Humboldt-Universität zu Berlin

Mathematisch-Naturwissenschaftliche Fakultät II

Institut für Informatik

Mobile eCard-API

Kristian Beilke

Diplomarbeit

Betreuer:	Dr. Wolf Müller
	Frank Dietrich
Gutachter:	Prof. Jens-Peter Redlich
	Dr. Manfred Paeschke

Berlin, den 4. Oktober 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACD	Application Capability Description
ACL	Access Control List
AID	Application Adentifier
AOD	Authentication Object Directory
APDU	Application Protocol Data Unit
API	Application Programming Interface
AR	Access Rule
ASN	Abstract Syntax Notation
ATS	Answer To Select, as defined in ISO/IEC 14443-3
BCD	Binary-Coded Decimal
BDr	Bundesdruckerei GmbH
BMI	Bundesinnenministerium
BSI	Bundesamt für Sicherheit in der Informationstechnik
CC	Common Criteria
CCD	Card Capability Description
CD	Certificate Directory
CDE	Cryptographic Data Element
CIA	Cryptographic Information Application
CIO	Cryptographic Information Object
CV	Card-Verifiable
DCOD	Data Container Object Directory
DDO	Discretionary Data Object
DF	Dedicated File
DH	Diffie-Hellman
DID	Differential-Identity
DNS	Domain Name System
DO	Data Object
DS	Data Set
DSA	Digital Signature Algorithm
DSI	Data Structure for Interoperability
EC	Elliptic Curve
ECC	European Citizen Card
ECP	Enhanced Client and Proxy
EF	Elementary File

FCP ..... File Control Parameters
FID ..... File Identifier
GCAL ..... Generic Card Access Layer
GCI ..... Generic Card Interface
HCI ..... Host Controller Interface nach ETSI TS 102 622
ICC ..... Integrated Circuit Card
IdP ..... Identity Provider
IFD ..... Interface Device
IPC ..... Inter-Process Communication
ISP ..... Internet Service Provider
J2ME ..... Java Mobile Edition
JSR ..... Java Specification Request
Le ..... expected length
MAC ..... Message Authentication Code
MF ..... Master File
NAT ..... Network Address Translation
NFC ..... Near Field Communication
OD ..... Object Directory
OID ..... Object Identifier
OSI ..... Open Systems Interconnection
PC/SC ..... Personal Computer/Smart Card
PKCS ..... Public-key Cryptography Standard
PrKD ..... Private Key Directory
PuKD ..... Public Key Directory
QES ..... Qualifizierte elektronische Signatur
RFID ..... Radio Frequency Identification
RFU ..... Reserved for Further Use
RP ..... Relying Party
RSA ..... Rivest-Shamir-Adleman
SAL ..... Service Access Layer
SAML ..... Security Assertion Markup Language
SDIO ..... Secure Digital Input Output
SFI ..... Short File Identifier
SICCT ..... Secure Interoperable ChipCard Terminal
SigG ..... Signaturgesetz (Gesetz über Rahmenbedingungen für elektronische Signaturen)
SKD ..... Secret Key Directory
SP ..... Service Provider
SSC ..... Send Sequence Counter
SSL ..... Secure Sockets Layer
SSO ..... Single Sign-On
SWP ..... Single Wire Protocoll nach ETSI TS 102 613
TLS ..... Transport Layer Security
TLV ..... Tag-Length-Value

UA ..... User Agent
UICC ..... Universal Integrated Circuit Card
UMTS ..... Universal Mobile Telecommunications System
URL ..... Uniform Resource Locator
USB ..... Universal Serial Bus
VPN ..... Virtual Private Network
WAP ..... Wireless Application Protocol
XML ..... Extensible Markup Language

Kapitel 1

Kurzdarstellung

Ziel der Arbeit ist es, eine Möglichkeit zu finden, auf einer mobilen oder anderweitig eingeschränkten Plattform die eID-Funktionalität des Personalausweises zu nutzen, wenn dort keine die eCard-API umsetzende Software zur Verfügung steht. Dabei können zwei wesentliche Ansätze verfolgt werden. Beim ersten soll die Kompatibilität zur eCard-API angestrebt werden. Für den zweiten wird diese Anforderung fallen gelassen. Gemeinsam ist beiden Ansätzen, dass versucht werden soll bestimmte Eigenschaften zu optimieren. Dazu gehört als erstes die benötigte Rechenleistung. Genauso wichtig ist eine Minimierung der nötigen Kommunikation vorrangig bezüglich Anzahl der Datenaustauschvorgänge, aber auch die Größe der ausgetauschten Nachrichten. Ein weiteres Ziel ist es, den Aufwand für eine Implementierung im Vergleich zu einer vollständigen Umsetzung der eCard-API Spezifikation verringern zu können. Am Ende sollen trotzdem die gleichen Sicherheitseigenschaften wie bei einem Einsatz z.B. der AusweisApp [2] gewährleistet werden.
Unter diesen Gesichtspunkten sollen bei der zu der eCard-API kompatibel ausgelegten Lösung die Machbarkeit einer eingeschränkten und einer verteilten Implementierung sowie einer Proxy-Lösung betrachtet werden. Die sich dabei als umsetzbar erweisenden Lösungen werden anschließend mit einem speziell für diesen Anwendungsfall konzipierten Protokoll verglichen, um zu einer Empfehlung zu gelangen, unter welchen Voraussetzungen welche Lösung am besten einsetzbar ist.

Kapitel 2

Einleitung

2.1 Einführung

Im Zuge der Einführung des neuen Personalausweises wird eine elektronische Infrastruktur für seine Nutzung im Internet etabliert. Ein wesentlicher Teil davon ist die eCard-API. Diese beschreibt eine Middleware, die die Benutzung von eCards auf eine einheitliche Weise ermöglicht. Im Fokus dieses Dokumentes steht dabei die Nutzung der eID-Funktionalität des neuen Personalausweises als erste in großer Zahl verfügbare eCard. Die eID-Funktion ist nicht auf einen hoheitlichen Einsatz beschränkt, sondern steht explizit auch für privatwirtschaftliche Zwecke zur Verfügung. Darin wird die Motivation für den breiten Einsatz der eCard-API gesehen. Daneben existiert der Wunsch nach einem mobilen Einsatz, welcher zusätzliche Nutzungsmöglichkeiten erschließt. Dadurch könnte die Akzeptanz des Personalausweises erhöht werden. Es sollen Anreize für Unternehmen und Behörden geschaffen werden weitere Dienste anzubieten, denn eine hohe Anzahl von nutzbaren Angeboten verbessert die Chancen für den Erfolg des Personalausweises.
Für den Einsatz der eCard-API auf mobilen oder ressourcenschwachen Geräten sind allerdings eine Reihe von Anpassungen der involvierten Spezifikationen wünschenswert, da eine Umsetzung der vollständigen TR-03112 [44] wie in Form der AusweisApp [2] auf solchen Geräten nur schwer zu realisieren ist.

2.2 Vorgehen

Zuerst einmal wird die eCard-API theoretisch und praktisch analysiert, um ihre Eigenschaften und Möglichkeiten herauszuarbeiten. Dies geschieht anhand der veröffentlichten Spezifikationen und Standards und zum anderen durch praktische Analyse der AusweisApp innerhalb eines Testszenarios. Die so gewonnenen Informationen sollten genügend Anhaltspunkte bieten, um die Machbarkeit der verschiedenen Lösungsansätze zu beurteilen. Soweit dies mit vertretbarem Aufwand möglich sein sollte und entsprechende Hardware zur Verfügung steht, wird ebenfalls eine beispielhafte Implementierung solch einer Lösung angestrebt. Den Abschluss bilden theoretische und möglicherweise praktische Analysen der Lösungen unter den eingangs genannten Gesichtspunkten sowie mittels sinnvoller Metriken

wie z.B. Ausführungszeit und Protokolloverhead. In einem weiteren Schritt, der aber nicht mehr inhaltlicher Teil der Arbeit ist, soll die Standardisierung einer Lösung durch das BSI angeregt werden.
Um zu diesem Ziel zu gelangen, werden zunächst die für das Verständnis notwendigen Standards und Spezifikationen erläutert. Danach wird auf dieser Basis die eCard-API analysiert. Anschließend werden mögliche Ansätze für eine mobilen Einsatz diskutiert und bewertet. Die sich daraus ergebenden Empfehlungen sollen als Vorlage für die Standardisierung dienen.

Kapitel 3

Identitätssysteme im Internet

In diesem Kapitel soll ein kurzer Überblick über die wichtigsten Ansätze für Authentifikationsysteme auf Basis digitaler Identitäten gegeben werden. Dazu werden die wichtigsten Spezifikation und Technologien in Kürze behandelt.

3.1 Authentifikationssysteme

Mit der zunehmenden Verlagerung und Ergänzung von Geschäftsprozessen und Dienstleistungen in das Internet gibt es einen Bedarf nach Authentifizierungsmöglichkeiten der Nutzer gegenüber Dienstleistern. Dazu müssen die Realprozesse in der Online-Welt abgebildet werden. In diesem Kontext gab es in den letzten Jahren einige Entwicklungen, wie z.B. die Liberty Alliance Spezifikationen [15], die das Ziel eines Internet-weiten und offenen Single-Sign-on-Standards verfolgt.
Generell gibt es bei der Authentifizierung mehrere Parteien. Im einfachsten Fall sind das genau zwei. Ein Nutzer, der im Besitz eines Identitätsnachweises ist, und eine Entität, die den Nutzer anhand seines Identitätsnachweises identifiziert. Der Einfachheit halber sei so eine Entität im Folgenden als Dienstanbieter bezeichnet, obwohl diese Beschreibung alle einen Identitätsnachweis konsumierenden Entitäten einschließt.
Generell gibt es in diesem Szenario das Problem des Vertrauens, welches einem Identitätsnachweis entgegen gebracht werden kann. In dem einfachen Beispiel einer Authentifizierung mit Benutzername und Passwort in der Online-Welt kann der Dienstanbieter nur ein beschränktes Vertrauen in den Identitätsnachweis haben, da Benutzername und Passwort in den meisten Fällen vom Nutzer selbst gewählt wurden und die damit verbundenen Daten unverifiziert sind. Damit ist also lediglich ein Pseudonym geschaffen, das ein Wiedererkennen möglich macht.
Um ein höheres Maß an Vertrauen gewährleisten zu können, kommt noch eine weitere Partei ins Spiel, ein Identitätsnachweisersteller, der einem Nutzer einen Identitätsnachweis ausstellt. Diesem Identitätsnachweisersteller muss von einem Dienstanbieter vertraut werden, damit dieses Vertrauen sich dann über den ausgestellten Identitätsnachweis auf den Authentifizierungsvorgang des Nutzers übertragen kann. Die Rolle eines Identitätsnachweiserstellers übernimmt in der Real-Welt üblicherweise ein Staat. Damit ist eine

so nachgewiesene Identität so vertrauenswürdig wie das Vertrauen in den ausstellenden Staat. In der Online-Welt kann die Rolle eines Identitätsnachweisersteller differenzierter sein. Je nach Umgebung und Einsatzzweck kann diese Rolle zum Beispiel dem Administrator eines Netzwerkes, einer staatlichen Stelle oder einer sonstigen dritten Partei zufallen. Damit einhergehend sind verschiedene Richtlinien und Prozeduren für das Erlangen eines Identitätsnachweises und dementsprechende Vertrauensgrade, die mit den verschiedenen Identitäten verbunden sind.
Für Nutzer und Dienstanbieter in der Online-Welt gibt es mehrere Gründe (Aufwand, Akzeptanz), die dafür sprechen, dass in bestimmten Situationen nicht jeder Dienstanbieter sein eigenes Identitätsnachweissystem aufbaut, sondern dass möglichst ein gemeinsames genutzt werden kann, sodass der Nutzer wie im realen Leben nur einen Identitätsnachweis benötigt. Dieses kann durch das Konzept der *Federated Network Identity* erreicht werden, welches dem Liberty Alliance Project zugrunde liegt. Beim förderierten Identitätsmanagement gibt es nur einen Ort, an dem die Identitätsinformationen gehalten werden. Wenn diese benötigt werden, können sie über festgelegte Standards geteilt werden. Der Nutzer kann dabei selber entscheiden, wem welche Informationen zugänglich gemacht werden, und welche Entitäten diese Information verknüpfen dürfen. Im Unterschied dazu, werden im nicht-förderierten Fall, die Informationen an jeder benötigten Stelle separat vorgehalten.
Den wichtigsten dieser Ansätze zum förderierten Identitätsmanagement ist eine bestimmte Struktur gemeinsam, wie sie Abbildung 3.1 andeutet. Es wird eine Dreiecksbeziehung zwischen dem Nutzer, Dienstanbieter und Identitätsnachweisersteller vorausgesetzt. Für eine Angleichung der verwendeten Begriffe, die in den referenzierten Standards und Spezifikationen gebraucht werden, seien im Folgenden der Nutzer mit U (User) oder UA (User Agent), der Dienstanbieter mit SP (Service Provider) oder RP (Relying Party), sowie der Identitätsnachweisersteller als IdP (Identity Provider) bezeichnet. In diesem Szenario kom-

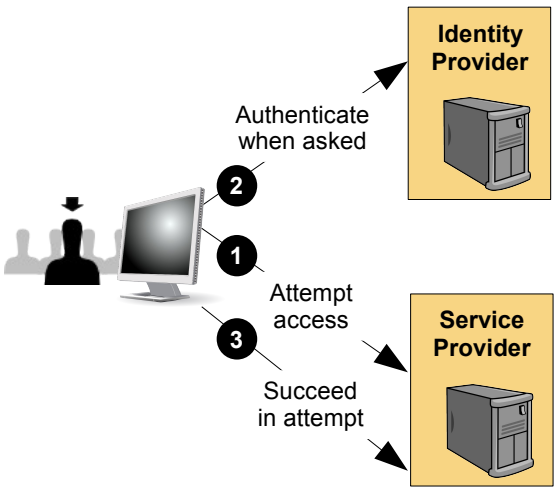

Abbildung 3.1: Auszug [41] Fig. 11: Dreiecksbeziehung

muniziert der Nutzer mit dem SP und dem IdP. Der SP hat zum Beispiel eine Ressource, auf die der Nutzer zugreifen will. Für diesen Zugriff verlangt der SP einen Identitätsnachweis in Form eines hier noch nicht näher spezifizierten Tokens. Diesen Nachweis stellt der IdP für den Nutzer aus, wenn dieser sich dort identifiziert hat. Dabei muss zwischen IdP und SP ein Vertrauensverhältnis bestehen, welches das Token legitimiert.
Das beschriebene Dreiecksmodell ist in verschiedenen Lösungen umgesetzt. Diese bauen allerdings auf teilweise verschiedenen Technologien und Standards auf. Über einige ausgewählte wird hier ein kurzer Überblick gegeben. Als wichtigster erscheint dabei die Security Assertion Markup Language (SAML).

3.2 SAML

SAML definiert ein XML-basiertes Framework für die Beschreibung und den Austausch von Authentifikations- und Autorisierungsinformationen. Diese Sicherheitsinformationen sind inhaltlich nichts weiter als Aussagen oder Zusicherungen (Security Assertions) über ein Subjekt. Dabei werden konkrete Regeln für das Anfragen, Erstellen, Versenden und Benutzen sowie die präzise Syntax dieser Assertions definiert.
SAML selbst besteht aus einer Reihe von Komponenten, die je nach Kombination verschiedene Anwendungsfälle zulassen, um Identitäts-, Authentifizierungs-, Authorisierungs- und Attributsinformationen auszutauschen. Die SAML-Kernspezifikation [55] beschreibt dabei den Aufbau der Assertions und der Protokollnachrichten, um Assertions auszutauschen. In so einer Assertion finden sich Aussagen über einen Nutzer (im SAML Kontext Principal gennant), welche der IdP (Asserting Party) bestätigt. Mit Hilfe dieser Protokollnachrichten werden üblicherweise Anfragen eines SP (Relying Party) mit einer Antwort, die eine Assertion enthält, von der Asserting Party beantwortet.
Die Art und Weise, wie diese Protokollnachrichten auf tiefer liegende Transportprotokolle (z.B. HTTP oder SOAP [50]) abgebildet werden, ist in der Bindings-Spezifikation [56] beschrieben.
Beim Entwurf von SAML standen bestimmte Anwendungsfälle im Fokus. Dazu gehören das Multi-Domain Web Single Sign-On und förderiertes Identitätsmanagement. Um die Nutzung von SAML weitestgehend kompatibel zu halten, gibt es Profile. Dies sind Einschränkungen bezüglich der Kombination der SAML-Komponenten Assertions, Protokolle und Bindings bezüglich eines Anwendungsfalles.
Assertions enthalten Aussagen einer Asserting Party über ein Subjekt. Dabei gibt es drei verschiedene Arten von Aussagen (Statements).

Authentification Statements: Diese enthalten mindestens wie und wann sich das Subjekt gegenüber der Asserting Party authentisiert hat.

Attribute Statements: Enthalten identifizierende Attribute über das Subjekt.

Authorization Decision Statements: Enthalten Informationen, dass das Subjekt berechtigt ist, etwas zu tun.

Folgende generalisierte Anfrage/Antwort-Protokolle sind definiert:

Authentication Request Protocol: Definiert, wie ein Principal eine Assertion mit Authentification Statements anfordert. Die dafür verwendete Anfrage enthält eine *AuthnRequest* Struktur. In der Antwort ist eine SAML Response Struktur, in der die angeforderten Authentication Statements enthalten sind.

Single Logout Protocol: Ermöglicht fast zeitgleiches Beenden aller aktiven Sitzungen eines Nutzers.

Assertion Query and Request Protocol: Definiert eine Menge von Anfragen, mit deren Hilfe SAML Assertions erhalten werden können.

