbeitung des Kommandos terminieren.

ii. RspApdu2 enthält in diesem Fall keine Antwortdaten und besteht lediglich aus dem Trailer IncorrectSmDo = ´69 88´.

b. True, dann MUSS

i. die gesicherte CmdApdu2 den normativen Vorgaben aus Kapitel 14.2 entsprechen. Falls ein DO mit Tag = ´87´ in der CmdApdu2 vorhanden ist, MUSS *flagCmdEnc* auf True gesetzt werden, sonst auf False.
Das COS KANN weitere Secure Messaging Formate akzeptieren.
Das COS KANN weitere Secure Messaging Formate ablehnen.

ii. die gesicherte CmdApdu2 analog zu den Regeln aus Kapitel 14.2 entsichert werden. Das Ergebnis ist dann CmdApdu3. Falls dabei ein Fehler festgestellt wird, genau dann

1. MUSS *flagSessionEnabled* auf den Wert False gesetzt werden.

2. MUSS der Sicherheitszustand des zugehörigen Aushandlungsschlüssels zurückgesetzt werden (siehe Kapitel 13.4).

3. MUSS die Bearbeitung des Kommandos terminieren.

4. RspApdu2 enthält in diesem Fall keine Antwortdaten und besteht lediglich aus dem Trailer IncorrectSmDo = ´69 88´.

(N318) Die vom Secure Messaging Layer („SecMes“ in Abbildung 1) gegebenenfalls weitergeleitete CmdApdu3 MUSS gemäß den Vorgaben aus Kapitel 15 bearbeitet werden, wobei eine Antwortnachricht RspApdu1 entsteht. In Kapitel 15 wird davon ausgegangen, dass der Secure Messaging Layer CmdApdu3 so aufbereitet, dass im CLA Byte kein Secure Messaging angezeigt wird und die Kanalnummer auf null gesetzt wurde.

(N319) Falls flagSessionEnabled den Wert

a. False besitzt, dann MUSS RspApdu2 = RspApdu1 gesetzt werden. Falls sowohl KD.i, als auch KD.e vorhanden sind (einer oder beide Werte wurden mit dem gerade bearbeiteten Kommando gesetzt), dann

i. MUSS *SessionkeyContext* mittels SessionkeyDerivation() geändert werden (siehe Kapitel 14.1).

ii. MÜSSEN KD.i und KD.e auf den Wert „nicht vorhanden“ gesetzt werden.

b. True besitzt, dann MUSS die ungesicherte RspApdu1 gemäß Kapitel 14.3 in die gesicherte RspApdu2 umgewandelt werden.

14.2 Sicherung einer Kommando APDU

Dieses Unterkapitel beschreibt, wie eine Kommando APDU gemäß Kapitel 12.5 zu sichern ist. Die hier beschriebenen Regeln richten sich an die Instanz, welche Kommando APDU sendet. Das COS führt die entsprechenden Umkehroperationen durch.

Generell wird hier ein Subset der Regeln aus [7816–4] Kapitel 6 beschrieben, wobei

- alle Kommando APDU mit Integritätsschutz übertragen werden,
- der Kommandoheader stets mit Integritätsschutz übertragen wird,
- Kommandodaten als Klartext oder verschlüsselt übertragen werden.

Hier sei angemerkt, dass der Sender zwar die Wahl hat zwischen der Übertragung von Kommandodaten im Klartext oder als Kryptogramm, aber die Zugriffsregel gegebenenfalls eine verschlüsselte Übertragung erzwingt.

Wie in Kapitel 12.5 beschrieben, besteht eine Kommando APDU generisch betrachtet aus den Oktette CLA, INS, P1 und P2 sowie aus dem optionalen Datenfeld Data und dem optionalen LeFeld. Somit gelten folgende Definitionen

Input:	CLA	Oktett gemäß (N265)
	INS	Oktett gemäß (N266)
	P1	Oktett gemäß (N267)
	P2	Oktett gemäß (N268)
	Data	optionaler Oktettstring gemäß Kapitel 12.5.5
	LeFeld	optionaler Oktettstring gemäß Kapitel 12.5.6, welcher gemäß (N276) oder (N279) in eine Zahl Ne enthält
	Kenc	symmetrischer Schlüssel für die Verschlüsselung, (N299)d.ii
	SSCenc	beliebige nicht negative Zahl, die als Send Sequence Counter bei der Verschlüsselung verwendet wird, (N299)d.iii
	Kmac	symmetrischer Schlüssel für die MAC Berechnung, (N299)d.iv
	SSCmac	beliebige nicht negative Zahl, die als Send Sequence Counter bei der MAC Berechnung verwendet wird, (N299)d.v
Output:	CmdApdu	gesicherte Kommando APDU
Errors:	–	keine

Zur Sicherung einer generischen Kommando APDU sind folgende Schritte durchzuführen:

(N320) Falls das optionale Datenfeld Data fehlt, gilt: ProtectedData = ´´ (leerer Oktett-string).

(N321) Falls das optionale Datenfeld Data vorhanden ist und im Klartext übertragen wird, gilt:

a. Setze lenPDO = OctetLength(Data), falls lenPDO im Intervall

i. [1, 127] liegt, gilt: LenP = I2OS(lenPDO, 1)

ii. [128, 255] liegt, gilt: LenP = ´81´ || I2OS(lenPDO, 1)

iii. [256, 65535] liegt, gilt: LenP = ´82´ || I2OS(lenPDO, 2)

b. Setze ProtectedData = ´81 || LenP || Data´

(N322) Falls das optionale Datenfeld Data vorhanden ist und verschlüsselt übertragen wird und Kenc ein 3TDES Schlüssel ist, gilt:

a. SSCenc = SSCenc + 1

b. C = 3TDES_CBC_ENC(Kenc, SSCenc, PaddingIso(Data, 8))

c. Setze lenCDO = OctetLength(C) + 1, falls lenCDO im Intervall

i. [1, 127] liegt, gilt: LenC = I2OS(lenCDO, 1)

ii. [128, 255] liegt, gilt: LenC = ´81´ || I2OS(lenCDO, 1)

iii. [256, 65535] liegt, gilt LenC = ´82´ || I2OS(lenCDO, 2)

d. Setze ProtectedData = ´87 || LenC || 01 || C´

(N323) Falls das optionale Datenfeld Data vorhanden ist und verschlüsselt übertragen wird und Kenc ein AES Schlüssel ist, gilt:

a. (C, SSCenc) = AES_CTR_ENC(Kenc, SSCenc, Data)

b. Setze lenCDO = OctetLength(C) + 1, falls lenCDO im Intervall

i. [1, 127] liegt, gilt: LenC = I2OS(lenCDO, 1)

ii. [128, 255] liegt, gilt: LenC = ´81´ || I2OS(lenCDO, 1)

iii. [256, 65535] liegt, gilt LenC = ´82´ || I2OS(lenCDO, 2)

c. Setze ProtectedData = ´87 || LenC || 02 || C´

(N324) Falls das optionale LeFeld

a. fehlt, gilt: LeDO = ´´, (leerer Oktettstring).

b. vorhanden ist, gilt: LeDO = ´97 || I2OS(OctetLength(LeFeld), 1) || LeFeld´

(N325) Setze: CLA’ = CLA OR ´0C´, die Bits b4 und b3 in CLA werden gesetzt Kom 3002

(N326) Setze: head = CLA’ || INS || P1 || P2

(N327) Setze: SSCmac = SSCmac + 1

(N328) Falls Kmac ein

a. 3TDES Schlüssel ist, gilt:

i. MACin = I2OS(SSCmac, 8)

ii. MACin = MACin || PaddingIso(head, 8)

iii. MACin = MACin || ProtectedData

iv. MACin = MACin || LeDO

v. MAC = CALCULATE_Retail_MAC(Kmac, MACin)

b. AES Schlüssel ist, gilt:

i. MACin = I2OS(SSCmac, 16)

ii. MACin = MACin || PaddingIso(head, 16)

iii. MACin = MACin || ProtectedData

iv. MACin = MACin || LeDO

v. MAC = CALCULATE_CMAC(Kmac, MACin)

(N329) Setze: MDO = ´8E || I2OS(OctetLength(MAC), 1) || MAC´

(N330) Setze: newD = ProtectedData || LeDO || MDO

Kom 3002

(N331) Case 1: Falls Data und LeFeld fehlen, dann setze:
CmdApdu = Case4S(CLA', INS, P1, P2, newD, WildCardShort)

(N332) Case 2: Falls Data fehlt und LeFeld vorhanden ist, dann setze:
CmdApdu = Case4E(CLA', INS, P1, P2, newD, WildCardExtended)

(N333) Case 3: Falls Data vorhanden ist und LeFeld fehlt und OctetLength(newD)

Kom 3002

a. kleiner gleich 255 ist, dann gilt:
CmdApdu = Case4S(CLA', INS, P1, P2, D, WildCardShort)

b. größer gleich 256 ist, dann gilt:
CmdApdu = Case4E(CLA', INS, P1, P2, D, WildCardExtended)

(N334) Case 4: Falls Data und LeFeld vorhanden sind, dann setze:
CmdApdu = Case4E(CLA', INS, P1, P2, newD, WildCardExtended)

Hinweis (43): Zu (N331) und (N333)a: Gemäß Kapitel 12.7.1 (12.7.3) hat die korrespondierende Response APDU zu einer Case 1 (Case 3) Kommando APDU niemals Antwortdaten. Zudem werden in Kapitel 14.3 ausschließlich symmetrischen Verfahren verwendet. Daraus folgt, dass eine gesicherte Response APDU zu einer ungesicherten Case 1 (Case 3) APDU niemals mehr als 256 Byte Antwortdaten enthält. Deshalb wird hier für die gesicherte Case 1 (Case 3) Kommando APDU eine Case 4 Short Kommando APDU verwendet.

Hinweis (44): Zu (N332) und (N334): Gemäß Kapitel 12.7.2.1 (12.7.4.1) hat die korrespondierende Response APDU zu einer Case 2 Short (Case 4 Short) Kommando APDU bis zu 256 Oktette Antwortdaten. Gemäß Kapitel 14.3 ist es deshalb möglich, dass eine gesicherte Response APDU zu einer ungesicherten Case 2 Short (Case 4 Short) APDU mehr als 256 Byte Antwortdaten enthält. Deshalb wird hier auch für die gesicherte Case 2 Short (Case 4 Short) Kommando APDU eine Case 4 Extended Kommando APDU verwendet.

14.3 Sicherung einer Response APDU

Dieses Unterkapitel beschreibt, wie eine Response APDU gemäß Kapitel 12.6 zu sichern ist. Die hier beschriebenen Regeln richten sich an das COS. Die Instanz, welche die korrespondierende Kommando APDU gesendet hat führt die entsprechenden Umkehroperationen durch.

Generell wird hier ein Subset der Regeln aus [7816–4] Kapitel 6 beschrieben, wobei

- alle Response APDU mit Integritätsschutz übertragen werden,
- der Trailer wird stets mit Integritätsschutz übertragen
- Responsedaten als Klartext oder verschlüsselt übertragen werden.

Hier sei angemerkt, dass das COS anhand der verwendeten Zugriffsbedingung entscheidet, ob vorhandene Responsedaten im Klartext oder als Kryptogramm übertragen werden.

Wie in Kapitel 12.6 beschrieben, besteht eine Response APDU generisch betrachtet aus dem optionalen Datenfeld Data und dem Trailer. Somit gelten folgende Definitionen:

Input:	Data	optionaler Oktettstring gemäß Kapitel 12.6.1
	Trailer	Oktettstring gemäß Kapitel 12.6.2
	Kenc	symmetrischer Schlüssel für die Verschlüsselung, (N299)d.ii
	SSCenc	beliebige nicht negative Zahl, die als Send Sequence Counter bei der Verschlüsselung verwendet wird, (N299)d.iii
	Kmac	symmetrischer Schlüssel für die MAC Berechnung, (N299)d.iv
	SSCmac	beliebige nicht negative Zahl, die als Send Sequence Counter bei der MAC Berechnung verwendet wird, (N299)d.v
Output:	RspApdu	gesicherte Response APDU
Errors:	–	keine

Zur Sicherung einer generischen Response APDU sind folgende Schritte durchzuführen:

(N335) Falls das optionale Datenfeld Data fehlt, gilt: ProtectedData = ´´ (leerer Oktettstring).

(N336) Falls das optionale Datenfeld Data vorhanden ist und im Klartext übertragen wird, gilt:

a. Setze lenPDO = OctetLength(Data), falls lenPDO im Intervall
   i. [1, 127] liegt, gilt: LenP = I2OS(lenPDO, 1)
   ii. [128, 255] liegt, gilt: LenP = ´81´ || I2OS(lenPDO, 1)
   iii. [256, 65535] liegt, gilt: LenP = ´82´ || I2OS(lenPDO, 2)

b. Setze ProtectedData = ´81 || LenP || Data´

(N337) Falls das optionale Datenfeld Data vorhanden ist und verschlüsselt übertragen wird (siehe (N299)d.vii und (N226)c) und Kenc ein 3TDES Schlüssel ist, gilt:

a. SSCenc = SSCenc + 1

b. C = 3TDES_CBC_ENC(Kenc, SSCenc, PaddingIso(Data, 8))

c. Setze lenCDO = OctetLength(C) + 1, falls lenCDO im Intervall
   i. [1, 127] liegt, gilt: LenC = I2OS(lenCDO, 1)
   ii. [128, 255] liegt, gilt: LenC = ´81´ || I2OS(lenCDO, 1)
   iii. [256, 65535] liegt, gilt LenC = ´82´ || I2OS(lenCDO, 2)

d. Setze ProtectedData = ´87 || LenC || 01 || C´

(N338) Falls das optionale Datenfeld Data vorhanden ist und verschlüsselt übertragen wird (siehe (N299)d.vii und (N226)c) und Kenc ein AES Schlüssel ist, gilt:

a. (C, SSCenc) = AES_CTR_ENC(Kenc, SSCenc, Data)

b. Setze lenCDO = OctetLength(C) + 1, falls lenCDO im Intervall

i. [1, 127] liegt, gilt: LenC = I2OS(lenCDO, 1)

ii. [128, 255] liegt, gilt: LenC = ´81´ || I2OS(lenCDO, 1)

iii. [256, 65535] liegt, gilt LenC = ´82´ || I2OS(lenCDO, 2)

c. Setze ProtectedData = ´87 || LenC || 02 || C´

(N339) Setze TDO = ´99 02 || Trailer´

(N340) Setze: SSCmac = SSCmac + 1

(N341) Falls Kmac ein

a. 3TDES Schlüssel ist, gilt:

i. MACin = I2OS(SSCmac, 8)

ii. MACin = MACin || ProtectedData

iii. MACin = MACin || TDO

iv. MAC = CALCULATE_Retail_MAC(Kmac, MACin)

b. AES Schlüssel ist, gilt:

i. MACin = I2OS(SSCmac, 16)

ii. MACin = MACin || ProtectedData

iii. MACin = MACin || TDO

iv. MAC = CALCULATE_CMAC(Kmac, MACin)

(N342) Setze: MDO = ´8E || I2OS(OctetLength(MAC), 1) || MAC´

(N343) Für die gesicherte Response APDU RspApdu gilt:

a. RspApdu.Datenfeld = ProtectedData || TDO || MDO

b. RspApdu.Trailer = Trailer

15 Kommandos (normativ)

(N344) Das COS KANN Kombinationen aus CLA, INS, P1 und P2 unterstützen, die in diesem Kapitel nicht aufgeführt sind.

15.1 Roll–Verhalten

In den folgenden Unterkapiteln ist gelegentlich davon die Rede, dass der persistente Speicher mittels „Roll–Forward“ oder „Roll–Back“ zu verändern ist. Als Oberbegriff wird „Transaktionsschutz“ verwendet um auszudrücken, dass Roll–Forward oder Roll–Back gemeint ist. Kurz gesagt verbirgt sich dahinter folgendes: Das persistente Speichern von Informationen dauert aus technischen Gründen einige Millisekunden. Da Chipkarten aus technischen Gründen nicht in der Lage sind, einen Ausfall der Spannungsversorgung zu puffern, ist es denkbar, dass der Ausfall zu einem Zeitpunkt geschieht, in welchem der Zustand des persistenten Speichers in einem undefinierten Zustand ist. Der Transaktionsschutz legt dann fest, wie mit diesem möglicherweise undefiniertem Zustand umzugehen ist.

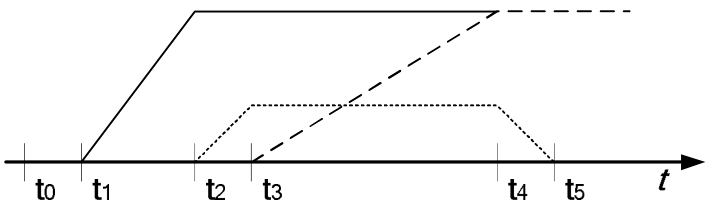

Abbildung 2: Zeitlicher Ablauf eines Roll–Back Kommandos

Für die Bearbeitung eines Kommandos mit Transaktionsschutz werden, wie in Abbildung 2 gezeigt, die folgenden Zeitpunkte definiert:

- Zum Zeitpunkt t_0 werde das erste Bit der Kommando APDU über die physikalische Schnittstelle (siehe Abbildung 1) zur Chipkarte gesendet.
- Zum Zeitpunkt t_1 sei die Kommandobearbeitung so weit vorgeschritten, dass der Zwischenspeicher bereit ist befüllt zu werden.
- Zum Zeitpunkt t_2 sei der Zwischenspeicher komplett befüllt, aber sein Inhalt noch nicht als gültig gekennzeichnet.
- Zum Zeitpunkt t_3 sei der Inhalt als gültig gekennzeichnet und die eigentliche persistente Änderung werde gestartet.
- Zum Zeitpunkt t_4 sei die eigentliche persistente Änderung abgeschlossen, aber der Inhalt des Zwischenpuffers sei noch als gültig gekennzeichnet.
- Zum Zeitpunkt t_5 sei der Inhalt des Zwischenspeicher als ungültig gekennzeichnet.
- Zum Zeitpunkt t_e (in Abbildung 2 nicht dargestellt) habe die Chipkarte das letzte Bit der Antwort APDU über die physikalische Schnittstelle versendet.

(N345) Dieses Dokument legt nicht fest, in welchem zeitlichen Zusammenhang t_e zu den anderen Zeiten steht. Damit KANN t_e zu einem beliebigen Zeitpunkt nach t_0 erfolgen.

15.1.1 Roll–Back

Roll–Back legt fest, dass die durch den Spannungsausfall unterbrochene Aktion rückgängig zu machen ist, wenn wieder eine Versorgungsspannung anliegt. Typischerweise wird dazu vor Durchführung der Änderung der *ursprüngliche* Inhalt in einen Zwischenspeicher geschrieben, dann die Änderung durchgeführt und anschließend der Inhalt des Zwischenspeichers als ungültig gekennzeichnet.

(N346) Findet der Ausfall der Versorgungsspannung

a. vor t_4 statt, so ist entweder noch keine Änderung erfolgt, oder der Inhalt des Zwischenpuffers definitiv auf gültig gesetzt und der (hier ursprüngliche) Inhalt des Zwischenspeichers MUSS nach Wiederanlegen der Versorgungsspannung wiederhergestellt werden.

b. nach t_5 statt, so ist der Inhalt des Zwischenpuffers definitiv auf ungültig gesetzt und damit eine Wiederherstellung des ursprünglichen Zustandes unmöglich.

c. zwischen t_4 und t_5 statt, so hängt es vom Zufall ab, ob der Zustand des Zwischenspeichers als gültig oder ungültig beurteilt wird (physikalische Speicher haben mitnichten ein zeit- oder wertediskretes Verhalten).
In diesem Fall KANN der ursprüngliche Zustand rekonstruiert werden.
In diesem Fall KANN der neue Zustand beibehalten werden.

15.1.2 Roll–Forward

Roll–Forward legt fest, dass die durch den Spannungsausfall unterbrochene Aktion fortzusetzen ist, wenn wieder eine Versorgungsspannung anliegt. Typischerweise wird dazu vor Durchführung der Änderung der *neue* Inhalt in einen Zwischenspeicher geschrieben, dann die Änderung durchgeführt und anschließend der Inhalt des Zwischenspeichers als ungültig gekennzeichnet.

(N347) Findet der Ausfall der Versorgungsspannung

a. vor t_2 statt, so ist der Inhalt des Zwischenspeichers definitiv nicht auf gültig gesetzt und damit ist ein Wechsel zum neuen Zustand unmöglich.

b. nach t_3 statt, so ist der Inhalt des Zwischenpuffers definitiv auf gültig gesetzt und der (hier neue) Inhalt des Zwischenspeichers MUSS nach Wiederanlegen der Versorgungsspannung wiederhergestellt werden.

c. zwischen t_2 und t_3 statt, so hängt es vom Zufall ab, ob der Zustand des Zwischenspeichers als gültig oder ungültig beurteilt wird (physikalische Speicher haben mitnichten zeit- oder wertediskretes Verhalten).
In diesem Fall KANN der ursprüngliche Zustand beibehalten werden.
In diesem Fall KANN der neue Zustand gesetzt werden.

15.2 Management des Objektsystems

15.2.1 ACTIVATE

Das Kommando ACTIVATE aktiviert reversibel ein File. Das betroffene File wird vor der Operation ausgewählt. Dies geschieht vor dem Senden dieses ACTIVATE Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*).

15.2.1.1 Use Case Aktivieren eines Ordners oder einer Datei

In dieser Variante enthält die APDU des ACTIVATE Kommandos keinen Parameter:

(N348) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 21 verwendet werden.

Tabelle 21: ACTIVATE aktuelles File

		Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´44´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–

15.2.1.2 Antwort der Karte auf Aktivieren eines Files

Tabelle 22: ACTIVATE Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Aktivierung
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 23: ACTIVATE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (45): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N349) Ein COS KANN zusätzliche Trailer verwenden.

15.2.1.3 Kommandoabarbeitung innerhalb der Karte

(N350) Das COS MUSS die ACTIVATE Variante aus 15.2.1.1 unterstützen.
Das COS KANN weitere ACTIVATE Varianten unterstützen.
Das COS KANN weitere ACTIVATE Varianten ablehnen.

(N351) Falls currentEF

a. auf eine Datei zeigt, dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

b. andernfalls MUSS *affectedObject* gleich *currentFolder* gesetzt werden.

(N352) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N353) Wenn der physikalische Wert von *lifeCycleStatus* von *affectedObject* bereits den Wert „Operational state (activated)“ besitzt, dann MUSS als Trailer NoError verwendet werden.

(N354) Der physikalische Wert von *lifeCycleStatus* von *affectedObject* MUSS mittels Transaktionsschutz auf den Wert „Operational state (activated)“ gesetzt werden.

(N355) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N356) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N357) Andernfalls MUSS als Trailer NoError gewählt werden.

(N358) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 23 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 23 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.2.2 CREATE

(N359) Das COS KANN dieses Kommando gemäß [7816–9] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–9] ablehnen.

Hinweis (46): Die entsprechende Funktionalität dieses Kommandos wird im Rahmen dieses Dokumentes durch das Kommando LOAD APPLICATION (siehe Kapitel 15.2.5) bereitgestellt.

15.2.3 DEACTIVATE

Das Kommando DEACTIVATE deaktiviert reversibel ein File. Das betroffene File wird vor der Operation ausgewählt. Dies geschieht vor dem Senden dieses DEACTIVATE Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*).

15.2.3.1 Use Case Deaktivieren eines Ordners oder einer Datei

In dieser Variante enthält die APDU des DEACTIVATE Kommandos keinen Parameter:

(N360) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 24 verwendet werden.

Tabelle 24: DEACTIVATE aktuelles File

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´04´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–

15.2.3.2 Antwort der Karte auf Deaktivieren eines Files

Tabelle 25: DEACTIVATE Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Deaktivierung
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 26: DEACTIVATE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (47): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N361) Ein COS KANN zusätzliche Trailer verwenden.

15.2.3.3 Kommandoabarbeitung innerhalb der Karte

(N362) Das COS MUSS die DEACTIVATE Variante aus 15.2.3.1 unterstützen.
Das COS KANN weitere DEACTIVATE Varianten unterstützen.
Das COS KANN weitere DEACTIVATE Varianten ablehnen.

(N363) Falls currentEF

a. auf eine Datei zeigt, dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

b. andernfalls MUSS *affectedObject* gleich *currentFolder* gesetzt werden.

(N364) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N365) Wenn der physikalische Wert von *lifeCycleStatus* von *affectedObject* bereits den Wert „Operational state (deactivated)" besitzt, dann MUSS als Trailer NoError verwendet werden.

(N366) Der physikalische Wert von *lifeCycleStatus* von *affectedObject* MUSS mittels Transaktionsschutz auf den Wert „Operational state (deactivated)" gesetzt werden.

(N367) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N368) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N369) Andernfalls MUSS als Trailer NoError gewählt werden.

(N370) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 26 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 26 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.2.4 DELETE

Das Kommando DELETE löscht Objekte vom Typ File aus dem Objektsystem. Das betroffene File wird vor der Operation ausgewählt. Dies geschieht vor dem Senden dieses DELETE Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*).

15.2.4.1 Use Case Löschen eines Ordners oder einer Datei

In dieser Variante enthält die APDU des DELETE Kommandos keinen Parameter:

(N371) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 27 verwendet werden.

Tabelle 27: DELETE aktuelles File

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´E4´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–

15.2.4.2 Antwort der Karte auf Löschen eines Files

Tabelle 28: DELETE Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Löschoperation
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 29: DELETE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (48): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N372) Ein COS KANN zusätzliche Trailer verwenden.

15.2.4.3 Kommandoabarbeitung innerhalb der Karte

(N373) Das COS MUSS die DELETE Variante aus 15.2.4.1 unterstützen.
Das COS KANN weitere DELETE Varianten unterstützen.
Das COS KANN weitere DELETE Varianten ablehnen.

(N374) Falls currentEF

a. auf eine Datei zeigt, dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

b. andernfalls MUSS *affectedObject* gleich *currentFolder* gesetzt werden.

(N375) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N376) Die Löschoperation MUSS mit Transaktionsschutz durchgeführt werden.

(N377) Wenn affectedObject vom Typ

a. Ordner ist, dann MUSS

   i. *currentEF* auf den Wert undefined gesetzt werden und

   ii. *currentFolder* auf den Vater von *affectedObject* gesetzt werden und

   iii. *affectedObject* inklusive Subtree gelöscht werden.

b. Datei ist, dann MUSS

   i. *currentEF* auf den Wert undefined gesetzt werden und

   ii. *affectedObject* gelöscht werden.

(N378) Der vormals von *affectedObject* allokierte Speicher MUSS so freigegeben werden, dass er zum Anlegen anderer Objekte verwendbar ist.

(N379) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N380) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N381) Andernfalls MUSS als Trailer NoError gewählt werden.

(N382) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 29 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 29 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N383) Im Fehlerfall MÜSSEN *currentFolder* und *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.2.5 LOAD APPLICATION

Das Kommando LOAD APPLICATION wird verwendet um neue Files im Objektsystem anzulegen. So ist es möglich
– einen neuen Ordner inklusive Subtree,
– eine neue Datei inklusive Inhalt (transparent oder strukturiert),
anzulegen. Das neu angelegte File wird im *currentFolder* gespeichert. Typischerweise ist die beim Anlegen neuer Files zur Karte transferierte Datenmenge so groß, dass sie nicht in einer einzigen Kommando APDU übertragbar ist. Deshalb unterstützt dieses Kommando „Command Chaining“.

15.2.5.1 Use Case Anlegen neues Objekt, nicht Ende der Kommandokette

Diese Variante ist zu wählen, wenn die Datenmenge nicht in einer Kommando APDU übertragbar ist. Sie kommt zum Einsatz von der ersten bis zur vorletzten Kommando APDU. Die letzte Kommando APDU wird gemäß 15.2.5.2 übertragen. In der hier beschriebenen Variante enthält die APDU des LOAD APPLICATION Kommandos zwei Parameter.

(N384) Das CLA–Byte zeigt an, dass diese Kommando APDU nicht die letzte einer Command Chaining Kette ist. Dies wird durch den Wert CLA = ´10´ kodiert.

(N385) Der Parameter *cmdData* enthält Daten, welche das neu anzulegende File beschreiben. Der Parameter *cmdData* ist ein Oktettstring mit beliebigem, herstellerspezifischem Inhalt. Die Länge von *cmdData* unterliegt nur den Beschränkungen aus Kapitel 12.5.5.

(N386) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 30 verwendet werden.

Tabelle 30: LOAD APPLICATION mit Command Chaining

	Inhalt	Beschreibung
CLA	´10´	CLA Byte gemäß [7816–4], Chaining Bit b5 gesetzt
INS	´EA´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–
Data	´XX…YY´	cmdData

15.2.5.2 Use Case Anlegen neues Objekt, Ende der Kommandokette

Diese Variante ist zu wählen, wenn die Datenmenge in einer Kommando APDU transferierbar ist, oder das letzte Kommando einer Chaining Kette zu übertragen ist. In dieser Variante enthält die APDU des LOAD APPLICATION Kommandos einen Parameter.

(N387) Der Parameter *cmdData* enthält Daten, welche das neu anzulegende File beschreiben. Der Parameter *cmdData* ist ein Oktettstring mit beliebigem, herstellerspezifischem Inhalt. Die Länge von *cmdData* unterliegt nur den Beschränkungen aus Kapitel 12.5.5.

(N388) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 31 verwendet werden.

Tabelle 31: LOAD APPLICATION ohne Command Chaining

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´EA´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–
Data	´XX…YY´	cmdData

15.2.5.3 Antwort der Karte auf Anlegen neues Objekt

Tabelle 32: LOAD APPLICATION Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreicher Ladevorgang
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 33: LOAD APPLICATION Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´6A 8A´	DfNameExists	AID des neuen Objektes wird bereits verwendet
´6A 89´	DuplicatedObject	Identifier des neuen Objektes wird bereits verwendet
´6A 84´	OutOfMemory	zu wenig Speicherplatz für neues Objekt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (49): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N389) Ein COS KANN zusätzliche Trailer verwenden.

15.2.5.4 Kommandoabarbeitung innerhalb der Karte

(N390) Das COS MUSS die LOAD APPLICATION Varianten aus 15.2.5.1 und 15.2.5.2 unterstützen.
Das COS KANN weitere LOAD APPLICATION Varianten unterstützen.
Das COS KANN weitere LOAD APPLICATION Varianten ablehnen.

(N391) Als *affectedObject* MUSS *currentFolder* verwendet werden.

(N392) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatus-NotSatisfied terminieren.

(N393) Das Kommando und damit eine eventuell aktive Chaining Kette KANN akzeptiert oder angelehnt werden, wenn das neu anzulegende File

a. ein Ordner ist und ein Attribut *applicationIdentifier* besitzt und dieser *applicationIdentifier* bereits einem anderen Ordner innerhalb des Objektsystems zugeordnet ist, oder

b. ein Ordner ist und ein Attribut *fileIdentifier* besitzt und dieser *fileIdentifier* bereits einem anderen File innerhalb von *currentFolder* zugeordnet ist, oder

c. eine Datei ist, deren Attribut *fileIdentifier* bereits einem anderen File innerhalb von *currentFolder* zugeordnet ist, oder

d. eine Datei ist, deren Attribut *shortFileIdentifier* bereits einer anderen Datei innerhalb von *currentFolder* zugeordnet ist.

(N394) Das Kommando MUSS akzeptiert werden, wenn das neu anzulegende Objekt

a. ein Ordner ist und ein Attribut *applicationIdentifier* besitzt und dieser *applicationIdentifier* keinem anderen Ordner innerhalb des Objektsystems zugeordnet ist,

b. ein Ordner ist und ein Attribut *fileIdentifier* besitzt und dieser *fileIdentifier* keinem anderen File innerhalb von *currentFolder* zugeordnet ist, oder

c. eine Datei ist, deren Attribut *fileIdentifier* keinem anderen File innerhalb von *currentFolder* zugeordnet ist, oder

d. eine Datei ist, deren Attribut *shortFileIdentifier* keiner anderen Datei innerhalb von *currentFolder* zugeordnet ist.

(N395) Wenn insgesamt ausreichender, aber nicht genügend zusammenhängender Speicherplatz vorhanden ist, dann MUSS das COS intern dafür sorgen, dass Kom 3002 diese Operation trotzdem erfolgreich durchführbar ist. Typischerweise wird diese Operation als „Defragmentieren“ bezeichnet.

(N396) Wenn nicht genügend Speicherplatz zum Anlegen des neuen Objektes vorhanden ist, genau dann MUSS das Kommando mit dem Trailer OutOfMemory terminieren.

(N397) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N398) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N399) Andernfalls MUSS als Trailer NoError gewählt werden.

(N400) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 33 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 33 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N401) Wenn dies das einzige oder letzte Kommando einer Command Chaining Kette ist und das neu angelegte Objekt ist

a. ein Ordner, dann MUSS im Erfolgsfall *currentFolder* auf den neu angelegten Ordner gesetzt werden. Dabei sind die Regeln zum Setzen des Kanalkontextes für den neuen Ordner zu berücksichtigen (siehe (N482)b).

b. eine Datei, dann MUSS im Erfolgsfall *currentEF* auf die neu angelegte Datei gesetzt werden.

(N402) Wenn das Kommando nicht erfolgreich verlief, dann DÜRFEN *currentFolder* und *currentEF* NICHT verändert werden.

(N403) Wenn dieses LOAD APPLICATION Kommando mit einem Trailer aus Tabelle 33 terminierte dann MUSS ein eventuell aktives Command Chaining abgebrochen werden.

(N404) Das neu angelegte bzw. neu anzulegende Objekt MUSS inklusive Freigabe eines eventuell von ihm allokierten Speichers aus dem Objektsystem gelöscht werden (komplettes Roll–Back der Chaining Kette), wenn

a. das LOAD APPLICATION Kommando mit einem Trailer aus Tabelle 33 terminierte, oder

b. das LOAD APPLICATION Kommando während der Kommandobearbeitung durch einen Reset abgebrochen wird, oder

c. eine Command Chaining Kette durch einen Reset abgebrochen wird.

Hinweis (50): Es ist denkbar, dass Anforderung (N395) wie folgt getestet wird:
a) Ausgangspunkt sei eine Chipkarte, deren Objektsystem nur eine sehr geringe (minimale) Anzahl von Objekten enthält.
b) Im ersten Schritt werde per LOAD APPLICATION Kommando eine Datei (transparent oder strukturiert wird zufällig bestimmt) mit 200 Byte Nutzdaten angelegt.
c) Schritt b) wird so lange wiederholt, bis das LOAD APPLICATION Kommando mit dem Trailer OutOfMemory terminiert.

d) Im zweiten Schritt werden zwei der zuvor angelegten Dateien zufällig ausgewählt und gelöscht (DELETE). Anschließend werde per LOAD APPLICATION eine neue Datei (transparent oder strukturiert wird zufällig bestimmt) angelegt. Wenn die Summe der Nutzdaten der in diesem Schritt gelöschten Dateien x ist, dann werde als Größe der Nutzdaten der in diesem Schritt neu angelegten Datei ebenfalls x gewählt. Es wird erwartet, dass dieses LOAD APPLICATION Kommando nicht mit OutOfMemory terminiert.
e) Schritt d) werde so lange wiederholt, bis nur noch eine Datei übrig ist, welche durch diesen Algorithmus angelegt wurde.

15.2.6 SELECT

Das Kommando SELECT sucht im Objektsystem nach einem File (Ordner oder Datei) und wählt dieses aus. Das Auswählen ist vielfach Voraussetzung, damit andere Use Cases (Lesen, Schreiben, …) erfolgreich durchführbar sind. Optional ist es möglich in den Antwortdaten die wesentlichen Attribute des Files zurückzumelden. Welches File selektiert wird, bestimmen Parameter in der Kommandonachricht.

15.2.6.1 Use Case Selektieren ohne AID, first, keine Antwortdaten

Diese Variante selektiert das Wurzelverzeichnis des Objektsystems. In dieser Variante enthält die APDU des SELECT Kommandos drei Parameter:

(N405) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N406) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N407) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N408) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 34 verwendet werden.

Tabelle 34: SELECT, kein AID, first occurrence, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier (hier leer)
P2	´0C´	*fileOccurrence* + *responseType* = first occurrence, keine Antwortdaten

15.2.6.2 Use Case Selektieren ohne AID, first, Antwortdaten mit FCP

Diese Variante selektiert das Wurzelverzeichnis des Objektsystems. In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N409) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N410) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N411) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N412) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N413) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 35 verwendet werden.

Tabelle 35: SELECT, kein AID, first occurrence, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier (hier leer)
P2	´04´	*fileOccurrence* + *responseType* = first occurrence, Antwortdaten mit FCP
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.3 Use Case Selektieren ohne AID, next, keine Antwortdaten

Der wiederholte Aufruf dieser Variante selektiert nacheinander alle Ordner, die das Attribut *applicationIdentifier* besitzen (Applikationen gemäß Kapitel 9.3.1.1 und ADF gemäß 9.3.1.3). In dieser Variante enthält die APDU des SELECT Kommandos drei Parameter:

(N414) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N415) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´2´ gewählt werden.

(N416) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N417) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 36 verwendet werden.

Tabelle 36: SELECT, kein AID, next occurrence, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier (hier leer)
P2	´0E´	*fileOccurrence* + *responseType* = next occurrence, keine Antwortdaten

15.2.6.4 Use Case Selektieren ohne AID, next, Antwortdaten mit FCP

Der wiederholte Aufruf dieser Variante selektiert nacheinander alle Ordner, die das Attribut *applicationIdentifier* besitzen (Applikationen gemäß Kapitel 9.3.1.1 und ADF gemäß 9.3.1.3). In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N418) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N419) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´2´ gewählt werden.

(N420) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N421) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N422) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 37 verwendet werden.

Tabelle 37: SELECT, kein AID, next occurrence, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier (hier leer)
P2	´06´	*fileOccurrence* + *responseType* = next occurrence, FCP–Antwortdaten
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.5 Use Case Selektieren per AID, first, keine Antwortdaten

In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N423) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N424) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N425) Der Parameter *aid* enthält einen Oktettstring gemäß (N102) oder dessen Anfang. Im Objektsystem wird nach einem Ordner mit dazu passendem Attribut *applicationIdentifier* gesucht.

(N426) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N427) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 38 verwendet werden.

Tabelle 38: SELECT, AID, first occurrence, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier
P2	´0C´	*fileOccurrence* + *responseType* = first occurrence, keine Antwortdaten
Data	´XX…YY´	*aid*, Oktettstring, Anzahl Oktette aus dem Intervall [1, 16]

15.2.6.6 Use Case Selektieren per AID, first, Antwortdaten mit FCP

In dieser Variante enthält die APDU des SELECT Kommandos fünf Parameter:

(N428) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N429) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N430) Der Parameter *aid* enthält einen Oktettstring gemäß (N102) oder dessen Anfang. Im Objektsystem wird nach einem Ordner mit dazu passendem Attribut *applicationIdentifier* gesucht.

(N431) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N432) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N433) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 39 verwendet werden.

Tabelle 39: SELECT, AID, first occurrence, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier
P2	´04´	*fileOccurrence* + *responseType* = first occurrence, Antwortdaten mit FCP
Data	´XX…YY´	*aid*, Oktettstring, Anzahl Oktette aus dem Intervall [1, 16]
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.7 Use Case Selektieren per AID, next, keine Antwortdaten

In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N434) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N435) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´2´ gewählt werden.

(N436) Der Parameter *aid* enthält einen Oktettstring gemäß (N102) oder dessen Anfang. Im Objektsystem wird nach einem Ordner mit dazu passendem Attribut *applicationIdentifier* gesucht.

(N437) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N438) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 40 verwendet werden.

Tabelle 40: SELECT, AID, next occurrence, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier
P2	´0E´	*fileOccurrence* + *responseType* = next occurrence, keine Antwortdaten
Data	´XX…YY´	*aid*, Oktettstring, Anzahl Oktette aus dem Intervall [1, 16]

15.2.6.8 Use Case Selektieren per AID, next, Antwortdaten mit FCP

In dieser Variante enthält die APDU des SELECT Kommandos fünf Parameter:

(N439) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´04´ gewählt werden.

(N440) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´2´ gewählt werden.

(N441) Der Parameter *aid* enthält einen Oktettstring gemäß (N102) oder dessen Anfang. Im Objektsystem wird nach einem Ordner mit dazu passendem Attribut *applicationIdentifier* gesucht.

(N442) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N443) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N444) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 41 verwendet werden.