Artifact Resolution Protocol: Ermöglicht, dass SAML Protokollnachrichten als Referenz (im Gegensatz zu der Nachricht selbst als Wert) übergeben werden. Diese Referenz ist ein sogenanntes Artifact. Der Empfänger des Artifacts nutzt dann das Artifact Resolution Protocol, um den Sender nach der eigentlichen Protokollnachricht zu fragen bzw. das Artifact zu dereferenzieren. Meistens wird das Artifact Resolution Protocol mit einem SOAP Binding durchgeführt.

Name Identifier Management/Mapping Protocol: SAML erlaubt es einem Principal Name Identifier zuzuordnen.

Die SAML Bindings spezifizieren genau, wie die verschiedenen SAML Protokollnachrichten auf darunter liegende Transportprotokolle abgebildet werden. SAML definiert dabei diese Bindings:

HTTP Redirect Binding: SAML Protokollnachrichten werden in HTTP redirect Nachrichten (Statuscode 302 oder 303) versendet.

HTTP POST Binding: Die Nachrichten werden im Base64 kodierten Inhalt eines HTML-Form-Controls transportiert. Beim Redirect Binding ist die Anzahl der in einer URL zu kodierenden Daten auf einen Browser-abhängigen Wert begrenzt. Das POST Binding hebt diese Begrenzung auf.

HTTP Artifact Binding: Beschreibt wie ein Artifact über HTTP transportiert wird.

SAML SOAP Binding: Die SAML Protokollnachrichten werden in SOAP Nachrichten über HTTP transportiert.

Reverse SOAP (PAOS) Binding: Definiert einen mehrphasigen Nachrichtenaustausch (PAOS Request-Response Message Exchange Pattern [14]), der es einem HTTP-Client ermöglicht auf SOAP Requests zu antworten.

SAML URI Binding: Mechanismus, um SAML Assertions durch URL-Auflösung zu erhalten.

Die SAML Profile spezifizieren wie die vorrangig genannten Assertions, Protocols und Bindings kombiniert werden, um für konkrete Anwendungsfälle eine hohe Interoperabilität zu gewährleisten. Für diese Arbeit relevant sind dabei nur zwei Profile.

3.2.1 Web Browser SSO Profile

Dieses Profil definiert wie mittels des Authentication Request Protocols und SAML Assertions eine Single Sign On Lösung mit einem unmodifizierten Web Browser realisiert wird. Nachrichten werden dabei mit einer Kombination aus dem HTTP Redirect und Post und optional auch dem Artifact Bindings ausgetauscht.
Obwohl in der Spezifikation noch zwischen IdP und SP initiiertem SSO unterschieden wird, wird hier nur letzteres betrachtet, da das IdP initiierte in keinem weiteren Szenario zur Anwendung kommt. Zu Beginn fordert der Nutzer mit seinen Browser beim SP eine Ressource an. Um diesen Zugriff zu gestatten, delegiert der SP den Browser des Nutzers zu einem IdP, damit er sich dort authentisiert. Dabei wird vom SP ein *AuthnRequest* generiert, welcher in dem Redirect enthalten ist. Der IdP erstellt eine Assertion, die die Authentifizierung des Nutzers beim IdP repräsentiert. Diese Assertion leitet der IdP innerhalb einer SAML *Response* über den Nutzer zurück zum SP, welcher die Assertions verarbeitet und abhängig davon den Zugriff dem Nutzer gewährt. In diesem Profil gibt es daher zwei SAML Protokollnachrichten. Einen Request vom SP zum IdP und eine Response vom IdP an den SP. Für den Request sind das HTTP Redirect, POST und Artifact Bindung zulässig. Für die Response nur die letzteren beiden. Daher ist für diesen Protokollnachrichtenaustausch ein asymmetrisches Binding möglich.
Der konkrete Ablauf bei einem HTTP Redirect/POST Binding ist in Abbildung 3.2 zu sehen. Auf den HTTP Request in Schritt 1 wird vom SP in Schritt 2 mit einem HTTP Redirect geantwortet, welcher das SAML AuthnRequest als Teil der URL enthält. Die Entscheidung, ob hier ein Redirect oder POST benutzt wird, ist konfigurationsabhängig und sollte auch die zu erwartende Größe des AuthnRequests berücksichtigen. Der Redirect veranlasst den Browser ein GET Request an den IdP mit dem AuthnRequest zu senden.
Obwohl in Schritt 3 und 4 die Authentifizierung des Nutzers beim IdP angedeutet wird, ist der tatsächliche Vorgang nicht Teil der SAML Spezifikation. Es wird keine Technologie, Protokoll oder Richtlinie vorgeschrieben, wie Identitäten bei IdPs erstellt werden und wie ein Nutzer sich anschließend beim IdP authentisiert. Es ist aber möglich, gewisse Kriterien über diesen Vorgang im AuthnRequest sowie in der folgenden Response anzugeben. Dazu sei auf den SAML Authentication Context [54] verwiesen. Darin werden Klassen von Authentifizierungsverfahren definiert, durch die es für den SP möglich wird abzuschätzen, welchen Vertrauensgrad er in den Authentifizerungsmechanismus des IdP haben kann.
Nach der Authentifizierung des Nutzers beim IdP wird in Schritt 5 als Antwort auf den HTTP GET Request aus Schritt 2 die Response an den Browser zurückgesendet. Diese enthält entsprechend dem HTTP POST Binding ein verstecktes HTML Formular mit der SAML Response und gewöhnlich "Scriptcode", um diese automatisch an den SP zu senden, was daher mit oder ohne Nutzerinteraktion in Schritt 6 passiert. Nachdem die SAML Response verarbeitet wurde, wird der Browser per Redirect auf die ursprünglich angefragte Ressource umgeleitet und kann darauf, abhängig von seinem Authorisationsstatus beim SP, zugreifen.

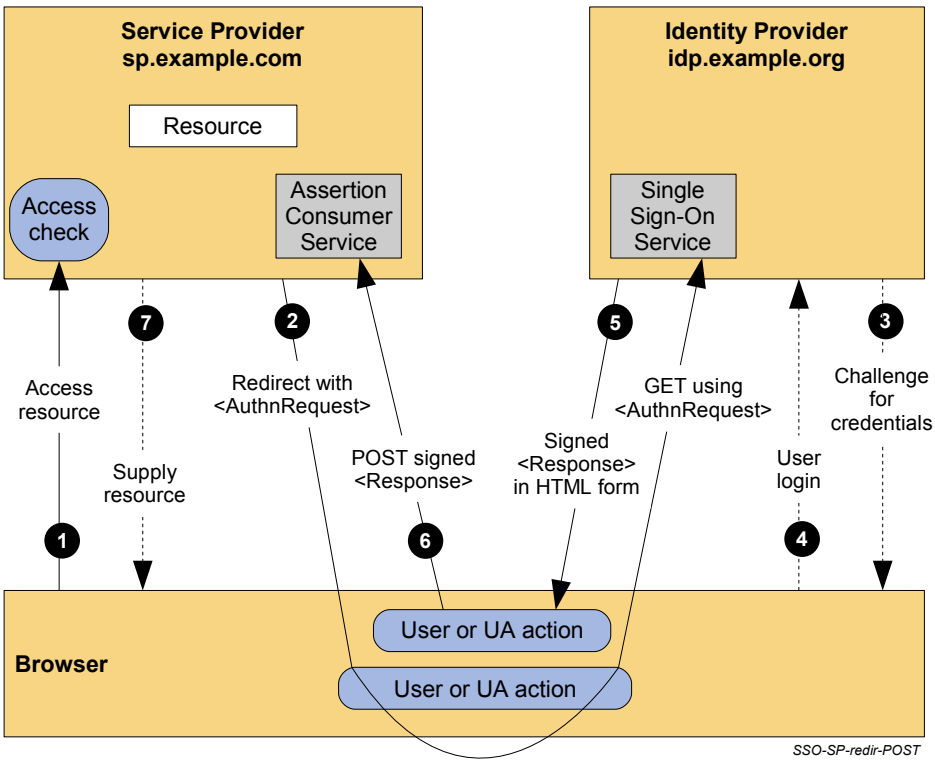

Abbildung 3.2: Auszug [41] Fig. 12: SAML SSO

3.2.2 Enhanced Client and Proxy (ECP) Profile

Das ECP Profil ist für einige spezielle Anwendungsfälle entworfen worden. Zum einen für Clients, die verglichen mit einem Standard-Webbrowser erweiterte Fähigkeiten haben und so z.B. selber bestimmen können, welchen IdP sie benutzen, indem sie die Nachricht vom SP in Schritt 2 selber verarbeiten und auswerten. Zum anderen für Clients, die hinter einem Proxy-Server sitzen (z.B. WAP-Gateway), der eingeschränkte Verbindungsmöglichkeiten (NAT) bietet. Deshalb wird ein PAOS und SOAP Binding benutzt, wie in Abbildung 3.3 ersichtlich.

Der tatsächliche PAOS Nachrichtenaustausch (Request-Response Message Exchange Pattern [14]) findet dabei nur zwischen Client und SP statt. Der AuthnRequest und die Response wird mit einem normalen SOAP-Aufruf an den IdP kommuniziert.

3.3 Personalausweis

Mit der Einführung des neuen Personalausweises im Rahmen der eCard-Strategie des Bundes soll es in Deutschland eine mit eID-Funktionalität ausgestattete Karte geben. Um die Nutzung dieser Funktionalität in Webanwendungen zu ermöglichen, ist eine Infrastruktur nötig, wie sie schematisch in Abbildung 3.4 dargestellt ist. Diese Infrastruktur ist in

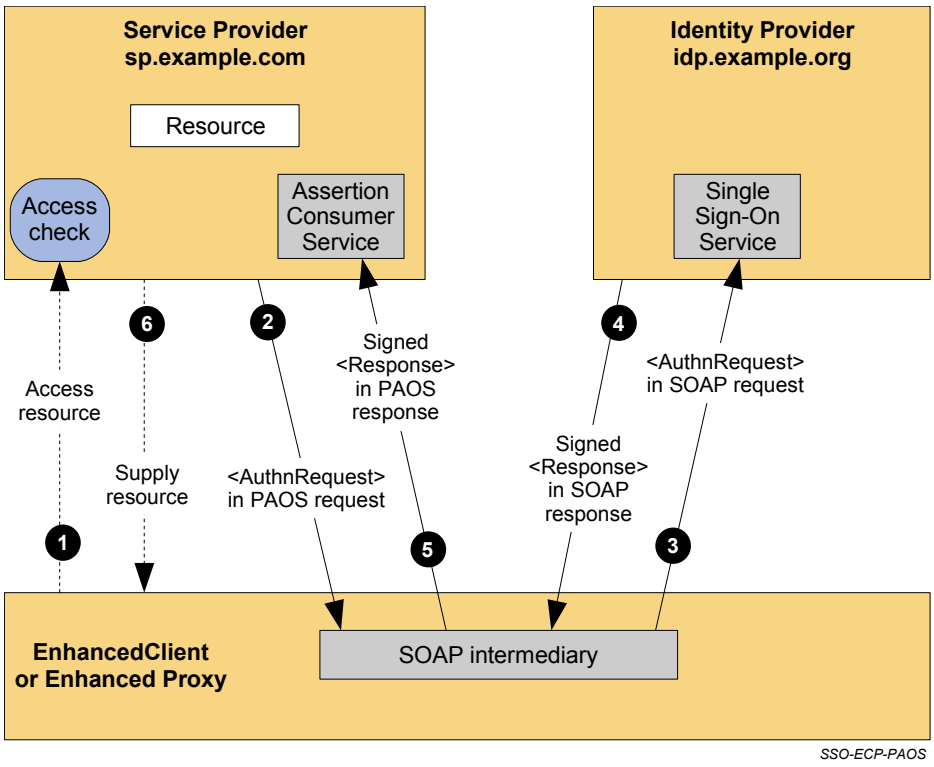

Abbildung 3.3: Auszug [41] Fig. 16: SAML ECP Profile

mehreren Technischen Richtlinien des BSIs [22] beschrieben und besteht aus den schon bekannten Komponenten, welche in Abbildung 3.4 skizziert sind. Dabei wird zwischen zwei Betriebsarten des eID-Servers [47], der dem IdP entspricht, unterschieden. Der IdP ist nicht zwangsweise eine eigene Entität, sondern logisch dem Dienstanbieter zugeordnet. Dieses ist in dem Konzept verankert, dass Dienstanbieter direkt ein Berechtigungszertifikat erhalten, mit dem sie sich gegenüber dem Ausweis authentisieren können. Daher ist der Vorgang des Auslesen direkt an den Dienstanbieter gebunden. Technisch ist der eID-Server in eigenständiges System, selbst wenn er beim Dienstanbieter betrieben wird. Die zweite Betriebsart des eID-Servers als ein eigenständiger eID-Service soll andeuten, dass es keine Notwendigkeit für den Betrieb eines eigenen eID-Servers bei einem Dienstanbieter gibt.
Eine Besonderheit ist die eCard-API genannte Komponente beim Nutzer. In SSO-Lösungen wie SAML werden der Ablauf und die Nachrichten beschrieben, die in so einem Szenario nötig sind. Die eigentliche Authentisierung des Nutzers beim IdP ist allerdings nicht Teil der Betrachtung. Diese Authentisierung wird durch eine Middleware auf Basis der eCard-API TR-03112 [44] realisiert. Konkret wird diese durch die AusweisApp umgesetzt.

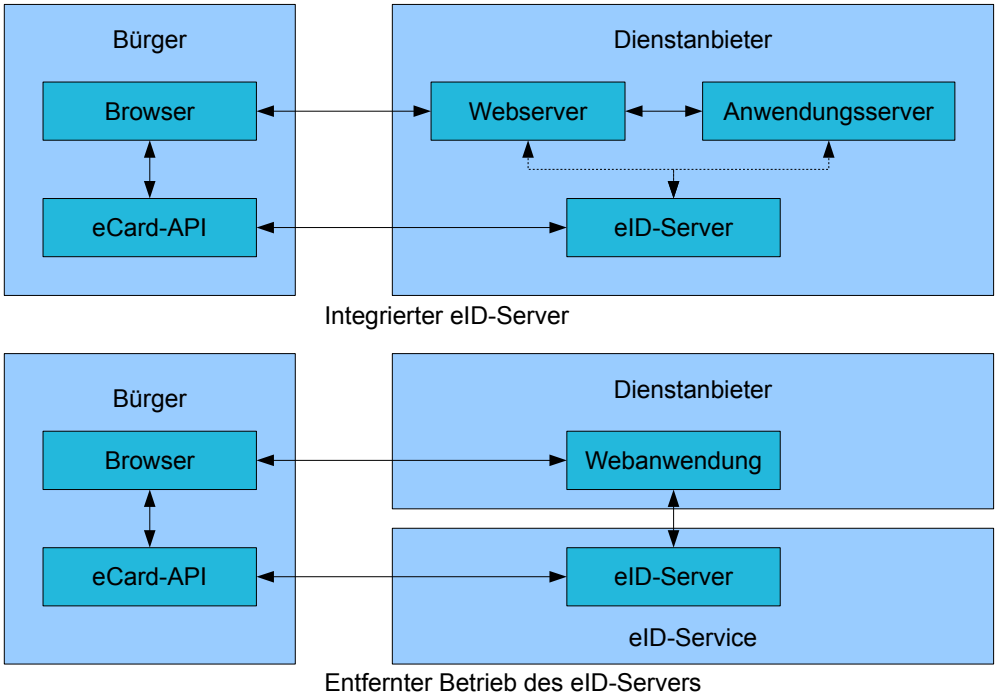

Abbildung 3.4: Kommunikationsbeziehungen Online-Authentisierung

3.3.1 Standards

Da sich die Komponenten teilweise noch in der Spezifikationsphase befinden und es jederzeit zu Änderungsanfragen kommen kann, können die folgenden Ausführungen gegebenenfalls gegenüber den endgültigen Komponenten abweichen. Unabhängig davon sind aber die Kernkomponenten schon soweit spezifiziert, dass grundlegende Änderungen unwahrscheinlich sind, so dass sich an den verwendeten Standards, welche die Grundlage der Infrastruktur bilden, nichts ändert.
Die Funktionalität wird durch Umsetzung vorhandener Standards erreicht. Der Austausch der Nachrichten zwischen den Entitäten erfolgt entsprechend dem SAML SSO Browser Profile mit Redirect und POST Binding. Die SAML Nachrichten sind kryptographisch auf XML-Ebene geschützt und zusätzlich sind die Kanäle auf Transportebene durch TLS [52] abgesichert.

3.3.2 eCard-API

Der eCard Strategie des Bundes zufolge soll durch das eCard-API-Framework, das eine Reihe von einfachen und Plattform-unabhängigen Schnittstellen umfasst, die Kommunika-

tion zwischen den jeweiligen Anwendungen und den unterschiedlichen Chipkarten, die in den Kartenprojekten der Bundesregierung ausgegeben oder genutzt werden, vereinheitlicht werden.
Zu wichtigsten Anforderungen an die eCard-API gehören das Bereitstellen einer einfachen und homogenen Schnittstelle, um in den verschiedenen Anwendungen eine einheitliche Nutzung der unterschiedlichen Chipkarten (eCards) zu ermöglichen. Dabei sollen möglichst geringe Einschränkungen für bereits bestehende Middleware-Infrastrukturen und kartenbasierte Applikationen entstehen. Geplant ist die Unterstützung der gängigsten Betriebssysteme (Windows, Linux, Mac OS) sowie eine Sprachen-unabhängige Implementierung der Schnittstellen (Nutzung von C, C++, C#, Java). Diese Middleware soll netzwerkfähig sein, um unterschiedliche Komponenten verteilt laufen lassen zu können. Im Kontext der Signaturfunktion der einzelnen Karten zählt die Evaluierbarkeit und Zertifizierungsfähigkeit (CC, QES, § 18 SigG) zu den Anforderungen. Damit einher geht das Ziel, möglichst den Austausch von ausführbarem Code, um neue Karten zu unterstützen, zu vermeiden, da sonst immer Rezertifizierungen nötig wären. Letztlich sollen die unterstützten Kartenterminal-Technologien abstrahiert werden, damit für Anwendungen der Zugriff transparent wird. So soll es keinen Unterschied machen, ob ein Leser mittels PC/SC [25] oder SICCT [39] angesprochen wird oder ob die Kommunikation über eine kontaktbehaftete oder eine kontaklose Schnittstelle stattfindet.
Die Software muss diese Anforderungen umsetzen und dabei das in TR-03112 [44] spezifizierte Interface nach außen anbieten. Die wichtigsten Details bezüglich Architektur und Funktionalität stecken allerdings in der Umsetzung der ISO/IEC 24727 [37] und der darauf aufbauenden CEN/TS 15480 ECC-3 [27].