Tabelle 41: SELECT, AID, next occurrence, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´04´	selectionMode = Ordnerselektion mit applicationIdentifier
P2	´06´	*fileOccurrence* + *responseType* = next occurrence, Antwortdaten mit FCP
Data	´XX…YY´	*aid*, Oktettstring, Anzahl Oktette aus dem Intervall [1, 16]
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.9 Use Case Selektieren, DF oder ADF, keine Antwortdaten

In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N445) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´01´ gewählt werden.

(N446) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N447) Der Parameter *fid* enthält einen Oktettstring gemäß (N67). Im *currentFolder* wird nach einem Ordner mit dazu passendem Attribut *fileIdentifier* gesucht.

(N448) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N449) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 42 verwendet werden.

Tabelle 42: SELECT, DF oder ADF mit FID, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´01´	*selectionMode* = Selektion eines Ordners mit *fileIdentifier*
P2	´0C´	*fileOccurrence* + *responseType* = first occurrence, keine Antwortdaten
Data	´XXYY´	fid

15.2.6.10 Use Case Selektieren, DF oder ADF, Antwortdaten mit FCP

In dieser Variante enthält die APDU des SELECT Kommandos fünf Parameter:

(N450) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´01´ gewählt werden.

(N451) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N452) Der Parameter *fid* enthält einen Oktettstring gemäß (N67). Im *currentFolder* wird nach einem Ordner mit dazu passendem Attribut *fileIdentifier* gesucht.

(N453) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N454) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N455) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 43 verwendet werden.

Tabelle 43: SELECT, DF oder ADF mit FID, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´01´	selectionMode = Ordnerselektion mit fileIdentifier
P2	´04´	*fileOccurrence* + *responseType* = first occurrence, Antwortdaten mit FCP
Data	´XXYY´	fid
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.11 Use Case Selektieren übergeordnetes Verzeichnis ohne FCP

Diese Variante selektiert den eine Ebene höher liegenden Ordner (siehe (N200), (N201)). In dieser Variante enthält die APDU des SELECT Kommandos zwei Parameter:

(N456) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´03´ gewählt werden.

(N457) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N458) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 44 verwendet werden.

Tabelle 44: SELECT, höhere Ebene keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´03´	*selectionMode* = Selektion des eine Eben höheren Ordners
P2	´0C´	*responseType* = keine Antwortdaten

15.2.6.12 Use Case Selektieren übergeordnetes Verzeichnis mit FCP

Diese Variante selektiert den eine Ebene höher liegenden Ordner (siehe (N200), (N201)). In dieser Variante enthält die APDU des SELECT Kommandos drei Parameter:

(N459) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´03´ gewählt werden.

(N460) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N461) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N462) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 45 verwendet werden.

Tabelle 45: SELECT, höhere Ebene, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´03´	*selectionMode* = Selektion des eine Eben höheren Ordners
P2	´04´	*responseType* = Antwortdaten mit FCP
Ne	*length*	Anzahl der erwarteten Oktette in den Antwortdaten

Kom
3002

15.2.6.13 Use Case Selektieren einer Datei, keine Antwortdaten

In dieser Variante enthält die APDU des SELECT Kommandos vier Parameter:

(N463) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´02´ gewählt werden.

(N464) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N465) Der Parameter *fid* enthält einen Oktettstring gemäß (N67). Im *currentFolder* wird nach einer Datei mit dazu passendem Attribut *fileIdentifier* gesucht.

(N466) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´0C´ gewählt werden.

(N467) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 46 verwendet werden.

Tabelle 46: SELECT, EF mit FID, keine Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´02´	*selectionMode* = Selektion einer Datei mit *fileIdentifier*
P2	´0C´	*fileOccurrence* + *responseType* = first occurrence, keine Antwortdaten
Data	´XXYY´	*fid*

15.2.6.14 Use Case Selektieren einer Datei, Antwortdaten mit FCP

In dieser Variante enthält die APDU des SELECT Kommandos fünf Parameter:

(N468) Der Parameter *selectionMode* bestimmt die Art der Suche. Für diesen Use Case MUSS *selectionMode* = ´02´ gewählt werden.

(N469) Der Parameter *fileOccurrence* bestimmt welches File aus einer Liste von passenden Files gefunden wird. Für diesen Use Case MUSS *fileOccurrence* = ´0´ gewählt werden.

(N470) Der Parameter *fid* enthält einen Oktettstring gemäß (N67). Im *currentFolder* wird nach einer Datei mit dazu passendem Attribut *fileIdentifier* gesucht.

(N471) Der Parameter *responseType* bestimmt die Art der Antwortdaten. Für diesen Use Case MUSS *responseType* = ´04´ gewählt werden.

(N472) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N473) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 47 verwendet werden.

Tabelle 47: SELECT, EF mit FID, Antwortdaten mit FCP

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´02´	*selectionMode* = Selektion einer Datei mit *fileIdentifier*
P2	´04´	*fileOccurrence* + *responseType* = first occurrence, FCP–Antwortdaten
Data	´XXYY´	fid
Ne	*length*	Anzahl der erwarteten Oktette in den Antwortdaten

15.2.6.15 Zusammenfassung der SELECT Kommando Varianten

Wegen der Vielzahl an Varianten für dieses Kommando werden hier alle auf einen Blick dargestellt. Es sei darauf hingewiesen, dass nicht alle Kombinationen der folgenden Tabelle in den vorangegangenen Kapiteln enthalten sind. Deshalb sind solche nicht zwingend zu unterstützen.

Tabelle 48: SELECT, Kommandoparameter im Überblick

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A4´	Instruction Byte gemäß [7816–4]
P1	´01´ ´02´ ´03´ ´04´	Selektion eines Ordners mit *fileIdentifier* Selektion einer Datei mit *fileIdentifier* Selektion des eine Ebene höher liegenden Ordners Selektion einer Ordners mit (möglicherweise leerem) *applicationIdentifier*
P2	´04´ ´06´ ´0C´ ´0E´	first occurrence, Antwortdaten mit FCP next occurrence, Antwortdaten mit FCP first occurrence, keine Antwortdaten next occurrence, keine Antwortdaten
Data	´XX…YY´	P1 = ´01´: zwei Byte *fid* P1 = ´02´: zwei Byte *fid* P1 = ´03´: abwesend P1 = ´04´: abwesend, oder bis zu 16 Oktette *aid*
Ne	length	P2 = ´04´: Anzahl der erwarteten Oktette in den Antwortdaten P2 = ´06´: Anzahl der erwarteten Oktette in den Antwortdaten P2 = ´0C´: abwesend P2 = ´0E´: abwesend

15.2.6.16 Antwort der Karte auf Selektieren eines Files

Tabelle 49: SELECT Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	FCP	abwesend, oder File Control Parameter
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Selektion eines Files
´62 83´	FileDeactivated	selektiertes File ist logisch oder physikalisch deaktiviert

Tabelle 50: SELECT Antwort APDU im Fehlerfall

Trailer	Inhalte	Beschreibung
´6A 82´	FileNotFound	zu selektierendes File wurde nicht gefunden

Hinweis (51): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N474) Ein COS KANN zusätzliche Trailer verwenden.

15.2.6.17 Kommandoabarbeitung innerhalb der Karte

Zur Beschreibung der Kommandobearbeitung werden folgende Definitionen eingeführt:

oldFolder — Dies ist die Bezeichnung von *currentFolder* vor Ausführung dieses SELECT Kommandos

newFile — Dies ist die Bezeichnung für das File (Ordner oder Datei), welches im Rahmen der Selektion ausgewählt wird.

path(*folder*) — Pfad eines Ordners mit Namen *folder*. Zum Pfad gehören nach dieser Definition sowohl der Ordner *folder*, als auch alle seine übergeordneten Verzeichnisse einschließlich *root*.

(N475) Das COS MUSS die SELECT Varianten aus 15.2.6.1, 15.2.6.2, 15.2.6.5, 15.2.6.6, 15.2.6.9, 15.2.6.10, 15.2.6.11, 15.2.6.12, 15.2.6.13 und 15.2.6.14 unterstützen.
Das COS KANN weitere SELECT Varianten unterstützen.
Das COS KANN weitere SELECT Varianten ablehnen.

(N476) Die Zugriffsbedingung für alle SELECT Varianten MUSS ALWAYS sein.

(N477) Wenn der Parameter P1 den Wert ´01´ besitzt, dann MUSS in *oldFolder.children* nach einem Ordner gesucht werden, der ein Attribut *fileIdentifier* besitzt, dessen Wert mit dem Parameter *fid* aus den Kommandodaten übereinstimmt. Wenn ein solcher Ordner existiert, dann MUSS *newFile* auf den gefundenen Ordner gesetzt werden. Andernfalls ist die Suche erfolglos.

(N478) Wenn der Parameter P1 den Wert ´02´ besitzt, dann MUSS in *oldFolder.children* nach einer Datei gesucht werden, deren Attribut *fileIdentifier* mit dem Parameter *fid* aus den Kommandodaten übereinstimmt. Wenn eine solche Datei existiert, dann MUSS *newFile* auf die gefundene Datei gesetzt werden. Andernfalls ist die Suche erfolglos.

(N479) Wenn der Parameter P1 den Wert ´03´ besitzt, und *currentFolder*

a. ist gleich *root*, dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. ist nicht *root*, dann ist *newFile* der im Vergleich zu *currentFolder* eine Ebene höher liegende Ordner (siehe (N200), (N201)).

(N480) Wenn der Parameter P1 den Wert ´04´ besitzt, dann enthält die Kommandonachricht einen (möglicherweise leeren) Parameter *aid*.

a. Im gesamten Objektsystem wird nach Ordnern gesucht, die ein Attribut *applicationIdentifier* besitzen, welches zu *aid* passt. Ein Attribut *applicationIdentifier* passt zu *aid*, wenn
   i. *applicationIdentifier* identisch zu *aid* ist, oder

b. Alle passenden Ordner werden zu einer Liste zusammengestellt.
   i. Ist *root* in dieser Liste enthalten, dann MUSS es das erste Listenelement sein.
   ii. Solange Objektsystem nicht verändert wird (DELETE oder LOAD APPLICATION) MUSS bei identischem *aid* stets dieselbe Liste erstellt werden.

c. Hat P2 einen Wert aus der Menge {´04´, ´0C}, dann MUSS *newFile* auf das erste Listenelement gesetzt werden.

d. Hat P2 einen Wert aus der Menge {´06´, ´0E} und
   i. *oldFolder* ist ebenfalls in der Liste und *oldFolder* ist
      1. nicht das letzte Listenelement, dann MUSS *newFile* auf das nächste Listenelement gesetzt werden.
      2. das letzte Listenelement, dann MUSS die Suche erfolglos sein.
   ii. *oldFolder* ist nicht in der Liste enthalten, dann
      1. KANN *newFile* auf irgendein Listenelement gesetzt werden, oder
      2. die Suche KANN erfolglos sein.

(N481) Wenn die Suche erfolglos war, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren. Dabei DÜRFEN *currentFolder*, *currentEF* sowie der Kanalkontext (siehe Kapitel 13) NICHT verändert werden.

(N482) Wenn

a. *newFile* eine Datei ist, dann wird *currentEF* auf *newFile* gesetzt.

b. *newFile* ein Ordner ist, dann MUSS
   i. *currentFolder* gleich *newFile* gesetzt werden.
   ii. *currentEF* auf den Wert „undefiniert“ gesetzt werden.
   iii. in allen Ordnern, die sowohl zu path(*oldFolder*) als auch zu path(*newFile*) gehören *seIdentifier* unverändert bleiben.
   iv. in allen anderen Ordnern *seIdentifier* auf den Wert 1 gesetzt werden.
   v. aus *dfSpecificSecurityList* MÜSSEN mittels clearSecurityStatus(…) alle Einträge entfernt werden, die weder zu path(*oldFolder*) noch zu path(*newFile*) gehören.
   vi. aus *dfSpecificPasswordList* MÜSSEN mittels clearPasswordStatus(…) alle Einträge entfernt werden, die weder zu path(*oldFolder*) noch zu path(*newFile*) gehören.
   vii. jedes Elemente von *keyReferenceList* auf den Wert „leer“ gesetzt werden.

Kom 3002

Kom 3002

(N483) Für das Datenfeld der Antwortnachricht gilt:

a. Wenn P2 einen Wert aus der Menge {´04´, ´06´} hat, genau dann MUSS das Datenfeld der Antwortnachricht die File Control Parameter gemäß Kapitel 9.3.3 enthalten.

b. Andernfalls fehlt das Datenfeld der Antwortnachricht.

(N484) Wenn der logische Wert von *newFile.lifeCycleStatus* (siehe (N206)) den Wert „Operational state (deactivated)" hat, genau dann MUSS als Trailer FileDeactivated gewählt werden.

(N485) Andernfalls MUSS als Trailer NoError gewählt werden.

(N486) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 50 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 50 MUSS eine höhere Priorität als FileDeactivated haben.

c. FileDeactivated MUSS eine höhere Priorität als NoError haben.

Hinweis (52): Gemäß den Regeln dieses Dokumentes ist es zulässig in deaktivierten Ordnern Unterordner oder Dateien mittels SELECT Kommando zu selektieren.

15.2.7 TERMINATE CARD USAGE

(N487) Das COS KANN dieses Kommando gemäß [7816–9] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–9] ablehnen.

15.2.8 TERMINATE DF

(N488) Das COS KANN dieses Kommando gemäß [7816–9] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–9] ablehnen.

15.2.9 TERMINATE EF

(N489) Das COS KANN dieses Kommando gemäß [7816–9] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–9] ablehnen.

15.3 Zugriff auf Daten in transparenten EF

Es ist möglich auf Daten im *body* eines transparenten EF lesend (READ BINARY) oder schreibend (UPDATE BINARY) zuzugreifen.

Die Kommandos in diesem Unterkapitel unterstützen zwei Varianten:

1. Variante ohne *shortFileIdentifier*: Diese Variante ist dadurch gekennzeichnet, dass für eine erfolgreiche Kommandoabarbeitung *currentEF* notwendigerweise ein transparentes EF ist.
2. Variante mit *shortFileIdentifier*: Diese Variante ist dadurch gekennzeichnet, dass das vom Kommando betroffene EF erst während der Kommandoabarbeitung gesetzt wird. Ein vorausgehendes SELECT Kommando ist also nicht notwendig.

15.3.1 ERASE BINARY

Das Kommando ERASE BINARY ersetzt bereits vorhandene Daten im *body* eines transparenten EF durch Oktette mit dem Wert ´00´. Das betroffene transparente EF wird vor der Löschoperation ausgewählt. Dies geschieht entweder vor dem Senden dieses ERASE BINARY Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*), oder innerhalb dieses ERASE BINARY Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welche Informationen im *body* gelöscht werden, bestimmt der Offset, der als Parameter in der Kommandonachricht enthalten ist.

15.3.1.1 Use Case Löschen ohne shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des ERASE BINARY Kommandos einen Parameter:

(N490) Der Parameter *offset* bestimmt ab welcher Position gelöscht wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 32767] = [`0000´, ´7FFF] sein (vergleiche (N115)).

(N491) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 51 verwendet werden.

Tabelle 51: ERASE BINARY ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´0E´	Instruction Byte gemäß [7816–4]
P1	´XX´	(*offset* – P2) / 256, MSByte von *offset*
P2	´XX´	*offset* mod 256, LSByte von *offset*

15.3.1.2 Use Case Löschen mit shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des ERASE BINARY Kommandos zwei Parameter:

(N492) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N493) Der Parameter *offset* bestimmt ab welcher Position geschrieben wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 255] = [´00´, ´FF´] sein.

(N494) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 52 verwendet werden.

Tabelle 52: ERASE BINARY mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´0E´	Instruction Byte gemäß [7816–4]
P1	´XX´	128 + shortFileIdentifier = ´80´ + shortFileIdentifier
P2	´XX´	offset

15.3.1.3 Antwort der Karte auf Löschen in transparenten EF

Tabelle 53: ERASE BINARY Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreicher Löschvorgang
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 54: ERASE BINARY Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht transparent
´6B 00´	OffsetTooBig	Parameter *offset* in Kommando APDU ist zu groß
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (53): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N495) Ein COS KANN zusätzliche Trailer verwenden.

15.3.1.4 Kommandoabarbeitung innerhalb der Karte

(N496) Das COS MUSS die ERASE BINARY Varianten aus 15.3.1.1 und 15.3.1.2 unterstützen.

Das COS KANN weitere ERASE BINARY Varianten unterstützen.
Das COS KANN weitere ERASE BINARY Varianten ablehnen.

(N497) Falls die APDU des ERASE BINARY Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche

i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.

ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. keinen shortFileIdentifier enthält

i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.

ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N498) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N499) Wenn *affectedObject* nicht vom Typ transparent EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N500) Wenn *offset* größer oder gleich *affectedObject.numberOfBytes* ist, genau dann MUSS das Kommando mit dem Trailer OffsetTooBig terminieren.

(N501) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS *affectedObject.body* mit Transaktionsschutz geändert werden

b. False hat, dann MUSS das COS entscheiden, ob *affectedObject.body* mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N502) Ab der durch *offset* gekennzeichneten Stelle MÜSSEN die in *affectedObject.body* enthaltenen Oktette auf den Wert ´00´ gesetzt werden.

(N503) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N504) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N505) Andernfalls MUSS als Trailer NoError gewählt werden.

(N506) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 54 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 54 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N507) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N508) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,
b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.3.2 READ BINARY

Das Kommando READ BINARY dient dem Auslesen von Informationen aus dem *body* eines transparenten EF. Deshalb enthält das Datenfeld der Antwortnachricht (Teile von) *body*. Das betroffene transparente EF wird vor der Leseoperation ausgewählt. Dies geschieht entweder vor dem Senden dieses READ BINARY Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*), oder innerhalb dieses READ BINARY Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welche Informationen aus *body* ausgelesen werden, bestimmen Offset und Länge, die als Parameter in der Kommandonachricht enthalten sind.

15.3.2.1 Use Case Lesen ohne shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des READ BINARY Kommandos zwei Parameter:

(N509) Der Parameter *offset* bestimmt ab welcher Position gelesen wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 32767] = [`0000´, ´7FFF] sein (vergleiche (N115)).

(N510) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N511) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 55 verwendet werden.

Tabelle 55: READ BINARY ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4] [7816–4]
INS	´B0´	Instruction Byte gemäß [7816–4]
P1	´XX´	(*offset* – P2) / 256, MSByte von *offset*
P2	´XX´	*offset* mod 256, LSByte von *offset*
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.3.2.2 Use Case Lesen mit shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des READ BINARY Kommandos drei Parameter:

(N512) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N513) Der Parameter *offset* bestimmt ab welcher Position gelesen wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 255] = [´00´, ´FF´] sein.

(N514) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N515) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 56 verwendet werden.

Tabelle 56: READ BINARY mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´B0´	Instruction Byte gemäß [7816–4]
P1	´XX´	128 + shortFileIdentifier = ´80´ + shortFileIdentifier
P2	´XX´	offset
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.3.2.3 Antwort der Karte auf Lesen in transparenten EF

Tabelle 57: READ BINARY Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´		ausgelesene Daten
Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Leseoperation
´62 82´	EndOfFileWarning	weniger Daten vorhanden, als mittels Ne angefordert
´62 81´	CorruptDataWarning	möglicherweise sind die Antwortdaten korrupt

Tabelle 58: READ BINARY Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht transparent
´6B 00´	OffsetTooBig	Parameter *offset* in Kommando APDU ist zu groß

Hinweis (54): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N516) Ein COS KANN zusätzliche Trailer verwenden.

15.3.2.4 Kommandoabarbeitung innerhalb der Karte

(N517) Das COS MUSS die READ BINARY Varianten aus 15.3.2.1 und 15.3.2.2 unterstützen.
Das COS KANN weitere READ BINARY Varianten unterstützen.
Das COS KANN weitere READ BINARY Varianten ablehnen.

(N518) Falls die APDU des READ BINARY Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche

i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.

ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. keinen shortFileIdentifier enthält

i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.

ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N519) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N520) Wenn *affectedObject* nicht vom Typ transparent EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N521) Wenn *offset* größer oder gleich *affectedObject.numberOfBytes* ist, genau dann MUSS das Kommando mit dem Trailer OffsetTooBig terminieren.

(N522) Wenn *affectedObject.flagChecksum* den Wert True hat und die Daten von *affectedObject.body* inkonsistent zur Checksumme sind, genau dann MUSS als Trailer CorruptDataWarning gewählt werden.

(N523) Wenn das Le Feld der Kommando APDU keine WildCard enthält und (*offset* + *length*) größer als *affectedObject.numberOfBytes* ist, genau dann MUSS als Trailer EndOfFileWarning gewählt werden.

(N524) Andernfalls MUSS als Trailer NoError gewählt werden.

(N525) Für das Datenfeld der Antwortnachricht gilt:

a. Aus dem Oktettstring von *affectedObject.body* MÜSSEN von der durch *offset* gekennzeichneten Position an die nachfolgenden Oktette übernommen werden.

b. Es DÜRFEN NICHT mehr Oktette übernommen werden, als durch Ne angegeben.

c. Die Übernahme der Oktette MUSS am Ende von *affectedObject.body* stoppen.

(N526) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 58 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 58 MUSS eine höhere Priorität als CorruptDataWarning haben.

c. CorruptDataWarning MUSS eine höhere Priorität als EndOfFileWarning haben.

d. EndOfFileWarning MUSS eine höhere Priorität als NoError haben.

(N527) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N528) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.3.3 SEARCH BINARY

(N529) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.3.4 UPDATE BINARY

Das Kommando UPDATE BINARY ersetzt bereits vorhandene Daten im *body* eines transparenten EF durch Daten, die im Datenfeld der Kommandonachricht enthalten sind. Das betroffene transparente EF wird vor der Schreiboperation ausgewählt. Dies geschieht entweder vor dem Senden dieses UPDATE BINARY Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit *shortFileIdentifier*), oder innerhalb dieses UPDATE BINARY Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welche Informationen im *body* geändert werden, bestimmen Offset und Schreibdaten, die als Parameter in der Kommandonachricht enthalten sind.

15.3.4.1 Use Case Schreiben ohne shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des UPDATE BINARY Kommandos zwei Parameter:

(N530) Der Parameter *offset* bestimmt ab welcher Position geschrieben wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 32767] = [`0000´, ´7FFF] sein (vergleiche (N115)).

(N531) Der Parameter *newData* enthält die neuen Daten, welche die ab der durch *offset* gekennzeichneten Stelle enthaltenen Daten in *body* ersetzen. Der Parameter *newData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *newData* unterliegt nur den Beschränkungen aus Kapitel 12.5.5.

(N532) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 59 verwendet werden.

Tabelle 59: UPDATE BINARY ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´D6´	Instruction Byte gemäß [7816–4]
P1	´XX´	(*offset* – P2) / 256, MSByte von *offset*
P2	´XX´	*offset* mod 256, LSByte von *offset*
Data	´XX…YY´	newData

15.3.4.2 Use Case Schreiben mit shortFileIdentifier in transparenten EF

In dieser Variante enthält die APDU des UPDATE BINARY Kommandos drei Parameter:

(N533) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N534) Der Parameter *offset* bestimmt ab welcher Position geschrieben wird. Der Wert von *offset* MUSS eine ganze Zahl im Intervall [0, 255] = [´00´, ´FF´] sein.

(N535) Der Parameter *newData* enthält die neuen Daten, welche die ab der durch *offset* gekennzeichneten Stelle enthaltenen Daten in *body* ersetzen. Der Parameter *newData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *newData* unterliegt nur den Beschränkungen aus Kapitel 12.5.5.

(N536) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 60 verwendet werden.

Tabelle 60: UPDATE BINARY mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´D6´	Instruction Byte gemäß [7816–4]
P1	´XX´	128 + shortFileIdentifier = ´80´ + shortFileIdentifier
P2	´XX´	offset
Data	´XX…YY´	newData

15.3.4.3 Antwort der Karte auf Schreiben in transparenten EF

Tabelle 61: UPDATE BINARY Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreicher Schreibvorgang
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 62: UPDATE BINARY Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht transparent
´6B 00´	OffsetTooBig	Parameter *offset* in Kommando APDU ist zu groß
´6A 84´	DataTooBig	Parameter *newData* ragt über das Dateiende hinaus
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (55): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N537) Ein COS KANN zusätzliche Trailer verwenden.

15.3.4.4 Kommandoabarbeitung innerhalb der Karte

(N538) Das COS MUSS die UPDATE BINARY Varianten aus 15.3.4.1 und 15.3.4.2 unterstützen.
Das COS KANN weitere UPDATE BINARY Varianten unterstützen.
Das COS KANN weitere UPDATE BINARY Varianten ablehnen.

(N539) Falls die APDU des UPDATE BINARY Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.
b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N540) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N541) Wenn *affectedObject* nicht vom Typ transparent EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N542) Wenn *offset* größer oder gleich *affectedObject.numberOfBytes* ist, genau dann MUSS das Kommando mit dem Trailer OffsetTooBig terminieren.

(N543) Wenn (*offset* + OctetLength(*newData*)) größer als *affectedObject.numberOfBytes* ist, genau dann MUSS das Kommando mit dem Trailer DataTooBig terminieren.

(N544) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS *affectedObject.body* mit Transaktionsschutz geändert werden
b. False hat, dann MUSS das COS entscheiden, ob *affectedObject.body* mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N545) Ab der durch *offset* gekennzeichneten Stelle MÜSSEN die in *affectedObject.body* enthaltenen Daten durch *newData* ersetzt werden.

(N546) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N547) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N548) Andernfalls MUSS als Trailer NoError gewählt werden.

(N549) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 62 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 62 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N550) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N551) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.3.5 WRITE BINARY

(N552) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.4 Zugriff auf strukturierte Daten

Es ist möglich auf Listenelemente von *recordList* in strukturierten EF lesend (READ RECORD) oder schreibend (UPDATE RECORD) zuzugreifen. Zudem lassen sich neue Listenelemente angelegen (APPEND RECORD). Listenelemente lassen sich nicht löschen. Es ist aber möglich den **Inhalt** eines Listenelementes zu löschen (ERASE RECORD). Listenelemente lassen sich aktivieren (ACTIVATE RECORD) und deaktivieren (DEACTIVATE RECORD), was sich auf die Nutzung des Rekordinhaltes auswirkt. Des Weiteren ist es möglich in der Liste nach Elementen zu suchen, deren Inhalt zu einem frei wählbaren Suchmuster passt (SEARCH RECORD).

Die Kommandos in diesem Unterkapitel unterstützen zwei Varianten:

1. Variante ohne shortFileIdentifier: Diese Variante ist dadurch gekennzeichnet, dass für eine erfolgreiche Kommandoabarbeitung *currentEF* notwendigerweise ein strukturiertes EF ist.
2. Variante mit shortFileIdentifier: Diese Variante ist dadurch gekennzeichnet, dass das vom Kommando betroffene EF erst während der Kommandoabarbeitung gesetzt wird. Ein vorausgehendes SELECT Kommando ist also nicht notwendig.

15.4.1 ACTIVATE RECORD

Das Kommando ACTIVATE RECORD aktiviert ein oder mehrere Listenelemente aus *recordList* eines strukturierten EF. Das betroffene strukturierte EF wird vor der Operation ausgewählt. Dies geschieht entweder vor dem Senden dieses ACTIVATE RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses ACTIVATE RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welche Listenelemente aktiviert werden, bestimmen Rekordnummer und Modus, welche als Parameter in der Kommandonachricht enthalten sind.

15.4.1.1 Use Case Aktivieren eines Rekords ohne shortFileIdentifier

In dieser Variante enthält die APDU des ACTIVATE RECORD Kommandos zwei Parameter:

(N553) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N554) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´04´ gewählt werden.

(N555) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 63 verwendet werden.

Tabelle 63: ACTIVATE RECORD, ein Rekord, ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´08´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	*mode*, Kodierung ´04´ bedeutet „nutze Listenelement P1“

15.4.1.2 Use Case Aktivieren eines Rekords mit shortFileIdentifier

In dieser Variante enthält die APDU des ACTIVATE RECORD Kommandos drei Parameter:

(N556) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N557) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N558) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´04´ gewählt werden.

(N559) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 64 verwendet werden.

Tabelle 64: ACTIVATE RECORD, ein Rekord, mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´08´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + mode = (shortFileIdentifier << 3) + ´04´ Kodierung ´04´ bedeutet „nutze Listenelement P1“

15.4.1.3 Use Case Aktivieren aller Rekords ab P1 ohne shortFileIdentifier

In dieser Variante enthält die APDU des ACTIVATE RECORD Kommandos zwei Parameter:

(N560) Der Parameter *recordNumber* bestimmt das erste betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N561) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´05´ gewählt werden.

(N562) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 65 verwendet werden.

Tabelle 65: ACTIVATE RECORD, alle Rekords ab P1, ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´08´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´05´	*mode*, Kodierung ´05´ bedeutet „nutze Listenelemente ab P1“

15.4.1.4 Use Case Aktivieren aller Rekords ab P1 mit shortFileIdentifier

In dieser Variante enthält die APDU des ACTIVATE RECORD Kommandos drei Parameter:

(N563) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N564) Der Parameter *recordNumber* bestimmt das erste betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N565) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´05´ gewählt werden.

(N566) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 66 verwendet werden.

Tabelle 66: ACTIVATE RECORD, alle Rekords ab P1, mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´08´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´05´	8 shortFileIdentifier + mode = (shortFileIdentifier << 3) + ´05´ Kodierung ´05´ bedeutet „nutze Listenelemente ab P1“

15.4.1.5 Antwort der Karte auf Aktivieren eines Rekords

Tabelle 67: ACTIVATE RECORD Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Aktivierung
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 68: ACTIVATE RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´69 85´	NoRecordLifeCycleStatus	Rekords in ausgewähltem EF besitzen keinen LCS
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (56): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N567) Ein COS KANN zusätzliche Trailer verwenden.

15.4.1.6 Kommandoabarbeitung innerhalb der Karte

(N568) Das COS MUSS die ACTIVATE RECORD Varianten aus 15.4.1.1, 15.4.1.2 15.4.1.3 und 15.4.1.4 unterstützen.
Das COS KANN weitere ACTIVATE RECORD Varianten unterstützen.
Das COS KANN weitere ACTIVATE RECORD Varianten ablehnen.

(N569) Falls die APDU des ACTIVATE RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N570) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N571) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N572) Wenn *affectedObject.flagRecordLifeCycleStatus* den Wert False besitzt, genau dann MUSS das Kommando mit dem Trailer NoRecordLifeCycleStatus terminieren.

(N573) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N574) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS der *lifeCycleStatus* mit Transaktionsschutz geändert werden.

b. False hat, dann MUSS das COS entscheiden, ob der *lifeCycleStatus* mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N575) Wenn *mode* = ´04´ ist und der physikalische Wert von *lifeCycleStatus* des durch *recordNumber* adressierten *record* in *affectedObject.recordList* bereits den Wert „Operational state (activated)“ besitzt, dann MUSS als Trailer NoError verwendet werden.

(N576) Der physikalische Wert von *lifeCycleStatus* der durch *recordNumber* und *mode* adressierten *record* in *affectedObject.recordList* MÜSSEN auf den Wert „Operational state (activated)“ gesetzt werden. Dabei gilt: Wenn *mode* den Wert

(N577) ´04´ besitz, dann ist nur das durch *recordNumber* adressierte Listenelement betroffen.

(N578) ´05´ besitz, dann ist das durch *recordNumber* adressierte Listenelement und alle folgenden betroffen.

(N579) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N580) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

a. Andernfalls MUSS als Trailer NoError gewählt werden.

b. Für die Priorität der Trailer gilt:

c. Die Priorität der Trailer in Tabelle 68 ist herstellerspezifisch.

d. Jeder Trailer in Tabelle 68 MUSS eine höhere Priorität als UpdateRetryWarning haben.

e. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N581) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N582) Im Fehlerfall

a. SOLL currentEF gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.2 APPEND RECORD

Das Kommando APPEND RECORD fügt ein neues Listenelement an *recordList* eines strukturierten EF an, wobei die Daten für den Oktettstring des neuen Listenelementes im Datenfeld der Kommandonachricht enthalten sind. Das betroffene strukturierte EF wird vor der Operation ausgewählt. Dies geschieht entweder vor dem Senden dieses APPEND RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses APPEND RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde.

15.4.2.1 Use Case Anlegen eines neuen Rekords ohne shortFileIdentifier

In dieser Variante enthält die APDU des APPEND RECORD Kommandos einen Parameter:

(N583) Der Parameter *recordData* enthält die Daten des neuen Rekords. Der Parameter *recordData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *recordData* unterliegt nur den Beschränkungen aus (N77).

(N584) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 69 verwendet werden.

Tabelle 69: APPEND RECORD ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´E2´	Instruction Byte gemäß [7816–4]
P1	´00´	Parameter ohne Bedeutung
P2	´00´	Parameter ohne Bedeutung
Data	´XX…YY´	recordData

15.4.2.2 Use Case Anlegen eines neuen Rekords mit shortFileIdentifier

In dieser Variante enthält die APDU des APPEND RECORD Kommandos zwei Parameter:

(N585) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N586) Der Parameter *recordData* enthält die Daten des neuen Rekords. Der Parameter *recordData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *recordData* unterliegt nur den Beschränkungen aus (N77).

(N587) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 70 verwendet werden.

Tabelle 70: APPEND RECORD mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´E2´	Instruction Byte gemäß [7816–4]
P1	´00´	Parameter ohne Bedeutung
P2	´XX´	8 shortFileIdentifier = shortFileIdentifier << 3
Data	´XX…YY´	recordData

15.4.2.3 Antwort der Karte auf Anlegen eines neuen Rekords

Tabelle 71: APPEND RECORD Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Hinzufügen eines Rekords
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 72: APPEND RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´67 00´	WrongRecordLength	*recordData* hat nicht die richtige Länge
´6A 84´	OutOfMemory	zu viele Oktette in *recordData*
´6A 84´	FullRecordList	Rekordliste lässt keine weiteren Elemente zu
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (57): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N588) Ein COS KANN zusätzliche Trailer verwenden.

15.4.2.4 Kommandoabarbeitung innerhalb der Karte

(N589) Das COS MUSS die APPEND RECORD Varianten aus 15.4.2.1 und 15.4.2.2 unterstützen.
Das COS KANN weitere APPEND RECORD Varianten unterstützen.
Das COS KANN weitere APPEND RECORD Varianten ablehnen.

(N590) Falls die APDU des APPEND RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.
b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N591) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatus-NotSatisfied terminieren.

(N592) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N593) Wenn affectedObject vom Typ

a. linear fixes EF ist und die Anzahl Oktette in *recordData* ungleich *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

b. zyklisches EF ist und die Anzahl Oktette in *recordData* ungleich *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

c. linear variables EF ist und

   i. die Anzahl Oktette in *recordData* größer als *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

   ii. die Anzahl Oktette in den Oktettstrings aller *record* von *recordList* nach durchgeführter Listenerweiterung größer als *affectedObject.numberOfBytes* wäre, genau dann MUSS das Kommando mit dem Trailer OutOfMemory terminieren.

(N594) Wenn die Anzahl der Listenelemente in *affectedObject.recordList* gleich *affectedObject.maximumNumberOfRecords* ist und *affectedObject* vom Typ linear fixes EF oder linear variables EF ist, genau dann MUSS das Kommando mit dem Trailer FullRecordList terminieren.

(N595) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS *affectedObject.recordList* mit Transaktionsschutz geändert werden

b. False hat, dann MUSS das COS entscheiden, ob *affectedObject.recordList* mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N596) Wenn affectedObject vom Typ

a. linear fixes EF oder linear variables EF ist, dann wird ein neuer Rekord an das Ende von *affectedObject.recordList* angehängt.

b. zyklische EF ist, dann wird ein neuer Rekord am Anfang von *affectedObject.recordList* eingefügt. Falls dadurch die Anzahl der Listenelemente größer als *affectedObject.maximumNumberOfRecords* wird, genau dann MUSS das letzte Element in *affectedObject.recordList* gelöscht werden.

(N597) Als Oktettstring des neuen Rekords wird *recordData* verwendet.

(N598) Falls *affectedObject.flagRecordLifeCycleStatus* den Wert True besitzt, dann wird der physikalische Wert des *lifeCycleStatus* des neuen Rekords auf „Operational state (activated)“ gesetzt.

(N599) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N600) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N601) Andernfalls MUSS als Trailer NoError gewählt werden.

(N602) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 72 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 72 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N603) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N604) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.3 DEACTIVATE RECORD

Das Kommando DEACTIVATE RECORD deaktiviert ein oder mehrere Listenelemente aus *recordList* eines strukturierten EF. Das betroffene strukturierte EF wird vor der Operation ausgewählt. Dies geschieht entweder vor dem Senden dieses DEACTIVATE RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses DEACTIVATE RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welche Listenelemente deaktiviert werden, bestimmen Rekordnummer und Modus, welche als Parameter in der Kommandonachricht enthalten sind.

15.4.3.1 Use Case Deaktivieren eines Rekords ohne shortFileIdentifier

In dieser Variante enthält die APDU des DEACTIVATE RECORD Kommandos Parameter:

(N605) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N606) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´04´ gewählt werden.

(N607) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 73 verwendet werden.

Tabelle 73: DEACTIVATE RECORD, ein Rekord, ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´06´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	*mode*, Kodierung ´04´ bedeutet „nutze Listenelement P1“

15.4.3.2 Use Case Deaktivieren eines Rekords mit shortFileIdentifier

In dieser Variante enthält die APDU des DEACTIVATE RECORD Kommandos drei Parameter:

(N608) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N609) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N610) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´04´ gewählt werden.

(N611) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 74 verwendet werden.

Tabelle 74: DEACTIVATE RECORD, ein Rekord, mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´06´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + 4 = (shortFileIdentifier << 3) + ´04´ Kodierung ´04´ bedeutet „nutze Listenelement P1"

15.4.3.3 Use Case Deaktivieren aller Rekords ab P1 ohne shortFileIdentifier

In dieser Variante enthält die APDU des DEACTIVATE RECORD Kommandos zwei Parameter:

(N612) Der Parameter *recordNumber* bestimmt das erste betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N613) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´05´ gewählt werden.

(N614) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 75 verwendet werden.

Tabelle 75: DEACTIVATE RECORD, alle Rekords ab P1, ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´06´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´05´	*mode*, Kodierung ´05´ bedeutet „nutze Listenelemente ab P1"

15.4.3.4 Use Case Deaktivieren aller Rekords ab P1 mit shortFileIdentifier

In dieser Variante enthält die APDU des DEACTIVATE RECORD Kommandos drei Parameter:

(N615) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N616) Der Parameter *recordNumber* bestimmt das erste betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N617) Der Parameter *mode* bestimmt die Art der Aktion. Für diesen Use Case MUSS *mode* = ´05´ gewählt werden.

(N618) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 76 verwendet werden.

Tabelle 76: DEACTIVATE RECORD, alle Rekords ab P1, mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´06´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´05´	8 shortFileIdentifier + mode = (shortFileIdentifier << 3) + ´05´ Kodierung ´05´ bedeutet „nutze Listenelemente ab P1“

15.4.3.5 Antwort der Karte auf Deaktivieren eines Rekords

Tabelle 77: DEACTIVATE RECORD Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Deaktivieren
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 78: DEACTIVATE RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´69 85´	NoRecordLifeCycleStatus	Rekords in ausgewähltem EF besitzen keinen LCS
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (58): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N619) Ein COS KANN zusätzliche Trailer verwenden.

15.4.3.6 Kommandoabarbeitung innerhalb der Karte

(N620) Das COS MUSS die DEACTIVATE RECORD Varianten aus 15.4.3.1, 15.4.3.2, 15.4.3.3 und 15.4.3.4 unterstützen.
Das COS KANN weitere DEACTIVATE RECORD Varianten unterstützen.
Das COS KANN weitere DEACTIVATE RECORD Varianten ablehnen.

(N621) Falls die APDU des DEACTIVATE RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.
b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N622) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N623) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N624) Wenn *affectedObject.flagRecordLifeCycleStatus* den Wert False besitzt, genau dann MUSS das Kommando mit dem Trailer NoRecordLifeCycleStatus terminieren.

(N625) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N626) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS der *lifeCycleStatus* mit Transaktionsschutz geändert werden
b. False hat, dann MUSS das COS entscheiden, ob der *lifeCycleStatus* mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N627) Wenn *mode* = ´04´ ist und der physikalische Wert von *lifeCycleStatus* des durch *recordNumber* adressierten *record* in *affectedObject.recordList* bereits den Wert „Operational state (deactivated)“ besitzt, dann MUSS als Trailer NoError verwendet werden.

(N628) Der physikalische Wert von *lifeCycleStatus* der durch *recordNumber* adressierten *record* in *affectedObject.recordList* MUSS auf den Wert „Operational state (deactivated)“ gesetzt werden. Dabei gilt: Wenn *mode* den Wert

a. ´04´ besitz, dann ist nur das durch *recordNumber* adressierte Listenelement betroffen.
b. ´05´ besitz, dann ist das durch *recordNumber* adressierte Listenelement und alle folgenden betroffen.

(N629) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N630) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N631) Andernfalls MUSS als Trailer NoError gewählt werden.

(N632) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 78 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 78 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N633) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N634) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.4 ERASE RECORD

Das Kommando ERASE RECORD ersetzt den Oktettstring eines bereits vorhandenen Listenelementes in *recordList* eines strukturierten EF durch einen Oktettstring, der nur Oktette mit dem Wert ´00´ besitzt. Das betroffene strukturierte EF wird vor der Operation ausgewählt. Dies geschieht entweder vor dem Senden dieses ERASE RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses ERASE RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welches Listenelement betroffen ist, bestimmt die Rekordnummer, welche als Parameter in der Kommandonachricht enthalten ist.

15.4.4.1 Use Case Löschen eines Rekordsinhaltes ohne shortFileIdentifier

In dieser Variante enthält die APDU des ERASE RECORD Kommandos einen Parameter:

(N635) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N636) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 79 verwendet werden.

Tabelle 79: ERASE RECORD ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´0C´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	Kodierung für „nutze Listenelement P1“

15.4.4.2 Use Case Löschen eines Rekordinhaltes mit shortFileIdentifier

In dieser Variante enthält die APDU des ERASE RECORD Kommandos zwei Parameter:

(N637) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N638) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N639) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 80 verwendet werden.

Tabelle 80: ERASE RECORD mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´0C´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + 4 = (shortFileIdentifier << 3) + ´04´ Kodierung ´04´ bedeutet „nutze Listenelement P1“

15.4.4.3 Antwort der Karte auf Löschen des Inhaltes eines Rekords

Tabelle 81: ERASE RECORD Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Löschen eines Rekordinhaltes
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 82: ERASE RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht
´62 87´	RecordDeactivated	adressierter Rekord ist deaktiviert
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (59): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N640) Ein COS KANN zusätzliche Trailer verwenden.

15.4.4.4 Kommandoabarbeitung innerhalb der Karte

(N641) Das COS MUSS die ERASE RECORD Varianten aus 15.4.4.1 und 15.4.4.2 unterstützen.
Das COS KANN weitere ERASE RECORD Varianten unterstützen.
Das COS KANN weitere ERASE RECORD Varianten ablehnen.

(N642) Falls die APDU des ERASE RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.
b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N643) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N644) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N645) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N646) Wenn der logische Wert von *lifeCycleStatus* des adressierten *record* den Zustand „Operational state (deactivated)“ hat, genau dann MUSS das Kommando mit dem Trailer RecordDeactivated terminieren.

(N647) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS der Rekordinhalt mit Transaktionsschutz geändert werden

b. False hat, dann MUSS das COS entscheiden, ob der Rekordinhalt mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N648) Alle Oktette im Oktettstring des durch *recordNumber* adressierten *record* in *affectedObject.recordList* werden auf den Wert ´00´ gesetzt.

(N649) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N650) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N651) Andernfalls MUSS als Trailer NoError gewählt werden.

(N652) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 82 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 82 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N653) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N654) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.5 READ RECORD

Das Kommando READ RECORD dient dem Auslesen (des Anfangs) eines Listenelementes aus *recordList* eines strukturierten EF. Das betroffene strukturierte EF wird vor der Leseoperation ausgewählt. Dies geschieht entweder vor dem Senden dieses READ RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses READ RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welches Listenelement (oder dessen Anfang) ausgelesen wird, bestimmen Rekordnummer und Länge, die als Parameter in der Kommandonachricht enthalten sind.

15.4.5.1 Use Case Lesen ohne shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des READ RECORD Kommandos zwei Parameter:

(N655) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N656) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N657) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion

dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 83 verwendet werden.

Tabelle 83: READ RECORD ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´B2´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	Kodierung für „nutze Listenelement P1“
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.4.5.2 Use Case Lesen mit shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des READ RECORD Kommandos drei Parameter:

(N658) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N659) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N660) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N661) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 84 verwendet werden.

Tabelle 84: READ RECORD mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´B2´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + 4 = (shortFileIdentifier << 3) + ´04´ Kodierung ´04´ bedeutet „nutze Listenelement P1“
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.4.5.3 Antwort der Karte auf Lesen in strukturierten EF

Tabelle 85: READ RECORD Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
	´XX…YY´	Ausgelesene Daten
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Leseoperation
´62 82´	EndOfRecordWarning	mittels Ne mehr Daten angefordert, als vorhanden sind
´62 81´	CorruptDataWarning	möglicherweise sind Antwortdaten korrupt

Tabelle 86: READ RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* adressiertes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht
´62 87´	RecordDeactivated	adressierter Rekord ist deaktiviert

Hinweis (60): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N662) Ein COS KANN zusätzliche Trailer verwenden.

15.4.5.4 Kommandoabarbeitung innerhalb der Karte

(N663) Das COS MUSS die READ RECORD Varianten aus 15.4.5.1 und 15.4.5.2 unterstützen.
Das COS KANN weitere READ RECORD Varianten unterstützen.
Das COS KANN weitere READ RECORD Varianten ablehnen.

(N664) Falls die APDU des READ RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.
b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N665) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N666) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N667) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N668) Wenn der logische Wert von *lifeCycleStatus* des adressierten *record* den Zustand „Operational state (deactivated)“ hat, genau dann MUSS das Kommando mit dem Trailer RecordDeactivated terminieren.

(N669) Wenn *affectedObject.flagChecksum* den Wert True hat und die Daten des adressierten Listenelementes inkonsistent zur Checksumme sind, genau dann MUSS als Trailer CorruptDataWarning gewählt werden.

(N670) Wenn *length* größer als die Länge des adressierten Listenelementes *affectedObject.numberOfBytes* ist, genau dann MUSS als Trailer EndOfRecordWarning gewählt werden.

(N671) Andernfalls MUSS als Trailer NoError gewählt werden.

(N672) Für das Datenfeld der Antwortnachricht gilt:

a. Aus dem Oktettstring des adressierten *record* MÜSSEN vom ersten Oktett an die nachfolgenden Bytes übernommen werden.

b. Es DÜRFEN NICHT mehr Bytes übernommen werden, als durch Ne angegeben.

c. Die Übernahme MUSS am Ende von *record* stoppen.

(N673) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 86 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 86 MUSS eine höhere Priorität als CorruptDataWarning haben.

c. CorruptDataWarning MUSS eine höhere Priorität als EndOfRecordWarning haben.

d. EndOfRecordWarning MUSS eine höhere Priorität als NoError haben.

(N674) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N675) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.6 SEARCH RECORD

Das Kommando SEARCH RECORD sucht in den Listenelementen von *recordList* eines strukturierten EF nach einem Muster, welches im Datenfeld der Kommandonachricht übergeben wird. Die Antwortdaten enthalten die Nummern der Rekords, welche das Muster enthalten. Das betroffene, strukturierte EF wird vor der Suchoperation ausgewählt. Dies geschieht entweder vor dem Senden dieses SEARCH RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses SEARCH RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Bei welchem Listenelement in *recordList* die Suche startet, wird durch die Rekordnummer bestimmt, die als Parameter in der Kommandonachricht enthalten ist.

15.4.6.1 Use Case Suchen ohne shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des SEARCH RECORD Kommandos drei Parameter:

(N676) Der Parameter *recordNumber* bestimmt das Listenelement, welches als erstes von der Suche betroffen ist. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N677) Der Parameter *searchString* enthält das Muster, nach welchem in den Oktettstrings der Listenelemente gesucht wird. Der Parameter *searchString* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *searchString* unterliegt nur den Beschränkungen aus (N77).

(N678) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N679) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 87 verwendet werden.

Tabelle 87: SEARCH RECORD ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A2´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	Kodierung für „suche in Listenelement P1 und allen folgenden“
Data	´XX…YY´	searchString
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.4.6.2 Use Case Suchen mit shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des SEARCH RECORD Kommandos vier Parameter:

(N680) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N681) Der Parameter *recordNumber* bestimmt das Listenelement, welches als erstes von der Suche betroffen ist. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N682) Der Parameter *searchString* enthält das Muster, nach welchem in den Oktettstrings der Listenelemente gesucht wird. Der Parameter *searchString* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *searchString* unterliegt nur den Beschränkungen aus (N77).

(N683) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N684) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 88 verwendet werden.

Tabelle 88: SEARCH RECORD mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´A2´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + 4 = (shortFileIdentifier << 3) + ´04´ ´04´ bedeutet: „suche in Listenelement P1 und allen folgenden“
Data	´XX…YY´	searchString
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.4.6.3 Antwort der Karte auf Suchen in strukturierten EF

Tabelle 89: SEARCH RECORD Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
Daten	´XX…YY´	Nummern der Listenelemente, in denen das Muster gefunden wurde
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Suchoperation
´62 82´	UnsuccessfulSearch	erfolglose Suche in adressierten Rekords
´62 81´	CorruptDataWarning	möglicherweise sind Antwortdaten korrupt

Tabelle 90: SEARCH RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* ausgewähltes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht

Hinweis (61): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N685) Ein COS KANN zusätzliche Trailer verwenden.

15.4.6.4 Kommandoabarbeitung innerhalb der Karte

(N686) Das COS MUSS die SEARCH RECORD Varianten aus 15.4.6.1 und 15.4.6.2 unterstützen.
Das COS KANN weitere SEARCH RECORD Varianten unterstützen.
Das COS KANN weitere SEARCH RECORD Varianten ablehnen.

(N687) Falls die APDU des SEARCH RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N688) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N689) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N690) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N691) Im Oktettstring des durch *recordNumber* adressierten *record* in *affectedObject.recordList* und allen folgenden Elementen der Liste MUSS nach dem Muster *searchString* gesucht werden.

(N692) Die Suche in einem Listenelement MUSS genau dann erfolgreich sein, wenn

a. der logische Wert des *lifeCycleStatus* des Listenelementes den Wert „Operational state (activated)“ hat UND

b. *searchString* im Oktettstring des Listenelementes vollständig enthalten ist.

(N693) Wenn die Suche in einem Listenelement erfolgreich war, dann MUSS die Nummer des Rekords (siehe (N76)) in einem Oktett (gemäß I2OS(*recordNumber*, 1)) kodiert zum Datenfeld der Antwortnachricht hinzugefügt werden.

(N694) Die Oktette im Datenfeld MÜSSEN aufsteigend sortiert sein.

(N695) Wenn *affectedObject.flagChecksum* den Wert True hat und die Daten wenigstens eines adressierten Listenelementes inkonsistent zur Checksumme sind, genau dann MUSS als Trailer CorruptDataWarning gewählt werden.

(N696) Wenn das Datenfeld leer ist, das heißt die Suche war in keinem der adressierten Listenelemente erfolgreich, genau dann MUSS als Trailer UnsuccessfulSearch verwendet werden.

(N697) Andernfalls MUSS als Trailer NoError gewählt werden.

(N698) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 90 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 90 MUSS eine höhere Priorität als CorruptDataWarning haben.

c. CorruptDataWarning MUSS eine höhere Priorität als UnsuccessfulSearch haben.

d. UnsuccessfulSearch MUSS eine höhere Priorität als NoError haben.

(N699) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N700) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.7 UPDATE RECORD

Das Kommando UPDATE RECORD ersetzt den Oktettstring eines bereits vorhandenen Listenelementes in *recordList* eines strukturierten EF durch Daten, die im Datenfeld der Kommandonachricht enthalten sind. Das betroffene strukturierte EF wird vor der Suchoperation ausgewählt. Dies geschieht entweder vor dem Senden dieses UPDATE RECORD Kommandos durch eine Select–Operation (SELECT Kommando, oder Kommando mit shortFileIdentifier), oder innerhalb dieses UPDATE RECORD Kommandos, falls diesem ein *shortFileIdentifier* als Parameter mitgeliefert wurde. Welches Listenelement in *recordList* ersetzt wird, bestimmt die Rekordnummer, die als Parameter in der Kommandonachricht enthalten ist.

15.4.7.1 Use Case Schreiben ohne shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des UPDATE RECORD Kommandos zwei Parameter:

(N701) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N702) Der Parameter *newData* enthält die neuen Daten, welche den Oktettstring des adressierten *record* ersetzen. Der Parameter *newData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *newData* unterliegt nur den Beschränkungen aus (N77).

(N703) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 91 verwendet werden.

Tabelle 91: UPDATE RECORD ohne shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´DC´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´04´	Kodierung für „nutze Listenelement P1“
Data	´XX…YY´	newData

15.4.7.2 Use Case Schreiben mit shortFileIdentifier in strukturierten EF

In dieser Variante enthält die APDU des UPDATE RECORD Kommandos drei Parameter:

(N704) Der Parameter *shortFileIdentifier* wählt während der Kommandoabarbeitung ein EF aus. Der Wert von *shortFileIdentifier* MUSS aus dem in (N70) definierten Bereich gewählt werden.

(N705) Der Parameter *recordNumber* bestimmt das betroffene Listenelement. Der Wert von *recordNumber* MUSS konform zu (N76) gewählt werden.

(N706) Der Parameter *newData* enthält die neuen Daten, welche den Oktettstring des adressierten *record* ersetzen. Der Parameter *newData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *newData* unterliegt nur den Beschränkungen aus (N77).

(N707) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 92 verwendet werden.

Tabelle 92: UPDATE RECORD mit shortFileIdentifier

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´DC´	Instruction Byte gemäß [7816–4]
P1	´XX´	recordNumber
P2	´XX´	8 shortFileIdentifier + 4 = (shortFileIdentifier << 3) + ´04´ Kodierung ´04´ bedeutet „nutze Listenelement P1“
Data	´XX…YY´	newData

15.4.7.3 Antwort der Karte auf Schreiben in strukturierten EF

Tabelle 93: UPDATE RECORD Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreicher Schreibvorgang
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 94: UPDATE RECORD Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 82´	FileNotFound	per *shortFileIdentifier* ausgewähltes EF nicht gefunden
´69 86´	NoCurrentEF	es ist kein EF ausgewählt
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 81´	WrongFileType	ausgewähltes EF ist nicht strukturiert
´6A 83´	RecordNotFound	Listenelement *recordNumber* existiert nicht
´62 87´	RecordDeactivated	adressierter Rekord ist deaktiviert
´67 00´	WrongRecordLength	*newData* hat nicht die richtige Länge
´6A 84´	OutOfMemory	zu viele Oktette in *newData*
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (62): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N708) Ein COS KANN zusätzliche Trailer verwenden.

15.4.7.4 Kommandoabarbeitung innerhalb der Karte

(N709) Das COS MUSS die UPDATE RECORD Varianten aus 15.4.7.1 und 15.4.7.2 unterstützen.
Das COS KANN weitere UPDATE RECORD Varianten unterstützen.
Das COS KANN weitere UPDATE RECORD Varianten ablehnen.

(N710) Falls die APDU des UPDATE RECORD Kommandos

a. einen *shortFileIdentifier* enthält, dann wird innerhalb von *currentFolder.children* nach einem EF mit diesem *shortFileIdentifier* gesucht. Falls die Suche
   i. erfolgreich verlief, dann MUSS *affectedObject* auf dieses EF gesetzt werden.
   ii. nicht erfolgreich verlief, genau dann MUSS das Kommando mit dem Trailer FileNotFound terminieren.

b. keinen shortFileIdentifier enthält
   i. und *currentEF* (siehe (N300)b) unbestimmt ist, genau dann MUSS das Kommando mit dem Trailer NoCurrentEF terminieren.
   ii. dann MUSS *affectedObject* gleich *currentEF* gesetzt werden.

(N711) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N712) Wenn *affectedObject* nicht vom Typ strukturiertes EF ist, genau dann MUSS das Kommando mit dem Trailer WrongFileType terminieren.

(N713) Wenn *recordNumber* größer als die Anzahl der Listenelemente in *affectedObject.recordList* ist, genau dann MUSS das Kommando mit dem Trailer RecordNotFound terminieren.

(N714) Wenn der logische Wert von *lifeCycleStatus* des adressierten *record* den Zustand „Operational state (deactivated)“ hat, genau dann MUSS das Kommando mit dem Trailer RecordDeactivated terminieren.

(N715) Wenn affectedObject vom Typ

a. linear fixes EF ist und die Anzahl Oktette in *newData* ungleich *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

b. zyklisches EF ist und die Anzahl Oktette in *newData* ungleich *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

c. linear variables EF ist und

i. die Anzahl Oktette in *newData* größer als *affectedObject.maximumRecordLength* ist, genau dann MUSS das Kommando mit dem Trailer WrongRecordLength terminieren.

ii. die Anzahl Oktette in den Oktettstrings aller *record* von *affectedObject.recordList* nach durchgeführter Ersetzung größer als *affectedObject.numberOfBytes* wäre, genau dann MUSS das Kommando mit dem Trailer OutOfMemory terminieren.

(N716) Wenn affectedObject.flagTransactionMode den Wert

a. True hat, genau dann MUSS der Rekordinhalt mit Transaktionsschutz geändert werden

b. False hat, dann MUSS das COS entscheiden, ob der Rekordinhalt mit oder ohne Transaktionsschutz (siehe Kapitel 15.1) geändert wird.

(N717) Der Oktettstring des durch *recordNumber* adressierten *record* in *affectedObject.recordList* wird durch *newData* ersetzt.

(N718) Falls *affectedObject* vom Typ linear variables EF ist, dann MÜSSEN alle folgenden Fälle unterstützt werden: *newData* enthält im Vergleich zum *record*, der durch *recordNumber* adressiert wird

a. weniger Oktette,

b. gleich viele Oktette, oder

c. mehr Oktette.

(N719) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N720) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N721) Andernfalls MUSS als Trailer NoError gewählt werden.

(N722) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 94 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 94 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N723) Im Erfolgsfall MUSS *currentEF* gleich *affectedObject* gesetzt werden.

(N724) Im Fehlerfall

a. SOLL *currentEF* gleich *affectedObject* gesetzt werden,

b. andernfalls MUSS *currentEF* unverändert auf dem Wert vor Ausführung des Kommandos belassen werden.

15.4.8 WRITE RECORD

(N725) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.5 Zugriff auf Datenobjekte

Hinweis (63): Gemäß Kapitel 9.6 sind die Kommandos dieses Kapitels nicht verpflichtend.

15.5.1 GET DATA

(N726) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.5.2 PUT DATA

(N727) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.6 Benutzerverifikation

Alle Kommandos dieses Kapitels benutzen bei der Kommandobearbeitung Passwortobjekte gemäß Kapitel 9.4. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in den Kommandodaten enthalten ist. Für diese Passwortreferenz gilt:

(N728) Der Parameter *passwordReference* besteht aus den zwei Teilen *location* und *identifier*. *location* zeigt an, ob ein globales oder DF–spezifisches Passwort von der Aktion betroffen ist. Als Wert für *location* MUSS ein Element der Menge {0, 128} = {´00´, ´80´} verwendet werden. Dabei gilt:

a. Der Wert *location* = ´00´ MUSS verwendet werden, wenn ein globales Passwortobjekt betroffen ist (siehe (N208)).

b. Der Wert *location* = ´80´ MUSS verwendet werden, wenn ein DF–spezifisches Passwortobjekt betroffen ist (siehe (N209)).

c. Der Parameter *identifier* bestimmt das betroffene Passwortobjekt. Der Wert von *identifier* MUSS konform zu (N150) gewählt werden.

d. Der Parameter *passwordReference* MUSS in einem Byte mit folgendem Wert kodiert werden: *passwordReference* = *location* + *identifier*.

15.6.1 CHANGE REFERENCE DATA

Das Kommando CHANGE REFERENCE DATA ersetzt das Attribut *secret* eines Passwortobjektes durch Daten, die im Datenfeld der Kommandonachricht enthalten sind. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist. Dieses Dokument spezifiziert die folgenden Varianten:

- Das Kommandodatenfeld enthält das alte und neue Benutzergeheimnis.
- Das Kommandodatenfeld enthält nur das neue Benutzergeheimnis.

15.6.1.1 Use Case Ändern eines Benutzergeheimnisses

In dieser Variante enthält die APDU des CHANGE REFERENCE DATA Kommandos drei Parameter:

(N729) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N730) Der Parameter *oldSecret* enthält das alte Benutzergeheimnis.

(N731) Der Parameter *newSecret* enthält das neue Benutzergeheimnis.

(N732) Die Parameter *oldSecret* und *newSecret* MÜSSEN gemäß (N81) kodiert sein.

(N733) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 95 verwendet werden.

Tabelle 95: CHANGE REFERENCE DATA mit altem und neuem Benutzergeheimnis

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´24´	Instruction Byte gemäß [7816–4]
P1	´00´	Data enthält altes und neues Benutzergeheimnis
P2	´XX´	passwordReference
Data	´XX…YY´	oldSecret || newSecret

15.6.1.2 Use Case Setzen eines Benutzergeheimnisses

In dieser Variante enthält die APDU des CHANGE REFERENCE DATA Kommandos zwei Parameter:

(N734) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N735) Der Parameter *newSecret* enthält das neue Benutzergeheimnis.

(N736) Der Parameter *newSecret* MUSS gemäß (N81) kodiert sein.

(N737) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 96 verwendet werden.

Tabelle 96: CHANGE REFERENCE DATA, nur neues Benutzergeheimnis

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´24´	Instruction Byte gemäß [7816–4]
P1	´01´	Data enthält neues Benutzergeheimnis
P2	´XX´	passwordReference
Data	´XX…YY´	newSecret

15.6.1.3 Antwort der Karte auf Ändern eines Benutzergeheimnisses

Tabelle 97: CHANGE REFERENCE DATA Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Ändern des Benutzergeheimnisses
´63 Cx´	WrongSecretWarning	*oldSecret* ist falsch
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 98: CHANGE REFERENCE DATA Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 83´	PasswordBlocked	abgelaufener Fehlbedienungszähler
´69 85´	ShortPassword	*newData* enthält ein zu kurzes Passwort
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (64): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N738) Ein COS KANN zusätzliche Trailer verwenden.

15.6.1.4 Kommandoabarbeitung innerhalb der Karte

(N739) Das COS MUSS die CHANGE REFERENCE DATA Variante aus 15.6.1.1 unterstützen.

(N740) Wenn das COS die Transportschutzvariante Leer–PIN_1 unterstützt, dann MUSS das COS die CHANGE REFERENCE DATA Variante aus 15.6.1.2 unterstützen.

(N741) Das COS KANN weitere CHANGE REFERENCE DATA Varianten unterstützen.

Das COS KANN weitere CHANGE REFERENCE DATA Varianten ablehnen.

(N742) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N743) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N744) Wenn *affectedObject.retryCounter* den Wert null hat, genau dann MUSS das Kommando mit dem Trailer PasswordBlocked terminieren.

(N745) Wenn die in *newSecret* kodierte Ziffernfolge für das Attribut *secret* des Passwortobjektes eine Länge hat, die kleiner als *affectedObject.minimumLength* ist, genau dann MUSS das Kommando mit dem Trailer ShortPassword terminieren.

(N746) Mittels clearPasswordStatus(*affectedObject*) MUSS der Sicherheitsstatus zurückgesetzt werden. Kom 3002

(N747) Wenn das Datenfeld der Kommandonachricht *oldSecret* enthält, genau dann MUSS das Attribut *affectedObject.secret* mit *oldSecret* verglichen werden.

a. Wenn der Vergleich fehlschlägt, genau dann MUSS

b. *affectedObject.retryCounter* um eins dekrementiert werden und

   i. das Kommando mit dem Trailer WrongSecretWarning terminieren. Das Lownibble des Trailers MUSS dabei auf den Wert ´F´ gesetzt werden, wenn *affectedObject.retryCounter* größer als fünfzehn ist, andernfalls auf den Wert von affectedObject.retryCounter.

   ii. Wenn die Vergleichsoperation durch einen Reset abgebrochen wird, dann MUSS *affectedObject.retryCounter* um eins dekrementiert werden.

c. Wenn der Vergleich erfolgreich ist, genau dann MUSS

i. das Attribut affectedObject.retryCounter auf den Wert affectedObject.startRetryCounter gesetzt werden und

ii. das Attribut *affectedObject.secret* auf den in *newSecret* kodierten Wert gesetzt werden und

iii. das Attribut *affectedObject.transportStatus* MUSS auf den Wert *regularPassword* geändert werden (siehe Tabelle 9).

(N748) Alle persistenten Änderungen in (N747)c MÜSSEN mit Transaktionsschutz ausgeführt werden.

(N749) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N750) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N751) Andernfalls MUSS als Trailer NoError gewählt werden.

(N752) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 98 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 98 MUSS eine höhere Priorität als WrongSecretWarning haben.

c. WrongSecretWarning MUSS eine höhere Priorität als UpdateRetryWarning haben.

d. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N753) Wenn die Ausführung dieses Kommandos durch einen Reset abgebrochen wird, dann gilt für die Änderung von *affectedObject.transportStatus* und *affectedObject.secret*:

a. Beide Attribute SOLLEN gemeinsam in einer Transaktion geändert werden.

b. *secret* KANN in einer eigenen Transaktion zeitlich vor *transportStatus* geändert werden.

c. *transportStatus* DARF NICHT persistent geändert sein, wenn nicht auch *secret* persistent geändert ist.

15.6.2 DISABLE VERIFICATION REQUIREMENT

Das Kommando DISABLE VERIFICATION REQUIREMENT ändert das Attribut *flagEnabled* eines Passwortobjektes (siehe Kapitel 9.4) so, dass das COS sich so verhält, als sei der Sicherheitszustand des Passwortes ständig gesetzt. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist.

15.6.2.1 Use Case Abschalten der Benutzerverifikation

In dieser Variante enthält die APDU des DISABLE VERIFICATION REQUIREMENT Kommandos einen Parameter:

(N754) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N755) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 99 verwendet werden.

Tabelle 99: DISABLE VERIFICATION REQUIREMENT ohne Verifikationsdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´26´	Instruction Byte gemäß [7816–4]
P1	´01´	keine Verifikationsdaten
P2	´XX´	passwordReference

15.6.2.2 Antwort der Karte auf Abschalten der Benutzerverifikation

Tabelle 100: DISABLE VERIFICATION REQUIREMENT Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	Erfolgreiches Abschalten des Passwortobjektes
´63 Cx´	UpdateRetryWarning	Wie NoError, aber Schreibschwierigkeiten

Tabelle 101: DISABLE VERIFICATION REQUIREMENT Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	Adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (65): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N756) Ein COS KANN zusätzliche Trailer verwenden.

15.6.2.3 Kommandoabarbeitung innerhalb der Karte

(N757) Das COS KANN die DISABLE VERIFICATION REQUIREMENT Variante aus 15.6.2.1 unterstützen.
Das COS KANN weitere DISABLE VERIFICATION REQUIREMENT Varianten unterstützen.
Das COS KANN alle DISABLE VERIFICATION REQUIREMENT Varianten ablehnen.

(N758) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N759) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N760) Wenn *affectedObject.flagEnabled* den Wert *False* besitzt, dann MUSS das Kommando mit dem Trailer NoError terminieren.

(N761) Das Attribut *affectedObject.flagEnabled* MUSS mit Transaktionsschutz auf den Wert False gesetzt werden.

(N762) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N763) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N764) Andernfalls MUSS als Trailer NoError gewählt werden.

(N765) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 101 ist herstellerspezifisch.
b. Jeder Trailer in Tabelle 101 MUSS eine höhere Priorität als UpdateRetryWarning haben.
c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.6.3 ENABLE VERIFICATION REQUIREMENT

Das Kommando ENABLE VERIFICATION REQUIREMENT ändert das Attribut *flagEnabled* eines Passwortobjektes (siehe Kapitel 9.4) so, dass der Sicherheitszustand des Passwortes nur durch eine erfolgreiche Benutzerverifikation gesetzt wird. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist.

15.6.3.1 Use Case Einschalten der Benutzerverifikation

In dieser Variante enthält die APDU des ENABLE VERIFICATION REQUIREMENT Kommandos einen Parameter:

(N766) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N767) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 102 verwendet werden.

Tabelle 102: ENABLE VERIFICATION REQUIREMENT ohne Verifikationsdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´28´	Instruction Byte gemäß [7816–4]
P1	´01´	keine Verifikationsdaten
P2	´XX´	passwordReference

15.6.3.2 Antwort der Karte auf Einschalten der Benutzerverifikation

Tabelle 103: ENABLE VERIFICATION REQUIREMENT Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Einschalten des Passwortobjektes
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 104: ENABLE VERIFICATION REQUIREMENT Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (66): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N768) Ein COS KANN zusätzliche Trailer verwenden.

15.6.3.3 Kommandoabarbeitung innerhalb der Karte

(N769) Das COS KANN die ENABLE VERIFICATION REQUIREMENT Variante aus 15.6.3.1 unterstützen.
Das COS KANN weitere ENABLE VERIFICATION REQUIREMENT Varianten unterstützen.
Das COS KANN alle ENABLE VERIFICATION REQUIREMENT Varianten ablehnen.

(N770) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N771) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N772) Wenn *affectedObject.flagEnabled* den Wert *True* besitzt, dann MUSS das Kommando mit dem Trailer NoError terminieren.

(N773) Das Attribut *affectedObject.flagEnabled* MUSS mit Transaktionsschutz auf den Wert True gesetzt werden.

(N774) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N775) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N776) Andernfalls MUSS als Trailer NoError gewählt werden.

(N777) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 104 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 104 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.6.4 GET PIN STATUS

Hinweis (67): Dieses Kommando ist nicht in der Normenreihe ISO/IEC 7816 enthalten. Es ließe sich kombinieren mit dem Kommando VERIFY (leere Kommandodaten). Einem DIN NIA17.4 Votum gemäß wurde auf eine derartige Kombination verzichtet.

Das Kommando GET PIN STATUS zeigt in den Antwortdaten an,

- ob der Sicherheitszustand des Passwortobjektes gesetzt ist.
- welchen Wert das Attribut *retryCounter* besitzt.
- ob ein Passwortobjekt durch ein Transport–PIN Verfahren geschützt ist und falls ja, welches Transportschutzverfahren vom Passwortobjekt verwendet wird.

Welches Passwortobjekt von diesem Kommando betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist.

15.6.4.1 Use Case Auslesen des Status eines Passwortobjektes

In dieser Variante enthält die APDU des GET PIN STATUS Kommandos einen Parameter:

(N778) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N779) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 105 verwendet werden.

Tabelle 105: GET PIN STATUS

	Inhalt	Beschreibung
CLA	´80´	CLA Byte gemäß [7816–4] wird hier „proprietary“ angezeigt
INS	´20´	Instruction Byte
P1	´00´	–
P2	´XX´	passwordReference

15.6.4.2 Antwort der Karte auf Auslesen des PIN Status

Tabelle 106: GET PIN STATUS Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	Verifikation nicht erforderlich
´62 Cx´: ´62 C1´ ´62 C2´ ´62 C4´ ´62 C5´ ´62 C7´ ´62 CF´	TransportStatus: Transport–PIN_Zufallszahl Transport–PIN_abgeleitet Leer–PIN_2 Transport–PIN_festerWert Leer–PIN_1 Transport–PIN_0000	Passwortobjekt ist mit Transportschutz versehen, Transportschutzverfahren enthalten im Least Significant Nibble (siehe auch Kapitel 9.2.5)
´63 Cx´	RetryCounter	Passwortobjekt ohne Transportschutz (das bedeutet regularPassword), Wert des Fehlbedienungszählers enthalten im Least Significant Nibble

Tabelle 107: GET PIN STATUS Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt

Hinweis (68): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N780) Ein COS KANN zusätzliche Trailer verwenden.

15.6.4.3 Kommandoabarbeitung innerhalb der Karte

(N781) Das COS MUSS die GET PIN STATUS Variante aus 15.6.4.1 unterstützen.
Das COS KANN weitere GET PIN STATUS Varianten unterstützen.
Das COS KANN weitere GET PIN STATUS Varianten ablehnen.

(N782) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N783) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N784) Falls das Attribut *affectedObject.flagEnabled* (siehe (N157)) den Wert False besitzt, dann MUSS als Trailer NoError verwendet werden.

(N785) Falls *affectedObject* in *globalPasswordList* (siehe (N299)i) oder in *dfSpecificPasswordList* (siehe (N299)j) enthalten ist und dort das Attribut *securityStatusEvaluationCounter* (siehe (N299)k) einen Wert ungleich null besitzt, dann MUSS als Trailer NoError verwendet werden.

Kom
3002

(N786) Falls das Attribut *affectedObject.transportStatus* (siehe (N156)) einen Wert ungleich regularPassword besitzt, genau dann MUSS als Trailer TransportStatus verwendet werden mit TransportStatus = ´62 Cx´. Dabei ist ´x´ durch die Kodierung von *affectedObject.transportStatus* gemäß Tabelle 9 zu ersetzen.

(N787) Falls das Attribut *affectedObject.transportStatus* (siehe (N156)) den Wert regularPassword besitzt, genau dann MUSS als Trailer RetryCounter verwendet werden mit RetryCounter = ´63 Cx´. Dabei ist ´x´ zu ersetzen durch das Minimum der beiden Zahlen 15 = ´F´ und *affectedObject.retryCounter*.

(N788) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 107 ist herstellerspezifisch.
b. Jeder Trailer in Tabelle 107 MUSS eine höhere Priorität als NoError haben.
c. NoError MUSS eine höhere Priorität als TransportStatus haben.
d. TransportStatus MUSS eine höhere Priorität als RetryCounter haben.

Hinweis (69): Gemäß (N786), (N787) und (N788) ist es nicht möglich den Fehlbedienungszähler auszulesen, wenn Transportschutz besteht.

15.6.5 RESET RETRY COUNTER

Das Kommando RESET RETRY COUNTER setzt das Attribut *retryCounter* eines Passwortobjektes auf den Startwert *startRetryCounter*. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist. Dieses Kommando gibt es in den Varianten

- mit und ohne PUK
- mit und ohne neuen Wert für das Attribut *secret* des Passwortobjektes.

Die Variante „mit PUK" wird typischerweise verwendet, wenn eine „nicht technische" Instanz (etwa der Kartenbesitzer) das Recht zur Durchführung dieser Aktion besitzt.

Die Variante „ohne PUK" wird typischerweise verwendet, wenn eine technische Instanz (etwa das CMS) diese Aktion durchzuführen hat. In den Zugriffsbedingungen wird dann typischerweise zumindest eine Rollenauthentisierung gefordert.

Die Variante „mit neuem Geheimnis" wird typischerweise gewählt, wenn der Karteninhaber sein Passwort definitiv vergessen hat.

Die Variante „ohne neues Geheimnis" wird typischerweise gewählt, wenn es der Instanz, welche zum Rücksetzen des Fehlbedienungszählers berechtigt ist, verboten ist, durch das Passwort geschützten Aktionen auszuführen. Dies ist häufig im Umfeld qualifizierter Signaturanwendungen und dem Signaturpasswort anzutreffen.

15.6.5.1 Use Case Entsperren mit PUK, mit neuem Geheimnis

In dieser Variante enthält die APDU des RESET RETRY COUNTER Kommandos drei Parameter:

(N789) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N790) Der Parameter *PUK* enthält ein Geheimnis, welches diese Aktion autorisiert.

(N791) Der Parameter *newSecret* enthält das neue Benutzergeheimnis.

(N792) Die Parameter *PUK* und *newSecret* MÜSSEN gemäß (N81) kodiert sein.

(N793) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion

dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 108 verwendet werden.

Tabelle 108: RESET RETRY COUNTER, mit PUK, mit *newSecret*

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2C´	Instruction Byte gemäß [7816–4]
P1	´00´	Data enthält *PUK* und neues Benutzergeheimnis
P2	´XX´	passwordReference
Data	´XX…YY´	PUK || newSecret

15.6.5.2 Use Case Entsperren mit PUK, ohne neuem Geheimnis

In dieser Variante enthält die APDU des RESET RETRY COUNTER Kommandos zwei Parameter:

(N794) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N795) Der Parameter *PUK* enthält ein Geheimnis, welches diese Aktion autorisiert.

(N796) Der Parameter *PUK* MUSS gemäß (N81) kodiert sein.

(N797) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 109 verwendet werden.

Tabelle 109: RESET RETRY COUNTER, mit PUK, ohne *newSecret*

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2C´	Instruction Byte gemäß [7816–4]
P1	´01´	Data enthält nur *PUK*
P2	´XX´	passwordReference
Data	´XX…YY´	PUK

15.6.5.3 Use Case Entsperren ohne PUK, mit neuem Geheimnis

In dieser Variante enthält die APDU des RESET RETRY COUNTER Kommandos zwei Parameter:

(N798) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N799) Der Parameter *newSecret* enthält das neue Benutzergeheimnis.

(N800) Der Parameter *newSecret* MUSS gemäß (N81) kodiert sein.

(N801) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion

dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 110 verwendet werden.

Tabelle 110: RESET RETRY COUNTER, ohne PUK, mit *newSecret*

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2C´	Instruction Byte gemäß [7816–4]
P1	´02´	Data enthält nur neues Benutzergeheimnis
P2	´XX´	passwordReference
Data	´XX…YY´	newSecret

15.6.5.4 Use Case Entsperren ohne PUK, ohne neuem Geheimnis

In dieser Variante enthält die APDU des RESET RETRY COUNTER Kommandos einen Parameter:

(N802) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N803) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 111 verwendet werden.

Tabelle 111: RESET RETRY COUNTER, ohne PUK, ohne *newSecret*

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2C´	Instruction Byte gemäß [7816–4]
P1	´03´	Kommandodatenfeld fehlt
P2	´XX´	passwordReference

15.6.5.5 Antwort der Karte auf Entsperren eines Benutzergeheimnisses

Tabelle 112: RESET RETRY COUNTER Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiches Rücksetzen des Fehlbedienungszählers
´63 Cx´	WrongSecretWarning	*PUK* ist falsch
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 113: RESET RETRY COUNTER Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 85´	ShortPassword	*newData* enthält ein zu kurzes Passwort
´69 83´	CommandBlocked	Bedienungszähler der PUK abgelaufen
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (70): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N804) Ein COS KANN zusätzliche Trailer verwenden.

15.6.5.6 Kommandoabarbeitung innerhalb der Karte

(N805) Das COS MUSS die RESET RETRY COUNTER Varianten aus 15.6.5.1, 15.6.5.2, 15.6.5.3 und 15.6.5.4 unterstützen.
Das COS KANN weitere RESET RETRY COUNTER Varianten unterstützen.
Das COS KANN weitere RESET RETRY COUNTER Varianten ablehnen.

(N806) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N807) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N808) Wenn die konditional vorhandene und in *newSecret* kodierte Ziffernfolge für das Attribut *secret* des Passwortobjektes eine Länge hat, die kleiner als *affectedObject.minimumLength* ist, genau dann MUSS das Kommando mit dem Trailer ShortPassword terminieren.

(N809) Mittels clearPasswordStatus(*affectedObject*) MUSS der Sicherheitsstatus zurückgesetzt werden. Kom 3002

(N810) Wenn das Kommandodatenfeld einen Parameter *PUK* enthält, genau dann MUSS dieser mit dem Attribut *affectedObject.PUK* verglichen werden.

a. Wenn *affectedObject.pukUsage* den Wert null besitzt, genau dann MUSS das Kommando mit dem Trailer CommandBlocked terminieren.

b. *affectedObject.pukUsage* MUSS mit Transaktionsschutz um eins dekrementiert werden.

c. Wenn der Vergleich fehlschlägt, genau dann MUSS das Kommando mit dem Trailer WrongSecretWarning terminieren. Das Lownibble des Trailers MUSS dabei auf den Wert ´F´ gesetzt werden, wenn *affectedObject.pukUsage* größer als fünfzehn ist, andernfalls auf den Wert von *affectedObject.pukUsage*.

(N811) Das Attribut affectedObject.retryCounter MUSS auf den Wert affectedObject.startRetryCounter gesetzt werden.

(N812) Wenn das Kommandodatenfeld einen Parameter *newSecret* enthält, genau dann MUSS das Attribut

a. *affectedObject.secret* auf den in *newSecret* kodierten Wert gesetzt werden.

b. *affectedObject.transportStatus* auf den Wert „regularPassword“ = ´6´ (siehe Tabelle 9) gesetzt werden.

(N813) Die persistenten Änderungen in (N812), (N813) und (N811) MÜSSEN mit Transaktionsschutz ausgeführt werden.

(N814) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N815) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N816) Andernfalls MUSS als Trailer NoError gewählt werden.