Kapitel 4

ISO/IEC 24727

Die ISO/IEC 24727 [36] Spezifikation beschreibt Programmierschnittstellen für die Interaktion von externen Applikationen mit auf ISO 7816-4 basierenden Chipkarten. Diese Schnittstellen stellen generische Dienste für verschiedene Anwendungsbereiche bereit. Dabei soll Interoperabilität zwischen verschiedenen Anwendungsbereichen und Implementationen gewährleistet sein. Das heißt, dass jede Kartenapplikation, die zu ISO 24727 kompatibel ist, mit jeder zu ISO 24727 kompatiblen Client-Anwendung interoperabel ist. Im Ergebnis soll dies ermöglichen Client-Applikationen mit einem einheitlichen Ansatz zur Benutzung von Chipkarten zu entwerfen, obwohl die Chipkarten bedeutende Unterschiede haben. So soll es zum Beispiel keinen Unterschied machen, ob eine Karte kontaktbehaftet oder kontaklos oder aber logisch als Dateisystem oder Java Card [11] organisiert ist.
Im Folgenden werden die einzelnen Standards ISO/IEC 24727-1 bis 4 erläutert, welche die Architektur und Schnittstellen einer Middleware mit diesen Eigenschaften beschreiben. Daneben wird kurz auf die ISO/IEC 7816-15 [9] eingegangen, da darin einige Konzepte beschrieben sind, die sich in ISO/IEC 24727 wiederfinden. Zum Abschluss werden noch die Erweiterung durch CardInfo-Dateien und die Spezifikation CEN/TS 15480 ECC-3 [27] erläutert, weil auf dieser Basis die Architektur des eCard-API-Frameworks verständlich wird.

4.1 ISO/IEC 24727-1

In der ISO/IEC 24727-1 werden der grundlegende Aufbau und die Komponenten einer Middleware beschrieben. Dazu werden zwei wesentliche Schnittstellen definiert. Die Erste zwischen einer Client-Applikation und der Dienst-Schnittschnelle (Service Interface), um allgemeine Eigenschaften von Chipkarten zu abstrahieren, und die Zweite zwischen Service Interface und einer generischen Kartenschnittstelle (Generic Card Interface), die direkt mit der Karte kommuniziert. Daneben gibt es noch Schnittstellen, die es ermöglichen die einzelnen funktionalen Bestandteile in einer verteilten Architektur unterzubringen. Dazu gehören eine Verbindungsschnittstelle (Connectivity Interface) und das Trusted Channel Interface. Letzteres dient dazu, sichere Verbindungen zwischen den verteilten Komponenten herzustellen.

Abstrakt betrachtet, existieren grundsätzlich auf einer Karte eine oder mehrere Kartenapplikationen, die auf Service-Interface-Ebene mit Hilfe einer AID selektiert werden können. Diese Kartenapplikationen verwalten Datenmengen (Data sets). Jedes dieser Data Sets hat einen Namen, den die Client-Applikation in Erfahrung bringen kann. Der Zugriff auf diese Data Sets wird mittels Zugriffs-Kontroll-Listen (Access Control Lists, ACL) geregelt. Die einzelnen Zugriffsregeln (Access Rule, AR) in diesen Listen legen die Bedingungen (Security Conditions) für Aktionen (Actions) in den Data Sets fest. Wie diese Security Conditions innerhalb einer Kartenapplikation realisiert sind, ist kartenspezifisch und damit über das Kartenbetriebssystem realisiert. Wichtig ist hier, das es ein allgemeines Modell gibt, um diese Bedingungen abzubilden.
Um diese Abstraktionsebene erreichen zu können, ist es nötig, von einer ISO-24727 unterstützenden Chipkarte Informationen über deren Struktur bzw. Fähigkeiten zu erhalten. Dazu muss die Karte diese Informationen besitzen und über einen festgelegten Mechanismus nach außen bekannt geben können. Zu diesem Zweck gibt es bestimmte Dateien, die auf einer Karte vorhanden sein können. Dieses sind eine Karten-Fähigkeiten-Beschreibung (Card Capabilities Description, CCD) und für jede Applikation eine Anwendungs-Fähigkeiten-Beschreibung (Application Capabilities Description, ACD).
Der logische Aufbau einer Middleware nach ISO/IEC 24727 ist in Abbildung 4.1 angedeutet.

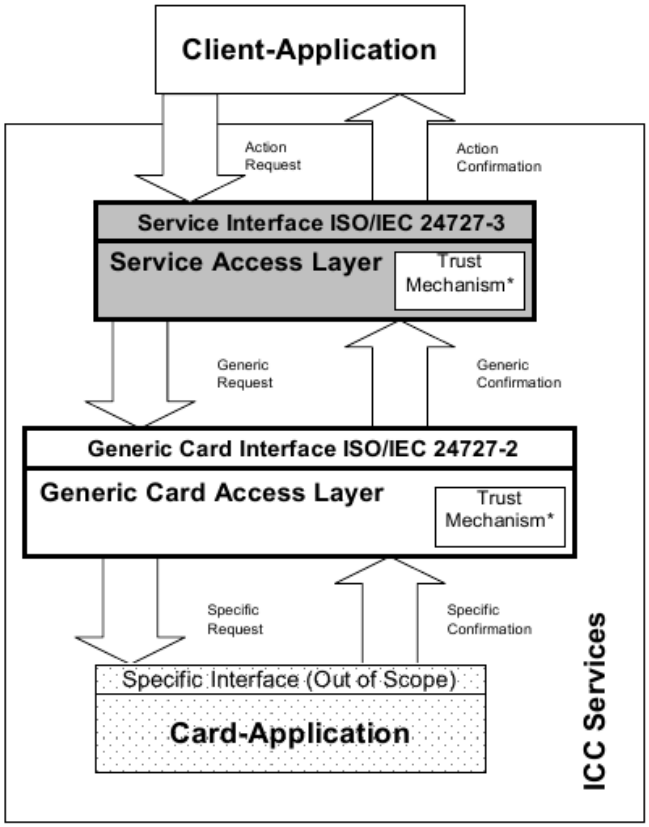

Abbildung 4.1: Auszug [36] Fig. 1: Logische Architektur von ISO/IEC 24727

Diese Middleware spricht auf der einen Seite mit der Client-Applikation mittels Action-Requests und Action-Confirmations und auf der anderen Seite mit Karten-Applikationen mittels APDUs. Dabei werden im Service Access Layer (SAL) diese Requests auf generische Karten Requests abgebildet, welche eine Untermenge von ISO/IEC 7816-4,8,9 [8] darstellen. Unter Umständen müssen diese generischen Requests im Generic Card Access Layer (GCAL) noch einmal in Karten-spezifische transformiert werden, bevor diese weiter an die Kartenapplikation gesendet werden.

4.2 ISO/IEC 7816-15

Um Schlüssel, PKI-Zertifikate und Authentisierungsobjekte auf ISO-7816 Chipkarten in einer kompatiblen Anordnung zu strukturieren, wurde ISO/IEC 7816-15 [9] spezifiziert. Diese basiert auf PKCS#15 [30] mit einigen Chipkarten-spezifischen Einschränkungen und Erweiterungen. Die Idee ist es, dass es eine Applikation auf der Karte gibt, die Informationen über die kryptographischen Funktionalitäten enthält. Die Syntax und das Format dieser Informationen sowie Mechanismen um diese Informationen zu finden werden in ISO/IEC 7816-15 beschrieben.
Dazu gibt es sogenannte Cryptographic Information Objects (CIO). Diese enthalten Informationen über Objekte auf der Karte, Beschreibungen wie diese Informationen verwendet werden sollen und wie man diese Informationen erhält. Dabei werden vier Klassen unterschieden: Schlüssel-, Zertifikats-, Datenobjekts- und Authentifikations-CIOs.
Entsprechend ihrer kryptographischen Unterteilung (z.B. private, öffentliche und geheime Schlüssel) werden diese Klassen noch weiter in Subklassen partitioniert. Konkrete CIOs besitzen Attribute, die von den ableitenden Klassen vererbt werden. Die eigentlichen Elemente (Cryptographic Data Element, CDE) auf die sich diese Informationen beziehen, können öffentlich oder privat sein. Für den Zugriff auf private CDEs muss üblicherweise eine Authentifizierung durchgeführt werden, welche wiederum in einem Authentication Information Object beschrieben ist.
Gewöhnlich ist ein CIO ein Elementary File (siehe ISO/IEC 7816-4 [33]) und befindet sich in einem Dedicated File (DF.CIA). Solch eine typische Anordnung ist in Abbildung 4.2 beschrieben. In EF.DIR sind die AIDs und Pfade der einzelnen Applikationen zu finden. Im DF.CIA sind die einzelnen CIOs zu finden. Vorgeschrieben ist aber nur das Vorhandensein von EF.CIAInfo und EF.OD (Object directory file). EF.CIAInfo kann kartenspezifische Daten wie Versionsnummer, Hersteller, Serialnummer usw. enthalten, EF.OD dagegen Referenzen auf sämtliche vorhandene CIOs.

4.3 ISO/IEC 24727-2

In ISO-24727-2 [40] wird das Generic Card Interface (GCI) spezifiziert. Auf Basis der von der Karte benötigten Informationen (CCD und ACD) lässt sich ein GCAL implementieren, der genau dieses GCI anbietet. Das GCI ist eine Untermenge von ISO/IEC 7816-4,8,9 [33, 32, 10]. Es werden z.B. keine SFI, logische Kanäle oder Dateien mit Record-Structure unterstützt. Falls die physische Karte so etwas benutzt, müssen entsprechende GCI APDUs

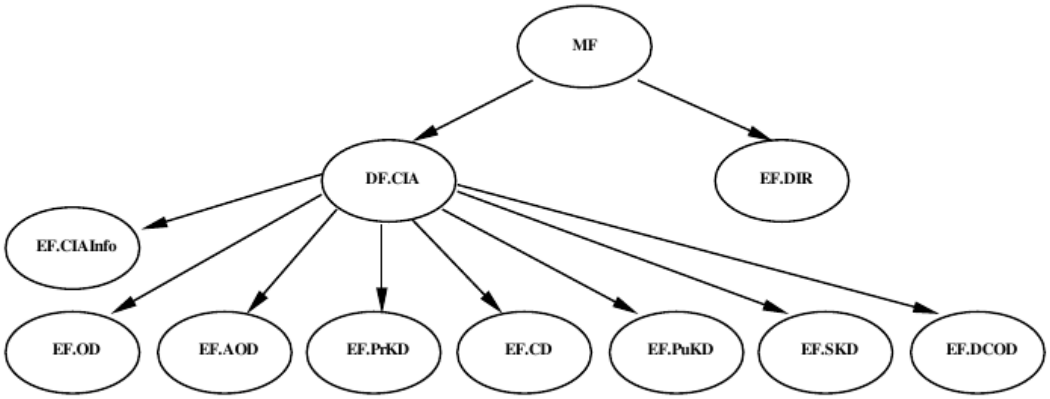

Abbildung 4.2: Auszug [9] Fig. 3: Beispielhafter Inhalt von DF.CIA

vom GCAL auf die kartenspezifischen APDUs abgebildet werden. Zusätzlich existieren spezielle GCI APDUs (CLA=0xFF), die direkt vom GCAL verarbeitet werden, also nicht an die Karte gesendet werden, wie z.B. *Reset* oder *ListReaders*. Dementsprechend gibt es spezielle Statuswörter, die direkt vom GCAL zurückgegeben werden, um dort aufgetretene Fehler zu signalisieren.

4.3.1 Prozedurale Elemente

Für die tatsächliche Abbildung von GCI APDUs auf kartenspezifische dienen so genannte Prozedurale Elemente. Sie sind ein Übersetzungscode für Requests und Confirmations am GCAL. Diese Elemente befinden sich direkt auf der Karte oder werden durch eine URL referenziert.

4.3.2 Capability Descriptions und Bootstrapping

Aufbauend auf ISO-7816-15 beschreiben so genannte Capability Descriptions die Karte und ihre kryptographischen Eigenschaften. Das ist zum einen die Card Capabilities Description (CCD) und zum anderen für Kartenapplikationen eine Application Capabilities Description (ACD). Zusätzlich muss eine Alpha-Applikation existieren, entweder auf der Karte oder im SAL simuliert, in welcher die CCD selektierbar ist. In einer CCD können eine Liste aller verfügbaren Kartenapplikationen und prozedurale Elemente für die Abbildung von GCI APDUs auf kartenspezifische enthalten sein. In den ACDs können neben der Beschreibung der verfügbaren Dienste in einer Applikation wieder prozedurale Elemente vorhanden sein. Damit ist es möglich Informationen, die für alle Kartenapplikation gelten, zentral in der CCD zu hinterlegen. Spezifische und damit nur eine Kartenapplikation betreffende Informationen befinden sich in der entsprechenden ACD.
Um die notwendigen Capability Descriptors zu finden, gibt es ein festes Schema. Allerdings ist dieses aufgrund von vielen möglichen Sonderfällen recht komplex. Prinzipiell wird zuerst die CCD eingelesen. Daraus oder aus dem EF.DIR wird die Liste der verfügbaren Kartenapplikationen erstellt, zu denen wiederum die zugehörigen ACDs gelesen werden.

Der erste Schritt beim Zugriff auf eine Karte ist immer dieser als Bootstrapping bezeichnete Vorgang, welcher der ISO/IEC 24727 konformen Middleware die Descriptions und damit die Kartenstruktur bekannt macht.

4.4 ISO/IEC 24727-3

Der dritte Teil der ISO/IEC 24727 beschreibt die Applikationsschnittstelle (Service interface). Die Hauptaufgabe der Schicht (Service Access Layer (SAL)), die diese Schnittstelle implementiert, ist die Interpretation von Service Anfragen (Action Requests) und deren Umwandlung in eine Menge von generischen (GCI) APDUs. Die Action Requests und Action Confirmations sind dabei unabhängig von konkreten Programmiersprachen spezifiziert. Abstrakt betrachtet gibt es zwischen einer Client-Applikation und einer Kartenapplikation eine Verbindung, in der eine Sitzung existiert, die einen Sicherheitskontext zu der Verbindung addiert. Diese Verbindung und Sitzung wird als Card-Application-Path bezeichnet. Durch diesen Card-Application-Path spricht die Client-Applikation mit der Kartenapplikation über die ISO-24727-3 Schnittstelle. Das konzeptionelle Modell, dem Chipkarten entsprechen, sieht wie in Abbildung 4.3 aus. Eine Kartenapplikation besteht aus einer Menge

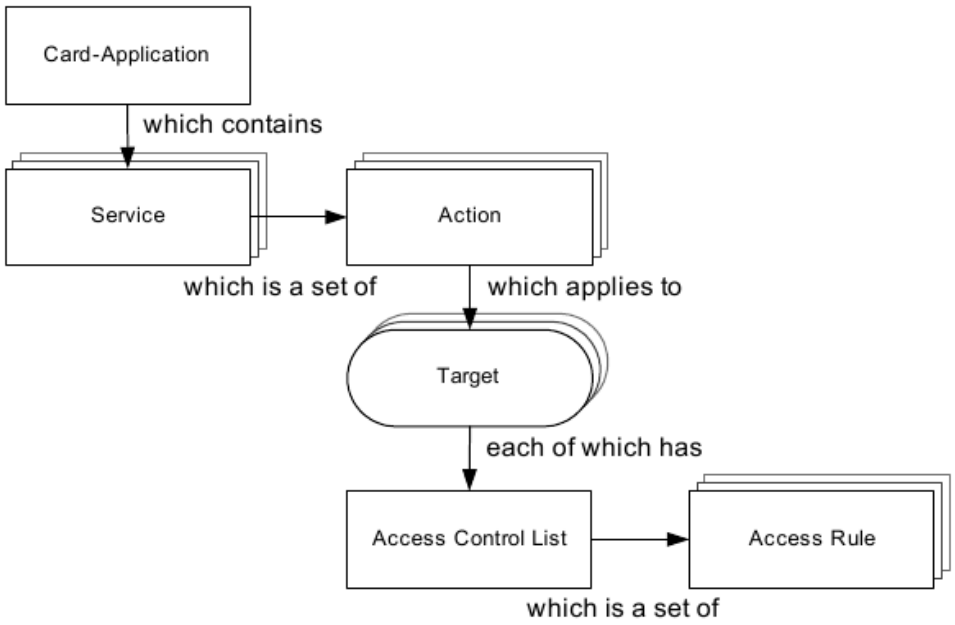

Abbildung 4.3: Auszug [37] Fig. 2: Modell der Entitäten und Beziehungen

von Zielen (Targets) und Diensten (Services). Die Dienste enthalten wiederum einen Menge von Aktionen (Actions). Eine Action operiert immer auf einem Target, dessen Typ von der Action abhängt. Der Typ kann z.B. die Kartenapplikation sein, ein Data-Set, eine Differential-Identiy (DID) oder ein Target selbst. Ein Target besitzt eine Zugriffskontrollliste (Access Control List, ACL), die aus einer Menge von Zugriffsregeln (Access

Rule, AR) besteht. Eine AR verbindet eine Action eines Service mit einer Sicherheitsbedingung (Security Condition). Eine Security Condition ist ein boolescher Ausdruck aus DID-Authorisationszuständen.
Bevor eine Client-Applikation auf einen Service zugreifen kann, muss die Information darüber, welche Kartenapplikation mit welchen Diensten verfügbar sind, vorhanden sein. Daher ist es notwendig mittels eines *Initialize* Aufrufes diese Informationen in der ISO/IEC 24727 Implementierung verfügbar zu machen. Infolge dessen wird über das GCI der Bootstrapping Mechanismus angestoßen.
Die am Service Interface angebotenen Services beinhalten Actions zur Verbindungs- und Sitzungserstellung sowie -beendigung, zur Verwaltung von Kartenapplikationen, Data-Sets und DIDs, sowie zur Kryptographie und Manipulation von ACLs.