(N817) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 113 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 113 MUSS eine höhere Priorität als WrongSecretWarning haben.

c. WrongSecretWarning MUSS eine höhere Priorität als UpdateRetryWarning haben.

d. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

(N818) Wenn die Ausführung dieses Kommandos durch einen Reset abgebrochen wird, dann gilt für die möglichen Änderung von *secret*, *transportStatus* und *retryCounter*:

a. Alle Änderungen SOLLEN gemeinsam in einer Transaktion geändert werden.

b. Die Änderungen KÖNNEN in jede in einer eigenen Transaktion durchgeführt werden. In diesem Fall MUSS *retryCounter* als letztes geändert werden.

15.6.6 VERIFY

Das Kommando VERIFY vergleicht das Attribut *secret* eines Passwortobjektes mit Daten, die im Datenfeld der Kommandonachricht enthalten sind. Falls der Vergleich erfolgreich ist, wird der Sicherheitszustand der Karte geändert. Welches Passwortobjekt betroffen ist, bestimmt eine Passwortreferenz, die als Parameter in der Kommandonachricht enthalten ist.

15.6.6.1 Use Case Vergleich eines Benutzergeheimnisses

In dieser Variante enthält die APDU des VERIFY Kommandos zwei Parameter:

(N819) Der Parameter *passwordReference* referenziert das von der Aktion betroffene Passwort und MUSS gemäß (N728) gewählt werden.

(N820) Der Parameter *verificationData* enthält das neue Benutzergeheimnis.

(N821) Der Parameter *verificationData* MUSS gemäß (N81) kodiert sein.

(N822) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 114 verwendet werden.

Tabelle 114: VERIFY

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´20´	Instruction Byte gemäß [7816–4]
P1	´00´	Data enthält Verifikationsdaten
P2	´XX´	passwordReference
Data	´XX…YY´	verificationData

15.6.6.2 Antwort der Karte auf Vergleich eines Benutzergeheimnisses

Tabelle 115: VERIFY Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreicher Vergleich
´63 Cx´	WrongSecretWarning	verificationData ist falsch
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 116: VERIFY Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	PasswordNotFound	adressiertes Passwort wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 83´	PasswordBlocked	abgelaufener Fehlbedienungszähler
´69 85´	PasswordNotUsable	Passwort im Transport–PIN Status
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (71): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N823) Ein COS KANN zusätzliche Trailer verwenden.

15.6.6.3 Kommandoabarbeitung innerhalb der Karte

(N824) Das COS MUSS die VERIFY Variante aus 15.6.6.1 unterstützen.
Das COS KANN weitere VERIFY Varianten unterstützen.
Das COS KANN weitere VERIFY Varianten ablehnen.

(N825) Es gilt *affectedObject* = searchPwd(*currentFolder*, *passwordReference*). Falls die Passwortsuche mit einem Fehler abbricht, genau dann MUSS das Kommando mit dem Trailer PasswordNotFound terminieren.

(N826) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N827) Wenn *affectedObject.retryCounter* den Wert null besitzt, genau dann MUSS das Kommando mit dem Trailer PasswordBlocked terminieren.

(N828) Wenn *affectedObject.transportStatus* nicht den Wert *regularPassword* besitzt, genau dann MUSS das Kommando mit dem Trailer PasswordNotUsable terminieren.

(N829) Das Attribut *affectedObject.secret* MUSS mit *verificationData* verglichen werden.

a. Wenn der Vergleich fehlschlägt, genau dann MUSS
   i. *affectedObject.retryCounter* um eins dekrementiert werden und
   ii. clearPasswordStatus(*affectedObject*) ausgeführt werden und
   iii. das Kommando mit dem Trailer WrongSecretWarning terminieren. Das Lownibble des Trailers MUSS dabei auf den Wert ´F´ gesetzt werden, wenn *affectedObject.retryCounter* größer als fünfzehn ist, andernfalls auf den Wert von *affectedObject.retryCounter*.

b. Wenn die Vergleichsoperation durch einen Reset abgebrochen wird, dann MUSS *affectedObject.retryCounter* um eins dekrementiert werden.

c. Wenn der Vergleich erfolgreich ist, genau dann MUSS
   i. setPasswordStatus(*affectedObject*) ausgeführt werden und
   ii. das Attribut *affectedObject.retryCounter* auf den Wert *affectedObject.startRetryCounter* gesetzt werden. Diese Änderung MÜSSEN mit Transaktionsschutz ausgeführt werden.

Kom 3002

Kom 3002

(N830) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N831) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N832) Andernfalls MUSS als Trailer NoError gewählt werden.

(N833) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 116 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 116 MUSS eine höhere Priorität als WrongSecretWarning haben.

c. WrongSecretWarning MUSS eine höhere Priorität als UpdateRetryWarning haben.

d. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.7 Komponentenauthentisierung

15.7.1 EXTERNAL AUTHENTICATE / MUTUAL AUTHENTICATE

Das Kommando EXTERNAL AUTHENTICATE prüft die Authentizität einer externen Instanz anhand der Antwort auf ein von der Karte generiertes Token mittels eines symmetrischen oder öffentlichen Schlüssels. Der Schlüssel wird vor der Authentisierungsoperation ausgewählt. Dies geschieht vor dem Senden dieses EXTERNAL AUTHENTICATE Kommandos durch ein MSE Set Kommando, siehe Kapitel 15.9.6.4, 15.9.6.5, 15.9.6.6. Die Antwort der externen Instanz ist als Parameter in der Kommandonachricht enthalten.

15.7.1.1 Use Case externe Authentisierung ohne Antwortdaten

In dieser Variante enthält die APDU des EXTERNAL AUTHENTICATE Kommandos einen Parameter:

(N834) Der Parameter *cmdData* enthält die Antwort der externen Instanz. Der Parameter *cmdData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *cmdData* ist abhängig von der mittels (N1013) oder (N1018) ausgewählten *algId*. Wenn *algId* gleich

a. aesRoleCheck ist, dann MUSS *cmdData* gleich 16 Oktett lang sein, informativ, ab Generation 2.
b. desRoleCheck ist, dann MUSS *cmdData* gleich 8 Oktett lang sein.
c. elcRoleCheck ist, dann MUSS *cmdData* … (für Generation 2 vervollständigen)
d. rsaRoleCheck ist, dann MUSS *cmdData* genauso viele Oktette enthalten, wie der Modulus des Authentisierungsschlüssels.
e. rsaSessionkey4SM ist, dann MUSS *cmdData* genauso viele Oktette enthalten, wie der Modulus des Authentisierungsschlüssels.
f. symSessionkey4TC ist, dann MUSS … (informativ, findet nur auf Seiten der Gegenstelle statt, siehe Abbildung 6)

(N835) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 117 verwendet werden.

Tabelle 117: EXTERNAL AUTHENTICATE ohne Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´82´	Instruction Byte gemäß [7816–4]
P1	´00´	Information zum Algorithmus bereits in der Karte vorhanden
P2	´00´	Schlüsselreferenz bereits in der Karte vorhanden
Data	´XX…YY´	cmdData

15.7.1.2 Use Case externe Authentisierung mit Antwortdaten

In dieser Variante enthält die APDU des MUTUAL AUTHENTICATE Kommandos zwei Parameter:

(N836) Der Parameter *cmdData* enthält die Antwort der externen Instanz. Der Parameter *cmdData* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *cmdData* ist abhängig von der mittels (N1023) ausgewählten *algId*. Wenn *algId* gleich

a. aesSessionkey4SM ist, dann MUSS *cmdData* … (für Generation 2 vervollständigen).

b. desSessionkey4SM ist, dann MUSS *cmdData* 104 Oktett lang sein.

(N837) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N838) Es MUSS eine Case 4 Kommando APDU Kapitel 12.7.4.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 118 verwendet werden.

Tabelle 118: MUTUAL AUTHENTICATE mit Antwortdaten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´82´	Instruction Byte gemäß [7816–4]
P1	´00´	Information zum Algorithmus bereits in der Karte vorhanden
P2	´00´	Schlüsselreferenz bereits in der Karte vorhanden
Data	´XX…YY´	cmdData
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.7.1.3 Antwort der Karte auf externe Authentisierung

Tabelle 119: EXTERNAL AUTHENTICATE Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	authData	Authentisierungsdaten (konditional, abhängig von *algId*)
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Authentisierung

Tabelle 120: EXTERNAL AUTHENTICATE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoRandom	keine Zufallszahl vorhanden, siehe auch Kapitel 16.1
´69 85´	NoKeyReference	kein Authentisierungsschlüssel ausgewählt
´6A 88´	KeyNotFound	Authentisierungsschlüssel nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´64 00´	AuthenticationFailure	Authentisierung fehlgeschlagen
´69 85´	NoPrkReference	kein Entschlüsselungsschlüssel ausgewählt
´6A 88´	PrKNotFound	Entschlüsselungsschlüssel nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt

Hinweis (72): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N839) Ein COS KANN zusätzliche Trailer verwenden.

15.7.1.4 Kommandoabarbeitung innerhalb der Karte

(N840) Das COS MUSS die EXTERNAL AUTHENTICATE Varianten aus 15.7.1.1 und 15.7.1.2 unterstützen.
Das COS KANN weitere EXTERNAL AUTHENTICATE Varianten unterstützen.

Das COS KANN weitere EXTERNAL AUTHENTICATE Varianten ablehnen.

(N841) Wenn keine *RND.ICC* zur Verfügung steht (siehe (N993), (N299)b und Kapitel 16), genau dann MUSS das Kommando mit dem Trailer NoRandom terminieren.

(N842) Wenn channelContext.keyReferenceList.externalAuthenticate

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist und eine Referenz auf ein öffentliches Authentisierungsobjekt enthält, dann wird *affectedObject* = SearchPublicKey(
   channelContext.keyReferenceList.externalAuthenticate.keyReference,
   channelContext.keyReferenceList.externalAuthenticate.algID,
   currentFolder
) gesetzt. Gemäß Kapitel 10.2.4 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

c. nicht leer ist und eine Referenz auf ein symmetrisches Authentisierungsobjekt enthält, dann wird *affectedObject* = SearchSecretKey(
   currentFolder,
   channelContext.keyReferenceList.externalAuthenticate.keyReference,
   channelContext.keyReferenceList.externalAuthenticate.algID
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler

   i. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

   ii. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

(N843) Falls *affectedObject* ein symmetrischer Schlüssel ist und AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N844) Die Antwort *cmdData* der externen Instanz wird wie folgt weiterverarbeitet:
Wenn *channelContext.keyReferenceList.externalAuthenticate.algID* den Wert

a. aesRoleCheck besitzt, dann gilt *authData* = ´´ (leerer Oktettstring) und
   i. (*resp'* , *T*) = AES_CTR_ENC(
      affectedObject.encKey,
      0,
      RND.ICC
)
   ii. Falls *resp'* ungleich *cmdData* ist, genau dann MUSS das Kommando mit dem Trailer AuthenticationFailure terminieren.
b. aesSessionkey4SM besitzt, dann … (für Generation 2 vervollständigen)
c. desRoleCheck besitzt, dann gilt *authData* = ´´ (leerer Oktettstring) und
   i. (*resp'* , *T*) = 3TDES_CBC_ENC(
      affectedObject.encKey,
      0,
      RND.ICC
)
   ii. Falls *resp'* ungleich *cmdData* ist, genau dann MUSS das Kommando mit dem Trailer AuthenticationFailure terminieren.
d. desSessionkey4SM besitzt, dann werden folgende Schritte ausgeführt:
   i. *cmdData* wird wie folgt aufgeteilt:
      1. *cmdData* = C_1 || MAC_1
      2. OctetLength(MAC_1) = 8.
   ii. *out* = VERIFY_Retail_MAC(*affectedObject.macKey*, MAC_1, C_1)
   iii. Falls *out* den Wert INVALID besitzt, dann MUSS das Kommando mit dem Trailer AuthenticationFailure terminieren.
   iv. S = 3TDES_CBC_DEC(*affectedObject.encKey*, 0, C_1)
   v. S wird wie folgt aufgeteilt:
      1. S = RND.ext || ICCSN8.ext || RND.int || ICCSN8.int || KD.e
      2. Es sei 8 = OctetLength(RND.ext)
      3. Es sei 8 = OctetLength(ICCSN8.ext)
      4. Es sei 8 = OctetLength(RND.ICC)
      5. Es sei 8 = OctetLength(ICCSN8.int)
      6. Es sei 64 = OctetLength(KD.e)
   vi. Das Kommando MUSS genau dann mit AuthenticationFailure terminieren, falls

1. RND.int ungleich *RND.ICC* (siehe (N841)) ist, oder
2. ICCSN8.int ungleich *iccsn8* (siehe (N199)c) ist.

vii. Setze KD.i = RAND(64)

viii. Setze *R* = RND.int || ICCSN8.int || RND.ext || ICCSN8.ext || KD.i

ix. Setze C_2 = 3TDES_CBC_ENC(*affectedObject.encKey*, 0, *R*)

x. Setze MAC_2 = CALCULATE_Retail_MAC(*affectedObject.macKey*, C_2)

xi. Setze *authData* = C_2 || MAC_2

xii. Die Oktettstrings KD.e und KD.i MÜSSEN an den Secure Messaging Layer übergeben werden (siehe 14.1).

e. elcRoleCheck besitzt, dann gilt … (für Generation 2 vervollständigen)

f. rsaRoleCheck besitzt, dann gilt *authData* = ´´ (leerer Oktettstring) und mit *PuK* = *affectedObject.publicKey*

i. Setze M_2 = RND.ICC || *iccsn8*

ii. (*out*, *M*) = RSA_ISO9796_2_DS1_VERIFY(*PuK*, *cmdData*, M_2)
Falls diese Operation mit einem Fehler abbricht, oder *out* gleich *False* ist, dann MUSS das Kommando mit dem Trailer AuthenticationFailure terminieren.

g. rsaSessionkey4SM besitzt, dann gilt *authData* = ´´ (leerer Oktettstring) und es werden folgende Schritte ausgeführt:

i. Wenn *channelContext.keyReferenceList.internalAuthenticate*

1. leer ist, genau dann MUSS das Kommando mit dem Trailer NoPrkReference terminieren.

2. nicht leer ist, dann wird *tmpObject* = SearchSecretKey(
currentFolder,
channelContext.keyReferenceList.internalAuthenticate.keyReference,
channelContext.keyReferenceList.internalAuthenticate.algID
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche den Fehler

A. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

B. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

ii. Wenn AccessRuleEvaluation(*tmpObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

iii. Setze tmpData = $\text{cmdData}^{\text{PrK.d}}$ mod PrK.n,
mit PrK = tmpObject.privateKey

iv. Setze M_2 = *RND.ICC* || *iccsn8*

v. (*out*, *M*) = RSA_ISO9796_2_DS1_VERIFY(*PuK*, *tmpData*, M_2)
Falls diese Operation mit einem Fehler abbricht, oder *out* gleich *False*

ist, dann MUSS das Kommando mit dem Trailer AuthenticationFailure terminieren.

vi. M wird aufgeteilt in $M = M_1 \| M_2$. Die 64 LSByte in M_1 MÜSSEN als KD.e an den Secure Messaging Layer übergeben werden (siehe 14.1).

(N845) Falls das Kommando mit dem Trailer AuthenticationFailure terminiert, genau dann MUSS clearSecurityStatus(*affectedObject*) ausgeführt werden.

(N846) Falls das Kommando mit dem Trailer NoError antwortet, genau dann MUSS setSecurityStatus(*affectedObject*) ausgeführt werden.

(N847) Als Datenfeld der Antwortnachricht MUSS der (möglicherweise leere) Oktettstring *authData* verwendet werden.

(N848) Als Trailer MUSS NoError gewählt werden.

(N849) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 120 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 120 MUSS eine höhere Priorität als NoError haben.

15.7.2 GENERAL AUTHENTICATE

(N850) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.7.3 GET SECURITY STATUS KEY

Hinweis (73): Dieses Kommando ist nicht in der Normenreihe ISO/IEC 7816 enthalten. Es ließe sich kombinieren mit dem Kommando EXTERNAL AUTHENTICATE.

Hinweis (74): Bei der Spezifikation der Kommando APDU wurde das MSE Set Kommando zugrunde gelegt.

Hinweis (75): Motivation für das Kommando: Ausgehend von der Annahme, dass ein bestimmtes Kommando mit dem Trailer SecurityStatusNotSatisfied terminiert, ist für die steuernde Software gelegentlich nicht offensichtlich, was zu tun ist, um Zugriff zu erhalten. Selbst bei Kenntnis der Zugriffsregel des Objektes lässt sich lediglich beurteilen, ob Secure Messaging Vorgaben eingehalten wurden. Falls eine PIN Verifikation erforderlich ist, so lässt sich der zugehörige Sicherheitszustand mittels GET PIN STATUS erfragen. Falls aber mehrere Sicherheitszustände von Schlüsseln ins Spiel kommen, dann hat die steuernde Software nur zwei Möglichkeiten: Entweder „try and error“, was schlecht für die Performance ist, oder sie baut die COS spezifische Zustandsmaschine für den Sicherheitsstatus von Schlüsseln nach (und hofft, dass interner und externer Zustand synchron bleiben).

Das Kommando GET SECURITY STATUS KEY wird verwendet um den Sicherheitszustand von Schlüsselobjekten zu erfragen. Welcher Sicherheitszustand ausgelesen wird, bestimmt eine Referenz, die als Parameter in den Kommandodaten enthalten ist.

15.7.3.1 Use Case Auslesen Sicherheitsstatus symmetrischer Schlüssels

In dieser Variante enthält die APDU des MSE Kommandos einen Parameter:

(N851) Der Parameter *cmdData* enthält eine Schlüsselreferenz. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N852) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 121 verwendet werden.

Tabelle 121: GET SECURITY STATUS KEY, symmetrischer Schlüssel

	Inhalt	Beschreibung
CLA	´80´	CLA Byte gemäß [7816–4] wird hier „proprietary“ angezeigt
INS	´82´	Instruction Byte
P1	´80´	Auslesen Sicherheitszustand
P2	´00´	–
Data	´XX…YY´	´83 01 || *cmdData* ´

15.7.3.2 Use Case Auslesen des Sicherheitsstatus einer Rolle

In dieser Variante enthält die APDU des MSE Kommandos einen Parameter:

(N853) Der Parameter *cmdData* enthält eine Rollenkennung. Wert und Kodierung MÜSSEN gemäß 8.1.1.4 und (N57) gewählt werden.

(N854) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 122 verwendet werden.

Tabelle 122: GET SECURITY STATUS KEY, Rollenkennung

	Inhalt	Beschreibung
CLA	´80´	CLA Byte gemäß [7816–4] wird hier „proprietary“ angezeigt
INS	´82´	Instruction Byte
P1	´80´	Auslesen Sicherheitszustand
P2	´00´	–
Data	´XX…YY´	´5F4C 07 || *cmdData* ´

15.7.3.3 Use Case Auslesen des Sicherheitsstatus einer Bitliste, G2 (informativ)

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.

15.7.3.4 Antwort der Karte auf Auslesen des Sicherheitsstatus eines Schlüssels

Tabelle 123: GET SECURITY STATUS KEY Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	Authentisierungsstatus ist gesetzt
´63 CF´	NoAuthentication	Authentisierungsstatus ist nicht gesetzt

Tabelle 124: GET SECURITY STATUS KEY Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	KeyNotFound	adressiertes Schlüsselobjekt wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt

Hinweis (76): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N855) Ein COS KANN zusätzliche Trailer verwenden.

15.7.3.5 Kommandoabarbeitung innerhalb der Karte

(N856) Das COS MUSS die GET SECURITY STATUS KEY Variante aus 15.7.3.1 und 15.7.3.2 unterstützen.
Das COS KANN weitere GET SECURITY STATUS KEY Varianten unterstützen.

Das COS KANN weitere GET SECURITY STATUS KEY Varianten ablehnen.

(N857) Falls *cmdData* die Referenz eines symmetrischen Schlüssels gemäß Kapitel 15.7.3.1 ist, dann

a. wird *affectedObject* = SearchSecretKey(
currentFolder,
cmdData,
„Wildcard“
) gesetzt. Gemäß Kapitel 10.2.3 und KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren. Der Fehler notSupported ist wegen der Wildcard Suche nicht möglich.

b. Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

c. Falls *affectedObject* in *globalSecurityList* (siehe (N299)e) oder in *dfSpecificSecurityList* (siehe (N299)f) enthalten ist, dann MUSS als Trailer NoError verwendet werden.

(N858) Falls *cmdData* eine Rolle gemäß Kapitel 15.7.3.2. ist und diese Rolle in *globalSecurityList* (siehe (N299)e) oder in *dfSpecificSecurityList* (siehe (N299)f) enthalten ist, dann MUSS als Trailer NoError verwendet werden.

(N859) Falls *cmdData* eine bitListe … (für Generation 2 vervollständigen)

(N860) Andernfalls MUSS als Trailer NoAuthentication verwendet werden.

(N861) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 124 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 124 MUSS eine höhere Priorität als NoError haben.

c. NoError MUSS eine höhere Priorität als NoAuthentication haben.

15.7.4 INTERNAL AUTHENTICATE

Das Kommando INTERNAL AUTHENTICATE berechnet Authentisierungsdaten zu einem Token mittels eines symmetrischen oder privaten Schlüssels. Der Schlüssel wird vor der Authentisierungsoperation ausgewählt. Dies geschieht vor dem Senden dieses INTERNAL AUTHENTICATE Kommandos durch ein MSE Set Kommando, siehe Kapitel 15.9.6.3. Das Token ist als Parameter in der Kommandonachricht enthalten.

15.7.4.1 Use Case interne Authentisierung

In dieser Variante enthält die APDU des INTERNAL AUTHENTICATE Kommandos zwei Parameter:

(N862) Der Parameter *token* enthält die zu authentisierenden Daten. Der Parameter *token* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *token* ist abhängig von der mittels (N1008) ausgewählten *algId*. Wenn *algId* gleich

a. aesRoleAuthentication ist, MUSS *token* gleich 16 Oktett lang sein.

b. desRoleAuthentication ist, MUSS *token* gleich 8 Oktett lang sein.

c. elc… (für Generation 2 vervollständigen)

d. rsaClientAuthentication ist, dann unterliegt die Länge von *token* nur den Beschränkungen aus Kapitel 12.5.5.

e. rsaRoleAuthentication ist, dann MUSS *token* gleich 16 Oktett lang sein.

f. rsaSessionkey4SM ist, dann MUSS *token* gemäß (N1067) gewählt werden.

g. symSessionkey4TC ist, dann MUSS … (informativ, findet nur auf Seiten der Gegenstelle statt, siehe Abbildung 6)

(N863) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N864) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 oder über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 125 verwendet werden.

Tabelle 125: INTERNAL AUTHENTICATE

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´88´	Instruction Byte gemäß [7816–4]
P1	´00´	Information zum Algorithmus bereits in der Karte vorhanden
P2	´00´	Schlüsselreferenz bereits in der Karte vorhanden
Data	´XX…YY´	token
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.7.4.2 Antwort der Karte auf interne Authentisierung

Tabelle 126: INTERNAL AUTHENTICATE Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	response	authentisierende Daten
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Authentisierungsoperation

Tabelle 127: INTERNAL AUTHENTICATE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoKeyReference	kein Authentisierungsschlüssel ausgewählt
´6A 88´	KeyNotFound	Authentisierungsschlüssel nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 85´	NoPukReference	kein Verschlüsselungsschlüssel ausgewählt
´6A 88´	PukNotFound	Verschlüsselungsschlüssel nicht gefunden

Hinweis (77): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N865) Ein COS KANN zusätzliche Trailer verwenden.

15.7.4.3 Kommandoabarbeitung innerhalb der Karte

(N866) Das COS MUSS die INTERNAL AUTHENTICATE Variante aus 15.7.4.1 unterstützen.
Das COS KANN weitere INTERNAL AUTHENTICATE Varianten unterstützen.
Das COS KANN weitere INTERNAL AUTHENTICATE Varianten ablehnen.

(N867) Wenn channelContext.keyReferenceList.internalAuthenticate

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist, dann wird affectedObject = SearchSecretKey(
   currentFolder,
   channelContext.keyReferenceList.internalAuthenticate.keyReference,
   channelContext.keyReferenceList.internalAuthenticate.algID
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen,

dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler

i. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

ii. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

(N868) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N869) Die Antwort response wird wie folgt berechnet:
Wenn channelContext.keyReferenceList.internalAuthenticate.algID den Wert

a. aesRoleAuthentication besitzt, dann gilt
(*response* , *T*) = AES_CTR_ENC(
affectedObject.encKey,
0,
token
)

b. desRoleAuthentication besitzt, dann gilt
(*response* , *T*) = 3TDES_CBC_ENC(
affectedObject.encKey,
0,
token
)

c. elc* besitzt, dann gilt … (für Generation 2 vervollständigen)

d. rsaClientAuthentication besitzt, dann gilt
response = RSASSA_PSS_SIGN(*affectedObject.privateKey, token*)

e. rsaRoleAuthentication besitzt, dann gilt mit
PrK = *affectedObject.privateKey*

i. PRND = RAND(OctetLength(*PrK.n*) – 34)

ii. *M* = PRND || *token*

iii. (*response*, M_2) = RSA_ISO9796_2_DS1_SIGN(*PrK*, *M*)
Hinweis: Die Länge von PRND ist so gewählt, dass M_2 = *token* ist. Darum ist es nicht nötig M_2 in die Antwortnachricht einzustellen.

f. rsaSessionkey4SM besitzt, dann gilt mit *PrK* = *affectedObject.privateKey*

i. KD.i= RAND(64)
Der Oktettstring KD.i MUSS an den Secure Messaging Layer übergeben werden (siehe 14.1).

ii. PRND = RAND(OctetLength(*PrK.n*) – 34 – OctetLength(KD.i)) Kom 3002

iii. *M* = PRND || KD.i || *token*

iv. (*sig*, M_2) = RSA_ISO9796_2_DS1_SIGN(*PrK*, *M*)
Hinweis: Die Länge von PRND ist so gewählt, dass M_2 = *token* ist. Darum ist es nicht nötig M_2 in die Antwortnachricht einzustellen.

v. Wenn channelContext.keyReferenceList.externalAuthenticate

1. leer ist, genau dann MUSS das Kommando mit dem Trailer NoPukReference terminieren.

2. nicht leer ist, dann wird *tmpObject* = SearchPublicKey(
   channelContext.keyReferenceList.externalAuthenticate.keyReference,
   channelContext.keyReferenceList.externalAuthenticate.algID
   currentFolder
) gesetzt. Gemäß Kapitel 10.2.4 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Genau dann MUSS das Kommando mit dem Trailer PukNotFound terminieren.

vi. response = $sig^{PuK.e}$ mod PuK.n, mit PuK = tmpObject.publicKey.

(N870) Als Datenfeld der Antwortnachricht MUSS *response* verwendet werden.

(N871) Als Trailer MUSS NoError gewählt werden.

(N872) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 127 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 127 MUSS eine höhere Priorität als NoError haben.

15.8 Kryptoboxkommandos

15.8.1 PSO Compute Digital Signature

Das Kommando PSO Compute Digital Signature signiert Daten mittels eines privaten Schlüssels. Der private Schlüssel und der zu verwendende Algorithmus wird vor der Signieroperation ausgewählt. Dies geschieht vor dem Senden dieses PSO Compute Digital Signature Kommandos durch ein MSE Set Kommando, siehe Kapitel 15.9.6.7. Die zu signierenden Daten sind als Parameter in der Kommandonachricht enthalten.

15.8.1.1 Use Case Signieren des Datenfeldes, ohne „message recovery“

Diese Variante gilt für folgende Algorithmen (siehe (N176) und (N183)): rsaClientAuthentication, signPKCS1_V1_5, signPSS, signECDSA.

In dieser Variante enthält die APDU des PSO Compute DS Kommandos zwei Parameter:

(N873) Der Parameter *dataToBeSigned* enthält die zu signierenden Daten. Der Parameter *dataToBeSigned* ist ein Oktettstring mit beliebigem Inhalt. Die Länge von *dataToBeSigned* ist abhängig von der mittels (N1028) ausgewählten *algId*. Wenn *algId* gleich

a. rsaClientAuthentication ist, dann unterliegt die Anzahl Oktette in *dataToBeSigned* nur den Beschränkungen aus Kapitel 12.5.5.

b. signPKCS1_V1_5 ist, dann MUSS die Anzahl Oktette in *dataToBeSigned* kleiner als 0,4 OctetLength(*n*) sein, mit *n* als Modulus des ausgewählten Signaturschlüssels.

c. signPSS ist, dann unterliegt die Anzahl Oktette in *dataToBeSigned* nur den Beschränkungen aus Kapitel 12.5.5.

d. signECDSA ist, dann unterliegt die Anzahl Oktette in *dataToBeSigned* nur den Beschränkungen aus Kapitel 12.5.5.

(N874) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N875) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 128 verwendet werden.

Tabelle 128: PSO Compute DS, Signieren des Datenfeldes ohne „message recovery“

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´9E´	Beschreibung der Antwortdaten, hier digitale Signature
P2	´9A´	Beschreibung der Kommandodaten, hier „zu signierende Daten“
Data	´XX…YY´	dataToBeSigned
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.8.1.2 Use Case Signieren des Datenfeldes, mit „message recovery“

Diese Variante gilt für folgenden Algorithmus (siehe (N183)): sign9796_2_DS2.

In dieser Variante enthält die APDU des PSO Compute DS Kommandos drei Parameter:

(N876) Der Parameter *M1* enthält den „recoverable part“ der zu signierenden Nachricht. Der Parameter *M1* ist ein Oktettstring mit beliebigem Inhalt.

(N877) Der Parameter *hashM2* enthält den Hashwert des „non recoverable part“ der zu signierenden Nachricht. Der Parameter *hashM2* ist ein Oktettstring mit beliebigem Inhalt.

(N878) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N879) Die Parameter *M1* und *hashM2* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N886)a spezifiziert.

(N880) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 129 verwendet werden.

Tabelle 129: PSO Compute DS, Signieren des Datenfeldes mit „message recovery“

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´9E´	Beschreibung der Antwortdaten, hier digitale Signature
P2	´AC´	Beschreibung der Kommandodaten, hier zu signierende Daten als DE
Data	´XX…YY´	signInput9796_2_DS2
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.8.1.3 Antwort der Karte auf Signieren von Daten

Tabelle 130: PSO Compute DS Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	signature	Signatur
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Signaturoperation

Tabelle 131: PSO Compute DS Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoKeyReference	kein Signierschlüssel ausgewählt
´6A 88´	KeyNotFound	Schlüssel nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´64 00´	KeyInvalid	Schlüsseldaten fehlen

Hinweis (78): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N881) Ein COS KANN zusätzliche Trailer verwenden.

15.8.1.4 Kommandoabarbeitung innerhalb der Karte

(N882) Das COS MUSS die PSO Compute DS Variante aus 15.8.1.1 und 15.8.1.2 unterstützen.
Das COS KANN weitere PSO Compute DS Varianten unterstützen.
Das COS KANN weitere PSO Compute DS Varianten ablehnen.

(N883) Wenn channelContext.keyReferenceList.signatureCreation

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist, dann wird affectedObject = SearchSecretKey(
    currentFolder,
    channelContext.keyReferenceList.signatureCreation.keyReference,
    channelContext.keyReferenceList.signatureCreation.algID
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler

    i. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

    ii. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

(N884) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N885) Wenn *affectedObject.keyAvailable* den Wert False besitzt, genau dann MUSS das Kommando mit dem Trailer KeyInvalid terminieren.

(N886) Die Signatur signature wird wie folgt berechnet:
Wenn channelContext.keyReferenceList.signatureCreation.algID den Wert

a. sign9796_2_DS2 besitzt, dann MUSS gelten

    i. signInput9796_2_DS2 = plainDO || hashDO.

    ii. *plainDO* = ´80 – L_{80} – *M1*´.

    iii. *hashDO* = ´90 – L_{90} – *hashM2*´.

    iv. *signature* = RSA_ISO9796_2_DS2_SIGN(
        affectedObject.privateRsaKey,

M1,
hashM2,
)

b. signPKCS1_V1_5 besitzt, dann gilt:
signature = RSASSA_PCKS1_V1_5_SIGN(
affectedObject.privateRsaKey,
dataToBeSigned
)

c. rsaClientAuthentication oder signPSS besitzt, dann gilt
signature = RSASSA_PSS_SIGN(
affectedObject.privateRsaKey,
dataToBeSigned
)

d. signECDSA besitzt, dann gilt
signature = *R* || *S*, mit
(*R*, *S*) = ELC_SIG(*affectedObject.privateElcKey*, *dataToBeSigned*)

(N887) Als Datenfeld der Antwortnachricht MUSS *signature* verwendet werden.

(N888) Als Trailer MUSS NoError gewählt werden.

(N889) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 131 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 131 MUSS eine höhere Priorität als NoError haben.

Hinweis (79): Falls bei der Operation in (N886)b die Anforderung aus (N873)a eingehalten wird, dann ist es nicht möglich, dass in (N32)a der Fehler DigestInfoTooLong geworfen wird. Deshalb ist dieser Fehler für dieses Dokument nicht relevant.

15.8.2 PSO Decipher

Das Kommando PSO Decipher entschlüsselt Daten mittels eines privaten Schlüssels. Der private Schlüssel und der zu verwendende Algorithmus wird vor der Entschlüsselungsoperation ausgewählt. Dies geschieht vor dem Senden dieses PSO Decipher Kommandos durch ein MSE Set Kommando, siehe 15.9.6.9. Die zu entschlüsselnden Daten sind als Parameter in der Kommandonachricht enthalten.

15.8.2.1 Use Case Entschlüsseln mittels RSA, G1 (normativ)

Diese Variante gilt für folgende Algorithmen (siehe (N179)): rsaDecipherPKCS1_V1_5, rsaDecipherOaep.

In dieser Variante enthält die APDU des PSO Decipher Kommandos zwei Parameter:

(N890) Der Parameter *C* enthält die zu entschlüsselnden Daten. Der Parameter *C* ist ein Oktettstring mit beliebigem Inhalt. Die Anzahl Oktette in *C* MUSS identisch sein zu OctetLength(*n*) mit *n* als Modulus des Schlüssels, der zum Entschlüsseln verwendet wird.

(N891) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N892) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 132 verwendet werden.

Tabelle 132: PSO Decipher, Entschlüsseln mittels RSA

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´80´	Beschreibung der Antwortdaten, hier Klartext
P2	´86´	Beschreibung der Kommandodaten, hier Chiffrat
Data	´XX…YY´	´00´ || *C* = PaddingIndicator || Cryptogram
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.8.2.2 Use Case Entschlüsseln mittels ELC, G2 (informativ)

Diese Variante gilt für folgenden Algorithmus (siehe (N179)): elcSharedSecretCalculation.

In dieser Variante enthält die APDU des PSO Decipher Kommandos vier Parameter:

(N893) Der Parameter PO_A enthält einen Punkt auf einer elliptischen Kurve. Dieser Punkt wurde vom Sender der Nachricht gewählt. Der Parameter PO_A ist ein Oktettstring, dessen Inhalt so gewählt werden SOLL, dass bei der Dekodierung kein Fehler auftritt, siehe (N48)a.

(N894) Der Parameter *C* enthält die zu entschlüsselnden Daten. Der Parameter *C* ist ein Oktettstring mit beliebigem Inhalt.

(N895) Der Parameter *T* enthält einen MAC, der die Integrität von *C* schützt. Der Parameter *T* ist ein Oktettstring, dessen Länge und Inhalt so gewählt werden SOLL, dass bei der MAC Prüfung kein Fehler auftritt.

(N896) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N897) Die Parameter PO_A, *C* und *T* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N903) spezifiziert.

(N898) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 133 verwendet werden.

Tabelle 133: PSO Decipher, Entschlüsseln mittels elliptischer Kurven

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´80´	Beschreibung der Antwortdaten, hier Klartext
P2	´86´	Beschreibung der Kommandodaten, hier Chiffrat
Data	´XX…YY´	cipher
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

Hinweis (80): Möglicherweise wird P2 in einer späteren Dokumentenversion geändert.

15.8.2.3 Antwort der Karte auf Entschlüsseln von Daten

Tabelle 134: PSO Decipher Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	plain	Klartext
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Entschlüsselungsoperation

Tabelle 135: PSO Decipher Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoKeyReference	kein Schlüssel für Entschlüsselung ausgewählt
´6A 88´	KeyNotFound	Schlüssel nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´6A 80´	WrongCiphertext	Fehler beim Entschlüsseln des Kryptogramms

Hinweis (81): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N899) Ein COS KANN zusätzliche Trailer verwenden.

15.8.2.4 Kommandoabarbeitung innerhalb der Karte

(N900) Das COS MUSS die PSO Decipher Variante aus 15.8.2.1 unterstützen.
Das COS KANN weitere PSO Decipher Varianten unterstützen.
Das COS KANN weitere PSO Decipher Varianten ablehnen.

(N901) Wenn channelContext.keyReferenceList.dataDecipher

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist, dann wird affectedObject = SearchSecretKey(
   currentFolder,
   channelContext.keyReferenceList.dataDecipher.keyReference,
   channelContext.keyReferenceList.dataDecipher.algID
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler

i. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

ii. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

(N902) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N903) Die Klartextnachricht *plain* wird wie folgt berechnet:
Wenn *channelContext.keyReferenceList.dataDecipher.algID* den Wert

a. rsaDecipherPKCS1_V1_5 besitzt, dann gilt
plain = RSAES_PKCS1_V1_5_DECRYPT(*affectedObject.privateRsaKey*, *C*)

b. rsaDecipherOaep besitzt, dann gilt
plain = RSAES_OAEP_DECRYPT(*affectedObject.privateRsaKey*, *C*)

c. elcSharedSecretCalculation besitzt, dann gilt

i. *cipher* = ´A6 – L_{A6} – (*keyDO* || *cipherDO* || *macDO*).

ii. *keyDO* = ´7F49 – L_{7F49} – (86 – L_{86} – PO_A)´.

iii. *cipherDO* = ´86 – L_{86} – 02 || *C*´.

iv. *macDO* = ´8E – L_{8E} – T´.
Hinweis: cipher ist hier identisch zu (N918)a und (N944)c definiert.

v. *plain* = ELC_DEC(PO_A, *affectedObject.privateElcKey*, *C*, *T*).

(N904) Wenn die Entschlüsselung gemäß (N903) mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren.

(N905) Als Datenfeld der Antwortnachricht MUSS *plain* verwendet werden.

(N906) Als Trailer MUSS NoError gewählt werden.

(N907) Für die Priorität der Trailer gilt:

Die Priorität der Trailer in Tabelle 135 ist herstellerspezifisch.

Jeder Trailer in Tabelle 135 MUSS eine höhere Priorität als NoError haben.

15.8.3 PSO Encipher, G2 (informativ)

Das Kommando PSO Encipher verschlüsselt Daten mittels eines öffentlichen Schlüssels. Die zu verschlüsselnden Daten und der öffentliche Schlüssel sind als Parameter in der Kommandonachricht enthalten.

15.8.3.1 Use Case Verschlüsseln von Daten

In dieser Variante enthält die APDU des PSO Encipher Kommandos fünf Parameter:

(N908) Der Parameter *algID* enthält Informationen zum Algorithmus, der für die Verschlüsselung verwendet wird. Der Parameter *algID* MUSS aus der Menge … (für Generation 2 festlegen) gewählt werden.

(N909) Der Parameter *oid* enthält einen Objektidentifier, der die zu verwendende elliptische Kurve referenziert. Der Parameter *oid* ist ein Oktettstring, dessen Länge und Inhalt so gewählt werden SOLLEN, dass damit eine in der Karte gespeicherte Kurve referenziert wird.

(N910) Der Parameter PO_B enthält einen Punkt auf einer elliptischen Kurve. Dieser Punkt repräsentiert den öffentlichen Schlüssel des Empfängers. Der Parameter PO_B ist ein Oktettstring, dessen Inhalt so gewählt werden SOLL, dass bei der Dekodierung kein Fehler auftritt, siehe (N48)a.

(N911) Der Parameter *M* enthält die zu verschlüsselnden Daten. Der Parameter *M* ist ein Oktettstring mit beliebigem Inhalt.

(N912) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N913) Die Parameter PO_B, *oid* und *M* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N917) und (N918) spezifiziert.

(N914) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 136 verwendet werden.

Tabelle 136: PSO Encipher, Verschlüsseln von Daten

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´86´	Beschreibung der Antwortdaten, hier Chiffrat
P2	´80´	Beschreibung der Kommandodaten, hier Klartext
Data	´XX…YY´	plain
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

Hinweis (82): Möglicherweise wird P1 in einer späteren Dokumentenversion geändert.

15.8.3.2 Antwort der Karte auf Verschlüsseln von Daten

Tabelle 137: PSO Encipher Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	cipher	Chiffrat
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Verschlüsselungsoperation

Tabelle 138: PSO Encipher Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´64 00´	CurveNotFound	referenzierte Kurve nicht gefunden
´64 00´	EncipherError	Verschlüsselung fehlgeschlagen

Hinweis (83): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N915) Ein COS KANN zusätzliche Trailer verwenden.

15.8.3.3 Kommandoabarbeitung innerhalb der Karte

(N916) Das COS MUSS die PSO Encipher Variante aus 15.8.2.1 unterstützen.
Das COS KANN weitere PSO Encipher Varianten unterstützen.
Das COS KANN weitere PSO Encipher Varianten ablehnen.

(N917) Das Datenfeld der Kommandonachricht wird wie folgt zerlegt:

a. plain = algDO || plainDO.

b. *algDO* = ´80 01 *algID* ´.

(N918) Das Chiffrat *cipher* wird wie folgt berechnet:
Wenn *algID* den Wert

a. elcSharedSecretCalculation besitzt, dann MUSS gelten:

i. *plainDO* = ´A0 – L_{A0} – (*oidDO* || *keyDO* || *mDO*).

ii. *oidDO* = ´06 – L_{06} – *oid* ´.

iii. *keyDO* = ´7F49 – L_{7F49} – (86 – L_{86} – PO_B)´.

iv. *mDO* = ´80 – L_{80} – *M*´.

v. Im Attribut *domainParamterList* (siehe (N199)f) wird nach einem Listenelement gesucht, dessen Attribut *OID* identisch ist zu *oid* aus der Kommandonachricht. Wenn ein solches Listenelement

1. nicht existiert, genau dann MUSS das Kommando mit dem Trailer CurveNotFound terminieren.

2. existiert, dann werden die darin enthaltenen Domainparameter im Folgenden mit *dP* bezeichnet.

vi. (PO_A, *C*, *T*) = ELC_ENC(*M*, PO_B, *dP*)
Wenn diese Funktion mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren. Andernfalls wird ein DER–TLV kodierter Oktettstring *cipher* wie folgt konstruiert:
Hinweis: cipher ist hier identisch zu (N903)c und (N944)c definiert.

vii. Setze *keyDO* = ´7F49 – L_{7F49} – (86 – L_{86} – PO_A)´.

viii. Setze *cipherDO* = ´86 – L_{86} – 02 || *C*´.

ix. Setze *macDO* = ´8E – L_{8E} – *T*´.

x. Setze *cipher* = ´A6 – L_{A6} – (*keyDO* || *cipherDO* || *macDO*).

(N919) Als Datenfeld der Antwortnachricht MUSS *cipher* verwendet werden.

(N920) Als Trailer MUSS NoError gewählt werden.

(N921) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 138 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 138 MUSS eine höhere Priorität als NoError haben.

15.8.4 PSO Hash

(N922) Das COS KANN dieses Kommando gemäß [7816–8] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–8] ablehnen.

15.8.5 PSO Transcipher

Das Kommando PSO Transcipher kombiniert die Kommandos PSO Decipher und PSO Encipher, ohne das die Klartextdaten die Karte verlassen. Es werden also Daten mittels eines privaten Schlüssels entschlüsselt und sofort wieder mittels eines öffentlichen Schlüssels verschlüsselt. Der private Schlüssel und der zu verwendende Algorithmus wird vor der Entschlüsselungsoperation ausgewählt. Dies geschieht vor dem Senden dieses PSO Transcipher Kommandos durch ein MSE Set Kommando, siehe 15.9.6.9. Die umzuschlüsselnden Daten und der öffentliche Schlüssel des Empfängers sind als Parameter in der Kommandonachricht enthalten.

15.8.5.1 Use Case Umschlüsseln von Daten mittels RSA Schlüssel, G1 (normativ)

Diese Variante gilt für folgende Algorithmen (siehe (N179)): rsaDecipherPKCS1_V1_5, rsaDecipherOaep.

In dieser Variante enthält die APDU des PSO Transcipher Kommandos vier Parameter:

(N923) Der Parameter *C* enthält die umzuschlüsselnden Daten. Der Parameter *C* ist ein Oktettstring mit beliebigem Inhalt.

(N924) Der Parameter *PuK* enthält den öffentlichen Schlüssel des Empfängers gemäß Kapitel 9.2.4.1. Der Parameter *PuK* ist ein Oktettstring, dessen Inhalt so gewählt werden MUSS, dass bei der Dekodierung kein Fehler auftritt, siehe (N944)b.iii.

(N925) Der Parameter *algID_enc* enthält den Algorithmus, der zur Verschlüsselung eingesetzt werden soll. Es MUSS ein Wert aus der Menge {rsaEncipherPKCS1_V1_5, rsaEncipherOaep} verwendet werden.

(N926) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N927) Die Parameter *C*, *PuK* und *algID_enc* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N941), (N942) und (N944) spezifiziert.

(N928) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 139 verwendet werden.

Tabelle 139: PSO Transcipher, Umschlüsseln von Daten mittels RSA

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´86´	Beschreibung der Antwortdaten, hier Chiffrat
P2	´B8´	Beschreibung der Kommandodaten, hier CRT für Verschlüsselung
Data	´XX…YY´	cipherIN
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.8.5.2 Use Case Umschlüsseln von Daten mittels ELC, G2 (informativ)

Diese Variante gilt für folgenden Algorithmus (siehe (N179)): elcSharedSecretCalculation.

In dieser Variante enthält die APDU des PSO Transcipher Kommandos sechs Parameter:

(N929) Der Parameter PO_A enthält einen Punkt auf einer elliptischen Kurve. Dieser Punkt wurde vom Sender der Nachricht gewählt. Der Parameter PO_A ist ein Oktettstring, dessen Inhalt so gewählt werden SOLL, dass bei der Dekodierung kein Fehler auftritt, siehe (N48)a.

(N930) Der Parameter *C* enthält die umzuschlüsselnden Daten. Der Parameter *C* ist ein Oktettstring mit beliebigem Inhalt.

(N931) Der Parameter *T* enthält einen MAC, der die Integrität von *C* schützt. Der Parameter *T* ist ein Oktettstring, dessen Länge und Inhalt so gewählt werden SOLL, dass bei der MAC Prüfung kein Fehler auftritt.

(N932) Der Parameter PO_B enthält einen Punkt auf einer elliptischen Kurve. Dieser Punkt repräsentiert den öffentlichen Schlüssel des Empfängers. Der Parameter PO_B ist ein Oktettstring, dessen Inhalt so gewählt werden SOLL, dass bei der Dekodierung kein Fehler auftritt, siehe (N48)a.

(N933) Der Parameter *algID_enc* enthält den Algorithmus, der zur Verschlüsselung eingesetzt werden soll. Es MUSS ein Wert aus der Menge { elcSharedSecretCalculation} verwendet werden.

(N934) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N935) Die Parameter PO_A, PO_B, *C*, *T* und *algID_enc* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N941), (N942)c und (N944)c spezifiziert.

(N936) Es MUSS eine Case 4 Kommando APDU gemäß Kapitel 12.7.4 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 4 Kommando APDU MÜSSEN die Angaben aus Tabelle 140 verwendet werden.

Tabelle 140: PSO Transcipher, Umschlüsseln von Daten mittels ELC

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´86´	Beschreibung der Antwortdaten, hier Chiffrat
P2	´86´	Beschreibung der Kommandodaten, hier ebenfalls Chiffrat
Data	´XX…YY´	cipherIN
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

Hinweis (84): Interessant wird es, wenn bei Transcipher RSA und ELC Schlüssel gemischt auftreten. das ist was für die Zukunft.

Hinweis (85): Möglicherweise werden P1 und P2 in einer späteren Dokumentenversion geändert.

15.8.5.3 Antwort der Karte auf Umschlüsseln von Daten

Tabelle 141: PSO Transcipher Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	cipherOUT	Chiffrat
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Umschlüsselungsoperation

Tabelle 142: PSO Transcipher Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoKeyReference	kein Schlüssel zur Entschlüsselung ausgewählt
´6A 88´	KeyNotFound	Schlüssel für Entschlüsselung nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´6A 80´	WrongCiphertext	Fehler beim Entschlüsseln des Kryptogramms

Hinweis (86): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N937) Ein COS KANN zusätzliche Trailer verwenden.

15.8.5.4 Kommandoabarbeitung innerhalb der Karte

(N938) Das COS MUSS die PSO Transcipher Variante aus 15.8.5.1 unterstützen.
Das COS KANN weitere PSO Transcipher Varianten unterstützen.
Das COS KANN weitere PSO Transcipher Varianten ablehnen.

(N939) Wenn channelContext.keyReferenceList.dataDecipher

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist, dann wird *affectedObject* gleich dem Schlüssel gesetzt, der durch SearchSecretKey(
currentFolder,
channelContext.keyReferenceList.dataDecipher.keyReference,
channelContext.keyReferenceList.dataDecipher.algID
) bezeichnet wird. Gemäß Kapitel 10.2.3 und (N1043) KANN es vorkommen,

dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler

i. keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

ii. notSupported meldet, genau dann MUSS das Kommando mit dem Trailer UnsupportedFunction terminieren.

(N940) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N941) Die Kommandodaten werden wie folgt aufgeteilt:

a. cipherIN = cipher || plainDO.

b. *cipher* MUSS ein DO mit Tag = ´A6´ sein.

c. *plainDO* MUSS ein DO mit Tag = ´A0´ sein.

(N942) Der Klartext wird wie folgt berechnet:
Wenn *channelContext.keyReferenceList.dataDecipher.algID* den Wert

a. rsaDecipherPKCS1_V1_5 besitzt, dann gilt:

i. *cipher* = ´A6 – L_{A6} – ($cipherDO_{in}$)´.

ii. $cipherDO_{in}$ = ´86 – L_{86} – 00 || C_{in}´.

iii. *M* = RSAES_PKCS1_V1_5_DECRYPT(
affectedObject.privateRsaKey,
C_{in}
).
Wenn diese Funktion mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren.

b. rsaDecipherOaep besitzt, dann gilt:

i. *cipher* = ´A6 – L_{A6} – ($cipherDO_{in}$)´.

ii. $cipherDO_{in}$ = ´86 – L_{86} – 00 || C_{in}´.

iii. *M* = RSAES_OAEP_DECRYPT(*affectedObject.privateRsaKey*, C_{in}).
Wenn diese Funktion mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren.

c. elcSharedSecretCalculation besitzt, dann gilt:

i. *cipher* = ´A6 – L_{A6} – ($keyDO_A$ || $cipherDO_{in}$ || $macDO_{in}$)´.

ii. $keyDO_A$ = ´7F49 – L_{7F49} – (86 – L_{86} – PO_A)´.

iii. $cipherDO_{in}$ = ´86 – L_{86} – 02 || C_{in}´.

iv. $macDO_{in}$ = ´8E – L_{8E} – T_{in}´.

v. Hinweis: *cipher* ist hier identisch zu (N903)c und (N918)a definiert.

vi. *M* = ELC_DEC(PO_A, *affectedObject.privateElcKey*, C_{in}, T_{in}).
Wenn diese Funktion mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren.

(N943) Suche in *plainDO* nach einem DO mit Tag = ´80´ dieses DO MUSS die Länge eins haben. Es gilt *algID_enc* = Wertfeld des DO mit Tag = ´80´.

(N944) Das Chiffrat *cipherOUT* wird wie folgt aus *M* berechnet: Wenn *algID_enc* den Wert

a. rsaEncipherPKCS1_V1_5 besitzt, dann gilt:
   i. *plainDO* = ´A0 – L_{A0} – (*algDO* || *keyDO*).
   ii. *algDO* = ´80 01 || *algID_enc*´.
   iii. *keyDO* = ´7F49 – L_{7F49} – [(81 – L_{81} – *PuK.n*) || (82 – L_{82} – *PuK.e*)]´.
   iv. *cipherOUT* = ´00´ || RSAES_PKCS1_V1_5_ENCRYPT(*PuK*, *M*).

b. rsaEncipherOaep besitzt, dann gilt:
   i. *plainDO* = ´A0 – L_{A0} – (*algDO* || *keyDO*).
   ii. *algDO* = ´80 01 || *algID_enc*´.
   iii. *keyDO* = ´7F49 – L_{7F49} – [(81 – L_{81} – *PuK.n*) || (82 – L_{82} – *PuK.e*)]´.
   iv. *cipherOUT* = ´00´ || RSAES_OAEP_ENCRYPT(*PuK*, *M*).

c. elcSharedSecretCalculation besitzt, dann gilt:
   i. *plainDO* = ´A0 – L_{A0} – (*algDO* || $keyDO_B$).
   ii. *algDO* = ´80 01 || *algID_enc*´.
   iii. $keyDO_B$ = ´7F49 – L_{7F49} – (86 – L_{86} – PO_B)´.
   iv. (PO_{out}, C_{out}, T_{out}) = ELC_ENC(
      M,
      PO_B,
      affectedObject.privateElcKey.domainParameter
)
   Wenn diese Funktion mit dem Fehler „*ERROR*“ terminiert, genau dann MUSS das Kommando mit dem Trailer WrongCiphertext terminieren. Andernfalls wird ein DER–TLV kodierter Oktettstring *cipherOUT* wie folgt konstruiert:
   Hinweis: cipherOUT ist hier identisch zu (N918)a definiert.
   v. Setze $keyDO_{out}$ = ´7F49 – L_{7F49} – (86 – L_{86} – PO_{out})´.
   vi. Setze $cipherDO_{out}$ = ´86 – L_{86} – 02 || C_{out}´.
   vii. Setze $macDO_{out}$ = ´8E – L_{8E} – T_{out}´.
   viii. cipherOUT = ´A6 – L_{A6} – ($keyDO_{out}$ || $cipherDO_{out}$ || $macDO_{out}$).

(N945) Als Datenfeld der Antwortnachricht MUSS *cipherOUT* verwendet werden.

(N946) Als Trailer MUSS NoError gewählt werden.

(N947) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 142 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 142 MUSS eine höhere Priorität als NoError haben.

15.8.6 PSO Verify Certificate

Das Kommando PSO Verify Certificate überprüft die Signatur eines Zertifikates mittels eines öffentlichen Schlüssels. Der öffentliche Schlüssel wird vor der Signieroperation ausgewählt. Dies geschieht vor dem Senden dieses PSO Verify Certificate Kommandos durch ein MSE Set Kommando, siehe Kapitel 15.9.6.8. Das Zertifikat ist als Parameter in der Kommandonachricht enthalten. Falls die Signatur des Zertifikates als gültig betrachtet wird, dann werden gewisse Attribute des im Zertifikat enthaltenen Schlüsselobjektes zur späteren Verwendung gespeichert.

15.8.6.1 Use Case Import RSA Schlüssels mittels Zertifikat, G1 (normativ)

In dieser Variante enthält die APDU des PSO Verify Certificate Kommandos zwei Parameter:

(N948) Der Parameter *certificateContent* enthält Attributsinformationen für das Schlüsselobjekt.

(N949) Der Parameter *signature* enthält die von einer CA erstellte Signatur über die Attributsinformationen.

(N950) Die Parameter *certificateContent* und *signature* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N959)a spezifiziert.

(N951) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 143 verwendet werden.

Tabelle 143: PSO Verify Certificate für RSA Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´00´	keine Antwortdaten
P2	´AE´	Kommandodaten mit Template, dessen Wertfelder zertifiziert sind
Data	´XX…YY´	certificate

15.8.6.2 Use Case Import ELC Schlüssels mittels Zertifikat, G2 (informativ)

In dieser Variante enthält die APDU des PSO Verify Certificate Kommandos zwei Parameter:

(N952) Der Parameter *certificateContent* enthält alle Attribute für das Schlüsselobjekt.

(N953) Der Parameter *signature* enthält die von einer CA erstellte Signatur über die Attribute.

(N954) Die Parameter *certificateContent* und *signature* MÜSSEN im Datenfeld der Kommandonachricht enthalten sein. Die Kodierung wird in (N959)b spezifiziert.

(N955) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser

Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 144 verwendet werden.

Tabelle 144: PSO Verify Certificate für ELC Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´2A´	Instruction Byte gemäß [7816–4]
P1	´00´	keine Antwortdaten
P2	´BE´	Kommandodaten mit zertifiziertem Template
Data	´XX…YY´	certificate

15.8.6.3 Antwort der Karte auf Vergleich eines Benutzergeheimnisses

Tabelle 145: PSO Verify Certificate Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Zertifikatsprüfung
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 146: PSO Verify Certificate Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´69 85´	NoKeyReference	kein Signaturprüfschlüssel ausgewählt
´6A 88´	KeyNotFound	Signaturprüfschlüssel nicht gefunden
´6A 80´	VerificationError	Prüfung des Zertifikates fehlgeschlagen
´6A 88´	InconsistentKeyReference	Signaturprüfschlüssel hat eine andere Referenz als CAR des Zertifikates
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich

Hinweis (87): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N956) Ein COS KANN zusätzliche Trailer verwenden.

15.8.6.4 Kommandoabarbeitung innerhalb der Karte

(N957) Das COS MUSS die PSO Verify Certificate Variante aus 15.8.6.1 unterstützen.

Das COS KANN weitere PSO Verify Certificate Varianten unterstützen.
Das COS KANN weitere PSO Verify Certificate Varianten ablehnen.

(N958) Wenn channelContext.keyReferenceList.verifyCertificate

a. leer ist, genau dann MUSS das Kommando mit dem Trailer NoKeyReference terminieren.

b. nicht leer ist, dann wird *affectedObject* = SearchPublicKey(
channelContext.keyReferenceList.verifyCertificate.keyReference,
verifyCertificate
currentFolder
) gesetzt. Gemäß Kapitel 10.2.4 und gemäß (N1043) KANN es vorkommen,

dass die Schlüsselsuche nicht erfolgreich ist. Genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren.

(N959) Wenn affectedObject vom Typ

a. *publicRsaKey* ist, dann gilt:

i. certificate = ´5F37–L_{5F37}–signature || 5F38–L_{5F38}–certificateContent´.

ii. (*out*, *M*) = RSA_ISO9796_2_DS1_VERIFY(
affectedObject.publicRsaKey,
signature,
certificateContent
)
Falls diese Operation mit einem Fehler abbricht, oder *out* gleich *False* ist, dann MUSS das Kommando mit dem Trailer VerificationError terminieren.

iii. Die Schlüsselattribute werden gemäß Kapitel 8.1.2 aus der Nachricht *M* extrahiert und ein öffentliches Schlüsselobjekt *pukObj* gebildet.

iv. Falls CPI gleich ´21´ ist, dann MUSS *pukObj* ein öffentliches Signaturprüfobjekt sein (siehe Kapitel 9.5.3.1), dessen Attribute mit den Definitionen aus Kapitel 8.1.2.1 wie folgt zu setzen sind:

1. *pukObj.oid* = OID.
2. pukObj.publicKey.n = Modulus.
3. *pukObj.publicKey.e* = öffentlicherExponent.
4. pukObj.keyIdentifier = CHR.

v. Falls CPI gleich ´22´ ist, dann MUSS *pukObj* ein öffentliches Authentisierungsobjekt sein (siehe Kapitel 9.5.3.2), dessen Attribute mit den Definitionen aus Kapitel 8.1.2.2 wie folgt zu setzen sind:

1. *pukObj.CHA* = CHA.
2. *pukObj.oid* = OID.
3. pukObj.publicKey.n = Modulus.
4. *pukObj.publicKey.e* = öffentlicherExponent.
5. pukObj.keyIdentifier = CHR.

vi. Falls *affectedObject.keyIdentifier* ungleich CAR ist, genau dann MUSS das Kommando mit dem Trailer InconsistentKeyReference terminieren.

vii. Das Objekt *pukObj* SOLL in der Liste *persistentPublicKeyList* des Objektsystems gespeichert werden.

viii. Das Objekt *pukObj* KANN in der Liste *publickeyList* des Objektsystems gespeichert werden.

ix. Das Objekt *pukObj* MUSS in der jeweiligen Liste zusammen mit einer Referenz auf *currentFolder* gespeichert werden.

b. *publicElcKey* ist, dann … (für Generation 2 vervollständigen)

(N960) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N961) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N962) Andernfalls MUSS als Trailer NoError gewählt werden.

(N963) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 146 ist herstellerspezifisch.
b. Jeder Trailer in Tabelle 146 MUSS eine höhere Priorität als UpdateRetryWarning haben.
c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.9 Verschiedenes

15.9.1 ENVELOPE

(N964) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.9.2 GENERATE ASYMMETRIC KEY PAIR

Das Kommando GENERATE ASYMMETRIC KEY PAIR (GAKP) dient dem Erzeugen von asymmetrischen Schlüsselpaaren und dem Auslesen eines dabei erzeugten öffentlichen Schlüssels. Das betroffene Schlüsselobjekt wird zuvor ausgewählt. Dies geschieht durch ein MANAGE SECURITY ENVIRONMENT Kommando (siehe Kapitel 15.9.6.7).

15.9.2.1 Use Case Schlüsselgenerierung ohne Ausgabe

In dieser Variante enthält die APDU des GAKP Kommandos einen Parameter:

(N965) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´84´ gewählt werden.

(N966) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 147 verwendet werden.

Tabelle 147: GAKP, Schlüsselgenerierung ohne Ausgabe

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´46´	Instruction Byte gemäß [7816–4]
P1	´84´	*operationMode* = Schlüsselgenerierung
P2	´00´	–

15.9.2.2 Use Case Auslesen eines zuvor erzeugten öffentlichen Schlüssels

In dieser Variante enthält die APDU des GAKP Kommandos zwei Parameter:

(N967) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´81´ gewählt werden.

(N968) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N969) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 148 verwendet werden.

Tabelle 148: GAKP, Auslesen öffentlicher Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´46´	Instruction Byte gemäß [7816–4]
P1	´81´	*operationMode* = Auslesen eines zuvor erzeugten Schlüssels
P2	´00´	–
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.9.2.3 Use Case Schlüsselgenerierung mit Ausgabe

In dieser Variante enthält die APDU des GAKP Kommandos zwei Parameter:

(N970) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´80´ gewählt werden.

(N971) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS aus dem in Kapitel 12.5.6 definierten Bereich gewählt werden.

(N972) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 149 verwendet werden.

Tabelle 149: GAKP, Schlüsselgenerierung mit Ausgabe des öffentlichen Schlüssels

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´46´	Instruction Byte gemäß [7816–4]
P1	´80´	*operationMode* = Schlüsselgenerierung und Ausgabe des Schlüssels
P2	´00´	–
Ne	length	Anzahl der erwarteten Oktette in den Antwortdaten

15.9.2.4 Antwort der Karte auf Schlüsselgenerierung

Tabelle 150: GAKP Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	publicKeyDO	abwesend, oder zuvor erzeugter öffentlicher Schlüssel
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Operation
´63 Cx´	UpdateRetryWarning	wie NoError, aber Schreibschwierigkeiten

Tabelle 151: GAKP Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	KeyNotFound	referenziertes Schlüsselobjekt wurde nicht gefunden
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´64 00´	KeyAlreadyPresent	Schlüssel wurde bereits erzeugt
´65 81´	MemoryFailure	Schreibvorgang nicht erfolgreich
´64 00´	KeyInvalid	auszulesende Schlüsseldaten fehlen

Kom 3002

Hinweis (88): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N973) Ein COS KANN zusätzliche Trailer verwenden.

15.9.2.5 Kommandoabarbeitung innerhalb der Karte

(N974) Das COS MUSS die GAKP Varianten aus 15.9.2.1, 15.9.2.2 und 15.9.2.3 unterstützen.
Das COS KANN weitere GAKP Varianten unterstützen.
Das COS KANN weitere GAKP Varianten ablehnen.

(N975) Falls im currentFolder das Attribute keyReferenceList im Element signatureCreation

a. leer ist, dann KANN das COS die Bearbeitung dieses Kommandos
   i. entweder mit einem beliebigen Trailer ablehnen,
   ii. oder versuchen mit Schlüsseln zu arbeiten, welche kein privates Signierobjekt darstellen.
b. nicht leer ist, dann wird *affectedObject* = SearchSecretKey(
   currentFolder,
   channelContext.keyReferenceList.signatureCreation.keyReference,
   Wildcard
) gesetzt. Gemäß Kapitel 10.2.3 und gemäß (N1043) KANN es vorkommen, dass die Schlüsselsuche nicht erfolgreich ist. Wenn die Schlüsselsuche mit den Fehler keyNotFound meldet, genau dann MUSS das Kommando mit dem Trailer KeyNotFound terminieren. Der Fehler notSupported ist wegen der Wildcard Suche nicht möglich.

(N976) Wenn AccessRuleEvaluation(*affectedObject*, *CLA*, *INS*, *P1*, *P2*) den Wert False zurückliefert, genau dann MUSS das Kommando mit dem Trailer SecurityStatusNotSatisfied terminieren.

(N977) Wenn *operationMode* Element der Menge {´80´, ´84´} ist und das Attribute *keyAvailable* den Wert

Kom 3002

a. True hat, genau dann MUSS das Kommando mit dem Trailer KeyAlreadyPresent terminieren.
b. False hat,
   i. dann MUSS ein Schlüsselpaar (*PrK*, *PuK*) erzeugt werden, dessen Eigenschaften zu den Attributen von *affectedObject* passen.
   ii. Anschließend MUSS das Attribute *keyAvailable* auf den Wert True geändert werden. Die Änderung von *keyAvailable* MUSS mit Transaktionsschutz ausgeführt werden.

(N978) Wenn *operationMode* Element der Menge {´80´, ´81´} ist und das Attribute *keyAvailable* den Wert False hat, genau dann MUSS das Kommando mit dem Trailer KeyInvalid terminieren.

Kom 3002

(N979) Wenn COS intern festgestellt wird, dass ein Schreibvorgang nicht beim ersten Versuch erfolgreich verlief, genau dann KANN das COS als Trailer UpdateRetryWarning wählen.

(N980) Wenn ein Schreibvorgang nicht erfolgreich verlief, genau dann MUSS als Trailer MemoryFailure verwendet werden.

(N981) Andernfalls MUSS als Trailer NoError gewählt werden.

(N982) Das DER–TLV kodierte Datenobjekt *publicKeyDO* MUSS wie folgt berechnet werden: Falls *PuK* ein

a. RSA Schlüssel ist, dann gilt:

   i. Setze L_{82} größer gleich OctetLength(*PuK.e*), aber kleiner als 128.

   ii. Setze *n* = I2OS(*PuK.n*, OctetLength(*PuK.n*)).

   iii. Setze *e* = I2OS(*PuK.e*, L_{82}).

   iv. *publicKeyDO* = ´7F49 – L_{7F49} – (81 – L_{81} – *n* || 82 – L_{82} – *e*)´.

b. ELC Schlüssel ist, dann gilt ab Generation 2
publicKeyDO = ´7F49 – L_{7F49} – (06 – L_{06} – *OID* || 86 – L_{86} – *P*)´.

(N983) Für das Datenfeld der Antwortnachricht gilt:

a. Wenn *operationMode* einen Wert aus der Menge {´80´, ´81} hat, dann enthält das Datenfeld der Antwortnachricht *publicKeyDO*.

b. Andernfalls fehlt das Datenfeld der Antwortnachricht.

(N984) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 151 ist herstellerspezifisch.

b. Jeder Trailer in Tabelle 151 MUSS eine höhere Priorität als UpdateRetryWarning haben.

c. UpdateRetryWarning MUSS eine höhere Priorität als NoError haben.

15.9.3 GET CHALLENGE

Das Kommando GET CHALLENGE erzeugt eine Zufallszahl. Diese steht kartenintern mindestens bei der Ausführung des nächsten Kommandos zur Verfügung. Typischerweise beinhaltet dieses nächste Kommando die Authentisierung einer externen Komponente.

15.9.3.1 Use Case Erzeugen Zufallszahl für DES oder RSA Authentisierung

In dieser Variante enthält die APDU des GET CHALLENGE Kommandos einen Parameter:

(N985) Der Parameter *length* bestimmt die Länge der erwarteten Antwortdaten. Der Wert von *length* MUSS gleich 8 sein.

(N986) Es MUSS eine Case 2 Kommando APDU gemäß Kapitel 12.7.2.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion

dieser Case 2 Kommando APDU MÜSSEN die Angaben aus Tabelle 152 verwendet werden.

Tabelle 152: GET CHALLENGE

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´84´	Instruction Byte gemäß [7816–4]
P1	´00´	–
P2	´00´	–
Ne	´08´	Anzahl der erwarteten Oktette in den Antwortdaten, hier acht

15.9.3.2 Use Case Erzeugen Zufallszahl für AES oder ELC Authentisierung

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.
Generation 2 mit Ne = 16 für AES und ELC

15.9.3.3 Antwort der Karte auf Erzeugen einer Zufallszahl

Tabelle 153: GET CHALLENGE Antwort APDU im Erfolgsfall

Daten	Inhalt	Beschreibung
´XX…YY´	RND.ICC	Zufallszahl
Trailer	**Inhalt**	**Beschreibung**
´90 00´	NoError	erfolgreiche Erzeugung einer Zufallszahl

Tabelle 154: GET CHALLENGE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
–	–	derzeit keine Fehlerfälle spezifiziert

Hinweis (89): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N987) Ein COS KANN zusätzliche Trailer verwenden.

15.9.3.4 Kommandoabarbeitung innerhalb der Karte

(N988) Das COS MUSS die GET CHALLENGE Variante aus 15.9.3.1 unterstützen.
Das COS KANN weitere GET CHALLENGE Varianten unterstützen.
Das COS KANN weitere GET CHALLENGE Varianten ablehnen.

(N989) Die Zugriffsbedingung für alle GET CHALLENGE Varianten MUSS ALWAYS sein.

(N990) Setze *RND.ICC* = RAND(Ne)

(N991) Als Trailer MUSS NoError gewählt werden.

(N992) Das Datenfeld der Antwortnachricht enthält *RND.ICC*.

(N993) *RND.ICC* MUSS zur Verwendung in nachfolgenden Kommandos gespeichert werden (siehe (N299)b).
RND.ICC MUSS im nachfolgenden Kommando verfügbar sein.
RND.ICC KANN im übernächsten Kommando verfügbar sein.

15.9.4 GET RESPONSE

(N994) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.9.5 MANAGE CHANNEL

(N995) Das COS KANN dieses Kommando gemäß [7816–4] unterstützen.
Das COS KANN dieses Kommando gemäß [7816–4] ablehnen.

15.9.6 MANGE SECURITY ENVIRONMENT

Das Kommando MANAGE SECURITY ENVIRONMENT (MSE) verändert im *currentFolder* die Attribute *seIdentifier* und im *channelContext* Elemente von *keyReferenceList*. Welche Aktion durchzuführen ist, welches Attribut oder Listenelement betroffen ist und auf welchen Wert sie zu ändern sind wird durch Parameter bestimmt, die in der Kommandonachricht enthalten sind.

(N996) Falls ein symmetrischer oder ein privater Schlüssel referenziert wird, dann besteht der Parameter *keyReference* aus den zwei Teilen *location* und *identifier*. *location* zeigt an, ob ein globaler oder DF–spezifischer Schlüssel von der Aktion betroffen ist. Als Wert für *location* MUSS ein Element der Menge {´00´, ´80´} verwendet werden. Dabei gilt:

a. Der Wert *location* = ´00´ MUSS verwendet werden, wenn ein globaler Schlüssel betroffen ist.

b. Der Wert *location* = ´80´ MUSS verwendet werden, wenn ein DF–spezifischer Schlüssel betroffen ist.

c. Der Parameter *identifier* bestimmt das betroffene Schlüsselobjekt. Der Wert von *identifier* MUSS konform zu (N164) bzw. (N171) gewählt werden.

d. Der Parameter *keyReference* MUSS in einem Oktett mit folgendem Wert kodiert werden: *keyReference* = *location* + *identifier*.

15.9.6.1 Use Case Ändern des SE–Identifiers

In dieser Variante enthält die APDU des MSE Kommandos zwei Parameter:

(N997) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´F3´ gewählt werden.

(N998) Der Parameter *seNo* MUSS gemäß (N79) gewählt werden.

(N999) Es MUSS eine Case 1 Kommando APDU gemäß Kapitel 12.7.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 1 Kommando APDU MÜSSEN die Angaben aus Tabelle 155 verwendet werden.

Tabelle 155: MSE, Restore Variante

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´F3´	*operationMode* = Auswahl eines SE–Identifiers
P2	´XX´	*seNo* = Wert des auszuwählenden SE–Identifiers

15.9.6.2 Use Case Schlüsselsauswahl zur internen, symmetrischen Authentisierung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1000) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´41´ gewählt werden.

(N1001) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´A4´ gewählt werden.

(N1002) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *internalAuthenticate*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1003) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *internalAuthenticate*. Wert und Kodierung MÜSSEN gemäß Tabelle 167 gewählt werden, wobei ein Wert aus der Menge {
desRoleAuthentication,
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1004) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 156 verwendet werden.

Tabelle 156: MSE, Selektion symmetrischer INTERNAL AUTHENTICATE Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´41´	*operationMode* = Setzen eines „internen“ Schlüssels
P2	´A4´	*crtTag* = betroffenes Listenelement ist *internalAuthenticate*
Data	´XX…YY´	´83 – L_{83} – *keyRef* || 80 01 || *algId* ´

Hinweis (90): L_{83} ist ein Oktett mit dem Wert I2OS(OctetLength(keyRef), 1).

15.9.6.3 Use Case Schlüsselsauswahl zur internen, asymmetrischen Authentisierung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1005) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´41´ gewählt werden.

(N1006) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´A4´ gewählt werden.

(N1007) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *internalAuthenticate*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1008) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *internalAuthenticate*. Wert und Kodierung MÜSSEN gemäß Tabelle 167 gewählt werden, wobei ein Wert aus der Menge {
rsaClientAuthentication,
rsaRoleAuthentication,
rsaSessionkey4SM
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1009) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 157 verwendet werden.

Tabelle 157: MSE, Selektion privater INTERNAL AUTHENTICATE Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´41´	*operationMode* = Setzen eines „internen“ Schlüssels
P2	´A4´	*crtTag* = betroffenes Listenelement ist *internalAuthenticate*
Data	´XX…YY´	´84 01 || *keyRef* || 80 01 || *algId* ´

15.9.6.4 Use Case Schlüsselsauswahl zur externen, symmetrischen Authentisierung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1010) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´81´ gewählt werden.

(N1011) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´A4´ gewählt werden.

(N1012) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1013) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß Tabelle 167 gewählt werden, wobei ein Wert aus der Menge {
desRoleCheck
} verwendet werden MUSS.

Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1014) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 158 verwendet werden.

Tabelle 158: MSE, Selektion symmetrischer EXTERNAL AUTHENTICATE Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´81´	*operationMode* = Setzen eines „externen" Schlüssels
P2	´A4´	*crtTag* = betroffenes Listenelement ist *externalAuthenticate*
Data	´XX…YY´	´83 – L_{83} – *keyRef* || 80 01 || *algId* ´

Hinweis (91): L_{83} ist ein Oktett mit dem Wert I2OS(OctetLength(keyRef), 1).

15.9.6.5 Use Case Schlüsselsauswahl zur externen, asymmetrischen Authentisierung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1015) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´81´ gewählt werden.

(N1016) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´A4´ gewählt werden.

(N1017) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß (N195) gewählt werden.

(N1018) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß Tabelle 167 gewählt werden, wobei ein Wert aus der Menge {
rsaRoleCheck,
rsaSessionkey4SM
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1019) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 159 verwendet werden.

Tabelle 159: MSE, Selektion öffentlicher EXTERNALAUTHENTICATE Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´81´	*operationMode* = Setzen eines öffentlichen Schlüssels
P2	´A4´	*crtTag* = betroffenes Listenelement ist *externalAuthenticate*
Data	´XX…YY´	´83 0C || *keyRef* || 80 01 || *algId* ´

15.9.6.6 Use Case Schlüsselsauswahl zur symmetrischen, gegenseitigen Authentisierung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1020) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´81´ gewählt werden.

(N1021) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´A4´ gewählt werden.

(N1022) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1023) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *externalAuthenticate*. Wert und Kodierung MÜSSEN gemäß Tabelle 167 gewählt werden, wobei ein Wert aus der Menge {
desSessionkey4SM
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1024) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 160 verwendet werden.

Tabelle 160: MSE, Selektion symmetrischer MUTUAL AUTHENTICATE Schlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´81´	*operationMode* = Setzen eines symmetrischen Schlüssels
P2	´A4´	*crtTag* = betroffenes Listenelement ist *externalAuthenticate*
Data	´XX…YY´	´83 – L_{83} – *keyRef* || 80 01 || *algId* ´

Hinweis (92): L_{83} ist ein Oktett mit dem Wert I2OS(OctetLength(keyRef), 1).

15.9.6.7 Use Case Schlüsselsauswahl für Signierschlüssel

Dieser Use Case wird verwendet um einen Signierschlüssel zu selektieren. Anschließend ist es möglich diesen Schlüssel zu erzeugen oder seinen öffentlichen Teil auszulesen (siehe Kapitel 15.9.2) oder mit diesem Schlüssel Signaturen zu erzeugen (siehe Kapitel 15.8.1). In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1025) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´41´ gewählt werden.

(N1026) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´B6´ gewählt werden.

(N1027) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *signatureCreation*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1028) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *signatureCreation*. Wert und Kodierung MÜSSEN gemäß Tabelle 169 gewählt werden, wobei ein Wert aus der Menge {
rsaClientAuthentication,
sign9796_2_DS2,
signPKCS1_V1_5,
signPSS
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1029) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 161 verwendet werden.

Tabelle 161: MSE, Selektion privater Signaturschlüssel

	Inhalt	**Beschreibung**
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´41´	*operationMode* = Setzen eines privaten Schlüssels
P2	´B6´	*crtTag* = betroffenes Listenelement ist *signatureCreation*
Data	´XX…YY´	´84 01 || *keyRef* || 80 01 || *algId* ´

15.9.6.8 Use Case Schlüsselsauswahl zum Prüfen von CV–Zertifikaten

In dieser Variante enthält die APDU des MSE Kommandos drei Parameter:

(N1030) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´81´ gewählt werden.

(N1031) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´B6´ gewählt werden.

(N1032) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *verifyCertificate*. Wert und Kodierung MÜSSEN gemäß (N191) gewählt werden.

(N1033) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter" in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 162 verwendet werden.

Tabelle 162: MSE, Selektion öffentlicher Zertifikatsprüfschlüssel

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´81´	*operationMode* = Setzen eines öffentlichen Schlüssels
P2	´B6´	*crtTag* = betroffenes Listenelement ist *verifyCertificate*
Data	´XX…YY´	´83 08 || *keyRef* ´

15.9.6.9 Use Case Schlüsselsauswahl zur Datenent- oder Datenumschlüsselung

In dieser Variante enthält die APDU des MSE Kommandos vier Parameter:

(N1034) Der Parameter *operationMode* bestimmt die durchzuführende Aktion. Für diesen Use Case MUSS *operationMode* = ´41´ gewählt werden.

(N1035) Der Parameter *crtTag* bestimmt das Listenelement in *keyReferenceList*, welches zu ändern ist. Für diesen Use Case MUSS gemäß *crtTag* = ´B8´ gewählt werden.

(N1036) Der Parameter *keyRef* enthält den neuen Wert für das Element *keyReference* im Listenelement *dataDecipher*. Wert und Kodierung MÜSSEN gemäß (N996) gewählt werden.

(N1037) Der Parameter *algId* enthält den neuen Wert für das Element *algorithmIdentifier* im Listenelement *dataDecipher*. Wert und Kodierung MÜSSEN gemäß Tabelle 168 gewählt werden, wobei ein Wert aus der Menge {
rsaDecipherPKCS1_V1_5,
rsaDecipherOaep
} verwendet werden MUSS.
Das COS KANN weitere Werte für *algId* akzeptieren.
Das COS KANN weitere Werte für *algId* ablehnen.

(N1038) Es MUSS eine Case 3 Kommando APDU gemäß Kapitel 12.7.3.1 über die Schnittstelle „Interpreter“ in Abbildung 1 geschickt werden. Für die Konstruktion dieser Case 3 Kommando APDU MÜSSEN die Angaben aus Tabelle 163 verwendet werden.