4.4.1 Sicherheitmodell

Das Sicherheitsmodell von ISO-24727 basiert auf in den ARs festgelegten Security Conditions. Ein gemeinsamer Sicherheitszustand zwischen Client-Applikation und Kartenapplikation wird durch das erfolgreiche Ausführen von benötigten Protokollen hergestellt. Dadurch erfüllte Security Conditions autorisieren wiederum das Ausführen von Actions auf Targets, die durch entsprechende ARs geschützt sind.
Um einen Sicherheitszustand herzustellen, soll ein Differential Authentication Mechanism benutzt werden. Eine Differential Identity (DID) besteht aus einem Namen, Authentisierungsprotokoll, einem Marker, einem impliziten Scope und einem optionalen Qualifier. Das Authentisierungsprotokoll ist ein Object Identifier (OID), der auf ein Protokoll verweist, welches der ISO/IEC 24727 bekannt sein muss. Ein erfolgreicher Durchlauf dieses Protokolls setzt den Authentisierungszustand dieser DID in der Kartenapplikation auf *TRUE*.
Der implizite Scope einer DID ist je nach Definitionsort global (in der Alpha-Applikation) oder lokal. Der Qualifier kann mittels OID auf eine externe Spezifikation verweisen. Der Marker beinhaltet oder verweist auf Informationen innerhalb der Kartenapplikation, die benötigt werden, um das Authentication Protocol durchzuführen. Seine Struktur ist daher vom Protokoll abhängig. Beispiele für einen Marker sind eine PIN, ein Passwort oder ein Schlüssel.
Einige Standardauthentisierungsprotokolle sind ein ISO/IEC 24727-3 schon enthalten, es können aber auch weitere aufgenommen werden. Dazu sei auf ISO/IEC 24727-6 [43] verwiesen.

4.4.2 DIDAuthenticate

Die Funktion *DIDAuthenticate* hat eine besondere Bedeutung. Sie wird zur Ausführung von Authentisierungsprotokollen von DIDs gerufen, und ist damit essentiell für das Herstellen von Sicherheitszuständen in einer Kartenapplikation. Ihre Parameter richten sich nach der referenzierten DID.

```
OUT    ReturnCode              DIDAuthenticate(
IN     ConnectionHandle                connectionHandle,
IN     DIDScope                        didScope,
```

```
IN     DIDName                          didName,
IN/OUT DIDAuthenticationData            authenticationProtocolData,
IN     ConnectionHandle                 samConnectionHandle
                                );
```

Dabei kann es notwendig sein, mehrere Aufrufe für dieselbe DID, referenziert durch *didName*, vorzunehmen, um ein Protokoll vollständig durchzuführen. Der Parameter *authenticationProtocolData* kapselt dabei die vom Protokoll abhängigen Ein- und Ausgabedaten. Seine Struktur muss daher in der Protokollspezifikation beschrieben werden.

4.5 ISO/IEC 24727-4

Im vierten Teil wird die Sicherheitsarchitektur des Middleware-Stacks beschrieben. Dies beinhaltet die Herstellung eines sicheren Ende-zu-Ende-Kanales zwischen der Kartenapplikation und der Client-Applikation. Dabei wird auf die verschiedenen Einsatzszenarien eingegangen, die sich aus einem verteilten Einsatz ergeben. Die Middleware ist immer zwischen der Client-Applikation und der Karte positioniert. Ist diese Middleware verteilt, besteht eine Man-in-the-Middle-Angriffs-Problematik, woraus sich der Bedarf nach abgesicherten Kanälen zwischen den einzelnen Komponenten ergibt.
ISO/IEC 24727-4 beschreibt die gültigen Stack-Konfigurationen und deren Instanziierung. Diese Konfigurationen reichen von einem Full-Networked-Stack (Abbildung 4.6), bei dem alle Komponenten verteilt sind, bis zu einem Loyal-Stack (Abbildung 4.7), in dem alle Komponenten auf derselben Plattform sind und damit kein explizites Absichern der Kanäle benötigt wird. Außerdem werden die Trusted Channel Schnittstelle zum Aufbauen von gesicherten Verbindungen und die Interface Device API für die Kartenleser-Abstraktion spezifiziert.
Abbildung 4.4 zeigt die generischen Elemente des Protokollstacks. Es gibt sechs zulässige Stack-Konfigurationen, die im Folgenden kurz erklärt werden.
Bei der Verteilung einer Komponente, die eine bekannte Schnittstelle implementiert, wird ein Proxy-Agent-Paradigma angenommen. Dies erlaubt die Verteilung einer Schnittstelle, indem alle Requests samt ihren Parametern einer Prozedur, die Marshalling genannt wird, unterworfen werden. Dies ist nichts anderes als eine Serialisierung der Daten. Abbildung 4.5 erklärt die Bedeutung der graphischen Elemente, mit denen die Stack-Konfigurationen dargestellt werden.

4.5.1 Full-Networked-Stack

In dieser Konfiguration, Abbildung 4.6, ist jede Komponente des Stacks von jeder anderen getrennt. Die Kommunikation der Komponenten findet über ein Netzwerk statt. Daher ist zwischen allen Komponenten eine separate kryptographische Absicherung der Kanäle notwendig.

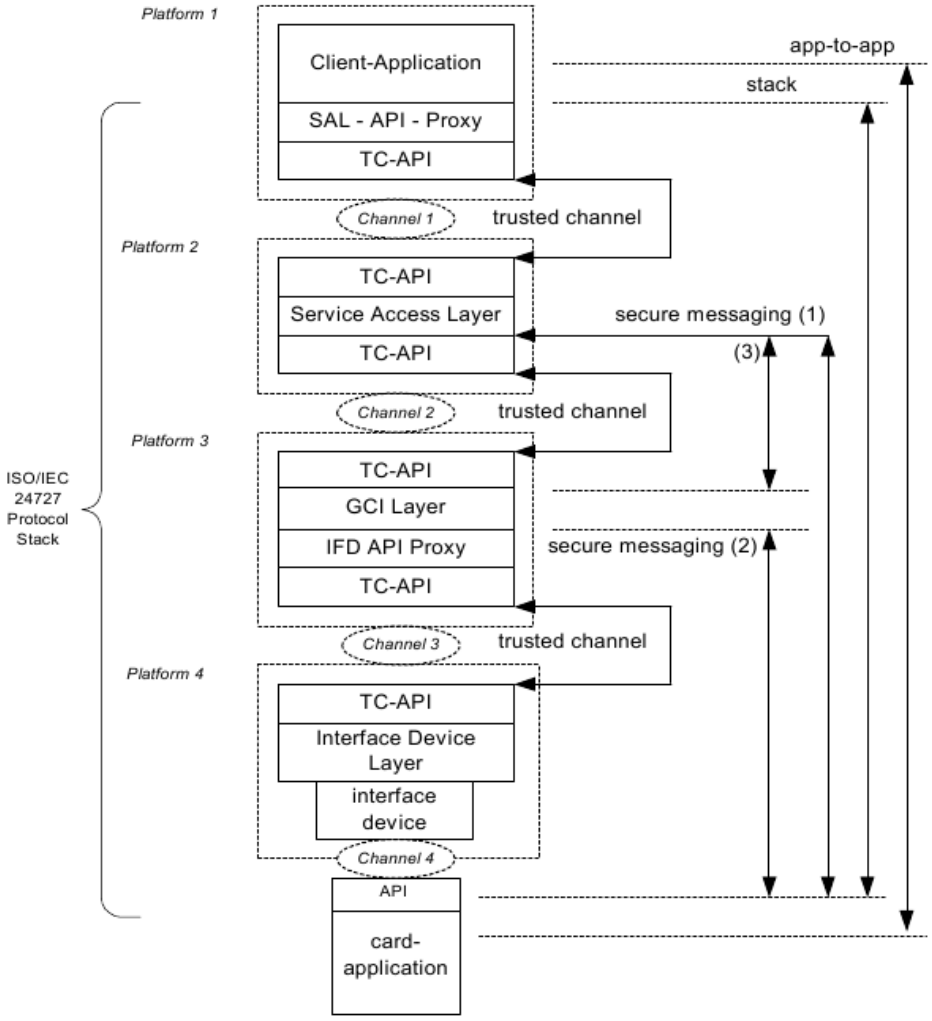

Abbildung 4.4: Auszug [38] Fig. 2: Generische Elemente des ISO/IEC 24727 Stacks

4.5.2 Loyal-Stack

In der Konfiguration Loyal-Stack sind alle Kanäle vertrauenswürdig. Lediglich der Kanal über das Interface-Device zur Kartenapplikation kann zusätzlich durch Secure-Messaging abgesichert werden, wie in Abbildung 4.7 zu sehen.

4.5.3 Opaque-ICC-Stack

In Abbildung 4.8 befinden sich die Instanzen des IFD und des GCIs in derselben Komponente wie die Karte. Ein Trusted Channel muss über die TC_API (Trusted Channel API) aufgebaut werden.

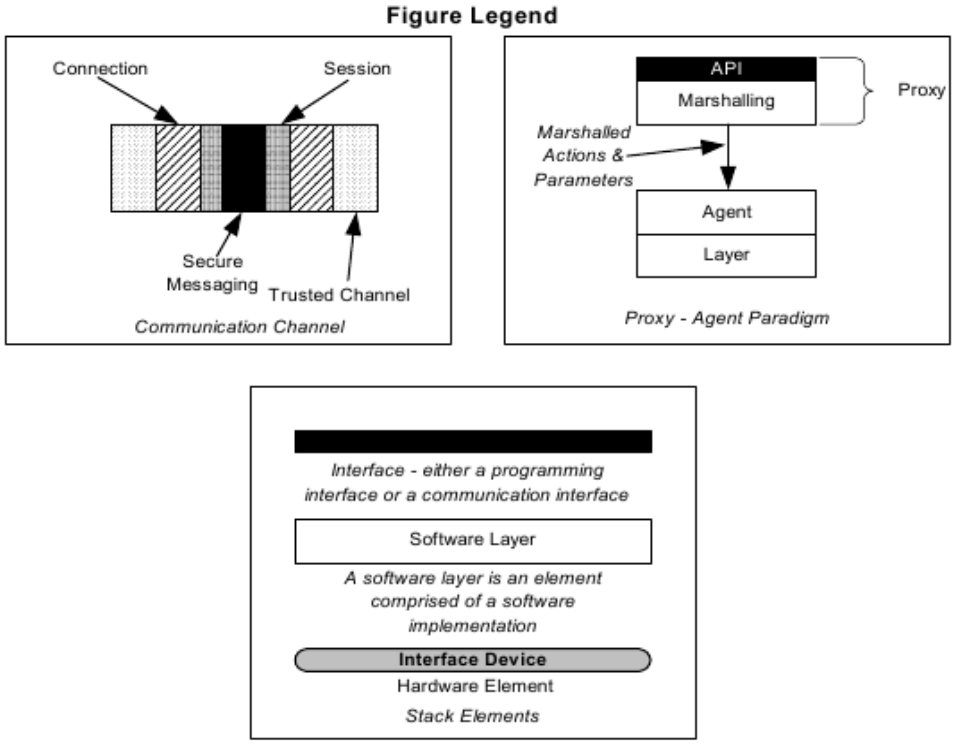

Abbildung 4.5: Auszug [38] Fig. 3: ISO/IEC 24727 Stack Legende

4.5.4 Remote-Loyal-Stack

Im Gegensatz zum Opaque-ICC-Stack ist hier die Trennung näher an der Client-Applikation. Damit sind alle Implementierungen der ISO/IEC 24727 Komponenten von der Client-Applikation getrennt. Nur ein SAL-Proxy kommuniziert über einen Trusted Channel mit dem Stack in Abbildung 4.9.

4.5.5 ICC-Resident-Stack

In dieser Konfiguration befindet sich ein kompletter ISO/IEC 24727 Stack innerhalb der Karte. Das bedeutet, dass die Chipkarte nicht über APDUs nach ISO/IEC 7816 [8] angesprochen wird, sondern programmatisch nach ISO/IEC 24727-3 mit Funktionen aus dem SAL. Dies könnte z.B. über Webservices realisiert sein.

4.5.6 Remote-ICC-Stack

Bei der Remote-ICC-Stack Konfiguration ist der gesamte ISO-24727 Stack auf derselben Plattform wie die Client-Applikation. Nur für die IFD-Schnittstelle gibt es ein Proxy-Interface, wie in Abbildung 4.10 dargestellt. Dies entspricht quasi einem entfernten Kartenleser, da die IFD-Interface Befehle sich direkt auf PC/SC [25] umsetzen lassen.

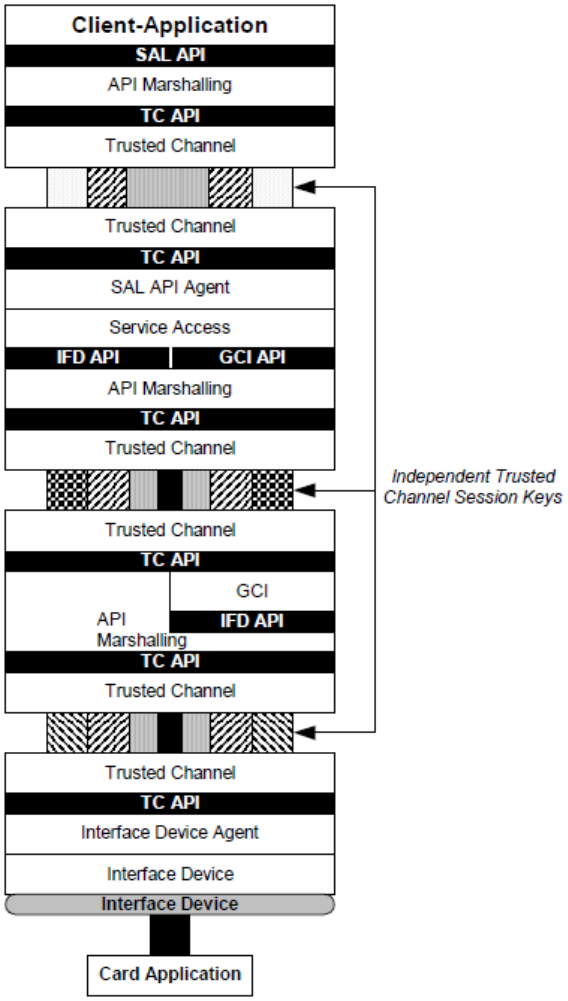

Abbildung 4.6: Auszug [38] Fig. 4: ISO/IEC 24727 Full-Network-Stack

4.5.7 Zusätzliche Schnittstellen

Die einzelnen Stackkonfigurationen müssen vor dem Zugriff durch Client-Applikationenen initialisiert werden. Die Sicherheitseigenschaften der Konfiguration werden der Client-Applikation durch einen *CardApplicationPath*-Request und -Response bekannt gemacht.

Trusted Channel Interface

Das Trusted Channel Interface dient dazu, zwischen verteilten Komponenten eine sichere Verbindung über ein Netzwerk herzustellen. Dabei muss mindestens das TLS [52] Protokoll unterstützt werden.

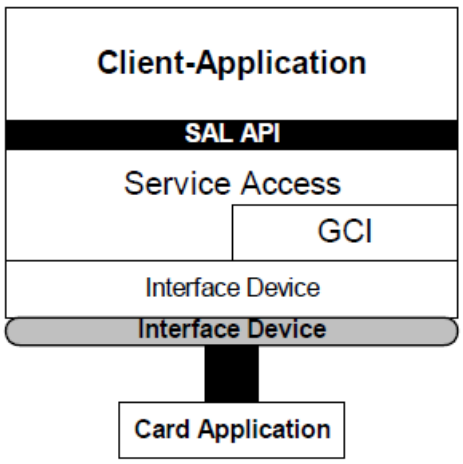

Abbildung 4.7: Auszug [38] Fig. 5: ISO/IEC 24727 Loyal-Stack

Interface Device API

Die Interface Device API besteht aus einer Gruppe von Requests, die sehr stark an PC/SC [25] angelehnt sind. Es gibt Requests bezüglich des Terminals, der Slots und des Benutzers. Zugriff auf die Interface Device API soll es nur über den SAL oder den GCAL geben.

4.6 CardInfo Dateien

Entsprechend ISO/IEC 24727-2 [40] muss auf einer Chipkarte die Information über ihre Fähigkeiten vorhanden sein, um sie als ISO/IEC 24727 Karte zu benutzen. Dies verhindert die Einbindung von schon existierenden Chipkarten ohne diese Information. Als Ausweg wird in [59] vorgeschlagen, diese Information in externen, XML-basierten CardInfo-Dateien bereitzustellen.
Dazu ist es nötig, dass die ISO/IEC 24727 Middleware die Karte identifizieren kann. Wenn die nötigen Erkennungsmerkmale im Programmcode der Middleware wären, würde das Hinzufügen von Unterstützung weiterer Chipkarten Codeveränderungen bedeuten. Die Lösung mit CardInfo-Dateien, welche als externe CCD und ACD betrachtet werden können, die einfacher austauschbar oder anpassbar sind, hat diesen Nachteil nicht. CardInfo Dateien müssen es erlauben eine Chipkarte zu erkennen, um die passenden GCI-Abbildungen anwenden zu können, und sie müssen die Informationen für das Abbilden von generischen APDUs auf kartenspezifische enthalten. Eine beispielhafte Abbildung ist in Abbildung 4.11 zu sehen. Um eine flexible Kartenerkennung mit CardInfo-Dateien zu realisieren, können diese Daten z.B. in einem Repository hinterlegt sein, so dass eine ISO/IEC 24727 Middleware diese Dateien lokal periodisch aktualisieren kann. Solange die notwendigen Protokolle, welche die DIDs benötigen, der Middleware bekannt sind, lassen sich so beliebige weitere Chipkarten dynamisch unterstützen.
Diese Erweiterung wird im eCard-API-Framework wie auch in der CEN/TS 15480 ECC-3

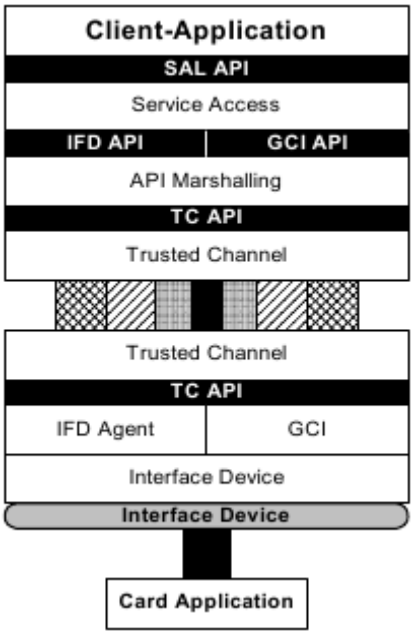

Abbildung 4.8: Auszug [38] Fig. 6: ISO/IEC 24727 Opaque-ICC-Stack

benutzt.

4.7 CEN/TS 15480 ECC-3

Obwohl die Spezifikation von CEN/TS 15480 ECC-3 [42] auf ISO/IEC 24727-3 [37] und TR-03112 [44] beruht, wird sie in dieser Arbeit noch vor TR-03112 beschrieben, da sie in vielen Details der Architektur ausführlichere Informationen zur Verfügung stellt.
CEN/TS 15480 ECC-3 [27] beschreibt die European Citizen Card (ECC), welche elektronische Identitätskarten (eID) umfasst. Dabei liegt der Fokus auf entfernten zivilen Dienstleistungen, die durch solche Chipkarten ermöglicht werden sollen. Dazu wird eine Middleware auf Basis von ISO/IEC 24727-3 beschrieben. Die grundlegende Struktur dieser Middleware ist in Abbildung 4.12 dargestellt. Dabei wird deutlich, dass im Unterschied zu Abbildung 4.1 dem GCI eine geminderte Bedeutung zukommt, da der SAL hier direkt mit dem IFD-Interface kommuniziert. Dies resultiert daraus, dass es meistens erforderlich ist, einen Ende-zu-Ende gesicherten Kanal zwischen Client-Applikation und Kartenapplikation zu haben, über den von der Client-Applikation mit Secure Messaging geschützte APDUs versendet werden. Damit wäre aber die Abbildungsfunktion des GCAL von generischen APDUs auf kartenspezifische APDUs nicht mehr möglich, da sonst der GCAL im Besitz der SM-Schlüssel sein müsste. Dies schränkt die möglichen Konfigurationen ein. So sind nur Loyal-Stack, Remote-Loyal-Stack und Remote-ICC-Stack sinnvoll. Die Aufgabe des GCALs ist damit auf das Bootstrapping reduziert.
Wenn eine Client-Applikation auf einem entfernten Server ist und die ECC-Karte bei einem

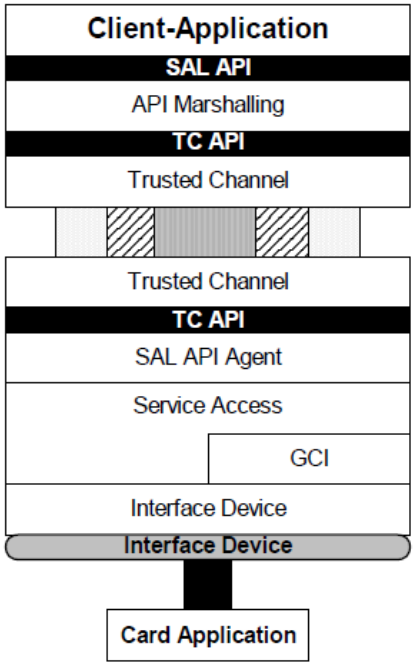

Abbildung 4.9: Auszug [38] Fig. 7: ISO/IEC 24727 Remote-Loyal-Stack

Nutzer lokal, werden die Remote-Loyal-Stack als auch die Remote-ICC-Stack Konfigurationen benutzt. In diesem Szenario ist es notwendig, dass die Client-Applikation gesichert direkt mit der Kartenapplikation kommuniziert, was dem Remote-ICC-Stack entspricht. Ohne diese Sicherung durch SM wird die Remote-Loyal-Stack Konfiguration verwendet. Zusätzlich soll ein lokaler Client einen Authorisierungsprozess mit der Karte durchführen können, welches wiederum eine Loyal-Stack Konfiguration darstellt. Daraus folgt, dass auf beiden Systemen der volle ECC-3 Middleware Stack zur Verfügung steht und dass eine Mischung der genannten Konfigurationen angewendet werden muss. Diese ist in Abbildung 4.13 zu sehen. Zu beachten sind die einzelnen Verbindungen 1 bis 4. Verbindung 1 dient zur anwendungspezifischen Kontaktaufnahme der lokalen Client-Applikation zum eService und wird nicht weiter betrachtet. Um über den SAL zu kommunizieren, wird Verbindung 2 hergestellt. Damit der eService direkt mit der Kartenapplikation mittels APDUs kommunizieren kann, wird Verbindung 3 benutzt. Diese erlaubt damit eine Ende-zu-Ende authentisierte und vertrauliche Verbindung. Die 4. Verbindung sichert die Kommunikation über die Proxies. Sie ist der Trusted Channel zwischen den verteilten Komponenten der Middleware. Verbindung 2 und 3 benutzen also beide den Kanal, der durch Verbindung 4 geschaffen wurde. Dieser Trusted Channel ist im ECC Falle, in dem die Kommunikation über Webservices stattfindet, mittels TLS [52] realisiert.
Der in ISO/IEC 24727-2 als Bootstrapping bezeichnete Prozess holt sich von der Karte die benötigten Daten. Außerdem wird für Legacy-Chipkarten oder auch einfach nicht ISO/IEC 24727-2 konformen Karten die Einbindung mittels XML-basierten CardInfo Dateien un-

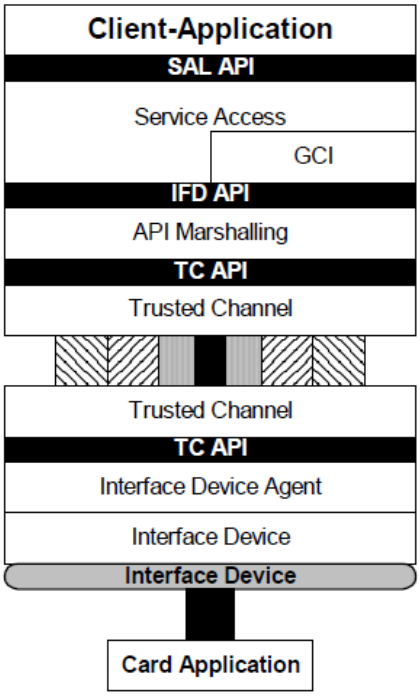

Abbildung 4.10: Auszug [38] Fig. 9: ISO/IEC 24727 Remote-ICC-Stack

terstützt. Der verwendete Kartenidentifikationsprozess ist in Abbildung 4.14 angedeutet. Daneben ist es möglich, dass eine Chipkarte nur einen reduzierten CCD enthält, der auf eine URL verweist, wo eine passende CardInfo Datei zu finden ist beziehungsweise wo ein CardInfo-Repository mit solchen Dateien verfügbar ist.
Das Ausführen von Authentisierungsprotokollen nach ISO/IEC 24727 [37] unterliegt einigen Regeln. Sind der eService und der SAL in der gleichen Umgebung, dann soll ein mittels *DIDAuthenticate* gestartetes Protokoll komplett im SAL ablaufen und sämtliche Komplexität dorthin verlagern. Sind der eService und der SAL nicht in der gleichen Umgebung, kann es nötig sein, dass mehrere Aufrufe von *DIDAuthenticate* nötig sind, zwischen denen der eService selbst einige Authentisierungsdaten bearbeitet.
Eine wichtige Erweiterung gegenüber ISO/IEC 24727-4 [38] ist die Fähigkeit der *Transmit* Funktion im IFD-Interface, eine ganzen Stapel von APDUs gleichzeitig entgegen zu nehmen und sequentiell an die Kartenapplikation zu senden. Dazu wird jeder APDU eine Menge von akzeptablen Statuswörtern zugeordnet. Sollte das Statuswort einer Response APDU nicht in den erwarteten enthalten sein, wird der Vorgang abgebrochen und alle bisher von der Karte erhaltenen R-APDUs zurückgegeben.

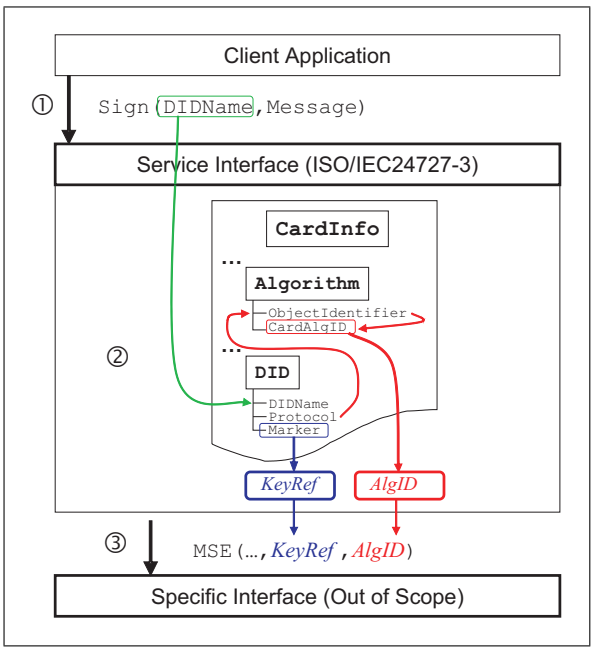

Abbildung 4.11: Auszug [59] Fig. 3: Mapping von generischem auf kartenspezifischen Request

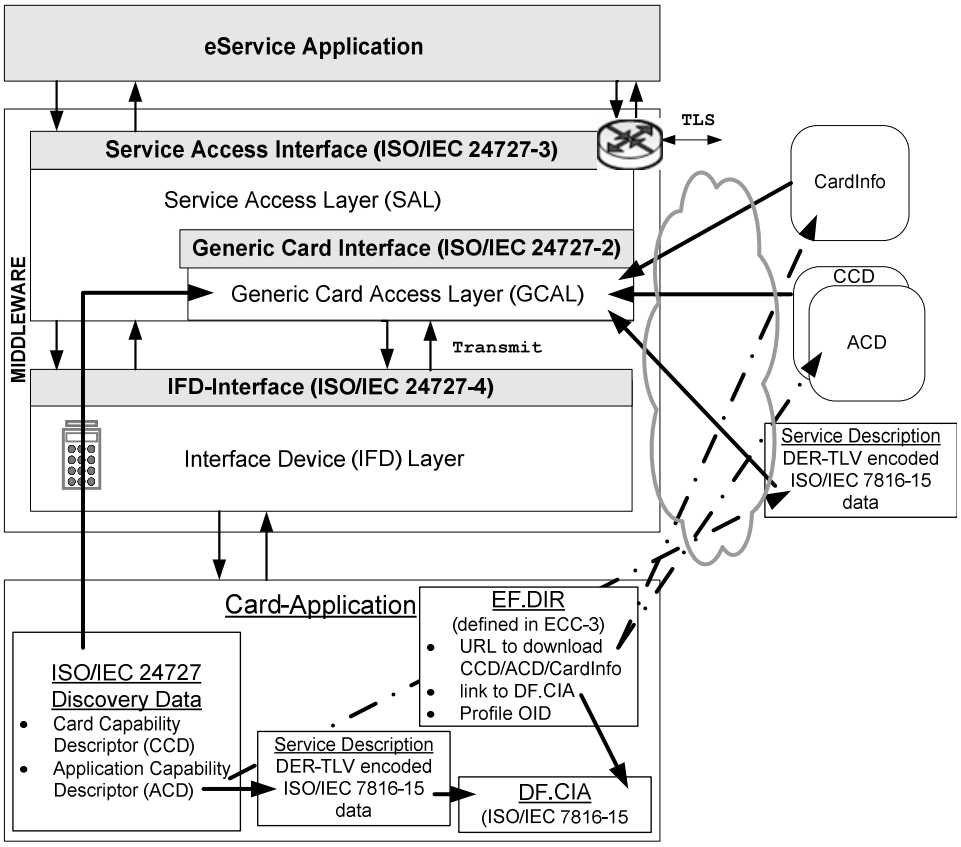

Abbildung 4.12: Auszug [42] Fig. 1: CEN/TS 15480 konforme Chipkarte im ISO/IEC 24727 Framework

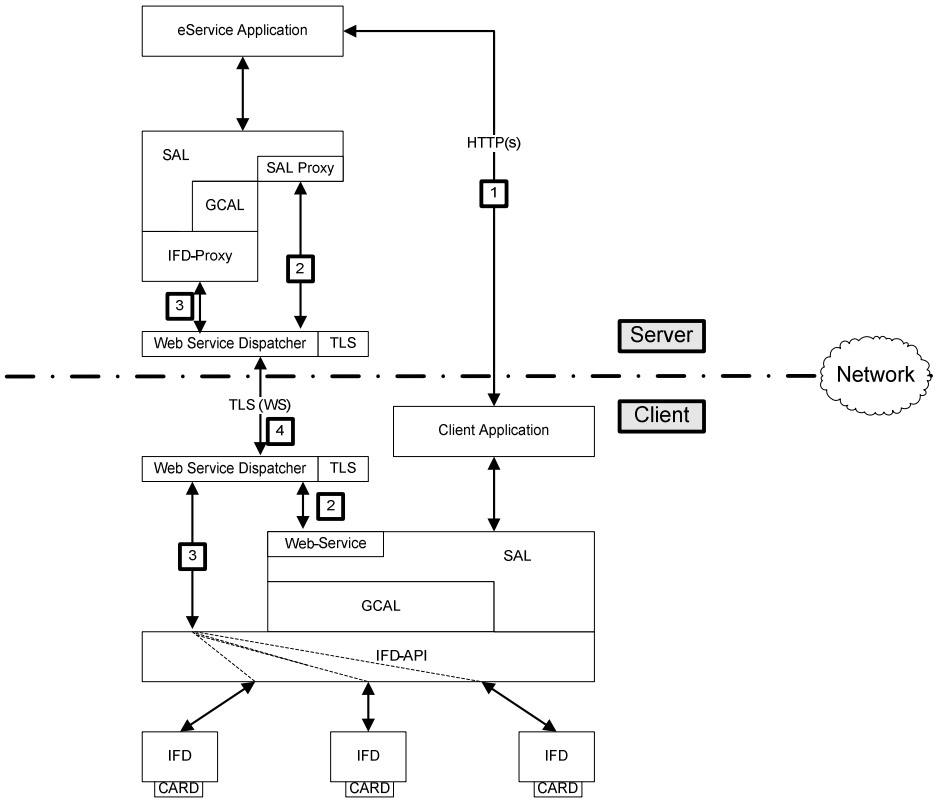

Abbildung 4.13: Auszug [42] Fig. 8: ECC-3 Middleware Architektur in einer Remote-ICC-Stack Konfiguration

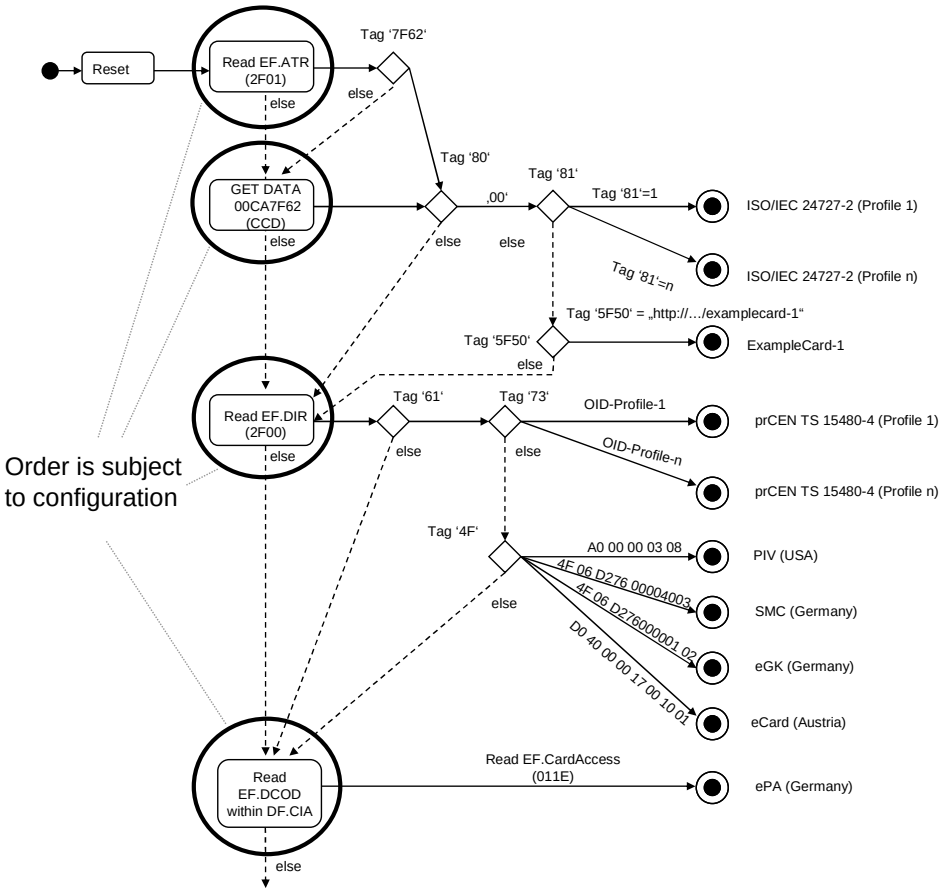

Abbildung 4.14: Auszug [42] Fig. 12: Kartenidentifikationsprozess

Kapitel 5

eCard-API Framework

Aufbauend auf den vorherigen Kapiteln ist es möglich, die Architektur und Funktion des eCard-API-Frameworks zu verstehen. Da der konkrete Funktionsumfang die reine Chipkartenkommunikaton übersteigt und auch zusätzliche Dienste enthalten sind, gibt es weitere Schnittstellen und Komponenten. Wegen der Fokusierung auf die eID-Funktionalität werden diese aber nur kurz erwähnt, da ihre Funktionalität bei einem mobilen Einsatz nicht benötigt wird.
Zunächst wird eine grobe Übersicht über die vorhandenen Schnittstellen und die darin enthaltenen Komponenten gegeben. Danach wird der Verbindungsaufbau zwischen zwei verteilten Instanzen beschrieben. Den Abschluss bildet eine Analyse des Ablaufs des EAC Protokolls, wie es beim Einsatz der eID-Funktion benutzt wird.