Tabelle 163: MSE, Schlüsselselektion zur Entschlüsselung

	Inhalt	Beschreibung
CLA	´00´	CLA Byte gemäß [7816–4]
INS	´22´	Instruction Byte gemäß [7816–4]
P1	´41´	*operationMode* = Setzen eines privaten Schlüssels
P2	´B8´	*crtTag* = betroffenes Listenelement ist *dataDecipher*
Data	´XX…YY´	´84 01 || *keyRef* || 80 01 || *algId* ´

15.9.6.10 Antwort der Karte auf Management des Security Environments

Tabelle 164: MSE Antwort APDU im Erfolgsfall

Trailer	Inhalt	Beschreibung
´90 00´	NoError	erfolgreiche Übernahme der Kommandodatenparameter

Tabelle 165: MSE Antwort APDU im Fehlerfall

Trailer	Inhalt	Beschreibung
´6A 88´	KeyNotFound	zu selektierendes Schlüsselobjekt wurde nicht gefunden
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht

Hinweis (93): Diese Tabelle enthält keine Fehler, die in den Komponenten I/O, ChannelSwitch und SecMes aus Abbildung 1 entdeckt wurden.

(N1039) Ein COS KANN zusätzliche Trailer verwenden.

15.9.6.11 Kommandoabarbeitung innerhalb der Karte

(N1040) Das COS MUSS die MSE Varianten aus 15.9.6.1, 15.9.6.2, 15.9.6.3, 15.9.6.4, 15.9.6.5, 15.9.6.6, 15.9.6.7, 15.9.6.8 und 15.9.6.9 unterstützen.
Das COS KANN weitere MSE Varianten unterstützen.
Das COS KANN weitere MSE Varianten ablehnen.

(N1041) Die Zugriffsbedingung für alle MSE Varianten MUSS ALWAYS sein.

(N1042) Wenn der Parameter *operationMode* in P1 den Wert ´F3´ besitzt, dann MUSS

a. in *currentFolder* das Attribut *seIdentifier* (siehe (N300)a) auf den Wert des Parameters *seNo* gesetzt werden.
b. jedes Listenelement in *keyReferenceList* (siehe (N299)c) auf den Wert gesetzt werden, der ein leeres Element kodiert.
c. jedes Element in *globalPasswordList* (siehe (N299)i) und in *dfSpecificPasswordList* (siehe (N299)j) mittels clearPasswordStatus(…) aus den genannten Listen entfernt werden, die currentFolder zugeordnet sind.
d. der Wert von *currentEF* unverändert bleiben.
e. jedes Element in *globalSecurityList* (siehe (N299)e) und in *dfSpecificSecurityList* (siehe (N299)f) mittels clearSecurityStatus(…) aus den genannten Listen entfernt werden, die *currentFolder* zugeordnet sind.
f. Attribute, die übergeordneter Ordner zugeordnet sind, DÜRFEN sich NICHT ändern.

(N1043) Wenn der Parameter *operationMode* in P1 einen Wert aus der Menge {´A4´, ´B6´, ´B8} besitzt, dann MUSS in *currentFolder* das durch die Parameter *operationMode* und *crtTag* gekennzeichnete Listenelement mit den Parametern aus dem Datenfeld der Kommandonachricht gefüllt werden.
ACHTUNG: Je nach COS Implementierung ist es möglich, dass die referenzierten Schlüsselobjekte mit passendem *algorithmIdentifier* bereits während der Ausführung dieses MSE Kommandos gesucht werden. In diesem Fall KANN dieses MSE Kommando mit einem Trailer KeyNotFound oder UnsupportedFunction terminieren. Es KANN sein, dass dieser Fehler dann beim Kommando, welches den Schlüssel nutzen will, nicht mehr vorkommt. Entweder dort oder hier MUSS der Fehler KeyNotFound oder UnsupportedFunction geworfen werden.

(N1044) Als Trailer MUSS NoError gewählt werden.

(N1045) Für die Priorität der Trailer gilt:

a. Die Priorität der Trailer in Tabelle 165 ist herstellerspezifisch.
b. Jeder Trailer in Tabelle 165 MUSS eine höhere Priorität als NoError haben.

16 Authentisierungsprotokolle (normativ)

Dieses Kapitel beschreibt, wie eine schlüsselbasierte Authentisierung zwischen einer externen Entität und dem COS einer eGK abläuft. Die externe Entität werde als Gegenstelle bezeichnet. Aus Symmetriegründen und der leichteren Referenzierbarkeit innerhalb dieses Dokumentes wird eine sprachliche Darstellung gewählt, welche die Gegenstelle als Chipkarte mit einem Funktionsumfang beschreibt, der dem Funktionsumfang dieses Dokumentes analog ist. Diese Darstellungsform beschränkt nicht die Allgemeingültigkeit, weil hier lediglich die Schnittstelle zwischen Gegenstelle und eGK betrachtet wird und für die Gegenstelle aus Sicht dieses Dokumentes auch eine beliebige andere Komponente mit entsprechender Funktionalität möglich ist.

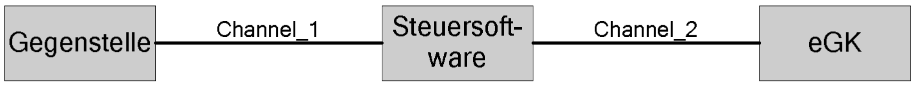

Abbildung 3: Kommunikationsmodell Gegenstelle, Steuersoftware und eGK

Das in diesem Kapitel verwendete Kommunikationsmodell ist wie folgt:

- Eine Komponente eGK besitzt die in diesem Dokument spezifizierten Schnittstelleneigenschaften.
- Eine Komponente Gegenstelle besitze zur eGK analoge Eigenschaften.
- Eine Komponente Steuersoftware besitze eine Ablauflogik. Zudem kommuniziert die Steuersoftware mit der Gegenstelle über den Kommunikationskanal Channel_1 und mit der eGK über den Kommunikationskanal Channel_2. Diese beiden Kanäle entsprechen der physikalischen Schnittstelle in Abbildung 1.

Jedem der hier behandelten Verfahren ist ein eigenes Unterkapitel gewidmet. Im Allgemeinen besteht die Authentisierung aus einer Sequenz von einem oder mehreren Kommandos, die über die Kommunikationskanäle gesendet werden. Für alle Authentisierungsprotokolle gilt:

(N1046) Falls die Sequenz eines Authentisierungsprotokolls aus mehr als einem Kommando Antwort Paar besteht (siehe Kapitel 12.1), so DARF diese Sequenz an der Schnittstelle „Channel_2" (siehe Abbildung 3) NICHT durch Kommandos unterbrochen werden, welche nicht zu dieser Sequenz gehören.

(N1047) Falls die Anforderung (N1046) von der externen Entität nicht eingehalten wird, mithin also die Sequenz durch ein unterbrechendes Kommando unterbrochen wird, dann KANN das COS der eGK

a. das unterbrechende Kommando akzeptieren.

b. das unterbrechende Kommando ablehnen.

c. ein Fortsetzen der unterbrochenen Sequenz akzeptieren.

d. ein Fortsetzen der unterbrochenen Sequenz ablehnen.

(N1048) Es MUSS möglich sein, dass jede der hier genannten Sequenzen ungeschützt (ohne Secure Messaging) übertragen wird. Falls auch die geschützte Übertragung der Sequenz zu unterstützen ist, so ist dies im entsprechenden Kapitel vermerkt. Wenn das erste Kommando dieser Sequenz

a. geschützt übertragen wird, dann MÜSSEN auch alle anderen Kommandos dieser Sequenz geschützt übertragen werden.

 i. ungeschützt übertragen wird, dann MÜSSEN auch alle anderen Kommandos dieser Sequenz ungeschützt übertragen werden.

(N1049) Falls die Anforderung (N1048) von der externen Entität nicht eingehalten wird, dann KANN das COS der eGK dies

a. akzeptieren.

b. ablehnen.

16.1 Externe Authentisierung

Dieses Kapitel behandelt die Authentisierung einer „Gegenstelle" gegenüber einer eGK.

16.1.1 Externe Authentisierung mittels symmetrischer Schlüssel (normativ)

Für den Fall, dass die Gegenstelle ihre Authentizität mittels eines symmetrischen Schlüssels nachweisen will und dabei keine Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 4):

(N1050) Das erste Kommando der Sequenz MUSS GET CHALLENGE gemäß Kapitel 15.9.3.1 sein und über Channel_2 geschickt werden.

(N1051) Das zweite Kommando über Channel_2 ist das letzte dieser Sequenz und MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich symRoleCheck sein.

(N1052) Es MUSS möglich sein, die Kommandos dieser Sequenz, welche über Channel_2 gesendet werden, geschützt (mit Secure Messaging) zu übertragen.

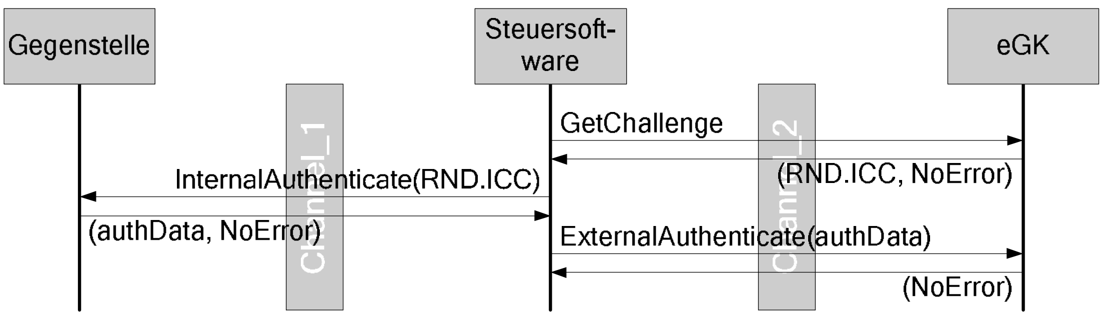

Abbildung 4: Sequenzdiagramm zur externen Authentisierung

16.1.2 RSA, asymmetrische Rollenauthentisierung (normativ)

Für den Fall, dass eine externe Entität ihre Authentizität mittels eines RSA Schlüsselpaares nachweisen will und dabei keine Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 4):

(N1053) Das erste Kommando der Sequenz MUSS GET CHALLENGE gemäß Kapitel 15.9.3.1 sein und über Channel_2 geschickt werden.

(N1054) Das zweite Kommando über Channel_2 ist das letzte dieser Sequenz und MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich asymRoleCheck sein.

(N1055) Es MUSS möglich sein, die Kommandos dieser Sequenz, welche über Channel_2 gesendet werden, geschützt (mit Secure Messaging) zu übertragen.

16.1.3 ELC, asymmetrische Berechtigungsnachweis (G2, informativ)

Für den Fall, dass eine externe Entität ihre Authentizität mittels eines ELC Schlüsselpaares nachweisen will und dabei keine Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 4):

(N1056) Das erste Kommando der Sequenz MUSS GET CHALLENGE gemäß Kapitel 15.9.3.2 sein und über Channel_2 geschickt werden.

(N1057) Das zweite Kommando über Channel_2 ist das letzte dieser Sequenz und MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich elcRoleCheck sein.

(N1058) Es MUSS möglich sein, die Kommandos dieser Sequenz, welche über Channel_2 gesendet werden, geschützt (mit Secure Messaging) zu übertragen.

16.2 Interne Authentisierung

Für den Fall, dass eine externe Entität die Authentizität der eGK mittels eines Schlüssels überprüfen will und dabei keine Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 5):

(N1059) Das erste Kommando über Channel_2 ist das letzte dieser Sequenz für Channel_2 und MUSS ein INTERNAL AUTHENTICATE gemäß Kapitel 15.7.4.1 mit beliebiger *algId* aus (N862) sein.

(N1060) Es MUSS möglich sein, die Kommandos dieser Sequenz, welche über Channel_2 gesendet werden, geschützt (mit Secure Messaging) zu übertragen.

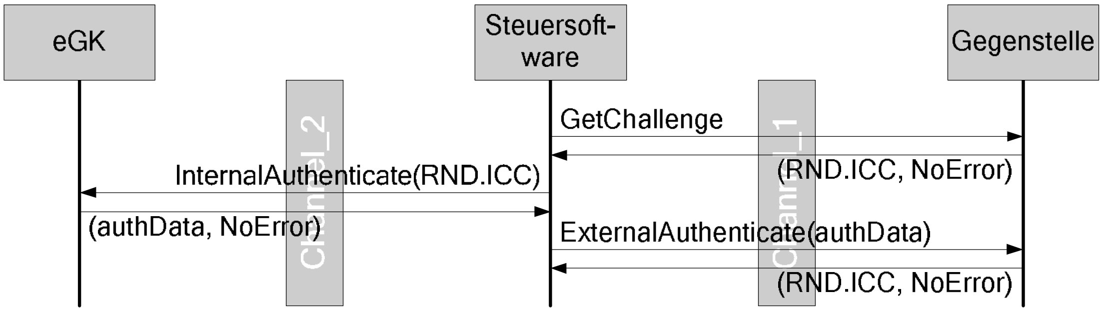

Abbildung 5: Sequenzdiagramm zur internen Authentisierung

16.3 Card–2–Card Authentisierung ohne Sessionkey Aushandlung

Die Card–2–Card Authentisierung ohne Sessionkey Aushandlung ist eine spezielle Variante der in den Kapiteln 16.1 und 16.2 behandelten Verfahren, wobei die Gegenstelle definitiv eine Chipkarte ist.

Falls eine einseitige Authentisierung (nur eine der beiden Komponenten Gegenstelle oder eGK authentisiert sich) beabsichtigt ist, wird je nach Richtung der Authentisierung entweder ein Algorithmus aus Kapiteln 16.1 oder aus Kapitel 16.2 gewählt.

Falls eine gegenseitige Authentisierung (beide Komponenten Gegenstelle und eGK authentisieren sich) beabsichtigt ist, wird sowohl ein Algorithmus aus Kapiteln 16.1 als auch ein Algorithmus aus Kapitel 16.2 gewählt. Typischerweise legen Zugriffsregeln der beteiligten Schlüssel fest, welche Komponente sich zuerst zu authentisieren hat.

16.4 Aushandlung von Sessionkey

Dieses Unterkapitel behandelt die gegenseitige Authentisierung zweier Entitäten, wobei gleichzeitig Sessionkeys ausgehandelt werden.

16.4.1 Symmetrische Aushandlung von Sessionkeys (normativ)

Für den Fall, dass eine gegenseitige Authentisierung mittels symmetrischer Schlüssel durchzuführen ist und dabei Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 6):

(N1061) Das erste Kommando der Sequenz MUSS GET CHALLENGE gemäß Kapitel 15.9.3.1 sein und über Channel_2 geschickt werden. Die dabei von der eGK erzeugte Zufallszahl wird mit RND.1 bezeichnet.

(N1062) Das zweite Kommando der Sequenz MUSS zur Gegenstelle gesendet werden und MUSS ein INTERNAL AUTHENTICATE gemäß Kapitel 15.7.4.1 mit *algId* gleich symSessionkey4TC sein. Dabei MUSS cmdData RND.1 enthalten. Die Antwortdaten der Gegenstelle werden mit rspData.2 bezeichnet.

(N1063) Das dritte Kommando der Sequenz MUSS zur eGK gesendet werden. Es MUSS ein MUTUAL AUTHENTICATE gemäß Kapitel 15.7.1.2 mit *algId* gleich symSessionkey4SM sein. Als Kommandodaten MÜSSEN rspData.2 verwendet werden. Die Antwortdaten der eGK werden mit rspData.3 bezeichnet.

(N1064) Das vierte Kommando ist das letzte dieser Sequenz und MUSS zur Gegenstelle gesendet werden. Es MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich symSessionkey4TC sein. Als Kommandodaten MÜSSEN rspData.3 verwendet werden.

(N1065) Die eGK KANN die geschützte Übertragung der Sequenz erlauben.
Die eGK KANN die geschützte Übertragung der Sequenz ablehnen.

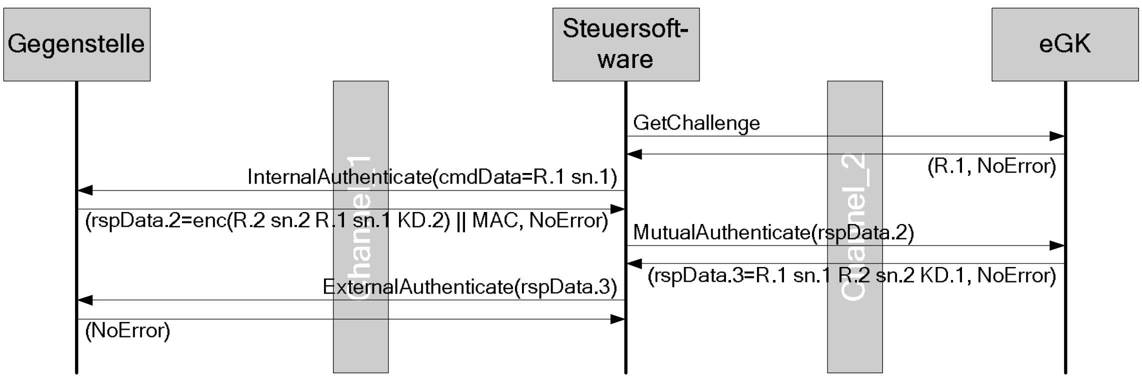

Abbildung 6: Sequenzdiagramm symmetrische Sessionkey Aushandlung

16.4.2 RSA Schlüssel (normativ)

Für den Fall, dass eine gegenseitige Authentisierung mittels asymmetrischer RSA Schlüssel durchzuführen ist und dabei Sessionkeys ausgehandelt werden, ist folgende Sequenz zu wählen (vergleiche Abbildung 7):

(N1066) Das erste Kommando MUSS zur Gegenstelle gesendet werden und MUSS GET CHALLENGE gemäß Kapitel 15.9.3.1 sein. Die dabei von der Gegenstelle erzeugten Antwortdaten werden mit rspData.1 bezeichnet.

(N1067) Das zweite Kommando MUSS zur eGK gesendet werden und MUSS ein INTERNAL AUTHENTICATE gemäß Kapitel 15.7.4.1 mit *algId* gleich rsaSessionkey4SM sein. Dabei MUSS cmdData.2 = rspData.1 || ICCSN8.User gelten mit ICCSN8.User gleich *iccsn8* (siehe (N199)c) aus der Gegenstelle. Wie die Instanz, welche die Kommando APDU sendet Kenntnis von ICCSN8.User erhält ist nicht Gegenstand dieses Dokumentes. Die korrespondierenden Antwortdaten der eGK werden mit rspData.2 bezeichnet.

(N1068) Das dritte Kommando MUSS zur Gegenstelle gesendet werden. Es MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich rsaSessionkey4TC sein. Als Kommandodaten MÜSSEN rspData.2 verwendet werden. Kom 3002

(N1069) Das vierte Kommando MUSS zur eGK gesendet werden und MUSS GET CHALLENGE gemäß Kapitel 15.9.3.1 sein. Die dabei von der eGK erzeugten Antwortdaten werden mit rspData.4 bezeichnet.

(N1070) Das fünfte Kommando MUSS zur Gegenstelle gesendet werden und MUSS ein INTERNAL AUTHENTICATE gemäß Kapitel 15.7.4.1 mit *algId* gleich rsaSessionkey4TC sein. Dabei MUSS cmdData.5 = rspData.4 || ICCSN8.COS gelten mit Kom 3002 ICCSN8.COS gleich *iccsn8* (siehe (N199)c) aus der eGK. Wie die Instanz, welche die Kommando APDU sendet Kenntnis von ICCSN8.COS erhält ist nicht Gegenstand dieses Dokumentes. Die korrespondierenden Antwortdaten von der Gegenstelle werden mit rspData.5 bezeichnet.

(N1071) Das sechste Kommando ist das letzte der Sequenz und MUSS zur eGK gesendet werden. Es MUSS ein EXTERNAL AUTHENTICATE gemäß Kapitel 15.7.1.1 mit *algId* gleich rsaSessionkey4SM sein. Als Kommandodaten MÜSSEN rspData.5 verwendet werden.

(N1072) Die eGK KANN die geschützte Übertragung der Sequenz erlauben.
Die eGK KANN die geschützte Übertragung der Sequenz ablehnen.

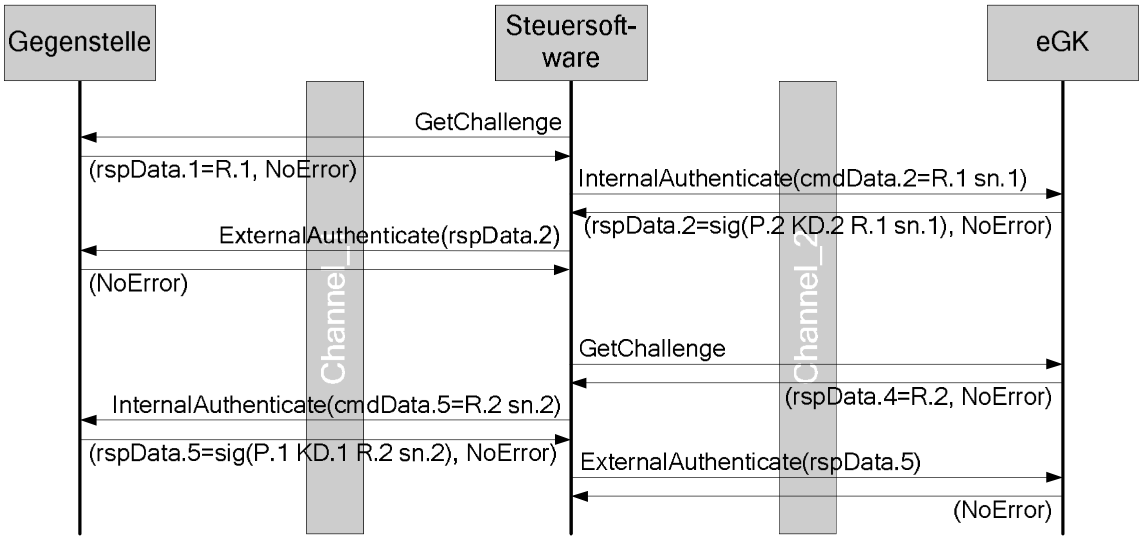

Abbildung 7: Sequenzdiagramm RSA Sessionkey Aushandlung

16.4.3 ELC Schlüssel (informativ)

Das Kapitel wird in einer späteren Version des Dokumentes ergänzt.

17 Verschiedenes (normativ)

17.1 Identifier

Hinweis (94): Es ist nicht leicht AlgIds für Authentisierungszwecke festzulegen, wenn man das gesamte Gesundheitswesen betrachtet und auch noch prEN14890 beachten will. Vor dem Hintergrund der derzeitigen Diskussion um Komfort- und Stapelsignatur werden diverse Verfahren diskutiert. die ggf. zu berücksichtigen sind. Hier mal der Versuch eines generischen Ansatzes.
Für einen privaten Authentisierungsschlüssel sehe ich derzeit vier Möglichkeiten (2 Bit) im Hinblick auf Sessionkeys (SK):
1) Authentisierung ohne SK Aushandlung
2) Authentisierung mit SK Aushandlung, wobei SK NICHT persistent gespeichert werden.
3) Authentisierung mit SK Aushandlung und Speichern der SK als IntroductionKeys.
4) Authentisierung mit SK Aushandlung und Speichern der SK als PairingKeys.
Unterschied IntroductionKeys zu Pairingkeys:
IntroductionKeys sind eine Art „performante Abkürzung“ zur asymmetrischen CV basierten Authentisierung, wie sie schon in der HPC Spec angedacht ist. Gedacht ist diese Abkürzung für beliebige Karten (vor allem HBA und SMC). Es ist unkritisch jeden beliebigen HBA jeder beliebigen SMC vorzustellen.
ParingKeys sind ähnlich zu IntroductionKeys, aber gedacht als Paarung zwischen zwei bestimmten Karten: Ein konkreter HBA mit einem konkreten Komfortsignatur–Token.
Verwendung der SK, zwei Möglichkeiten (1 Bit): Entweder für Secure Messaging, oder für PSO Kommandos.
Ableitung von SK, zwei Möglichkeiten (1 Bit): Nach alter oder neuer Spec.
SK Typ vier Möglichkeiten (2Bit): 2TDES, 3TDES, AES–128, AES–256
Bleiben bei einer 1 Byte AlgId noch 2 Bit übrig um den Authentisierungsmechanismus auszuwählen: Mit SHA–1 (wird hier nicht gebraucht, aber es gibt ja noch andere Projekte ;-), SHA–256, …

Hinweis (95): Vorschlag für die Kodierung von algorithmIdentifier zur Sessionkey Aushandlung basierend auf Hinweis (94):
a) AlgId wird in einem Oktett kodiert.
b) Bits b2 b1 kodieren Authentisierungsmechanismus
00 => Verfahren gemäß dieser Spec.
andere „out of scope“
c) Bit b4 b3 kodieren SK Typ
00 => 2TDES (wird hier nicht gebraucht, aber woanders schon)
01 => 3TDES
10 => AES–128
11 => AES–256
d) Bit b5 kodiert Ableitungsverfahren
0 => [14890–1]
1 => Kapitel 7.2
e) Bit b6 kodiert Verwendung der SK (in Zukunft soll SMC auch Remote–PIN können)
0 => für Secure Messaging
1 => für PSO Kommandos (bzw. ENVELOPE)
f) Bits b8 b7 kodieren Möglichkeiten
00 => keine SK Aushandlung, Kodierung der übrigen Bit ergibt sich mittels Hinweis (96):
01 => SK Aushandlung ohne persistente Speicherung der SK
10 => SK Aushandlung, Speicherung als IntroductionKeys
11 => SK Aushandlung, Speicherung als PairingKeys

Hinweis (96): Vorschlag zur Kodierung von algorithmIdentifier für Authentisierung ohne Sessionkey Aushandlung:
a) AlgId wird in einem Oktett kodiert.
b) Bits b8 b7 kodieren Sessionkey Aushandlung
00 => keine SK Aushandlung, andere Werte werden in Hinweis (95): behandelt.
c) Die übrigen Bits werden auf 0 gesetzt, weil derzeit jedem Schlüsseltyp (AES, DES, RSA, ELC) nur ein Verfahren zugeordnet ist.

Tabelle 166: Generische AlgorithmIdentifier für Authentisierungszwecke

Name	Oberbegriff für	
asymClientAuthentication		rsaClientAuthentication
asymRoleAuthentication	elcRoleAuthentication,	rsaRoleAuthentication
asymRoleCheck	elcRoleCheck	rsaRoleCheck
asymSessionkey4SM		rsaSessionkey4SM
symRoleAuthentication	aesRoleAuthentication	desRoleAuthentication
symRoleCheck	aesRoleCheck	desRoleCheck
symSessionkey4SM	aesSessionkey4SM	desSessionkey4SM
symSessionkey4TC	aesSessionkey4TC	desSessionkey4TC

Tabelle 167: Konkrete AlgorithmIdentifier für Authentisierungszwecke

Name	Kodierung	Verwendung
aesRoleAuthentication	$0000\ 0000_2$ = ´00´	symmetrisch, InternalAuth
aesRoleCheck	$0000\ 0000_2$ = ´00´	symmetrisch, ExternalAuth
aesSessionkey4SM	$0101\ 0100_2$ = ´54´	symmetrisch, MutualAuth Sessionkeys für Secure Messaging
aesSessionkey4TC	$0111\ 0100_2$ = ´74´	symmetrisch, ExternalAuth, InternalAuth, Sessionkeys für PSO Kommandos
desRoleAuthentication	$0000\ 0000_2$ = ´00´	symmetrisch, InternalAuth
desRoleCheck	$0000\ 0000_2$ = ´00´	symmetrisch, ExternalAuth
desSessionkey4SM	$0101\ 0100_2$ = ´54´	symmetrisch, MutualAuth Sessionkeys für Secure Messaging
desSessionkey4TC	$0111\ 0100_2$ = ´74´	symmetrisch, ExternalAuth, InternalAuth, Sessionkeys für PSO Kommandos
rsaClientAuthentication	$0000\ 0101_2$ = ´05´	InternalAuth, siehe [14890–2] Table 11-1
rsaRoleAuthentication	$0000\ 0000_2$ = ´00´	asymmetrisch InternalAuth
rsaRoleCheck	$0000\ 0000_2$ = ´00´	asymmetrisch ExternalAuth
rsaSessionkey4Intro	$1001\ 0100_2$ = ´94´	asymmetrisch, ExternalAuth und InternalAuth, Sessionkeys werden persistent gespeichert
rsaSessionkey4SM	$0101\ 0100_2$ = ´54´	asymmetrisch, ExternalAuth und InternalAuth, Sessionkeys für SecureMessaging
rsaSessionkey4TC	$0111\ 0100_2$ = ´74´	asymmetrisch, ExternalAuth und InternalAuth, Sessionkeys für PSO Kommandos
elc…		

Hinweis (97): Fehlenden Kodierungen werden erst ab Generation 2 benötigt.

Tabelle 168: AlgorithmIdentifier für Ver- und Entschlüsselung

Name	Kodierung	Verwendung
desSessionkey4TC	0000 0000_2 = ´00´	PSO Decipher + Encipher
rsaDecipherOaep	1000 0101_2 = ´85´	PSO Decipher mit RSA, siehe PI in [14890–2] Table 11-3
rsaDecipherPKCS1_V1_5	1000 0001_2 = ´81´	
rsaEncipherOaep	0000 0101_2 = ´05´	PSO Encipher + Transcipher mit RSA, entspricht AlgorithmIdentifier für Decipher mod 128
rsaEncipherPKCS1_V1_5	0000 0001_2 = ´01´	
elcSharedSecretCalculation	0000 1011_2 = ´0B´	PSO Decipher mit ELC, siehe [14890–2] Table 11-2

Tabelle 169: AlgorithmIdentifier für Signaturerzeugung und Signaturprüfung

Name	Kodierung	Verwendung
sign9796_2_DS2	0000 0111_2 = ´07´	PSO Compute DS
signPKCS1_V1_5	0000 0010_2 = ´02´	PSO Compute DS, siehe [14890–1] Table 13-1
signPSS	0000 0101_2 = ´05´	PSO Compute DS, siehe [14890–1] Table 13-1
signECDSA		PSO Compute Digital Signature
verifyCertificate	´XX´	Zertifikatsprüfung, da dieser Identifier nie an der physikalischen Schnittstelle verwendet wird, braucht er hier nicht festgelegt zu werden

Hinweis (98): Der algorithmIdentifier für sign9796_2_DS2 wird in diesem Dokument definiert, weil das Signatur Schema DS2 aus [9796–2] in [14890–1] nicht behandelt wird.

Hinweis (99): Fehlenden Kodierungen werden erst ab Generation 2 benötigt.

17.2 Kodierungen für Trailer

Tabelle 170: Trailer à Fehlername

Wert	Name	Bedeutung
´62 81´	CorruptDataWarning	die Integrität der Antwortdaten ist nicht gewährleistet
´62 82´	EndOfFileWarning	es wurden mehr Daten angefordert als die Datei enthält
´62 82´	EndOfRecordWarning	es wurden mehr Daten angefordert als der Rekord enthält
´62 82´	UnsuccessfulSearch	Pattern wurde in keinem der adressierten Rekords gefunden
´62 83´	FileDeactivated	File, auf welches sich die Operation bezieht, ist deaktiviert
´62 87´	RecordDeactivated	Rekord, auf welchen sich Operation bezieht, ist deaktiviert
´62 Cx´	TransportStatus	Indikation des Transportschutzverfahrens
´63 CF´	NoAuthentication	keine Authentisierung mit dem referenzierten Schlüssel
´63 Cx´	RetryCounter	Wert des Fehlbedienungszählers
´63 Cx´	UpdateRetryWarning	Schreibschwierigkeiten
´63 Cx´	WrongSecretWarning	falsches Passwort in den Kommandodaten
´64 00´	AuthenticationFailure	Authentisierung fehlgeschlagen
´64 00´	CurveNotFound	Domainparameter der elliptischen Kurve nicht gefunden
´64 00´	EncipherError	fehlerhafte Verschlüsselungsoperation
´64 00´	KeyAlreadyPresent	Schlüsseldaten bereits vorhanden, Generierung unmöglich
´64 00´	KeyInvalid	Schlüsseldaten fehlen, Generierung erforderlich
´65 81´	MemoryFailure	Schreibfehler
´67 00´	WrongRecordLength	falsche Rekordlänge
´68 81´	ChannelClosed	logischer Kanal nicht geöffnet
´69 81´	WrongFileType	Datei unterstützt das aktuelle Kommando nicht
´69 82´	SecurityStatusNotSatisfied	Zugriffsregel nicht erfüllt
´69 83´	CommandBlocked	Rücksetzen des Fehlbedienungszählers nicht mehr möglich
´69 83´	PasswordBlocked	Fehlbedienungszähler abgelaufen
´69 85´	NoKeyReference	Schlüsselreferenz fehlt, MSE Set ist notwendig
´69 85´	NoPrkReference	Schlüsselreferenz fehlt, MSE Set ist notwendig
´69 85´	NoPukReference	Schlüsselreferenz fehlt, MSE Set ist notwendig
´69 85´	NoRandom	keine Zufallszahl, GET CHALLENGE ist notwendig
´69 85´	NoRecordLifeCycleStatus	Datei unterstützt das aktuelle Kommando nicht
´69 85´	PasswordNotUsable	Transportschutz aktiv, CHANGE REF. DATA notwendig
´69 85´	ShortPassword	neues Passwort zu kurz
´69 86´	NoCurrentEF	Kommandobearbeitung unmöglich, da keine Datei selektiert
´69 88´	IncorrectSmDo	fehlerhaftes Secure Messaging
´6A 80´	VerificationError	fehlerhaftes CV–Zertifikat
´6A 80´	WrongCiphertext	fehlerhaftes Kryptogramm
´6A 81´	UnsupportedFunction	Schlüssel unterstützt den angegeben Algorithmus nicht
´6A 82´	FileNotFound	referenzierte Datei nicht gefunden
´6A 83´	RecordNotFound	referenzierter Rekord nicht verwendbar
´6A 84´	DataTooBig	zu viele Daten
´6A 84´	FullRecordList	Rekordliste bereits komplett gefüllt
´6A 84´	OutOfMemory	zu wenig Speicherplatz
´6A 88´	InconsistentKeyReference	Schlüsselreferenz im CV–Zertifikat fehlerhaft
´6A 88´	KeyNotFound	referenzierten Schlüssel nicht gefunden
´6A 88´	PasswordNotFound	referenziertes Passwort nicht gefunden
´6A 88´	PrKNotFound	referenzierten Schlüssel nicht gefunden
´6A 88´	PukNotFound	referenzierten Schlüssel nicht gefunden
´6A 89´	DuplicatedObject	neu anzulegendes Objekt existiert bereits
´6A 8A´	DfNameExists	neu anzulegende Applikation existiert bereits
´6B 00´	OffsetTooBig	Offset zu groß
´90 00´	NoError	normale Kommandoausführung, kein Fehler, keine Warnung

Anhang A (informativ)

A1 – Abkürzungen

Kürzel	Erläuterung
AID	Application Identifier, siehe [7816–5], Identifikator für eine Chipkartenapplikation
APDU	Application Protocol Data Unit, siehe [7816–3], kleinste Nachrichteneinheit, die auf der Applikationsschicht mit der Chipkarte ausgetauscht wird
BGT	„block guard time“, siehe [7816–3] Kapitel 11.2
COS	Chipcard Operating System, Betriebssystem einer Chipkarte
DF	Dedicated File, siehe [7816–4], Bezeichnung für einen Ordner innerhalb des Objektsystems einer Chipkarte
EF	Elementary File, siehe [7816–4], Bezeichnung für eine Datei innerhalb des Objektsystems einer Chipkarte
ELC	Elliptic Curve Cryptography, Kryptographie mittels elliptischer Kurven
G1	Abkürzung für Generation 1, bezeichnet diese Version des Dokumentes, in der Regel ergänzt um den Zusatz „normativ“
G2	Abkürzung für Generation 2, bezeichnet eine spätere Version dieses Dokumentes, in der Regel ergänzt um den Zusatz „informativ“
IFD	Interface Device
IFSC	maximum Information Field Size for the Card, siehe [7816–3] Kapitel 11.4.2
IFSD	maximum Information Field Size for the Interface Device, siehe [7816–3] Kapitel 11.4.2
MAC	Message Authentication Code
MSBit	Most Significant Bit
MSByte	Most Significant Byte
LSBit	Least Significant Bit
LSByte	Least Significant Byte
TPDU	Transport Protocol Data Unit, siehe [7816–3], kleinste Nachrichteneinheit, die auf dem Data Link Layer mit der Chipkarte ausgetauscht wird

A2 – Glossar

Begriff	Erläuterung
ephemer	nur für kurze Zeit bestehend, flüchtig, ohne bleibende Bedeutung (siehe http://de.wiktionary.org/wiki/ephemer)
Nibble	Ein Nibble oder Nybble ist eine Datenmenge, die 4 Bits umfasst, es wird auch Halbbyte, Tetrade oder Quadrupel genannt (siehe http://de.wikipedia.org/wiki/Nibble).

Das Projektglossar wird als eigenständiges Dokument zur Verfügung gestellt.

A3 – Abbildungsverzeichnis

A4 – Tabellenverzeichnis

A5 – Referenzierte Dokumente

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[7816–3]	ISO/IEC 7816-3: 2006 (2nd edition) Identification cards — Integrated circuit cards with contacts — Part 3: Electrical interface and transmission protocols
[7816–4]	ISO/IEC 7816-4: 2004 (2nd edition) Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange
[7816–5]	ISO/IEC 7816-5: 2004 (2nd edition) Identification cards — Integrated circuit cards — Part 5: Registration of application providers
[7816–8]	ISO/IEC 7816-8: 2004 (2nd edition) Identification cards — Integrated circuit cards — Part 8: Commands for security operations
[7816–9]	ISO/IEC 7816-9: 2004 (2nd edition) Identification cards — Integrated circuit cards — Part 9: Commands for card management
[9796–2]	Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Integer factorization based mechanisms Second edition, 2002-10-01
[14890–1]	EUROPEAN STANDARD, DRAFT, prEN 14890-1, February 2007 Application Interface for smart cards used as secure signature creation devices – Part 1: Basic services
[14890–2]	EUROPEAN STANDARD, DRAFT, prEN 14890-2, February 2007 Application Interface for smart cards used as secure signature creation devices – Part 2: Additional services
[AES–128]	Federal Information Processing Standards Publication 197, (FIPS–197), November 26, 2001, Announcing the ADVANCED ENCRYPTION STANDARD (AES) http://csrc.nist.gov/publications/
[AES–CTR]	Section 6.5 of NIST Special Publication 800-38A, *Recommendation for Block, Cipher Modes of Operation, Methods and Techniques*, Morris Dworkin, December 2001 Edition, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
[ANSI X3.92]	American National Standard X3.92 – 1981, Data Encryption Algorithm
[ANSI X9.62]	Public Key Cryptography for the Financial Services Industry, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005
[ANSI X9.63]	Public Key Cryptography for the Financial Services Industry, *Key Agreement and Key Transport Using Elliptic Curve Cryptography*, 2001
[BinPrefix]	Prefix for binary multiples IEC INTERNATIONAL STANDARD 60027-2, Third edition, 2005-08, Letter symbols to be used in electrical technology – Part 2: Telecommunications and electronics, clause 3.8.3 http://physics.nist.gov/cuu/Units/binary.html http://de.wikipedia.org/wiki/Bin%C3%A4rpr%C3%A4fix http://en.wikipedia.org/wiki/Binary_prefix
[BrainPool]	ECC Brainpool Standard Curves and Curve Generation v. 1.0 19.10.2005 http://www.teletrust.de/fileadmin/files/oid/oid_ECC-Brainpool-Standard-curves-V1.pdf
[CMAC]	NIST Special Publication 800-38B, Recommendation for Block, Cipher Modes of Operation: The CMAC Mode for Authentication, Morris Dworkin, May 2005, http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[gemPKI_Reg]	gematik (18.03.2008): Einführung der Gesundheitskarte – Registrierung einer CVC–CA der zweiten Ebene Version 1.5.0, www.gematik.de
[gemSpec_eGK_P2]	gematik (25.03.2008): Einführung der Gesundheitskarte – Spezifikation elektronische Gesundheitskarte; Teil 2: Grundlegende Applikationen Version 2.2.0, www.gematik.de
[gemSpec_Krypt]	gematik (26.03.2008): Einführung der Gesundheitskarte – Verwendung kryptographischer Algorithmen in der Telematikinfrastruktur Version 1.3.0, www.gematik.de
[RFC2119]	Network Working Group, Request for Comments: 2119, S. Bradner Harvard, University, March 1997, Category: Best Current Practice Key words for use in RFCs to Indicate Requirement Levels http://www.apps.ietf.org/rfc/rfc2119.html
[PKCS#1]	PKCS #1 v2.1: RSA Cryptography Standard, RSA Laboratories, June 14, 2002 ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf
[SHA–256]	Federal Information Processing Standards Publication 180-2 Change Notice to include SHA-224), *section 6.2* 2002 August 1, Announcing the, SECURE HASH STANDARD http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[TR–03111]	Technical Guideline TR-03111, Elliptic Curve Cryptography, Based on ISO 15946, Version 1.00 http://www.bsi.de/literat/tr/tr03111/BSI-TR-03111.pdf
[TR–03116]	BSI TR-03116, Technische Richtlinie für die eCard-Projekte der Bundesregierung, Version: 1.0, Datum: 23.03.2007, Status: veröffentlichte Version, Fassung: 2007 http://www.bsi.de/literat/tr/tr03116/BSI-TR-03116.pdf

A6 – Klärungsbedarf

Kap.	Offener Punkt	Zuständig
–	–	A. Fiedler

Anhang B: Vorgaben zur Performance (normativ)

B.1 – Einführung (informativ)

Die Akzeptanz der elektronischen Gesundheitskarte und der damit verbundenen neuen Abläufe in den medizinischen Einrichtungen (Praxis, Apotheke, Krankenhaus, …) hängt stark von den Zeiten ab, die für einzelne Aktionen benötigt werden. Um im Gesamtsystem zu akzeptablen Werten zu kommen, sind für die einzelnen Komponenten Vorgaben zu treffen, die als zulassungsrelevant in die jeweiligen Spezifikationen aufgenommen werden. In diesem Dokument werden entsprechende Vorgaben für die elektronische Gesundheitskarte festgelegt.