5.1 Überblick

Der oberflächliche Aufbau der äußeren Schnittstelle der AusweisApp ist in TR-03112 [44] spezifiziert und ist schematisch in Abbildung 5.1 dargestellt. Darin befinden sich vier Ebenen. Die Oberste, der Application-Layer, ist nicht Teil der Spezifikation. Darin befinden sich die verschiedenen Anwendungen, die das eCard-API-Framework für den Zugriff auf die eCards und damit verbundene Funktionen nutzen wollen.
Darunter befindet sich der Identity-Layer. Dieser beinhaltet das Management Interface, über das die Funktionalität des Framework administriert werden kann. Dazu stehen Funktionen für das Update des Frameworks und die Verwaltung von vertrauenswürdigen Identitäten, Chipkarten, Kartenterminals sowie des Default-Verhaltens zur Verfügung. Daneben existiert das eCard Interface, welches auf hoher Ebene den Bezug von Zertifikaten sowie die Verschlüsselung, Signatur und Zeitstempelung von Dokumenten ermöglicht. Daneben existieren Funktionen für die Verwaltung und Nutzung elektronischer Identitäten.
Unter dem Identity-Layer befindet der Service-Access-Layer. In diesem gibt es ein Support-Interface, welches Funktionalität bereitstellt, die keine Chipkarteninteraktion benötigt. Dies sind unter anderem das Kodieren und Dekodieren von Daten, das Validieren von XML-Dokumenten oder das Einbinden weiterer CardInfo-Dateien. Daneben existiert das ISO24727-3-Interface. Dieses bietet eine Webservice Schnittstelle zu den in ISO/IEC 24727-

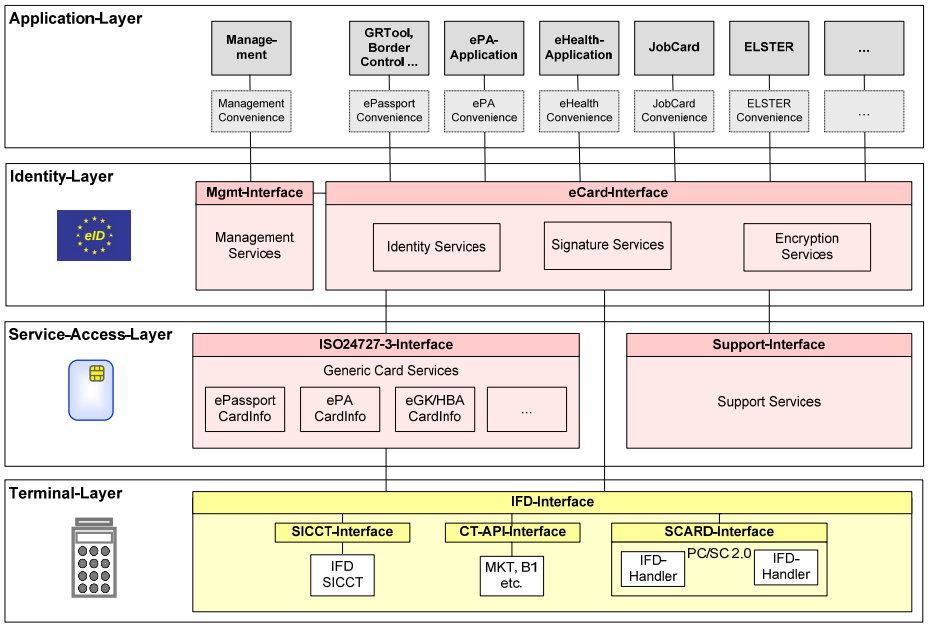

Abbildung 5.1: Auszug [45] Fig. 1: Architektur des eCard-API Frameworks

3 [37] spezifizierten Funktionen, um (kryptographisch geschützte) Verbindungen zu Chipkarten herzustellen, Chipkartenapplikationen zu verwalten, Daten zu lesen oder zu schreiben, kryptographische Operationen auszuführen sowie das entsprechende Schlüsselmaterial (in Form von Differential-Identities) zu verwalten. Hierbei sind alle Funktionen, die Differential-Identities nutzen oder verwalten, über protokollspezifische Object Identifier parametrisiert, sodass die unterschiedlichen in eCard-API Teil 7 [45] definierten Protokolle über eine einheitliche Schnittstelle genutzt werden können.
Als letztes gibt es noch den Terminal-Layer, der genau wie in CEN-15480 ECC-3 [27] fast gänzlich dem IFD-Interface aus ISO/IEC 24727-4 [38] entspricht. Dieses übernimmt die Generalisierung von konkreten Lesertypen und verschiedenen Schnittstellen sowie die Kommunikation mit der Chipkarte.
Zusätzlich zu den in den Interfaces enthaltenen Funktionen gibt es noch *TC_API_OPEN* und *TC_API_CLOSE*, sowie *StartPAOS*, die eine sichere Verbindung bei einer verteilten Konfiguration mit dem PAOS-Binding ermöglichen.
Die Ebenen-Aufteilung in mehrere Layer ist allerdings keine Layer-Struktur wie sie z.B. im OSI-Modell [29] angewendet wird. Einer Applikation stehen generell alle Interfaces zur Verfügung. Es ist nicht so zu verstehen, dass eine Anwendung nur mit dem Identity-Layer interagieren darf.
Zur vertrauenswürdigen Anzeige von zu signierenden Dokumenten existiert ein sogenannter Trusted Viewer. Diese Komponente soll sicherstellen, dass der Benutzer sich darauf verlassen kann, dass er wirklich das angezeigte Dokument signiert und nicht irgendein anderes.

Die Kommunikation zwischen zwei eCard-API-Framework-Instanzen geschieht aud der Basis von Webservices. Dabei wird ein Request-Response-Nachrichtenaustausch eingesetzt, um einen entfernten Funktionsaufruf auszuführen. Dementsprechend werden aufgetretene Fehler innerhalb der Response an den Aufrufer übermittelt. Dazu dient das zwingend vorhandene *Result*-Element, das in seinen Unterelementen entweder die fehlerlose Ausführung der gerufenen Funktion mitteilt oder eine möglichst genauen Fehlercode enthält. Eine Liste der vordefinierten Fehlercodes ist im ersten Teil der TR-03112 [44] enthalten.
Die einzelnen Schnittstellen sind in den Teilen zwei bis sechs der TR-03112 beschrieben. Im siebten Teil werden die notwendigen Protokolle nach [37] spezifiziert, insbesondere EAC (Version 2) [48]. Zusätzlich wird die sichere Verbindungsherstellung für das SOAP [50] und PAOS [14, 13] Binding über TLS [52] angeführt.

5.2 Verbindungsaufbau

Die Protokolle, welche verschiedene Instanzen des eCard-API-Frameworks involvieren, kommunizieren potenziell über ein ungesichertes Netzwerk. Daher wird zur Absicherung der Verbindung zwischen Modulen und Instanzen des eCard-API-Frameworks die Benutzung von TLS [52] vorausgesetzt. Dabei können entweder auf beiden Seiten X.509 [51] Zertifikate benutzt werden, was aber adäquaten Schutz des privaten Schlüssels auf beiden Seiten bedingt, oder anonyme TLS Ciphersuites nach RFC 4346 [52] und RFC 4492 [49], für die aber Maßnahmen gegen Man-in-the-Middle Angriffe berücksichtigt werden müssen.

5.2.1 Verbindungsaufbau mit SOAP Binding

Bei der Nutzung des SOAP Bindings ist der Verbindungsaufbau unkompliziert. Hier werden die SOAP Requests von Client-Applikationen zur Kartenapplikation gesendet. Damit entsprechen die Rollen Client und Server beim Aufbau der TLS-Verbindung auch den Rollen beim SOAP Nachrichtenaustausch.

5.2.2 Verbindungsaufbau mit PAOS Binding

Beim PAOS Binding sind die Rollen allerdings vertauscht. Hier fällt dem Webservice-Client die Rolle des TLS/HTTPS Servers zu, aber der TLS/HTTPS Client, und damit Webservice-Server, muss die Verbindung aufbauen. Da die eCard-API-Middleware in der Regel eine eigenständige Software ist, die zwar von einer Anwendung benutzt wird, trotzdem aber von ihr unabhängig existiert, werden zwei separate geschützte Kanäle aufgebaut, wie in Abbildung 4.13 sichtbar. Daraus ergibt sich die Notwendigkeit von zwei separaten TLS-Kanälen, deren Beziehung kryptographisch abgesichert sein soll. Dazu muss das Erstellen eines TLS Kanals nach RFC 4279 [53] mittels eines vorher ausgetauschten PSK (Pre-Shared Key) unterstützt werden.
Da ein Client üblicherweise nicht in der Lage ist eingehende Verbindungen anzunehmen (z.B. wegen NAT), soll er die TLS-Verbindung initiieren. Dazu wird der Ansatz eines Browser-Plugins verfolgt. Die Idee ist, dass der Browser auf ein HTTPS Request an den IdP (oder eService in diesem Kontext) eine Response erhält, die ein HTML-Object-Tag

enthält, welche das Browser-Plugin dazu veranlasst, die eCardAPI-Framework Instanz anzusprechen, um die PAOS-Verbindung zu initiieren. Das Object-Tag soll dabei folgende Parameter enthalten:

```
<object type="application/vnd.ecard-client">
    <param name="ServerAddress" value="http://sal.example.com">
    <param name="SessionIdentifier" value="1A23...B129">
    <param name="Binding" value="urn:liberty:paos:2006-08">
    <param name="PathSecurity-Protocol" value="urn:ietf:rfc:4279">
    <param name="PathSecurity-Parameters" value="<PSK>4BC1 ... A0B5</PSK>">
    <param name="RefreshAddress" value="http://service.example.com">
</object>
```

Da Microsoft Browser unter Umständen Plugins nicht immer aktivieren, kann in diesem Fall ein Browser Helper Object (BHO) eingesetzt werden. Der PAOS Verbindungsaufbau ist in Abbildung 5.2 dargestellt. Die Prozedur zum Erstellen der Verbindung läuft in folgenden

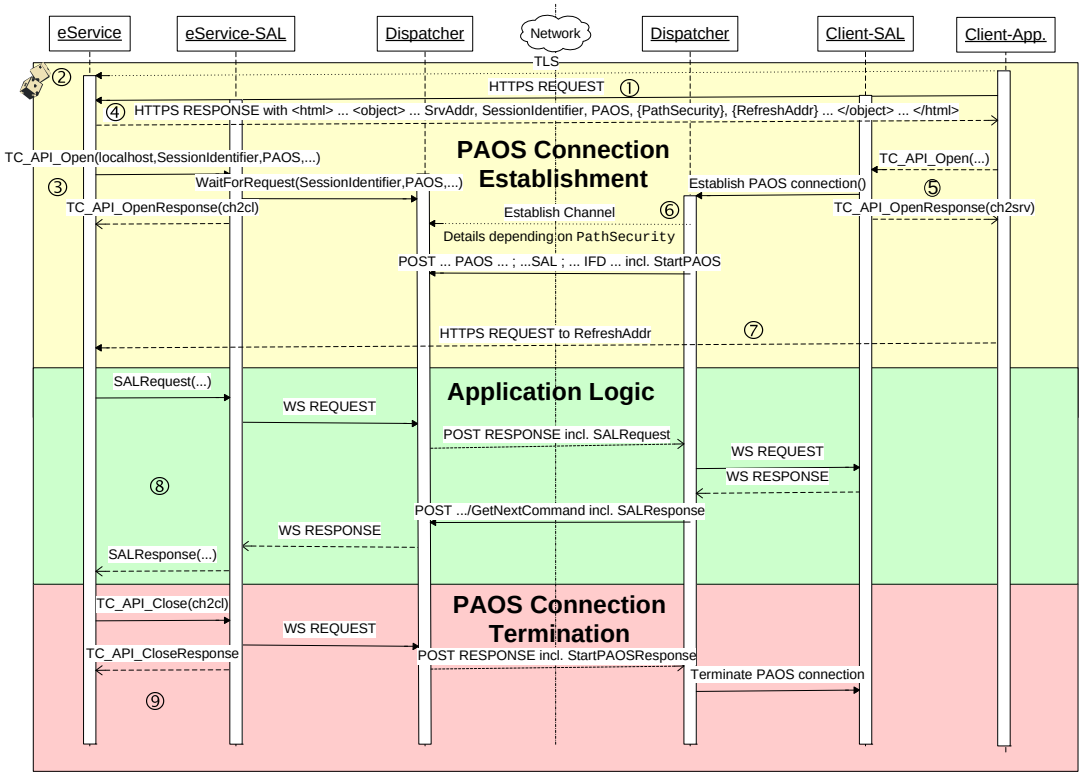

Abbildung 5.2: Auszug [45] Fig. 2: Verbindungsaufbau mit PAOS Binding

Schritten ab:

1. Innerhalb einer schon bestehenden TLS-Verbindung sendet der Client (WebBrowser)

dem eService ein HTTP Request, in dem er seine Bereitschaft zur Initialisation einer PAOS-Verbindung mitteilt.

2. Der eService generiert einen eindeutigen SessionIdentifier und ein PSK (pre-shared key). Der SessionIdentifier dient zur Identifizierung der Sitzung und durch den PSK wird die Beziehung zwischen der neu zu erstellenden TLS-Verbindung und der schon bestehenden kryptographisch abgesichert.

3. Der eService sendet an seinen lokalen SAL einen *TC_API_Open* Aufruf, in dessen Parameter *ChannelHandle* der SessionIdentifier und der PSK enthalten sind. Optional können diese Paramter auch nicht gesetzt sein, dann werden sie vom SAL selbst erzeugt und in der *TC_API_OpenResponse* an den eService übergeben.

4. Der Client erhält die HTTP Response, die sein Browser Plugin anspricht.

5. Die eCardAPI-Framework Instanz auf Client-Seite kann per *TC_API_Open* angesprochen werden. Dies hängt aber von der Implementation und Integration des Browser Plugins mit dem restlichen Middlewarestack ab.

6. Entsprechend dem *PathSecurity* Paramter, den das Browser-Plugin auswertet, wird vom Client-SAL eine sichere Verbindung hergestellt. Daraufhin wird das PAOS Request-Response Message Exchange Pattern [14] begonnen. Um den SessionIdentifier an den eService zu übergeben sendet der Client-SAL einen GET oder POST Request mit einem PAOS spezifischen Header.

   (i) Bei einem GET Request wird der SessionIdentifier als URL-Parameter übergeben.

   ```
   GET /eService?SessionIdentifier=12345 http/1.1
   Host: example.com
   Accept: text/html; application/vnd.paos+xml
   PAOS: ver=''urn:liberty:paos:2003-08'';
   ''urn:iso:std:iso-iec:24727:tech:schema''
   ```

   (ii) Alternativ wird ein POST Request benutzt, in dem ein SOAP-Envelop enthalten ist. In diesem steckt der *StartPAOS* Aufruf, der wiederum den SessionIdentifier enthält. Daneben sollten Informationen über die Kartenterminals und vorhandenen Chipkarten beim Client enthalten sein. Das ermöglicht, dass die bei einem Get Request im nächsten Schritt notwendigen *CardApplicationPath* und *CardApplicationConnect* Aufrufe weggelassen werden können. Daher wird die POST Request Variante empfohlen.

   ```
   POST /eService?SessionIdentifier=12345 http/1.1
   Host: example.com
   Accept: text/html; application/vnd.paos+xml
   PAOS: ver=''urn:liberty:paos:2003-08'';
   ''urn:iso:std:iso-iec:24727:tech:schema''
   action=''StartPAOS''
   ```

```
<S:Envelope xmlns:S=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:iso=urn:iso:std:iso-iec:24727:tech:schema>
  <S:Body>
    <iso:StartPAOS>
      <iso:SessionIdentifier>12345
      </iso:SessionIdentifier>
      <iso:ConnectionHandle>... </iso:ConnectionHandle>
      ...
      <iso:ConnectionHandle>... </iso:ConnectionHandle>
    </iso:StartPAOS>
  </S:Body>
</S:Envelope>
```

7. Wenn der Client-SAL den *TC_API_Open* Aufruf mit einer *TC_API_OpenResponse* beantwortet, wird das Browser-Plugin instruiert, in regelmäßigen Intervallen an die *RefreshAddress* Anfragen zu stellen. Wenn die eigentliche Applikation ein Ergebnis erreicht hat, wird dieses über die *RefreshAddress* dem Client mitgeteilt.