Es werden sowohl die Randbedingungen als auch die Messparameter festgelegt, um reproduzierbare und aussagekräftige Messungen zu ermöglichen. Es werden Vorgaben für verschiedene Szenarien gemacht. Die ermittelten Messwerte werden über ein Punktesystem bewertet. Die Gesamt–Punktzahl setzt sich aus einer gewichteten Addition aller Einzel–Punktzahlen zusammen. Das Betriebssystem wird nur dann von der gematik zugelassen, wenn es eine definierte Mindest–Punktzahl (siehe (N1097)) erreicht.

B.2 – Messaufbau (normativ)

Die Bearbeitungszeiten eines Kommandos werden mit dem im Folgenden beschriebenen Messaufbau durchgeführt:

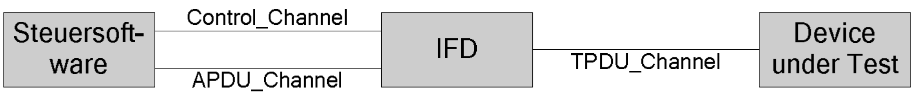

Abbildung 8: Komponentendiagramm Performance Messplatz

Die Komponente **Steuersoftware** führt die Performancemessung durch und protokolliert dabei die Messergebnisse. Diese Komponente besitzt logisch gesehen zwei Kommunikationskanäle zum Interface Device (IFD, Kartenleser). Der eine Kanal transportiert Kommando und Antwort APDUs, die vom IFD zum Prüfling weitergeleitet werden. Der zweite Kanal dient der Steuerung des IFD sowie dem Auslesen der Laufzeiten, die vom IFD gemessen werden.

Die Komponente **Interface Device (IFD)** transferiert APDUs gemäß der elektrischen Schnittstelle vom und zum Prüfling. Dabei ermittelt das IFD die Bearbeitungszeit eines Kommando–Antwort APDU Paares. Wichtig ist dabei, dass das IFD die maximale Übertragungskapazität des Prüflings unterstützt (siehe Anhang B.4).

Die Komponente **„Device under Test“ (DuT)** repräsentiert den Prüfling, dessen Performance zu ermitteln ist.

B.3 – Anforderungen an die Steuersoftware (normativ)

(N1073) Die Steuersoftware MUSS in der Lage sein, die einzelnen Schritte der Performancemessung auszuführen, zu protokollieren und das Gesamtergebnis der Messung aus den einzelnen Resultaten zu ermitteln.

B.4 – Anforderungen an das Interface Device (IFD) (normativ)

(N1074) Der Übertragungskanal TPDU_Channel des IFD SOLL die maximale Übertragungskapazität des Prüflings DuT unterstützen. Im Einzelnen bedeutet das:

(N1075) Das IFD MUSS im Rahmen einer PPS Sequenz (siehe Kapitel 12.2.4) alle Werte für PPS1 aus der Menge {´18´, ´95´, ´96´, ´97´} unterstützen.

Tabelle 171: Bedeutung PPS1 gemäß [7816–3] Table 7 und 8

PPS1	Teilerfaktor	f_{max} / [MHz]	*C* / [kBaud]
´18´	372/12 = 31	5	161
´95´	512/16 = 32	5	156
´96´	512/32 = 16	5	313
´97´	512/64 = 8	5	625

(N1076) Das IFD MUSS eine Versorgungsspannung U_{cc} gemäß [7816–3] Table 1 für Class A und Class B liefern.

(N1077) Das IFD MUSS einen Versorgungsstrom *Icc* gemäß [7816–3] Table 1 für Class A und Class B liefern.

(N1078) Das IFD MUSS während der Performancemessung ein Clocksignal mit einer Frequenz aus dem Intervall [4,95, 5,05] MHz liefern.

(N1079) Das IFD MUSS „direct convention“ (siehe [7816–3] Kapitel 8.1) unterstützen.

(N1080) Falls das Byte TC_1 im ATR fehlt, dann MUSS das IFD beim Senden eine „extra guard time“ von 12 etu verwenden (siehe [7816–3] Kapitel 8.3).

(N1081) Falls das Byte TC_1 im ATR vorhanden ist, dann MUSS das IFD beim Senden eine „extra guard time“ von 11 etu verwenden (siehe [7816–3] Kapitel 8.3 und 11.2).

(N1082) Das IFD MUSS beim Empfang eine „character guard time“ von 11 etu unterstützen (siehe [7816–3] Kapitel 11.2).

(N1083) Das IFD MUSS *BGT* = 22 etu verwenden (siehe [7816–3] Kapitel 11.2).

(N1084) Das IFD MUSS für IFSC einen Wert von 254 unterstützen (siehe [7816–3] Kapitel 11.4.2).

(N1085) Das IFD MUSS für IFSD einen Wert von 254 unterstützen (siehe [7816–3] Kapitel 11.4.2).

(N1086) Das IFD MUSS 5.000 und SOLL 32.800 als APDU Länge unterstützen.

B.5 – Allgemeines (normativ)

B.5.1 – Normale Zeitmessung

Hier wird die standardmäßig verwendete Zeitmessung beschrieben. Es ist möglich, dass hiervon im Einzelfall abgewichen wird. Darauf wird gegebenenfalls explizit hingewiesen.

(N1087) Die normale Zeitmessung MUSS die Zeitspanne $t_{Run} = t_{End} - t_{Start} - t_{IO}$ ermitteln, wobei folgende Definitionen gelten:

a. Der Zeitpunkt t_{Start} ist durch den Beginn des ersten Startbits des ersten Characters (siehe [7816–3] Figure 7) der ersten TPDU (siehe [7816–3] Figure 23) gekennzeichnet, welche zur Übertragung einer Kommando APDU verwendet wird.

b. Der Zeitpunkt t_{End} ist durch das Ende des letzten Paritybits des letzten Characters der letzten TPDU gekennzeichnet, welche zur Übertragung der zugehörigen Response APDU verwendet wird.

c. Die Zeitspanne t_{IO} berücksichtigt die Übertragungszeit von Kommando und Antwort APDU. Sie MUSS wie folgt berechnet werden:

   i. Sei N_{cmd_APDU} die Anzahl Oktette in der Kommando APDU, dann gilt für die Anzahl übertragener Oktette N_{cmd_TPDU} auf TPDU Ebene falls N_{cmd_APDU} größer gleich 1 ist und das Epilogue Field einen „longitudinal redundanc code“ enthält (siehe [7816–3] Kapitel 11.3.4):
   $N_{cmd\_TPDU} = N_{cmd\_APDU} + 8\ \mathrm{ceil}(\ N_{cmd\_APDU} / 254\ ) - 4.$

   ii. Für die Anzahl N_{cmd_etu} folgt unter der Vorraussetzung, dass pro Oktett 11 etu benötigt werden und eine „block guard time“ von 22 etu berücksichtigt wird (siehe [7816–3] Kapitel 11.2):
   $N_{cmd\_etu} = 11\ N_{cmd\_TPDU} + 22\ \mathrm{ceil}(\ N_{cmd\_APDU} / 254\ ).$

   iii. Analog gilt für die Antwort APDU:
   $N_{rsp\_TPDU} = N_{rsp\_APDU} + 8\ \mathrm{ceil}(\ N_{rsp\_APDU} / 254\ ) - 4$
   und $N_{rsp\_etu} = 11\ N_{rsp\_TPDU} + 22\ \mathrm{ceil}(\ N_{rsp\_APDU} / 254\ ).$

   iv. Daraus folgt: $t_{IO} = (\ N_{cmd\_etu} + N_{rsp\_etu}\ )\ \text{Teilerfaktor} / f_{max}$ mit Teilerfaktor und f_{max} in Abhängigkeit von PPS1 aus Tabelle 171.

B.5.2 – Reguläre Aktivierung der eGK

Hier wird die standardmäßige Aktivierung der Karte beschrieben, die der Testvorbereitung dient. Es ist möglich, dass hiervon im Einzelfall abgewichen wird. Darauf wird gegebenenfalls explizit hingewiesen.

(N1088) Schritt 1: Im Rahmen der regulären Aktivierung der eGK MUSS zunächst eine Aktivierung gemäß (N238) und (N239) erfolgen.

(N1089) Schritt 2: Es MUSS eine PPS Sequenz gemäß Kapitel 12.2.4 erfolgen. Die Bits 4 bis 1 in PPS0 MÜSSEN das Übertragungsprotokoll T=1 anzeigen. Als PPS1 MUSS der Wert von TA_1 aus dem ATR verwendet werden.

(N1090) Schritt 3: Das IFD MUSS der eGK den Wert von IFSD = 254 präsentieren.

(N1091) Schritt 4: Es MUSS eine beliebige Kommando APDU bearbeitet werden. Einzige Anforderung an diese Kommando APDU ist, dass sie so gewählt werden MUSS, dass der Trailer NoError anzeigt und den auszuführenden Testfall nicht beeinflusst. Hier SOLL eine Selektion des *root* Ordners gemäß Kapitel 15.2.6.1 verwendet werden.

Hinweis (100): Die hier beschriebene reguläre Aktivierung hat das Ziel, die eGK vollständig zu booten. Typischerweise sind bei aktuellen Chipkartenbetriebssystemen die Initialisierungen so umfangreich, dass sie nicht vollständig innerhalb der Sendezeit des ATR ausführbar sind. Deshalb wird der eGK durch das abschließende Kommando genügend Zeit zur Verfügung gestellt, die typischerweise nicht relevant für die Performancemessung ist (zur einzigen Ausnahme siehe Anhang B.6.1).

B.5.3 – Punkteermittlung

Input:	$\underline{\underline{t}}$	n–Tupel mit Zeiten von n Einzelmessungen
	T_R	Referenzzeit
Output:	P	gewichtete Qualität der Messzeit
Errors:	–	keine
Notation:		P = points($\underline{\underline{t}}$, T_R)

Es gelten folgende Definitionen:

X = Mittelwert (Erwartungswert) aller Einzelmessungen im n–Tupel $\underline{\underline{t}}$.

σ = Standardabweichung aller Einzelmessungen im n–Tupel $\underline{\underline{t}}$.

(N1092) $f_1 = e^{-\sigma/X}$.

(N1093) Es gilt:

a. $f_i = e^{-\frac{s}{X}}$.

b. $f_2 = 1 - \left(\frac{X}{2 \cdot T_R}\right)^2$.

(N1094) Es gilt: $P = f_1\, f_2\, T_R$.

Die Funktion P besteht aus den Faktoren f_1 und f_2 und wird durch T_R gewichtet.

Falls die Standardabweichung σ null ist, dann ist der Faktor f_1 exakt eins. Größere Standardabweichungen führen zu kleineren Faktoren und damit zu geringeren Punktzahlen P.

Der Faktor f_2 setzt den Mittelwert X in Relation zur Referenzzeit T_R. Deshalb wird f_2 als Funktion von X aufgefasst, die durch T_R parametrisiert wird. Trivialerweise sollte ein kleiner Wert von X zu einem größeren Wert von f_2 führen. Daraus folgt, dass f_2 streng monoton fallend ist, mithin also die Ableitung von f_2 nach X kleiner null ist. Zudem ist es ratsam, auch die zweite Ableitung von f_2 nach X kleiner gleich null zu wählen, weil es dann eher lohnt schlechte Mittelwerte zu verbessern, als gute Mittelwerte weiter zu optimieren.

Falls der Mittelwert X der Messergebnisse gleich dem Referenzwert T_R ist, dann ist der Faktor f_2 0,75.

Das Produkt aus f_1 und f_2 wird zur Ermittlung des Wertes P mit der Referenzzeit T_R gewichtet. Durch diese Gewichtung korreliert das Gewicht eines Prüfpunktes mit dem Beitrag des Prüfpunkt im Rahmen zusammengesetzter Kommandosequenzen.

B.5.4 – Gesamtbewertung

Die Gesamtbewertung wird durch gewichtetes Addieren der Einzelpunktzahlen ermittelt. Die Gewichte korrelieren mit der Häufigkeit des Auftretens der Einzelpositionen im Feld.

Tabelle 172: Gesamtbewertung

Prüfpunkt		**T_{Ri} / [ms]**	**Gewicht g_i**	**P_i / [s]**	**$g_i\, P_i$ / [s]**
Kanalkapazität	B.5.5 *PIO*	35	100		
Karte starten	B.6.1 *PKarteStarten*	500	110		
RsaRoleAuthentication	B.7.1.1 *PRsaRoleAuthentication*	720	100		
Schlüsselimport	B.6.2 *PImport*	260	100		
RsaRoleCheck	B.6.2 *PRsaRoleCheck*	120	100		
Signieren 2	B.7.1.5 *PSignPKCS1_V1_5*	700	86		
Lesen VD	B.6.4 *PLesenVD*	80	83		
Verordnung lesen	B.6.7 *PVerordnungLesen*	350	57		
Entschlüsseln 1	B.7.1.4 *PRsaDecipherOaep*	800	57		
Verordnung schreiben	B.6.6 *PVerordnungSchreiben*	400	50		
Asym. mut. Auth.	B.6.2 *PRsaSK*	1700	30		
Benutzerverifikation	B.7.2.1 *PVerify*	70	25		
Ticket verbergen	B.7.3 *PDeactivateRecord*	300	20		
Ticket entsperren	B.7.3 *PActivateRecord*	300	20		
Passwort ändern	B.7.2.2 *PChangeReferenceData*	90	15		
Lesen NFD	B.6.5 *PLesenNFD*	240	15		
Entschlüsseln 2	B.7.1.3 *PRsaDecipherPKCS1_v1_5*	680	10		
Sym. mut. Auth.	B.6.3 *PDesMutual*	175	8		
Aktualisieren VD	B.6.3 *PAktualisierungVD*	1000	5		
Client Authentication	B.7.1.2 *PRsaClientAuthentication*	670	5		
Signieren 1	B.7.1.6 *PSignPSS*	800	2		
Signieren 3	B.7.1.7 *PSign9796_2_DS2*	710	2		
Summe		10700	1000		

Für die Gesamtbewertung gilt:

(N1095) $$P_{gesamt} = \sum_i g_i \cdot P_i = \sum_i g_i \cdot f_{1i} \cdot f_{2i} \cdot T_{Ri}\ .$$

Unter den vereinfachenden Annahmen, dass die Messwerte innerhalb einer Reihe identisch sind (Standardabweichung ist null) und das Verhältnis X_i / T_{Ri} in allen Messreihen konstant ist folgt dann:

$$P_{gesamt\_einfach} = f_2 \sum_i g_i \cdot T_{Ri} = 455{,}68 \cdot f_2$$, mit f_2 als Funktion von X_i / T_{Ri}.

Abbildung 9 zeigt diesen Zusammenhang graphisch mit dem Verhältnis X_i / T_{Ri} als Abszisse und $P_{gesamt\_einfach}$ als Ordinate. Demnach erreicht eine unendlich schnelle Karte 455,68 Punkte. Eine Karte, die in allen Prüfpunkten die Referenzzeit benötigt erreicht rund 342 Punkte. Eine Karte, die überall die doppelte Referenzzeit benötigt null Punkte.

(N1096) Ein COS DARF KEINE Zulassung erhalten, wenn in wenigstens einem Prüfpunkt das Verhältnis X_i / T_{Ri} größer als vier ist.

Kom
3002

(N1097) Ein COS DARF KEINE Zulassung erhalten, wenn die gemäß (N1095) ermittelte Punktzahl kleiner als 200 s ist. Dem entspricht gemäß Abbildung 9 eine relative Ausführungszeit, die etwa 1,5-mal so groß ist wie die Referenzzeit.

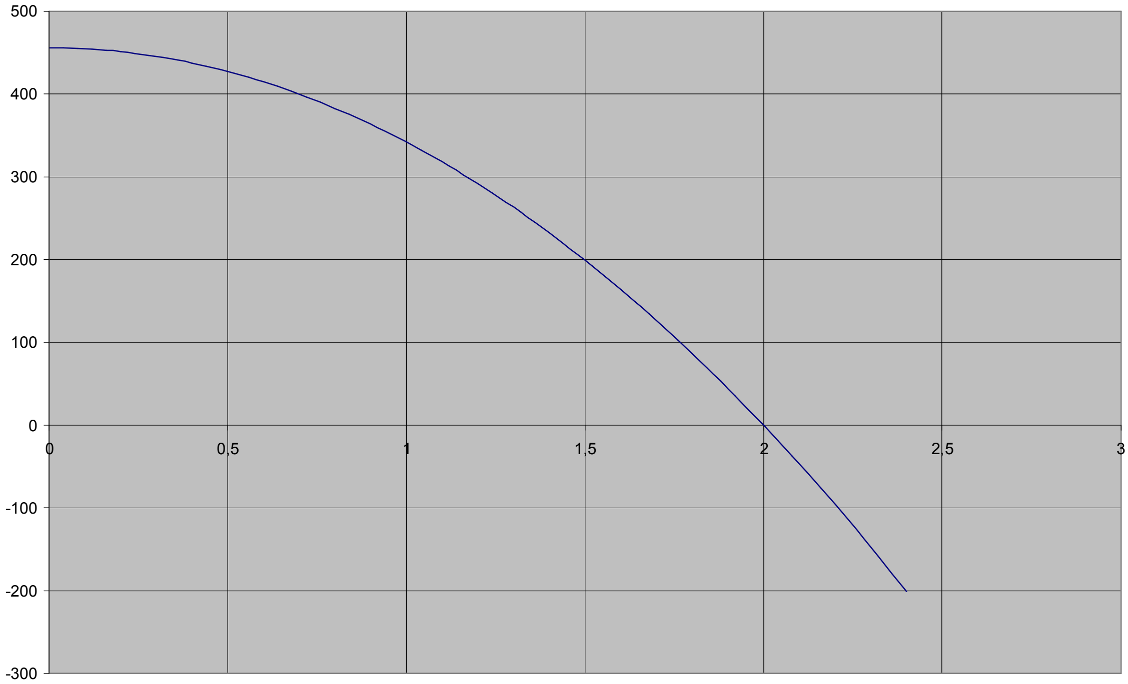

Abbildung 9: Graphische Darstellung von $P_{gesamt\_einfach}$

B.5.5 – Übertragungsgeschwindigkeit

Dieser Prüfpunkt berücksichtigt die Kanalkapazität des Kanals TPDU_Channel in Abbildung 8. Diese wird wegen der starken Abhängigkeit von der externen Taktfrequenz des IFD nicht experimentell, sondern rechnerisch aus den Bytes TA_1 und TC_1 im ATR und der Kapazität C aus Tabelle 171 ermittelt. Für die (rechnerische) Übertragungszeit von 1.000 Oktett zur Karte gilt:

$$t_T = \frac{1000 \cdot CGT}{C}$$

Mit C aus Tabelle 171 für den Fall PPS1 = TA_1 und CGT in Abhängigkeit von TC_1 im ATR gemäß folgender Tabelle:

Tabelle 173: Character Guard Time (CGT) gemäß [7816–3] Kapitel 11.2

TC_1	´FF´	´00´	´01´	´02´	…	´FD´	´FE´
CGT	11	12	13	14	…	253	254

Es gilt: P_{IO} = points((t_T, t_T), T_{IO}).

B.6 – Business Use Cases (normativ)

B.6.1 – Startsequenz

Dieser Prüfpunkt behandelt eine Sequenz, die typischerweise dann durchlaufen wird, wenn eine eGK dem nutzenden System unbekannt ist.

Der Prüfpunkt beinhaltet:

1. Die Aktivierung (Bootvorgang) des Betriebssystems.
2. Die Aushandlung einer höheren Übertragungsrate.
3. Die Aushandlung einer Puffergröße für die Datenübertragung.
4. Auslesen von Übertragungsparametern aus EF.ATR.
5. Auslesen der Seriennummer der Karte aus EF.GDO.
6. Auslesen der Versionsnummern für Betriebssystem und Applikation aus EF.Version.

Testvorbereitung:

Keine.

Testdurchführung:

(N1098) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 8 ausgeführt. Abweichend von den Festlegungen in Anhang B.5.1 wird hier die Zeit pro Schleifendurchlauf anders ermittelt.

(N1099) Schritt 1: Die Zeit t_{Start} gibt den Zeitpunkt an, zu welchem die eGK gemäß (N1088) aktiviert wird. Genauer, den Zeitpunkt des Wechsels von RST von L nach H (siehe [7816–3] Figure 1). Anschließend MUSS der ATR empfangen werden.

(N1100) Schritt 2: Es MUSS eine PPS Sequenz gemäß Kapitel 12.2.4 erfolgen. Die Bits 4 bis 1 in PPS0 MÜSSEN das Übertragungsprotokoll T=1 anzeigen. Als PPS1 MUSS der Wert von TA_1 aus dem ATR verwendet werden.

(N1101) Schritt 3: Das IFD MUSS der eGK den Wert von IFSD = 254 präsentieren.

(N1102) Schritt 4: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.ATR, *offset* = 0 und *length* = WildCardShort.

(N1103) Schritt 5: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.GDO, *offset* = 0 und *length* = WildCardShort.

(N1104) Schritt 6: Auslesen aller Rekords in EF.Version:

a. Angefangen mit Rekord 1 MÜSSEN alle Rekords in EF.Version gelesen werden. Jede Leseoperation MUSS gemäß (N1104)b ausgeführt werden.

b. Das erste READ RECORD Kommando MUSS gemäß Kapitel 15.4.5.2 mit den Parametern *shortFileIdentifier* von EF.Version, *recordNumber* = 1 und *length* = WildCardShort ausgeführt werden. Alle weiteren READ RECORD

Kommandos MÜSSEN gemäß Kapitel 15.4.5.1 mit einer um eins inkrementierten *recordNumber* und *length* = WildCardShort ausgeführt werden.

(N1105) Schritt 7: Auslesen der ersten vier Rekords in EF.DIR:

a. Angefangen mit Rekord 1 MÜSSEN die ersten vier Rekords in EF.DIR gelesen werden. Jede Leseoperation MUSS gemäß (N1105)b ausgeführt werden.

b. Das erste READ RECORD Kommando MUSS gemäß Kapitel 15.4.5.2 mit den Parametern *shortFileIdentifier* von EF.DIR, *recordNumber* = 1 und *length* = WildCardShort ausgeführt werden. Alle weiteren READ RECORD Kommandos MÜSSEN gemäß Kapitel 15.4.5.1 mit einer um eins inkrementierten *recordNumber* und *length* = WildCardShort ausgeführt werden.

(N1106) Schritt 8: Der Zeitpunkt t_{End} ist definiert durch das Ende der letzten TPDU, welche in (N1105)b übertragen wird. Damit gilt für den *i*-ten Schleifendurchlauf:

$t_{Run8,i} = t_{End} - t_{Start}$.

Testauswertung:

Es gilt: $P_{KarteStarten}$ = points($(t_{Run8,1}, t_{Run8,2}, \ldots, t_{Run8,100})$, $T_{KarteStarten}$).

Testnachbereitung:

Keine.

Besser wäre es, wenn auch hier die Messung aus B.5.1 angewendet werden könnte, gibt es dafür passendes Equipment, insbesondere für die Schritte 1 bis 3?

B.6.2 – Schlüsselimport und Authentisierungsprotokolle

In diesem Kapitel wird der Import von Authentisierungsschlüsseln mittels CV–Zertifikaten behandelt, wobei die öffentlichen Schlüssel aller verwendeten CA bei Produktion des Prüflings bekannt sind. Zudem wird die Rollenüberprüfung (eine andere Komponente authentisiert sich gegenüber der eGK) sowie die asymmetrische Aushandlung von Sessionkeys betrachtet.

Der Prüfpunkt beinhaltet einerseits:

1. Den Import eines Authentisierungsschlüssels.
2. Eine Rollenprüfung zur Erlangung eines Sicherheitszustandes.

Zudem wird hier ebenfalls geprüft

3. Den asymmetrischen Aufbau eines Trusted–Channels zur Karte.

Testvorbereitung:

Es werden zehn Test CA benutzt, denen von der Root–CA (siehe auch Anhang B.8.1) folgende CV–Zertifikate zugeordnet werden:

Test CA0	7f218201465f37820100337e6f08c7c6b73d83b4ba588a77b86f1b4312e1cf10e63040b4d2a b401f3c7549e10dc15b3d564055d9d31cb10d668b369b93df255a92f87cbe810ba35f14ccd4 260bf403622f6efe8f5c8f83380e72305e67ca253631dfdf0618772e30f8809c786abad84e0 cc814ee5f2890bfeb68b2652682e11253e6673a24fc732ffee7c380a9d49d24a609f8db1fed d5ecd0968dff042ca1a12f4f49525d83f5a492e86df65886b9a10945dc8c7c3d31d441ea21d 0c2d6b562969487495818497 4ea538476d749f32e36635429f8aa0196b1a9fa7916e5ca7360 87f66cc241da508328398efee9bd0ce84c8ad66853ed7427b0ed46f3ce1385d7ffa2a96f942 a7adfa95f383e802b826a6750346f9c911c371b6335fb8afd620826bd04b28bcad1af9db3ee 2d01673500010001 2b2403040202044445585858 6f000044455858 58600007
Test CA1	7f218201465f378201002224f38581b381e8ccf9e2c8dc2e5f93fea0ee522ed5eb5a2980388 75955ecaa9e24e6346971aade9766041fcaab8a29e2e5f89b491abd2f0b676ca84c7ea386b3 047f24a9cf3967591ec22694f52b3ad0ccf24556dbcc829f9055f33a941bec6f0a3fe4ae0c4 052f8944a294181af448d6075b680bf5cfc9c96c063c5067c5f32af36ba74af270cfd0ee4d4 656867a667f5a396a9b7adfe05f797dff9856267acdf6d5231e98689bc0ee6f367df5bc2d3c 5660b52213cce3fec57925cd7263cd8a1701b604ef0a11faf61096eea9abf670d2c8d980a71 e71ffdf15a328202f9eccb857a9d80def967f3f49fd1743e6cb087da61494a454a45bad9238 ca034c75f383e962b75f7f5b057fd5840f3e123f1990e957643d000cb73d1c78e30965bfef5 68abd135000100012b240304020204444558585 86f000144455858586000 07
Test CA2	7f218201465f37820100418498d75e93f8cd1259e5bb1aacaee5b57c3515b09615131ef3a36 004d21e60ebb36657ec13cbab0d740deb7242339fd740855559bf19f27ba4caf5fd43b9e99a a480ce75c980fa6075154c2910df2b48ed09d59caf2caa8539ccc27285a208b35cf02c12cee 7840c1e39a398bea513c4ffefd5125b2b71a9828a137939b1664dcbd10daca09c65289f0108 2993d4c3f2f4cbcfed71a78e686972d270289911c9b13b77eb995f7dfc14e5472b0f616053d b1fec66f79518b6142de04ae3f9cfb6d97cbaf96a374153769dd1b5af58424ccb4c27b2c601 2cd89b9571de6cfc5037f5956208759111b8a3fc71bdcae00a0e46f36ea657e1dee0925c665 741d1935f383ec141ec9be7330b62ae964d30bced07a1e1a126128aaf6d84527e587daf1861 533ab3b3000100012b2403040202044445585858 6f00024445585858600007
Test CA3	7f218201465f3782010004364de27591aa68edabdc4dbbddeedd3b64868dcb5591fde7a57ff b5d6e1daeabfd8f0f2f54ad2e91f4e723ba4b11eace7c1c8ede448bbb9de85cff0787bfcdd1 d5a6d7eeab9912aa11656809f278af443739c3b198b020fa3961b41626925be91a1243ffd8f a2b96906c6227df0a8df6c98318cad1600858ca18bfb5c79ae4946a5191202e9a14c2ebef1c 13e40a55b1f9a6fb367b606b3b6abffb4b87a5ea90d5171c05eb6e71d03d32fd68a2688d75e 28176ad602b21037ee512c7d8daf40403741dca1d5a5c9a1f5529f09b470bcf06a49ed3df32 60d3c21a490b82acb1b38a780a1af21216adc38a0f53628145260133ee2aa709ab2b96e0a01 645d6fd5f383e5c334af5daeb8f9bfd65e1dfcc3b3e75fbeb29334737964284dd152a2b737a 7f9f9a87000100012b2403040202044445585858 6f00034445585858600007
Test CA4	7f218201465f378201001d350e066c45ceb83b6632d74b6dcc6e8de1ac0ca1ed8e09dd0b11c 47eb5b296abe43a7e6dd601bc1f5fe0da3179fd56d48523e33783dd0cd2a9cca2d3a6705d01 3a1ae3154547510c6d574adc29deee0611683bef22be8512acaf741055fa7df2af608cfd658 8d4c89ec9adaae3f85cd647b55bf20caf6b112690f43ffe8417165438a7eafee85cc8b6f62b 61f2446baeda0f96b4f756feead1c35dc3f61a9c24c7e88001972e32a1735421e8f672f0f4a f8829917d550ab0ede2dc28008185b7baa5b4f2338b506c66f1f0bce569f74f71245ba4d86e 8fac67417b87b7a40d255798ef6f256daa955ab8763579cd3d84a28ebd577d682873ea8d8d5 29dbb325f383e4e44e0a92b055f8baa3d6f48a15f2157e59375e0c28c0f1fa5492c07758242 1944186b000100012b2403040202044445585858 6f00044445585858600007
Test CA5	7f218201465f378201004270db3d35aeeb9059e50951e806dfdcb5b47b8be9c25ec1abe8817 f9cc81215da0543c634cc5b25ed8f0caca2906409985402f4d1e95e4be3f6ce1afc64a55d59 b761d0fad591a272fa299685abba557a0fea5125b40310febe57b8c33535434c34ec776c77b 956f5e3c34854abadc088dffd52e78ed17ae536abd579e5e7005a91c1c9792938d0a9133cd5 c2b2a86ace16e65ac898d276042934c94c4424f6de0070ed2be16e8eb5154be41b8904087bd 63a86046f31246729ba8377a66c58bf7ec2ee2a6f199ebb8a660eaff6b0d8784933d5090c01 6454b561906c6b2a184740226c90a7fba4e97bb6638d973cbd021a68e9a33dfafdb32e44c00 08ff0555f383ed0a493d766acb031f881bff984b380cd1d032d1107562ef1e762e88af4ce0c 9c869c77000100012b2403040202044445585858 6f00054445585858600007
Test CA6	7f218201465f3782010047de1c03d39ce99ca55d4ab3295a27a7084a053ad40c62e2cc22123 671a71d6a3a39e5e6a5e8c860a178cece1fb616409db66e37c1afadc3d0e37739cb00866935 1360d192ae2e1b369c565d0c64e6af7bcb8e6846099b2a89c4bdba7ff197b12038e3acd5ff7 36e76337dbbdd387d8acdcad2719188873472054bbc4c820fbb64fad382b661aab4aad7c0a3 01d0852b3efc8b5f11e5f62d2c77af8f3d7968db6196441d7580440ddc42a5ad09ceb594993 cef01d9e1e804f3a4f2d990b92b61c447e955d13566624052af958b60c54d66c380b84d6243 d818cbc1796463ae4888b7cea2df4841811f772fe55baae10a5863f6b1ed030b965da1ff06e d4375d25f383e9d483c91c5bb23c9a2fe2a49486a3d589f26870e8888e7281c8e451ed8557e ffc5b7b7000100012b2403040202044445585858 6f00064445585858600007
T	7f218201465f37820100564d5199cc3f6422ea4d7e530ce92f9a0e3cbbfab2f071d2d7a404b

estCA7	3aef6ab41feca85c700a3654285ed76734aa5f7d6d406ea6cd8ade53cd5c01d776b759a06e3 05806d8d6d2365fc79ce64157c7348cb85a84a14b693ad1bafe1a70dc177732cf28b482e56e 2aa5e8ea076db513407617747ffed0db1405fd1e6407a7c72b31663de1d587ac751e4590096 f803248dcf91cc2c5d65ad8cdf14b213c1d3e6cc184fea6a4432c5fec226e3e9474d67dd349 f16cdb3b839b527b85b6cd259dd22f3dd84df7fca37462a4d784f7a1519b05f68a122cf1fe0 de850bdb40f0d9f17b95c584f16ed98e06ef62e7aa685d42f603b25ae6b2769ca50e58995d3 8dae9a65f383e694eb7f2846ee1882cbf82aa963cf67110865c2e87c5ff070c705194f088ed 65e0badd000100012b240304020204444558585 86f0007444558585860000 7
TestCA8	7f218201465f3782010048c0b951eda01b5b73bb3c08112a3782c81964fd763adfcf3051fd2 a2e08508b1211f4eacb9c75b821377fc1500bf27e91d3010e9ed5b51f93b3b2a2bfe78c7727 e161926deb9f09025c915c6e3a4e210be152724cd569b1ad5671d45382ce172bc903f782f40 a51384dce8fc2d47d7305453df19c3b5db21bd70b86a9853d59909eef7e3d0153458506b266 6ca7453d2cca6b7714df1b00a9616ee65881e01a4831bb327ddf96a8d9e0e5f52272efd410c 9392a03d4531794652ef191563a3bf20661a6e17f06555d61aff74d6229e71da94aa3be6a3b 0979f776f0dcec4065853d5668a4df79d555bcbd10d841f7b220d0d2c03c10beff371cc9da1 155462c5f383efca28f902c460608781d8d4d11ac62fd335cf44e51003916d1500fa85a631d cd9777f1000100012b24030402020444455858586f00084445585858600007
TestCA9	7f218201465f378201003184b73b3fe7e4075b76707d99820987d6005fadb3924062bb28058 74c0823a4cb2bf43108554daa21ce625d7463ca7df97df2f8f28d01e537650dc8d0343fa8fe 1019f44f8331484bf5f4f52322898188e7a96504ce4a6e49e6af1a43fc9dd7eb37a89e2510f 281b666ccc8d63ad39c2a684a9dd7aa9d45067152aa61b9f1e5e1c37a278da73571fbf58ac6 20263ac3eb7670ca9d75a4fe570b4812786ec1eccea2995166cc9a2fdc0fc2c2cb1b7eecbef 4eb77244ed64968635a2faad54ec60d9e15f73fd9d123ed64be812b5f092242482650c351f7 7da445cc857c006d0dc06b24a062b5877dc90ceed6b258dd556d93cece059dc8b6b66a2aa7c 3769d215f383e2ac2314b076695394767e03dc6da9c03146febfff16128c4eabbb2afb56e46 c511b5af000100012b24030402020444455858586f00094445585858600007

Von jeder Test_CA werden zehn CV–Zertifikate gemäß Kapitel 8.1.2.2 erzeugt. Insgesamt ergeben sich so einhundert CV–Zertifikate für Authentisierungsschlüssel.

(N1107) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1108) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

Testdurchführung:

(N1109) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 0 bis 14 ausgeführt.

(N1110) Schritt 0: MSE Restore Kommando gemäß Kapitel 15.9.6.1 mit *seNo* = 1. Dadurch werden alle Elemente aus der Liste *globalSecurityList* (siehe (N299)e entfernt. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

(N1111) Schritt 1: Aus den einhundert CV–Zertifikaten mit Authentisierungsschlüssel wird ein bislang noch nicht verwendetes gezogen (Ziehen ohne Zurücklegen). Durch die Ziehung wird folgende CV–Zertifikatskette gebildet:
CVC_RCA **à** CVC_Test_CAx **à** CVC_ICCy.
Die erste Ziehung ist beliebig. Bei allen weiteren Ziehungen MUSS die Nebenbedingung beachtet werden, dass CVC_Test_CAx verschieden ist vom unmittelbar vorher verwendeten Zertifikat CVC_Test_CAx. Dann MUSS mit Schritt 4 fortgefahren werden.

(N1112) Schritt 2: MSE Set Kommando gemäß Kapitel 15.9.6.8 wobei als *keyRef* der Wert *keyIdentifier* aus (N1261) verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

(N1113) Schritt 3: PSO Verify Certificate Kommando gemäß Kapitel 15.8.6.1 wobei als Parameter *certificate* CVC_Test_CAx verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run3,i}$ bezeichnet.

(N1114) Schritt 4: MSE Set Kommando gemäß Kapitel 15.9.6.8 wobei als *keyRef* der Wert CAR aus CVC_ICCy verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run4,i}$ bezeichnet. Falls dieses Kommando nicht mit NoError beendet wird, fahre mit Schritt 2 fort, sonst mit Schritt 5.

(N1115) Schritt 5: PSO Verify Certificate Kommando gemäß Kapitel 15.8.6.1 wobei als Parameter *certificate* CVC_ICCy verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run5,i}$ bezeichnet. Falls dieses Kommando nicht mit NoError beendet wird, fahre mit Schritt 2 fort, sonst mit Schritt 6.

(N1116) Schritt 6: MSE Set Kommando gemäß Kapitel 15.9.6.5 wobei als *keyRef* der Wert CHR aus CVC_ICCy und als *algId* rsaRoleCheck verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run6,i}$ bezeichnet.

(N1117) Schritt 7: GET CHALLENGE Kommando gemäß Kapitel 15.9.3.1. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run7,i}$ bezeichnet.

(N1118) Schritt 8: EXTERNAL AUTHENTICATE Kommando gemäß Kapitel 15.7.1.1 und (N844)f. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run8,i}$ bezeichnet.

(N1119) Schritt 9: MSE Restore Kommando gemäß Kapitel 15.9.6.1 mit *seNo* = 2. Dadurch werden alle Elemente aus der Liste *globalSecurityList* entfernt und PrK.eGK.AUT_CVC ist zur Aushandlung von Sessionkeys verwendbar. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run9,i}$ bezeichnet.

(N1120) Schritt 10: MSE Set Kommando gemäß Kapitel 15.9.6.3 wobei als *keyRef* der Wert *keyIdentifier* von PrK.eGK.AUT_CVC und als *algId* rsaSessionkey4SM verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run10,i}$ bezeichnet.

(N1121) Schritt 11: MSE Set Kommando gemäß Kapitel 15.9.6.5 wobei als *keyRef* der Wert CHR aus CVC_ICCy und als *algId* rsaSessionkey4SM verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run11,i}$ bezeichnet.

(N1122) Schritt 12: INTERNAL AUTHENTICATE Kommando gemäß Kapitel 15.7.4.1 und (N869)f. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run12,i}$ bezeichnet.

(N1123) Schritt 13: GET CHALLENGE Kommando gemäß Kapitel 15.9.3.1. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run13,i}$ bezeichnet.

(N1124) Schritt 14: EXTERNAL AUTHENTICATE Kommando gemäß Kapitel 15.7.1.1 und (N844)g. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run14,i}$ bezeichnet.