8. Innerhalb des PAOS Request-Response Message Exchange Pattern [14] kann nun der eService SAL-Requests an den Client senden. Aus der PAOS Spezifikation des Request-Response Message Exchange Pattern geht nicht hervor, das es zwischen dem Initial Request und der Response zum Initial Request mehr als ein Paar von PAOS Request-Response Nachrichten geben kann. Praktisch ist dies aber so umgesetzt.
   Der Inhalt der einzelnen Requests und Responses ist von der jeweiligen Applikationslogik abhängig. Beispielhaft sei hier ein Aufruf angedeutet für den Fall, dass *CardApplicationPath* und im Anschluss daran eine weitere Funktion gerufen werden soll.

```
HTTP 200
Content-Type: application/vnd.paos+xml
Content-Length: ...
<soap:Envelope xmlns:soap=”http:// ...”>
  <soap:Header>
  <paos:Request xmlns:paos="urn:liberty:paos:2003-08"
    responseConumerURL="/eService?SessionIdentifier=12345"
    service="urn:iso:std:iso-iec:24727:tech:schema"
    mustUnderstand="1"
    messageID="4711"
    actor=”http://schemas.xmlsoap.org/soap/actor/next”/>
  </soap:Header>
  <soap:Body>
  <soap-env:CardApplicationPath ... >
```

   Der Client sendet daraufhin eine *CardApplicationPathResponse* zurück und fordert gleichzeitig das nächste Kommando an.

```
POST /eService?SessionIdentifier=12345 HTTP/1.1
Host: example.com
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:paos:2003-08";
"urn:iso:std:iso-iec:24727:tech:schema"
Content-Type: application/vnd.paos+xml
Content-Length: ...
<Soap:Envelope ... >
  <soap:Header>
    <paos:Response xmlns:paos="urn:liberty:paos:2003-08"
      refToMessageID="4711"
      mustUnderstand="1"
      actor=''http://schemas.smlsoap.org/soap/actor/next''/>
  </soap:Header>
  <Soap:Body ...>
  <soap-env:CardApplicationPathResponse ...>
    ...

HTTP 200
Content-Type: application/vnd.paos+xml
Content-Length: ...
<soap:Envelope
    ...
```

9. Zuletzt wird zum Schließen der PAOS Verbindung vom eService ein *TC_API_Close* an den eService-SAL gesendet, was in einem POST an den Client resultiert, welcher einen *StartPAOSResponse* Element enthält.

5.3 ISO/IEC 24727 Protokolle

Um die konkrete Funktionalität des Personalausweises über ISO/IEC 24727-3 [37] zu nutzen, sind die notwendigen Protokolle und dazugehörigen DID Strukturen nötig. Hier soll wegen der Fokussierung auf die eID-Funktionalität nur auf EAC Version 2 [48] eingegangen werden.

5.3.1 Extended Access Control

Das in TR-03110 [48] beschriebene EAC Protokoll dient unter anderem zur Herstellung einer sicheren Ende-zu-Ende Verbindung und gleichzeitiger gegenseitiger Authentisierung. Dabei existieren vier Subprotokolle: Password Authenticated Connection Establishment (PACE), Chip Authentication (CA), Terminal Authentication (TA) und Restricted Identification (RI). Letzteres ist für den Verbindungsaufbau allerdings nicht relevant. Der Ablauf von EAC zur Authentifizierung über das Internet mittels eines eCard-API-Frameworks ist in Abbildung 5.3 skizziert. Der Vorgang wird in 3 Phasen eingeteilt.

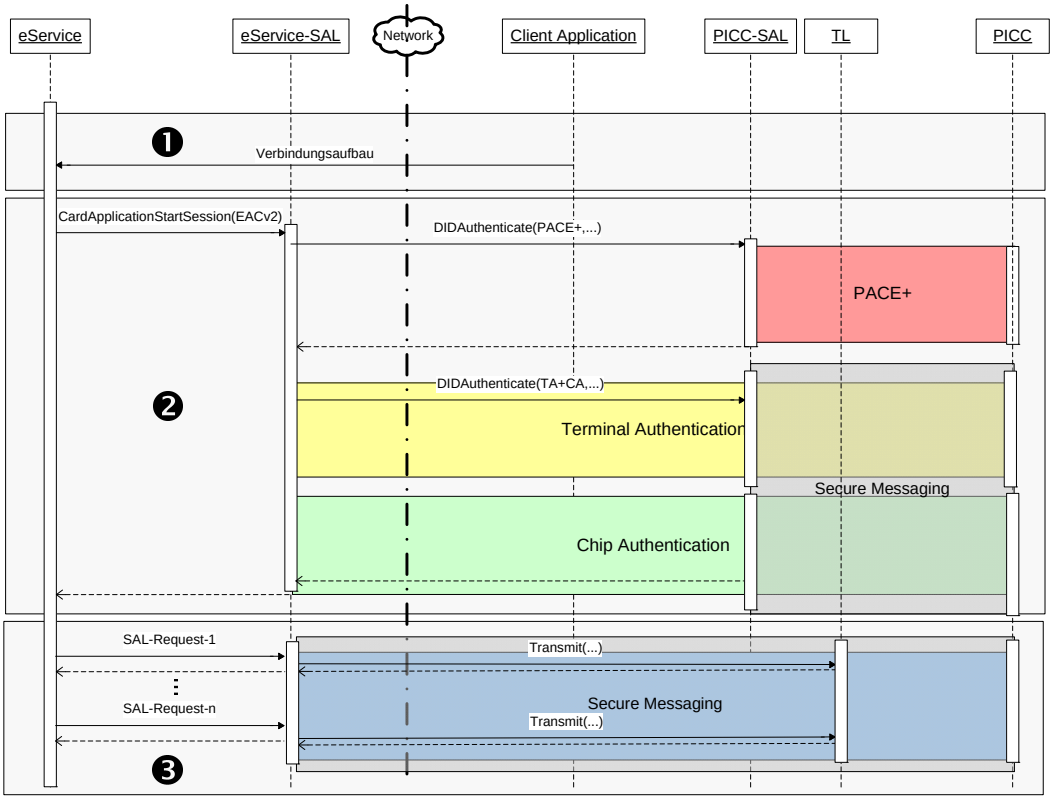

Abbildung 5.3: Auszug [45] Fig. 3: EAC (Version 2) zur Authentifizierung im Internet

1. Verbindungsaufbau
   In dieser Phase wird mittels *TC_API_Open* eine sichere Verbindung zwischen den Schnittstellen der Frameworks hergestellt. Dabei kommt der Verbindungsaufbau mit PAOS zum Einsatz.

2. Herstellung eines geschützten Kanals zu Chipkarte
   Durch den Aufruf von *CardApplicationStartSession* wird das EACv2 Protokoll durchgeführt und dadurch einen Secure Messaging Kanal vom eService-SAL zur Karte herstellt. In dieser Phase können bei einem eID Durchlauf schon die zu lesenden Datengruppen oder die Berechnung des sektorspezifischen Pseudonyms (RI), für die die notwendigen Berechtigungen vorliegen, über Parameter angezeigt werden, obwohl dieser Vorgang so in Abbildung 5.3 nicht zu erkennen ist. Der SAL auf der eID-Server-Seite führt daraufhin den gesamten Vorgang durch, ohne das ein weiterer Aufruf nötig wäre.

3. Benutzen des geschützten Kanals
   Wenn *CardApplicationStartSession* zurückkehrt, kann der SM-Kanal genutzt werden, um z.B. weitere Datengruppen zu lesen oder andere Kartenfunktionen zu nutzen. Zu diesen Zweck sendet der eService SAL Requests (*DSIRead*, *DataSelect*, etc.) an den eService-SAL. Dieser generiert daraus SM abgesicherte APDUs, die an den IFD-Layer

des Clients und somit an die Karte gesendet werden. Dabei wird die *Transmit* Funktion genutzt.

5.3.2 *CardApplicationStartSession* und *DIDAuthenticate*

Die Logik der Protokollausführung ist durch die Funktionen *CardApplicationStartSession* und *DIDAuthenticate* gekapselt. Daher besitzen sie eine zentrale Bedeutung im Kontext der ISO/IEC 24727. Der Aufruf der Funktion *CardApplicationStartSession* soll eine Sitzung zwischen der Kartenapplikation und der Client-Applikation herstellen. Die Funktion *DIDAuthenticate* dient dazu, ein mit einer DID verbundenes Authentisierungsprotokoll durchzuführen. Beide Funktionen haben dieselbe Signatur.

```
OUT ReturnCode CardApplicationStartSession(
IN ConnectionHandle connectionHandle,
IN DIDScope didScope,
IN DIDName didName,
IN/OUT DIDAuthenticationData authenticationProtocolData,
IN ConnectionHandle samConnectionHandle
);

OUT ReturnCode DIDAuthenticate(
IN ConnectionHandle connectionHandle,
IN DIDScope didScope,
IN DIDName didName,
IN/OUT DIDAuthenticationData authenticationProtocolData,
IN ConnectionHandle samConnectionHandle
);
```

Der Unterschied liegt nur darin, dass die DID, die mit *DIDName* referenziert ist, bei *CardApplicationStartSession* dazu in der Lage sein muss, eine Sitzung herzustellen bzw. im Parameter *authenticationProtocolData* so etwas wie z.B. Sitzungsschlüssel zurückzugeben. Beiden gemeinsam ist, dass es zur Durchführung des Protokolls zu mehreren Aufrufen mit derselben DID kommen kann.

5.3.3 EAC Parameter

Die einzelnen Funktionen und entsprechenden Nachrichten, die durch den Aufruf von *CardApplicationStartSession* ausgelöst werden, sind in Abbildung 5.4 zu sehen. Die tatsächlich übertragenen Nachrichten können dem Anhang A entnommen werden. Der genaue Ablauf der Herstellung der geschützten Verbindung erfolgt in folgenden Schritten:

1. Der eService-SAL ruft *DIDAuthenticate* mit dem *DIDName* Parameter für PACE gesetzt und *DIDAuthenticationData* vom Type *EAC1InputType*. Darin befindet sich unter anderem das eService Zertifikat, welches die Berechtigungen zur Kommunikation mit der Karte enthält. Daneben werden entsprechende Zertifikatsbeschreibungen und, falls

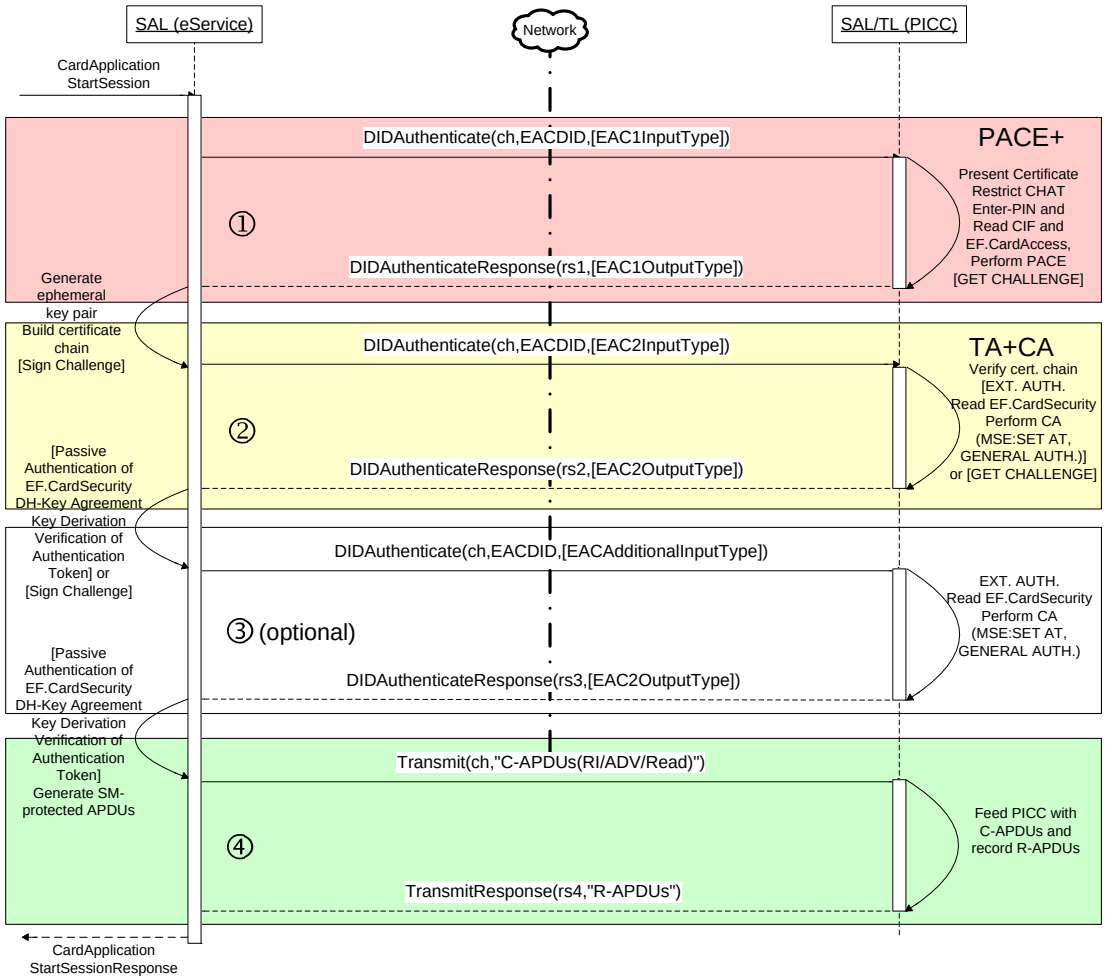

Abbildung 5.4: Auszug [45] Fig. 4: Nachrichtenaustausch zwischen SALs nach *CardApplicationStartSession* EAC

notwendig, ein alternativer *CardHolderAuthorizationTemplate* (CHAT) sowie *AuthenticatedAuxilaryData* für den Chip übertragen. Der Inhalt des Zertifikats und vorhandene Zertifikatsbeschreibungen sind dem Nutzer anzuzeigen. Zusätzlich kann der Nutzer die effektiven Zugriffsrechte des eService weiter einschränken bevor er sein Passwort eingibt und damit sein Einverständnis zu dem Vorgang signalisiert.
Als erste Kartenoperation wird die Datei EF.CardAccess gelesen. Nachdem das PACE-Protokoll durchgelaufen ist, wird vom Chip eine Challenge für die folgende TA angefordert. Dies setzt voraus, dass der Chip während TA in der Lage ist, die Zertifikatskette zu prüfen, nachdem er die Challenge erstellt hat, aber bevor die Signatur überprüft wird. Sollte dies nicht der Fall sein, so fehlt der *Challenge* Parameter im *EAC1OutputType*, der in der *DIDAuthenticateResponse* zurückgegeben wird, und ein zusätzlicher Nachrichtenaustausch (Schritt 3) wird dann benötigt.

Zusätzlich enthält die *DIDAuthenticationData* Struktur innerhalb der *DIDAuthenticateResponse* die ASN.1 kodierte *SecurityInfo* Struktur aus dem EF.CardAccess, den vom Nutzer möglicherweise eingeschränkten CHAT und bis zu zwei *CertificateAuthorityReference* Elemente, die die in der Karte vorhandenen Wurzel-Schlüssel zur Verifikation der Zertifikatskette angeben. In der *SecurityInfo* Struktur finden sich die Domain Parameter, die im nächsten Schritt zur Generierung eines neuen Schlüsselpaars verwendet werden.

2. Der eService-SAL wählt eine zu den Wurzelschlüsseln passende Zertifikatskette aus und generiert ein den Domain Parametern entsprechendes Schlüsselpaar und, wenn möglich, signiert die Challenge. Mit diesen Daten ruft der eService-SAL erneut *DIDAuthenticate*, wobei der Parameter *DIDName* für CA gesetzt ist. Der *DIDAuthenticationData* Parameter ist dabei vom Typ *EAC2InputType*. Darin sind neben der Zertifikatskette der öffentliche Schlüssel des soeben generierten Schlüsselpaares und, falls vorhanden, die Signatur der Challenge enthalten.
   Die Zertifikatskette wird nun vom Ausweis verifiziert. Danach wird entweder schon die Signatur mittels *EXTERNAL AUTHENTICATE* geprüft oder erst die Challenge angefordert, sollte das nicht schon vor der Zertifikatskettenverifikation in Schritt 1 möglich gewesen sein.
   Wenn bereits möglich, wird die Datei EF.CardSecurity gelesen und die CA mittels *MSE:SET AT* und *GENERAL AUTHENTICATE* ausgeführt. Schließlich werden die Ergebnisse dem eService-SAL über *DIDAuthenticateResponse* im Parameter *DIDAuthenticationData* vom Typ *EAC2OutputType* übergeben. Dies sind entweder der Inhalt von EF.CardSecurity, ein *AuthenticationToken* sowie eine Nonce, um das *AuthenticationToken* zu überprüfen, oder das *Challenge* Element.

3. Dieser Schritt ist optional und muss nur durchgeführt werden, wenn die Karte das Anfordern der Challenge nicht vor der Zertifikatskettenverifikation unterstützt. Daher würde jetzt hier die Signatur im Parameter *DIDAuthenticationData* vom Typ *EACAdditionalInputType* an die Karte gesendet werden. Die Rückgabe entspricht dann aber der aus Schritt 2, wenn die Challenge schon übertragen wurde und enthält alle Daten, die deswegen in Schritt 2 noch nicht enthalten waren.

4. Wenn die Signatur aus dem EF.CardSecurity und das *AuthenticationToken* verifiziert wurden, kann der eService-SAL direkt mit dem Ausweis durch SM geschützte APDUs kommunizieren. Je nachdem, welche Informationen in *CardApplicationStartSession* angefragt wurden, können ein Sektor-spezifisches Pseudonym (RI) erstellt oder einfach nur Datengruppen ausgelesen werden. Die notwendigen APDUs können in einem Stapel mit der *Transmit* Funktion versendet werden. Die Response APDUs werden in *TransmitResponse* zurückgegeben.