Testauswertung:

Bei der Bearbeitung einer Schleifeniteration sind folgende Fälle denkbar:

a. Der öffentliche Schlüssel der CA ist bereits in der Karte gespeichert, weil er bei der Kartenproduktion in der *persistentPublicKeyList* oder durch einen früheren Zertifikatsimport in *publicKeyList* gespeichert wurde. In diesem Fall besteht der *i*-te Schleifendurchlauf aus der Schrittfolge 4, 5, 6, 7, …. Die Laufzeiten der Schritte 4, 5 und 6 werden summiert zu $t_{Import,i}$.

b. Der öffentliche Schlüssel der CA ist nicht in der Karte gespeichert und dies wird bereits während der Schlüsselselektion bemerkt. Dann besteht der *i*-te Schleifendurchlauf aus der Schrittfolge 4, 2, 3, 4, 5, 6, 7, …. Die Laufzeiten der Schritte 4, 2, 3, 4, 5 und 6 werden summiert zu $t_{Import,i}$.

c. Der öffentliche Schlüssel der CA ist nicht in der Karte gespeichert und dies wird erst in Schritt 5 bemerkt, wenn der Schlüssel benutzt wird. Dann besteht der *i*-te Schleifendurchlauf aus der Schrittfolge: 4, 5, 2, 3, 4, 5, 6, 7, …. Die Laufzeiten der Schritte 4, 5, 2, 3, 4, 5 und 6 werden summiert zu $t_{Import,i}$.

d. Die Laufzeiten der Schritte 9, 10, 11, 12, 13 und 14 werden aufsummiert zu $t_{SK,i}$.

Es gilt: P_{Import} = points(($t_{Import,1}$, $t_{Import,2}$, …, $t_{Import,100}$), T_{Import}).

Es gilt: $P_{RsaRoleCheck}$ = points(($t_{Run7,1}$ + $t_{Run8,1}$, …, $t_{Run7,100}$ + $t_{Run8,100}$), $T_{RsaRoleCheck}$).

Es gilt: P_{RsaSK} = points(($t_{SK,1}$, $t_{SK,2}$, …, $t_{SK,100}$), T_{RsaSK}).

Testnachbereitung:

Keine.

Hintergrund:

Es wird erwartet, dass bei der Nutzung einer eGK innerhalb der Telematikinfrastruktur die eGK auf eine Vielzahl Karten von Leistungserbringern trifft (CVC_ICCy). Es erscheint unrealistische alle öffentlichen Schlüssel aus CVC_ICCy zu cachen. Demgegenüber ist es wahrscheinlich, dass es nur eine begrenzte Anzahl von CA geben wird, so dass sich das Speichern dieser Schlüssel aus Performancegründen lohnt (cachen oder personalisieren).

B.6.3 – Versichertendaten aktualisieren

In diesem Kapitel wird eine gegenseitige symmetrische Authentisierung bei gleichzeitiger Aushandlung von Sessionkeys betrachtet. Dazu wird das Schlüsselobjekt SK.VSDD verwendet. Zudem werden die Versichertendaten verändert.

Dieser Prüfpunkt beinhaltet:

1. Selektion der Applikation DF.HCA.
2. Aufbau eines Trusted–Channels mittels SK.VSDD.
3. Änderung von EF.StatusVD zur Kennzeichnung, dass eine Transaktion durchgeführt werden soll.
4. Aktualisierung von Daten in EF.PD, EF.VD und EF.GVD
5. Änderung von EF.StatusVD zur Kennzeichnung, dass die Transaktion zur Änderung der Versichertendaten abgeschlossen ist.

Testvorbereitung:

(N1125) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

Testdurchführung:

(N1126) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 11 ausgeführt.

(N1127) Schritt 1: SELECT Kommando gemäß Kapitel 15.2.6.5 mit *aid* gleich dem Attribut *applicationIdentifier* des Ordners *root*. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

(N1128) Schritt 2: MSE Restore Kommando gemäß Kapitel 15.9.6.1 mit *seNo* = 1. Dadurch werden alle Elemente aus der Liste *globalSecurityList* entfernt. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

(N1129) Schritt 3: SELECT Kommando gemäß Kapitel 15.2.6.5 mit *aid* gleich dem Attribut *applicationIdentifier* des Ordners DF.HCA. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run3,i}$ bezeichnet.

(N1130) Schritt 4: MSE Set Kommando gemäß Kapitel 15.9.6.6 wobei als *keyRef* der Wert des Attributes *keyIdentifier* von SK.VSDD und als *algId* desSessionkey4SM verwendet wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run4,i}$ bezeichnet.

(N1131) Schritt 5: GET CHALLENGE Kommando gemäß Kapitel 15.9.3.1. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run5,i}$ bezeichnet.

(N1132) Schritt 6: MUTUAL AUTHENTICATE Kommando gemäß Kapitel 15.7.1.2 und (N844)d. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run6,i}$ bezeichnet.

(N1133) Schritt 7: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.2, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.StatusVD verwendet wird. Weiterhin MUSS gelten: *offset* = 0 und *newData* ist ein zufällig gewählter Oktettstring, dessen Länge dem Attribute *numberOfBytes* von EF.StatusVD entspricht. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run7,i}$ bezeichnet.

(N1134) Schritt 8: Aktualisieren des kompletten Inhalts von EF.PD:

a. Es MUSS eine Oktettstring gemäß *z* = RAND(*numberOfBytes*) erzeugt werden, wobei dessen Länge identisch zum Attribute *numberOfBytes* von EF.PD.

b. Der Oktettstring *z* wird gemäß den (impliziten oder expliziten) Angaben im EF.ATR zum Kommando UPDATE BINARY in einen oder mehrere, aber möglichst wenige Teile zerlegt. Für jeden Teil wird Schritt (N1134)c mit entsprechendem *offset* ausgeführt. Die Laufzeit aller Kommandos in (N1134)c in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run8,i}$.

c. Das erste (oder einzige) UPDATE BINARY Kommando MUSS gemäß Kapitel 15.3.4.2 ausgeführt werden, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.PD verwendet wird. Falls vorhanden, MÜSSEN alle weiteren UPDATE BINARY Kommandos gemäß Kapitel 15.3.4.1 ausgeführt werden. Die Parameter *offset* und *newData* sind stets passend zum entsprechenden Teil z aus (N1134)b zu wählen.

(N1135) Schritt 9: Aktualisieren des kompletten Inhalts von EF.VD:

a. Es MUSS eine Oktettstring gemäß z = RAND(*numberOfBytes*) erzeugt werden, wobei dessen Länge identisch zum Attribute *numberOfBytes* von EF.VD.

b. Der Oktettstring z wird gemäß den (impliziten oder expliziten) Angaben im EF.ATR zum Kommando UPDATE BINARY in einen oder mehrere Teile zerlegt. Für jeden Teil wird Schritt (N1135)c mit entsprechendem *offset* ausgeführt. Die Laufzeit aller Kommandos in (N1135)c in der i-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run9,i}$.

c. Das erste (oder einzige) UPDATE BINARY Kommando MÜSS gemäß Kapitel 15.3.4.2 ausgeführt werden, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.VD verwendet wird. Falls vorhanden MÜSSEN alle weiteren UPDATE BINARY Kommandos gemäß Kapitel 15.3.4.1 ausgeführt werden. Die Parameter *offset* und *newData* sind stets passend zum entsprechenden Teil z aus (N1135)b zu wählen.

(N1136) Schritt 10: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.2, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.GVD verwendet wird. Weiterhin MUSS gelten: *offset* = 0 und *newData* ist ein zufällig gewählter Oktettstring, dessen Länge dem Attribute *numberOfBytes* von EF.GVD entspricht. Die Laufzeit dieses Kommandos in der i-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run10,i}$ bezeichnet.

(N1137) Schritt 11: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.2, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.StatusVD verwendet wird. Weiterhin MUSS gelten: *offset* = 0 und *newData* ist ein zufällig gewählter Oktettstring, dessen Länge dem Attribute *numberOfBytes* von EF.StatusVD entspricht. Die Laufzeit dieses Kommandos in der i-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run11,i}$ bezeichnet.

Testauswertung:

Als Gesamtdauer der Authentisierung gilt: $t_{a,i} = t_{Run3,i} + t_{Run4,i} + t_{Run5,i} + t_{Run6,i}$.

Als Gesamtdauer der Aktualisierung gilt: $t_{u,i} = t_{Run7,i} + t_{Run8,i} + t_{Run9,i} + t_{Run10,i} + t_{Run11,i}$.

Es gilt: $P_{DesMutual}$ = points(($t_{a,1}, t_{a,2}, \ldots, t_{a,100}$), $T_{DesMutual}$).

Es gilt: $P_{AktualisierungVD}$ = points(($t_{u,1}, t_{u,2}, \ldots, t_{u,100}$), $T_{AktualisierungVD}$).

Testnachbereitung:

Keine.

B.6.4 – Versichertendaten lesen

Dieser Prüfpunkt liest die Versichertendaten. Der dazu notwendige Schlüsselimport und die Rollenprüfung wurden bereits in Anhang B.6.2 behandelt und sind daher für diesen Prüfpunkt irrelevant.

Dieser Prüfpunkt beinhaltet:

1. Auslesen des Inhaltes von EF.StatusVD
2. Auslesen von EF.PD, EF.VD und EF.GVD

Testvorbereitung:

(N1138) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1139) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1140) Schritt 3: Es MUSS eine externe Authentisierung mit der Rolle CHA.2 durchgeführt werden.

(N1141) Schritt 4: Das DF.HCA MUSS selektiert werden.

Testdurchführung:

(N1142) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 4 ausgeführt.

(N1143) Schritt 1: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.StatusVD, *offset* = 0 und *length* = WildCardShort. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run1,i}$ bezeichnet.

(N1144) Schritt 2: Lesen des kompletten Inhalts von EF.PD:

a. Für das Auslesen MÜSSEN möglichst wenige Leseoperationen verwendet werden, wobei die (implizite oder explizite) Angabe in EF.ATR zum Kommando READ BINARY beachtet werden MUSS. Jede Leseoperation MUSS gemäß (N1144)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1144)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run2,i}$.

b. Das erste (oder einzige) READ BINARY Kommando MUSS gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.PD und *offset* = 0 ausgeführt werden. Falls vorhanden, MÜSSEN alle weiteren READ BINARY Kommandos gemäß Kapitel 15.3.2.1 mit passendem *offset* ausgeführt werden.

(N1145) Schritt 3: Lesen des kompletten Inhalts von EF.VD:

a. Für das Auslesen MÜSSEN möglichst wenige Leseoperationen verwendet werden, wobei die (implizite oder explizite) Angabe in EF.ATR zum Kommando READ BINARY beachtet werden MUSS. Jede Leseoperation MUSS gemäß (N1145)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1145)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run3,i}$.

b. Das erste (oder einzige) READ BINARY Kommando MUSS gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.VD und *offset* = 0 ausgeführt werden. Falls vorhanden, MÜSSEN alle weiteren READ BINARY Kommandos gemäß Kapitel 15.3.2.1 mit passendem *offset* ausgeführt werden.

(N1146) Schritt 4: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.GVD, *offset* = 0 und *length* = WildCardExtended. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run4,i}$ bezeichnet.

Testauswertung:

Für Gesamtdauer der Leseoperation gilt: $t_{r,i} = t_{Run1,i} + t_{Run2,i} + t_{Run3,i} + t_{Run4,i}$.

Es gilt: $P_{LesenVD}$ = points(($t_{r,1}$, $t_{r,2}$, …, $t_{r,100}$), $T_{LesenVD}$).

Testnachbereitung:

Keine.

B.6.5 – Notfalldaten lesen

Dieser Prüfpunkt liest die Notfalldaten. Der dazu notwendige Schlüsselimport und die Rollenprüfung wurden bereits in Anhang B.6.2 behandelt und sind daher für diesen Prüfpunkt irrelevant.

Dieser Prüfpunkt beinhaltet:

1. Auslesen des Inhaltes von EF.StatusNotfalldaten.
2. Auslesen des gesamten Inhaltes der Datei EF.Notfalldaten.

Testvorbereitung:

(N1147) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1148) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1149) Schritt 3: Es MUSS eine externe Authentisierung mit der Rolle CHA.7 durchgeführt werden.

(N1150) Schritt 4: Das DF.HCA MUSS selektiert werden.

Testdurchführung:

(N1151) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1152) Schritt 1: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.StatusNotfalldaten, *offset* = 0 und *length* = WildCardShort. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run1,i}$ bezeichnet.

(N1153) Schritt 2: Lesen des kompletten Inhalts von EF.Notfalldaten:

a. Für das Auslesen MÜSSEN möglichst wenige Leseoperationen verwendet werden, wobei die (implizite oder explizite) Angabe in EF.ATR zum Kommando READ BINARY beachtet werden MUSS. Jede Leseoperation MUSS gemäß (N1153)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1153)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run2,i}$.

b. Das erste (oder einzige) READ BINARY Kommando MUSS gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF. Notfalldaten und *offset* = 0 ausgeführt werden. Falls vorhanden, MÜSSEN alle weiteren READ BINARY Kommandos gemäß Kapitel 15.3.2.1 mit passendem *offset* ausgeführt werden.

Testauswertung:

Für Gesamtdauer der Leseoperation gilt: $t_{r,i} = t_{Run1,i} + t_{Run2,i}$.

Es gilt: $P_{LesenNFD}$ = points(($t_{r,1}$, $t_{r,2}$, …, $t_{r,100}$), $T_{LesenVD}$).

Testnachbereitung:

Keine.

B.6.6 – Verordnung schreiben

Dieser Prüfpunkt überträgt eine 4.000 Oktett große „Verordnung“ auf die eGK. Der dazu notwendige Schlüsselimport und die Rollenprüfung wurden bereits in Anhang B.6.2 behandelt und sind daher für diesen Prüfpunkt irrelevant.

Dieser Prüfpunkt beinhaltet:

1. Lesen von EF.StatusVerordnung.
2. Ändern von EF.StatusVerordnung um anzuzeigen, dass eine Transaktion aktiv ist.
3. Suchen nach einem freien Speicherplatz in EF.eVerordnungTicket.
4. Schreiben eines Tickets nach EF.eVerordnungTicket.
5. Auslesen der Speicherbelegung in EF.eVerordnungContainer.
6. Aktualisieren der Speicherbelegung in EF.eVerordnungContainer.
7. Schreiben eines 4.000 Oktett großen Blockes nach EF.eVerordnungContainer.
8. Ändern von EF.StatusVerordnung um anzuzeigen, dass die Transaktion abgeschlossen ist.

Testvorbereitung:

(N1154) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1155) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1156) Schritt 3: Es MUSS eine externe Authentisierung mit der Rolle CHA.2 durchgeführt werden.

(N1157) Schritt 4: Das DF.HCA MUSS selektiert werden.

(N1158) Schritt 5: Der Inhalt aller Rekords in EF.eVerordnungTicket MUSS so gesetzt werden, dass kein Rekord das Suchmuster ´0000 0000´ enthält.

Testdurchführung:

(N1159) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 0 bis 8 ausgeführt.

(N1160) Schritt 0: UPDATE RECORD Kommando gemäß Kapitel 15.4.7.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungTicket, einer zufällig gewählten Rekordnummer und *newData* = ´00…00´. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

(N1161) Schritt 1: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.StatusVerordnung, *offset* = 0 und *length* = WildCardShort. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run1,i}$ bezeichnet.

(N1162) Schritt 2: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.1 mit den Parametern *offset* = 0 und *newData* gleich einem zufällig gewählten Oktettstring, dessen Länge dem Attribute *numberOfBytes* von EF.StatusVerordnung entspricht. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

(N1163) Schritt 3: SEARCH RECORD Kommando gemäß Kapitel 15.4.6.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungTicket, *recordNummer* = 1, *searchString* gemäß (N1158) und *length* = WildCardShort. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run3,i}$ bezeichnet.

(N1164) Schritt 4: UPDATE RECORD Kommando gemäß Kapitel 15.4.7.1 mit den Parametern *recordNummer* aus (N1163) und *newData* mit zufällig gewähltem Inhalt. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run4,i}$ bezeichnet.

(N1165) Schritt 5: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungContainer, *offset* = 0 und *length* = ´40´ = 64. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run5,i}$ bezeichnet.

(N1166) Schritt 6: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.1 mit den Parametern *offset* gleich einem zufällig gewählten Wert aus der Menge {0, 8, 16, 24, 32, 40, 48, 56} und *newData* = RAND(8). Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run6,i}$ bezeichnet.

(N1167) Schritt 7: Aktualisieren eines 4.000 Oktette großen Bereiches in EF.eVerordnungContainer:

a. Es MUSS eine Oktettstring gemäß *z* = RAND(4000) erzeugt werden. Aus dem Intervall [64, *numberOfBytes* – 4000] MUSS ein Startoffset ausgewählt werden.

b. Der Oktettstring *z* wird gemäß den (impliziten oder expliziten) Angaben im EF.ATR zum Kommando UPDATE BINARY in einen oder mehrere Teile zerlegt. Für jeden Teil wird Schritt (N1167)c mit entsprechendem *offset* ausgeführt. Die Laufzeit aller Kommandos in (N1167)c in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run7,i}$.

c. Alle UPDATE BINARY Kommandos MÜSSEN gemäß Kapitel 15.3.4.1 ausgeführt werden. Die Parameter *offset* und *newData* sind stets passend zum entsprechenden Teil z aus (N1167)b zu wählen.

(N1168) Schritt 8: UPDATE BINARY Kommando gemäß Kapitel 15.3.4.2, wobei als *shortFileIdentifier* das entsprechende Attribut von EF.StatusVerordnung verwendet wird. Weiterhin MUSS gelten: *offset* = 0 und *newData* ist ein zufällig gewählter Oktettstring, dessen Länge dem Attribute *numberOfBytes* von EF.StatusVerordnung entspricht. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run8,i}$ bezeichnet.

Testauswertung:

Für Gesamtdauer der Operation gilt: $t_{r,i} = t_{Run1,i} + t_{Run2,i} + \ldots + t_{Run8,i}$.

Es gilt: $P_{VerordnungSchreiben}$ = points(($t_{r,1}$, $t_{r,2}$, …, $t_{r,100}$), $T_{VerordnungSchreiben}$).

Testnachbereitung:

Keine.

B.6.7 – Verordnung lesen

Dieser Prüfpunkt liest eine 4.000 Oktett große „Verordnung“ aus der eGK. Der dazu notwendige Schlüsselimport und die Rollenprüfung wurden bereits in Anhang B.6.2 behandelt und sind daher für diesen Prüfpunkt irrelevant.

Dieser Prüfpunkt beinhaltet:

1. Auslesen von EF.StatusVerordnung.
2. Auslesen aller Tickets aus EF.eVerordnungTicket.
3. Auslesen der Speicherbelegung in EF.eRzeptContainer.
4. Auslesen eines 4.000 Oktett großen Blockes aus EF.eVerordnungContainer.
5. „Löschen“ des Tickets in EF.eVerordnungTicket.

Testvorbereitung:

(N1169) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1170) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1171) Schritt 3: Es MUSS eine externe Authentisierung mit der Rolle CHA.3 durchgeführt werden.

(N1172) Schritt 4: Das DF.HCA MUSS selektiert werden.

(N1173) Schritt 5: Alle Rekords in EF.eVerordnungTicket MÜSSEN aktiviert sein.

Testdurchführung:

(N1174) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 6 ausgeführt.

(N1175) Schritt 1: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.StatusVerordnung, *offset* = 0 und *length* = WildCardShort. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run1,i}$ bezeichnet.

(N1176) Schritt 2: Auslesen aller Tickets aus EF.eVerordnungTicket:

a. Angefangen mit Rekord 1 MÜSSEN alle Rekords in EF.eVerordnungTicket gelesen werden. Jede Leseoperation MUSS gemäß (N1176)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1176)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run2,i}$.

b. Das erste READ RECORD Kommando MUSS gemäß Kapitel 15.4.5.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungTicket, *recordNumber* = 1 und *length* = WildCardShort ausgeführt werden. Alle weiteren READ RECORD Kommandos MÜSSEN gemäß Kapitel 15.4.5.1 mit einer um eins inkrementierten *recordNumber* und *length* = WildCardShort ausgeführt werden.

(N1177) Schritt 3: READ BINARY Kommando gemäß Kapitel 15.3.2.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungContainer, *offset* = 0 und *length* = ´40´ = 64. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run3,i}$ bezeichnet.

(N1178) Schritt 4: Lesen eines 4.000 Oktett großen Blocks aus EF.eVerordnungContainer:

a. Für das Auslesen MÜSSEN möglichst wenige Leseoperationen verwendet werden, wobei die (implizite oder explizite) Angabe in EF.ATR zum Kommando READ BINARY beachtet werden MUSS. Jede Leseoperation MUSS gemäß (N1178)c ausgeführt werden. Die Laufzeit aller Kommandos in (N1178)c in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run4,i}$.

b. Aus dem Intervall [64, *numberOfBytes* – 4000] MUSS ein Startoffset ausgewählt werden.

c. Alle READ BINARY Kommandos MÜSSEN gemäß Kapitel 15.3.2.1 mit passendem *offset* ausgeführt werden.

(N1179) Schritt 5: UPDATE RECORD Kommando gemäß Kapitel 15.4.7.2 mit den Parametern *shortFileIdentifier* von EF.eVerordnungTicket, einer zufällig gewählten Rekordnummer und *newData* = ´00…00´. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run5,i}$ bezeichnet.

(N1180) Schritt 6: UPDATE RECORD Kommando gemäß Kapitel 15.4.7.1 mit den Parametern *recordNummer* aus (N1179) und *newData* mit zufällig gewähltem Inhalt. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

Testauswertung:

Für Gesamtdauer der Operation gilt: $t_{r,i} = t_{Run1,i} + t_{Run2,i} + t_{Run3,i} + t_{Run4,i} + t_{Run5,i}$.

Es gilt: $P_{VerordnungLesen} = \text{points}(\ (t_{r,1}, t_{r,2}, \ldots, t_{r,100}\ ),\ T_{VerordnungLesen}\ )$.

Testnachbereitung:

Keine.

B.7 – Einzelkommandos (normativ)

B.7.1 – Asymmetrische Kryptographie

B.7.1.1 – Rollenauthentisierung

In diesem Kapitel wird die Rollenauthentisierung der eGK betrachtet, das bedeutet, die eGK authentisiert sich gegenüber einer anderen Komponente.

Testvorbereitung:

(N1181) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1182) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1183) Schritt 3: Damit das Ergebnis der Authentisierung prüfbar ist SOLL der zu PrK.eGK.AUT_CVC gehörende öffentliche Schlüssel aus dem Zertifikat in EF.C.eGK.AUT_CVC extrahiert werden.

(N1184) Schritt 4: MSE Set Kommando gemäß Kapitel 15.9.6.3 wobei als *keyRef* das Attribute *keyIdentifier* von PrK.eGK.AUT_CVC und als *algId* rsaRoleAuthentication verwendet wird.

Testdurchführung:

(N1185) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1186) Schritt 1: Es wird ein zufälliger Oktettstring *token* = RAND(16) erzeugt.

(N1187) Schritt 2: INTERNAL AUTHENTICATE Kommando gemäß Kapitel 15.7.4.1 und (N869)e. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{RsaRoleAuthentication}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,100}$), $T_{RsaRoleAuthentication}$).

Testnachbereitung:

Keine.

B.7.1.2 – Client / Server Authentisierung

In diesem Kapitel wird die Client / Server Authentisierung betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.AUT in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1188) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1189) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1190) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1191) Schritt 4: Damit das Ergebnis der Authentisierung prüfbar ist SOLL der zu PrK.CH.AUT gehörende öffentliche Schlüssel aus dem Zertifikat in EF.C.CH.AUT extrahiert werden.

(N1192) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.3 PrK.CH.AUT ausgewählt werden, wobei als *algId* rsaClientAuthentication verwendet wird.

Testdurchführung:

(N1193) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1194) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring z = RAND(i) erzeugt werden.

(N1195) Schritt 2: INTERNAL AUTHENTICATE Kommando gemäß Kapitel 15.7.4.1 und (N869)d mit *token* = *z*. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{RsaClientAuthentication}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,100}$), $T_{RsaClientAuthentication}$).

Testnachbereitung:

Keine.

B.7.1.3 – Entschlüsselung mittels PKCS #1 v1.5

In diesem Kapitel wird eine Entschlüsselung gemäß [PKCS#1] Kapitel 7.2.2 betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.ENC in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1196) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1197) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1198) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1199) Schritt 4: Zwecks Verschlüsselung SOLL der zu PrK.CH.ENC gehörende öffentliche Schlüssel PuK.CH.ENC aus dem Zertifikat in EF.C.CH.ENC extrahiert werden.

(N1200) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.9 PrK.CH.ENC ausgewählt werden, wobei als *algId* rsaDecipherPKCS1_V1_5 verwendet wird.

Testdurchführung:

(N1201) Die Testdurchführung MUSS eine Schleife 245-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1202) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring M =RAND(i) erzeugt werden. Dieser wird gemäß C = RSAES_PKCS1_V1_5_ENCRYPT(PuK.CH.ENC, M) verschlüsselt.

(N1203) Schritt 2: PSO Decipher Kommando gemäß Kapitel 15.8.2.1 und (N903)a. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{RsaDecipherPKCS1\_v1\_5}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,245}$), $T_{RsaDecipherPKCS1\_v1\_5}$).

Testnachbereitung:

Keine.

B.7.1.4 – Entschlüsselung mittels OAEP

In diesem Kapitel wird die Entschlüsselung gemäß [PKCS#1] Kapitel 7.1.2 betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.ENC in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1204) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1205) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1206) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1207) Schritt 4: Zwecks Verschlüsselung SOLL der zu PrK.CH.ENC gehörende öffentliche Schlüssel PuK.CH.ENC aus dem Zertifikat in EF.C.CH.ENC extrahiert werden.

(N1208) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.9 PrK.CH.ENC ausgewählt werden, wobei als *algId* rsaDecipherOaep verwendet wird.

Testdurchführung:

(N1209) Die Testdurchführung MUSS eine Schleife 190-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1210) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring *M* =RAND(*i*) erzeugt werden. Dieser wird gemäß *C* = RSAES_OAEP_ENCRYPT(PuK:CH.ENC, *M*) verschlüsselt.

(N1211) Schritt 2: PSO Decipher Kommando gemäß Kapitel 15.8.2.1 und (N903)b. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{RsaDecipherOaep}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,190}$), $T_{RsaDecipherOaep}$).

Testnachbereitung:

Keine.

B.7.1.5 – Signieren gemäß [PKCS#1] v1.5

In diesem Kapitel wird die elektronische Signatur gemäß [PKCS#1] Kapitel 8.1.1 und 9.1.1 betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.AUT in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1212) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1213) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1214) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1215) Schritt 4: Damit das Ergebnis der Authentisierung prüfbar ist SOLL der zu PrK.eGK.AUT_CVC gehörende öffentliche Schlüssel aus dem Zertifikat in EF.C.eGK.AUT_CVC extrahiert werden.

(N1216) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.7 PrK.CH.AUT ausgewählt werden, wobei als *algId* signPKCS1_V1_5 verwendet wird.

Testdurchführung:

(N1217) Die Testdurchführung MUSS eine Schleife 102-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1218) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring z = RAND(i) erzeugt werden.

(N1219) Schritt 2: PSO Compute Digital Signature Kommando gemäß Kapitel 15.8.1.1 und (N886)c, wobei *dataToBeSigned* = z gesetzt wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{SignPKCS1\_V1\_5}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,102}$), $T_{SignPKCS1\_V1\_5}$).

Testnachbereitung:

Keine.

B.7.1.6 – Signieren gemäß [PKCS#1] PSS

In diesem Kapitel wird die elektronische Signatur gemäß [PKCS#1] Kapitel 8.2.1 und 9.2 betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.AUT in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1220) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1221) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1222) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1223) Schritt 4: Damit das Ergebnis der Authentisierung prüfbar ist SOLL der zu PrK.eGK.AUT_CVC gehörende öffentliche Schlüssel aus dem Zertifikat in EF.C.eGK.AUT_CVC extrahiert werden.

(N1224) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.7 PrK.CH.AUT ausgewählt werden, wobei als *algId* signPSS verwendet wird.

Testdurchführung:

(N1225) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1226) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring z = RAND(i) erzeugt werden.

(N1227) Schritt 2: PSO Compute Digital Signature Kommando gemäß Kapitel 15.8.1.1 und (N886)b, wobei *dataToBeSigned* = z gesetzt wird. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{SignPSS}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,100}$), $T_{SignPSS}$).

Testnachbereitung:

Keine.

B.7.1.7 – Signieren gemäß [9796–2]

In diesem Kapitel wird die elektronische Signatur gemäß [9796–2] Kapitel 7 und 9 betrachtet. Dabei wird mit dem Schlüsselobjekt PrK.CH.AUT in DF.ESIGN gearbeitet.

Testvorbereitung:

(N1228) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1229) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1230) Schritt 3: Der Ordner DF.ESIGN MUSS selektiert werden.

(N1231) Schritt 4: Damit das Ergebnis der Authentisierung prüfbar ist SOLL der zu PrK.eGK.AUT_CVC gehörende öffentliche Schlüssel aus dem Zertifikat in EF.C.eGK.AUT_CVC extrahiert werden.

(N1232) Schritt 5: Anschließend MUSS mittels MSE Set Kommando gemäß Kapitel 15.9.6.7 PrK.CH.AUT ausgewählt werden, wobei als *algId* sign9796_2_DS2 verwendet wird.

Testdurchführung:

(N1233) Die Testdurchführung MUSS eine Schleife 256-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1234) Schritt 1: In der *i*-ten Schleifeniteration MUSS eine Oktettstring *M* =RAND(*i*) erzeugt werden.

(N1235) Schritt 2: PSO Compute Digital Signature Kommando gemäß Kapitel 15.8.1.2 und (N886)a, wobei als zu signierende Nachricht *M* verwendet wird um *signInput9796_2_DS2* zu berechnen. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit $t_{Run2,i}$ bezeichnet.

Testauswertung:

Es gilt: $P_{Sign9796\_2\_DS2}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,256}$), $T_{Sign9796\_2\_DS2}$).

Testnachbereitung:

Keine.

B.7.2 – Passwort

B.7.2.1 – Benutzerverifikation

Die Performancemessung zur Benutzerverifikation wird mit dem Objekt PIN.CH durchgeführt.

Testvorbereitung:

(N1236) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1237) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1238) Schritt 3: Es MUSS eine erfolgreiche Verifikation mit dem Objekt PIN.CH durchgeführt werden. Dadurch wird das Attribut *retryCounter* von PIN.CH auf einen definierten Wert gesetzt.

Testdurchführung:

(N1239) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 0 bis 4 ausgeführt.

(N1240) Schritt 0: MSE Restore Kommando gemäß Kapitel 15.9.6.1 mit *seNo* = 1. Dadurch wird das Attribut *securityStatusEvaluationCounter* auf den Wert null gesetzt. Die Laufzeit dieses Kommandos ist für diesen Prüfpunkt irrelevant.

(N1241) Schritt 1: VERIFY Kommando gemäß Kapitel 15.6.6.1 mit korrekten *verificationData*. Die Laufzeit $t_{Run1,i}$ dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden.

(N1242) Schritt 2: VERIFY Kommando gemäß Kapitel 15.6.6.1 mit falschen *verificationData*. Die Laufzeit $t_{Run2,i}$ dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden.

(N1243) Schritt 3: VERIFY Kommando gemäß Kapitel 15.6.6.1 mit falschen *verificationData*. Die Laufzeit $t_{Run3,i}$ dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden.

(N1244) Schritt 4: VERIFY Kommando gemäß Kapitel 15.6.6.1 mit korrekten *verificationData*. Die Laufzeit $t_{Run4,i}$ dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden.

Testauswertung:

Es gilt: $P_1 = \text{points}(\ (t_{Run1,1}, t_{Run1,2}, \ldots, t_{Run1,100}\ ), 0{,}8\ T_{Verify}\ )$.

Es gilt: $P_2 = \text{points}(\ (t_{Run2,1}, t_{Run2,2}, \ldots, t_{Run2,100}\ ), T_{Verify}\ )$.

Es gilt: $P_3 = \text{points}(\ (t_{Run3,1}, t_{Run3,2}, \ldots, t_{Run3,100}\ ), T_{Verify}\ )$.

Es gilt: $P_4 = \text{points}(\ (t_{Run4,1}, t_{Run4,2}, \ldots, t_{Run4,100}\ ), T_{Verify}\ )$.

Es gilt: $P_{Verify} = 0{,}8\ P_1 + 0{,}08\ P_2 + 0{,}02\ P_3 + 0{,}1\ P_4$.

Testnachbereitung:

Keine.

Hintergrund:

In Schritt 1 wird die Laufzeit eines erfolgreichen VERIFY Kommandos gemessen unter der Voraussetzung, dass das vorherige VERIFY Kommando ebenfalls erfolgreich war. Da sich hier der Wert des Attributes *retryCounter* vor und nach der Kommandoausführung nicht unterscheidet, ist hier die Sollzeit geringer als bei den anderen Schritten.

In Schritt 2 wird die Laufzeit eines erfolglosen VERIFY Kommandos gemessen unter der Voraussetzung, dass das vorherige VERIFY Kommando erfolgreich war.

In Schritt 3 wird die Laufzeit eines erfolglosen VERIFY Kommandos gemessen unter der Voraussetzung, dass das vorherige VERIFY Kommando ebenfalls erfolglos war.

In Schritt 4 wird die Laufzeit eines erfolgreichen VERIFY Kommandos gemessen unter der Voraussetzung, dass das vorherige VERIFY Kommando erfolglos war.

B.7.2.2 – Passwort ändern

Die Performancemessung zur Änderung eines Passwortes wird mit dem Objekt PIN.CH durchgeführt.

Testvorbereitung:

(N1245) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1246) Schritt 2: Dann MUSS der Ordner *root* selektiert werden.

(N1247) Schritt 3: Es MUSS eine erfolgreiche Verifikation mit dem Objekt PIN.CH durchgeführt werden. Dadurch wird das Attribut *retryCounter* von PIN.CH auf einen definierten Wert gesetzt.

Testdurchführung:

(N1248) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1249) Schritt 1: Es wird ein zufälliger Wert für *newSecret* gebildet, dessen Länge zufällig und gleichverteilt aus dem Intervall [6,12] gewählt wird.

(N1250) Schritt 2: CHANGE REFERENCE DATA Kommando gemäß Kapitel 15.6.1.1 mit korrektem *oldSecret* und dem in Schritt 1 erzeugtem *newSecret*. Die Laufzeit dieses Kommandos in der *i*-ten Schleifeniteration MUSS gemäß Anhang B.5.1 gemessen werden und wird mit t_{Run_i} bezeichnet.

(N1251) Alle während des Schleifendurchlaufes benutzten Werte für *newSecret* MÜSSEN paarweise verschieden sein.

Testauswertung:

Es gilt: $P_{ChangeReferenceData}$ = points(($t_{Run,1}$, $t_{Run,2}$, …, $t_{Run,100}$), $T_{ChangeReferenceData}$).

Testnachbereitung:

Der Wert des Attributes *secret* im Objekt PIN.CH MUSS auf den Ausgangswert gesetzt werden.

B.7.3 – Verordnungen verbergen und sichtbar machen

Dieser Prüfpunkt behandelt das Verbergen und Sichtbarmachen von Verordnungstickets. Absichtlich wird dabei jedes Ticket einzeln behandelt um zu einer Worst–Case Einschätzung zu gelangen.

Testvorbereitung:

(N1252) Schritt 1: Der Prüfling MUSS gemäß Anhang B.5.2 aktiviert werden.

(N1253) Schritt 2: PIN.home MUSS erfolgreich präsentiert werden.

(N1254) Schritt 3: Der Ordner DF.HCA MUSS selektiert werden.

(N1255) Schritt 4: Die Datei EF.eVerordnungTicket MUSS selektiert werden.

(N1256) Schritt 5: Alle Rekords in EF.eVerordnungTicket MÜSSEN aktiviert werden.

Testdurchführung:

(N1257) Die Testdurchführung MUSS eine Schleife 100-mal durchlaufen. In jedem Schleifendurchlauf werden die Schritte 1 bis 2 ausgeführt.

(N1258) Schritt 1: Verbergen aller Tickets

a. Angefangen mit Rekord 1 MÜSSEN alle Rekords in EF.eVerordnungTicket deaktiviert werden. Jedes Verbergen MUSS gemäß (N1258)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1258)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run1,i}$.
b. Alle DEACTIVATE RECORD Kommandos MÜSSEN gemäß Kapitel 15.4.3.1 ausgeführt werden.

(N1259) Schritt 2: Sichtbarmachen aller Tickets

a. Angefangen mit Rekord 1 MÜSSEN alle Rekords in EF.eVerordnungTicket aktiviert werden. Jedes Sichtbarmachen MUSS gemäß (N1259)b ausgeführt werden. Die Laufzeit aller Kommandos in (N1259)b in der *i*-ten Schleifeniteration MÜSSEN gemäß Anhang B.5.1 gemessen werden und summiert werden zu $t_{Run2,i}$.
b. Alle ACTIVATE RECORD Kommandos MÜSSEN gemäß Kapitel 15.4.1.1 ausgeführt werden.

Testauswertung:

Es gilt: $P_{DeactivateRecord}$ = points(($t_{Run1,1}$, $t_{Run1,2}$, …, $t_{Run1,100}$), $T_{DeactivateRecord}$).

Es gilt: $P_{ActivateRecord}$ = points(($t_{Run2,1}$, $t_{Run2,2}$, …, $t_{Run2,100}$), $T_{ActivateRecord}$).

Testnachbereitung:

Keine.

B.8 – Kartenkonfiguration (normativ)

(N1260) Grundsätzlich MUSS der Prüfling gemäß [gemSpec_eGK_P2] konfiguriert sein. Dabei MÜSSEN die Personalisierungsinformationen aus diesem Anhang berücksichtigt werden.

B.8.1 – / MF / PuK.RCA.CS

Zusätzlich zu den in [gemSpec_eGK_P2] Tabelle 15 beschriebenen Eigenschaften wird hier gefordert:

(N1261) Das Attribut *keyIdentifier* MUSS den Wert ´4445585858600007´ besitzen.

(N1262) Das Attribut *publicKey.n* MUSS den Wert
`´ca6333598b4577568ee7259a62b008be4baa2ed6f86a099cfe2612d430a`
`900ea8d0783de7ba174233e7a6ef6a197752dcd762d981c6354268cf0507`
`ce46d026f37f6c3e50efac2d79098b0be4d3374defe4bcf4e3b1f26f4f21`
`cab16367d1f8b6d257efa93bf0cd5ff933fb0ef30f5c75042c5fce9263f5`
`d7e845c7ff732a5238fcde15b970df296646a63c74a802dd8dc4d92e891e`
`2d044a83f4a664eb94babbb5e809c1e19f1b9cab688c8b005e7191983f87`
`b9e8a37a6ee6d8b3252f7392985419df26e31433c98547711d252ac28ea9`
`daad93f65a615f847dd52ccd326b1f5d8828f7387c6332e4f75bc93edc9a`
`9b2bc29a17de1782d0957e0426cce6a67´`
besitzen.

(N1263) Das Attribut *publicKey.e* MUSS den Wert ´01 0001´ = 65537 besitzen.

B.8.2 – / MF / PIN.CH

Zusätzlich zu den in [gemSpec_eGK_P2] Tabelle 12 beschriebenen Eigenschaften wird hier gefordert:

(N1264) Das Attribut *secret* MUSS den Wert „123456“ besitzen.

(N1265) Das Attribut *transportStatus* MUSS den Wert „regularPassword“ besitzen.

Der Wert des Attributes *PUK* ist für dieses Dokument irrelevant.

B.8.3 – / MF / PIN.home

Zusätzlich zu den in [gemSpec_eGK_P2] Tabelle 13 beschriebenen Eigenschaften wird hier gefordert:

(N1266) Das Attribut *secret* MUSS den Wert „123456“ besitzen.

(N1267) Das Attribut *transportStatus* MUSS den Wert „regularPassword“ besitzen.

Der Wert des Attributes *PUK* ist für dieses Dokument irrelevant.

B.8.4 – / MF / SK.VSDD

Zusätzlich zu den in [gemSpec_eGK_P2] Tabelle 18 beschriebenen Eigenschaften wird hier gefordert:

(N1268) Das Attribut *encKey* MUSS den Wert
´0102 0408 1020 4080   0204 0810 2040 8001   0408 1020 4080 0102´
besitzen.

(N1269) Das Attribut *macKey* MUSS den Wert
´0810 2040 8001 0204   1020 4080 0102 0408   2040 8001 0204 0810´
besitzen.

Anhang X: Erläuterungen zu Wertebereichen (informativ)

0 für *startSsec* macht keinen Sinn

250 als obere Intervallgrenze für *startSsec* lässt für 1–Byte Kodierungen noch genügend Luft für die Kodierung von „unendlich“, Escape Geschichten, RFU und ähnliches.

Anhang Y: Sicherheitshinweise (informativ)

1. Passwortvergleich: Laufzeit
2. GetChallenge: Stärke des Zufallszahlengenerators
3. GenAsymKeP: Stärke des Zufallszahlengenerators
4. Wäre sicherer, wenn bei PSO Transcipher Modulusläne PuK.enc >= Moduluslänge PrK.dec (ist für diese Version irrelevant).