5.4 Beobachtungen an bestehenden Implementationen

Die tatsächliche Implementation in Form der im Anwendungstest verwendeten eCard-API Schnittstelle des eID-Servers hat einige Eigenheiten, welche aus der Spezifizierung durch

die TR-03112 [44] so nicht hervorgehen. Relevant sind diese für den Versuch einer Implementierung der Client-seitigen eCard-API, welche mit dem Anwendungstest-eID-Server kommunizieren soll. Einige der bei so einer Entwicklung beobachteten Details werden im Folgenden betrachtet.
Ein Aspekt, der eine Entwicklung erschwert, besteht darin, dass der eID-Server bei Daten, die zwar syntaktisch, nicht aber semantisch korrekt sind, in einigen Fällen keine Fehlermeldungen produziert. Diese inhaltlichen Fehler sind dabei meistens logisch, so dass vermutlich der interne Programmablauf des eID-Servers in eine Fehlerbedingung läuft. Der Client erhält also als Antwort auf so eine Nachricht nicht eine Art von Fehlermeldung, sondern überhaupt keine Antwort.
Eine solche Situation entsteht beispielsweise bei einer mehrfachen Verwendung des gleichen *ContextHandle* und *SessionHandle* in verschiedenen eCard-API-Sessions. Obwohl man davon ausgeht, dass *ContextHandle* und *SessionHandle* nur bezüglich eines *SessionIdentifier* eindeutig sein müssen, bleiben sie unabhängig vom *SessionIdentifier* für eine Zeit lang gespeichert, nachdem sie dem eID-Server einmal bekannt gemacht wurden. Eine neue Sitzung, welche diese gespeicherten Werte in der *StartPAOS* Nachricht übergibt, erhält wie eingangs beschrieben keine Antwort.
Problematisch ist die vermutliche Realisierung des Aufrufes der *RefreshAddress* durch das Browser-Plugin. So wurde am Live-System beobachtet, dass es genau einen Aufruf der *RefreshAddress* gibt. Dabei gibt es an der spezifizierten Adresse immer eine SAML-Response. Diese enthält nur dann keine Fehlermeldung, wenn der eID-Durchlauf erfolgreich bis zur *TransmitResponse* Nachricht fortgeschritten ist. Wie in Anhang A ersichtlich, sollte nun vom eID-Server eine Nachricht mit *CardApplicationEndSession* gesendet werden. Dieses geschieht allerdings nicht. Vermutlich wird dieser Verbindungsabbau auf eID-Server-Seite erst durch das Abfragen der *RefreshAddress* durch den Browser des Clients angestoßen. Für einen Client stellt dieses nicht wirklich ein Problem dar, weil die eCard-API-Kommunikation aus seiner Sicht damit direkt nach *TransmitResponse* als abgeschlossen angesehen werden kann. In welcher Weise das aber zu nicht freigegebenen Ressourcen auf dem eID-Server führen kann, bleibt offen.
Eine weitere zu beachtende Tatsache ist die zeitliche Beschränkung für den kompletten Ablauf der eID-Funktionalität. Diese muss in ungefähr zwei Minuten abgeschlossen sein. Andernfalls werden auf dem Server vermutlich schon Ressourcen freigegeben. Daraufhin erhält der Client in der Regel eine Fehlermeldung als Antwort, deren Inhalt allerdings nicht auf eine Zeitüberschreitung schließen lässt. Die AusweisApp zeigt diese Zeitbeschränkung lokal als ablaufenden Zähler an. Dieser ist dabei aber nicht vom Server gesteuert, sondern ein vom Client lokal geschätzter Wert.
Schließlich bleibt noch zu erwähnen, dass das Terminalzertifikat in zwei Nachrichten vom eID-Server zum Client übertragen wird. Beim ersten Mal zur Ausführungen von PACE und zum zweiten Mal für die Ausführung von TA. Dabei unterscheiden sich die Zertifikate natürlich nicht. Aus der TR-03112 Teil 7 [45] geht hervor, dass diese doppelte Übertragung nicht notwendig ist und bei TA nicht stattfinden muss, wenn die Zertifikatskette vom Chip verifiziert werden kann, was in diesem Anwendungsfall gegeben ist. Dieses, genau wie die derzeitige Nichtnutzung der optimierten EAC-Ausführung, sind aber sicherlich nur Kinderkrankheiten, die im Laufe der Zeit verbessert werden zumal sie inhaltlich keine Fehler

darstellen.
Die hier erwähnten Beobachtungen beziehen sich alle auf den eID-Server von Openlimit [2]. Tests mit alternativen Produkten wurden nicht durchgeführt, bzw. scheiterten schon beim Aufbau des TLS/PSK-Kanales mit den verwendeten Bibliotheken.

Kapitel 6

Mobile Einsatzszenarien

In diesem Kapitel werden die Ansätze für einen mobilen Einsatz beleuchtet. Dabei wird unterschieden, welche Probleme sich in Bezug auf die Software ergeben und welche Möglichkeiten zur Nutzung von Hardware bestehen. Grundsätzlich sind die beschriebenen Probleme der Grund, warum die derzeitigen Spezifikationen den mobilen Einsatz noch nicht abdecken. Die im nächsten Kapitel beschriebenen Ansätze sollen dabei ein Schritt zur Lösung dieser Probleme darstellen.

6.1 Software

Aus dem Funktionsumfang der eCard-API, wie er in TR-03112 [44] spezifiziert ist, ergibt sich, dass eine vollständige Portierung der AusweisApp in ihrer jetzigen Form auf weitere Plattformen wahrscheinlich nicht stattfinden wird.
Zu der geplanten Einführung des Personalausweises in Deutschland wird die AusweisApp kostenlos zur Verfügung gestellt, um die Akzeptanz in der Bevölkerung zu erhöhen. Dabei wird es eine Implementierung für Windows, Mac, und einige Linux-Distributionen geben. Wegen den Kosten ist mit einer Portierung der AusweisApp auf diverse, weitere Plattformen, welche jeweils nur einen vergleichsweise geringen Marktanteil haben, nicht zu rechnen. Gleichzeitig prüfen BMI und BSI eine Open-Source Strategie, um die Verfügbarkeit auch auf weniger verbreiteten Plattformen zu fördern. Dennoch ist eine Implementation des eCard-API-Frameworks nicht trivial. Für einige der verwendeten Mechanismen existieren keine freien Bibliotheken, welche die spezifizierten Standards umsetzen. Obwohl es unzählige, teils sehr mächtige Frameworks für Webservices auf Basis von SOAP gibt, ist dasselbe mittels PAOS gegenwärtig nicht zu finden. Das bedeutet, dass für die Webservice-Kommunikation nicht auf eine fertige Bibliothek zurückgegriffen werden kann. Ein weiteres Problem ist die Umsetzung von TLS mit PSK nach RFC 4279 [53]. Die bekannteste Bibliothek, OpenSSL [24], unterstützt TLS/PSK erst seit Version 1.0, welche offiziell am 29. März 2010 erschienen ist. GnuTLS [23] bietet zwar schon länger Unterstützung, doch ist in beiden Fällen die Integration in andere Bibliotheken meistens nur auf den Fall mit einem Server-Zertifikat ausgelegt und erfordert somit einige Anpassungen. Die gesamte spezifizierte Schnittstelle umfasst über 100 einzelne Funktionen mit teilweise sehr komplexen

Datentypen.
Diese Faktoren machen eine Umsetzung aufwändig. Daher sollen im Folgenden Möglichkeiten betrachtet werden, wie dieser Aufwand reduziert werden kann. Dazu werden zunächst die möglichen Zielplattformen betrachtet, ihre Besonderheiten untersucht und der minimal notwendige Funktionsumfang einer mobilen eCard-API bestimmt. Im nächsten Kapitel werden einige Lösungsansätze vorgestellt, die versuchen, die Komplexität auf ein Minimum zu reduzieren. Schlussendlich werden resultierende Empfehlungen für Anpassungen der betreffenden Technischen Richtlinien gegeben.

6.1.1 Mobile Browser Plugins

Ein Aspekt, der im Weiteren nicht betrachtet wird, ist die Unterstützung für das benötigte Plugin im Browser. Entsprechende Erweiterungen werden aktuell auf mobilen Plattformen nur selten unterstützt. Eine Ausnahme, die Unterstützung anbietet, ist die mobile Version von Firefox [17]. Diese ist allerdings zur Zeit nur für Maemo [16] verfügbar. Eine Portierung auf die wichtigsten zukünftigen Plattformen in Form von iPhone [7] und Android [1] ist aber im Gange. Auf dem iPhone ist es zum jetzigen Zeitpunkt nicht möglich, ein Plugin in den Safari [21] Browser zu integrieren. Dieses hat weniger technische Gründe, sondern liegt an Lizenz-rechtlichen Einschränkungen seitens Apple. Als Ausweg lässt sich ein auf Webkit [26] basierendes Projekt erstellen, welches die Funktionalität eines Browsers mit der Unterstützung für Plugins enthält.
Im Zusammenhang mit den hiernach noch beschriebenen Hardwareproblemen kann dieser Aspekt aber vorerst noch ignoriert werden.

6.1.2 Zielplattformen und Zielstellungen

Ursprünglich wurde als Zielplattform ein Mobiltelefon mit integriertem NFC Lesegerät anvisiert, da es seit einigen Jahren Modelle für Pilotversuche gibt. Doch geht die Entwicklung in diesem Bereich weitaus langsamer voran als angenommen. Es sind gegenwärtig nur zwei Modelle von Nokia auf dem freien Markt verfügbar. Diese sind aber für einen autonomen Einsatz als Endgerät ungeeignet.
Dennoch wird erwartet, dass es in absehbarer Zeit Mobiltelefone geben wird, welche die notwendigen technischen Voraussetzungen erfüllen, um die Kommunikation mit der Chipkarte zu bewältigen. Alternativ bietet sich die Möglichkeit, externe RFID oder NFC Lesegeräte mit einem solchen Gerät zu verbinden. Die Verbindung kann dabei über eine beliebige Hardware-Schnittstelle erfolgen (z.B. USB, Bluetooth, Seriell, SDIO), solange für Anwendungen eine nutzbare Programm-Schnittstelle existiert. Neben Mobiltelefonen kann man die potenziellen Plattformen generell auf eingebettete Systeme erweitern. Grundsätzlich muss es eine auf ISO/IEC 14443 [31] basierende Verbindung zur Chipkarte und eine Internet-Verbindung zum Dienstanbieter sowie eID-Server geben. Dieses lässt sich theoretisch ebenfalls auf einem Geldautomat oder ähnlichem Terminal realisieren. Nachfolgend sei der Fokus aber primär auf den Einsatz von Mobiltelefonen gelegt, auch wenn sich fast alle Ergebnisse auf eingebettete Systeme übertragen lassen.
Mobilen Plattformen und eingebettete Systeme verfügen nur über eingeschränkte Ressour-

cen. Daher lassen sich einige Annahmen formulieren, wie die Komplexität verringert werden kann. Als erstes macht es Sinn, jede nicht zwingend notwendige Operation nicht auf dem Gerät selbst auszuführen. Abhängig von der gegebenen Netzwerkanbindung soll der Datenverkehr möglichst gering sein. Da aber die Infrastruktur für mobile Netzwerke sich ständig verbessert und praktisch nur eine vergleichsweise hohe Latenz festzustellen ist, liegt der Fokus der Optimierung hier in erster Linie auf der Reduzierung der Datenaustauschvorgänge zwischen mobilem Endgerät und entferntem Server und erst danach kommt die Reduzierung der Datenmengen. Ein weiteres Ziel ist es, die notwendigen Anpassungen möglichst gering zu halten oder im besten Fall auf Anpassungen verzichten zu können. Dabei geht es unter anderem darum, den Aufwand einer erneuten Evaluierung zu verringern.

6.1.3 Notwendige Funktionalität

Wegen der Einschränkung auf den mobilen Einsatz der eID-Funktionalität lässt sich ein Minimum an notwendiger Funktionalität definieren. Da ein autonomer Einsatz des mobilen Gerätes angestrebt wird, muss entsprechend des EAC Protokolls [48] (konkret EAC Version 2, im Folgenden einfach als EAC bezeichnet) eine bestimmte Interaktion auf dem Gerät stattfinden. Dazu gehören das Anzeigen der Berechtigungszertifikatsbeschreibung, die Auswahl der auszulesenden Attribute und die Eingabe der PIN. An die PIN-Eingabe ist die lokale Ausführung des PACE-Protokolls gebunden, weil bei einer entfernten Ausführung die PIN das Gerät verlassen müsste, was ein zusätzliches Sicherheitsrisiko darstellt.
Während der Durchführung der EAC Protokolls besteht die Kommunikation des eID-Servers mit dem Client aus ISO/IEC 24727-3 Funktionsaufrufen, die vom SAL des Client bearbeitet werden. Nach der Ausführung des EAC Protokolls besteht ein durch Secure Messaging geschützter Kanal zwischen dem Ausweis und dem eID-Server. Die weitere Kommunikation ist somit kryptographisch abgesichert und besteht nur noch aus APDUs. Diese APDUs werden in der ISO/IEC 24727-4 Funktion *Transmit* versendet, und im IFD-Layer des Clients verarbeitet. Die Software auf dem mobilen Gerät muss diese APDUs vom eID-Server an den Personalausweis und die zugehörigen Antworten zurück an den eID-Server weiterleiten.
Daraus folgt, dass eine Reduzierung der Applikationslogik auf Client-Seite nur bei der Ausführung von TA und CA sowie bei den Framework-Funktionen zur Kommunikation mit der Server-Seite möglich ist.

6.2 Hardware

In diesem Abschnitt wird eine Übersicht über die derzeitig verfügbare Hardware gegeben und erläutert, warum diese momentan ungeeignet ist, um eine mobile AusweisApp umzusetzen. Im Anschluss werden die für die Zukunft angekündigten Lösungen kurz sowie aktuelle Entwicklungen erwähnt. Danach werden die Anforderungen beschrieben, welche eine mobile eCard-API an solche Hardware und zugehörige Software stellt.

6.2.1 Nokia NFC

Die beiden derzeit verfügbaren Mobiltelefone von Nokia mit integriertem NFC-Chip sind das 6131 NFC und das 6210 NFC. Neben den offensichtlichen äußerlichen Unterschieden gibt es auch einige interne Unterschiede. Bei dem neueren 6210 verläuft die Antenne um das gesamte Gehäuse herum. Dies ist für den Anwendungsfall der NFC P2P Kommunikation zwischen zwei Telefonen von Vorteil, da diese so beliebig nebeneinander gelegt werden können. In der Kommunikation mit Chipkarten nach ISO/IEC 14443 Standard [31] wird dies aber zum Problem, da das Feld des NFC-Lesers am stärksten direkt hinter oder direkt vor dem Gerät ist. In mehreren Tests mit JavaCards [11] hat sich gezeigt, dass die verlässlichste Kommunikation zu erreichen ist, wenn die Chipkarte direkt auf der Tastatur liegt. Dies verhindert allerdings die Bedienung des Telefons. An anderen Positionen reicht die Feldstärke nicht aus, um eine stabile Kommunikation zu ermöglichen.
Beim 6131 ist die Antenne direkt auf der Spitze der Klappe. Zusätzlich ist das erzeugte Feld stärker, was die Kommunikation vereinfacht. Allerdings ist die Antennenposition weniger gut geeignet, um eine Chipkarte direkt davor zu positionieren, da eine unpraktische Positionierung verlangt wird.
Beiden Geräten ist gemeinsam, dass ein S40 Betriebssystem eingesetzt wird. Dieses auf NokiaOS basierende System unterstützt kein Multitasking und bietet keine native API für Anwendungen. Einzig in Java geschrieben Midlets auf Basis von J2ME [12] sind ausführbar. Die Anbindung der NFC-Funktionalität wird über JSR-257 [34] realisiert, wie in Abbildung 6.1 ersichtlich. Diese Schnittstelle zum NFC-Controller entspricht nicht dem, wofür der Terminal-Layer des eCard-API-Frameworks konzipiert wurde. Über die JSR-257 wäre eine Implementation des IFD-Layers nur mit großen Einschränkungen möglich, da es programmatisch z.B. nicht möglich ist zu erkennen, ob eine Chipkarte aus dem Feld des Readers entfernt wurde, ohne mit ihr zu kommunizieren, was in diesem Fall eine Ausnahme provozieren würde. Ein Ereignis-getriebenes System wie PC/SC, welches ständig den Leser-/Kartenstatus abfragt, kann damit nur mit Einschränkungen umgesetzt werden. Dazu fehlen die notwendigen Zugriffsmöglichkeiten über JSR-257 [34].
Wie bereits erwähnt, erlaubt das S40 System ausschließlich Anwendungen in Form von J2ME Midlets. Darin liegt das wichtigste Problem beim Einsatz dieser Telefone. Die nach der eCard-API vorgesehene Integration eines Plugins in den Browser ist hier nicht möglich. Ein gleichzeitiger Betrieb des S40 Browser und eines eCard-API-Midlets ist genauso ausgeschlossen. Lediglich die Möglichkeit, ein eigenes Midlet zu erstellen, das die Funktionalität des Browser sowie der eCard-API gleichzeitig übernimmt, bleibt bestehen. Dies erhöht allerdings die Komplexität der Anwendung beträchtlich und ist vermutlich mit den beschränkten Ressourcen gar nicht umsetzbar.
Ein Ausweg könnte die Nutzung der Funktionalität sein, dass der Browser ein Midlet starten kann, sich dabei allerdings beenden muss. Dieses so gestartete Midlet kann daraufhin wieder den Browser starten, muss sich dazu aber ebenfalls wieder beenden. Diese Idee ist aktuell Forschungsgegenstand des Autors von MobilePACE [58], Moritz Horsch.
Ein weiterer interner Unterschied ist die Unterstützung der Java Crypto-API auf dem 6210, welche auf dem 6131 fehlt. Im hier betrachteten Fall werden für die kryptographischen Operationen im Wesentlichen elliptische Kurven eingesetzt. Diese gehören nicht zu den von der

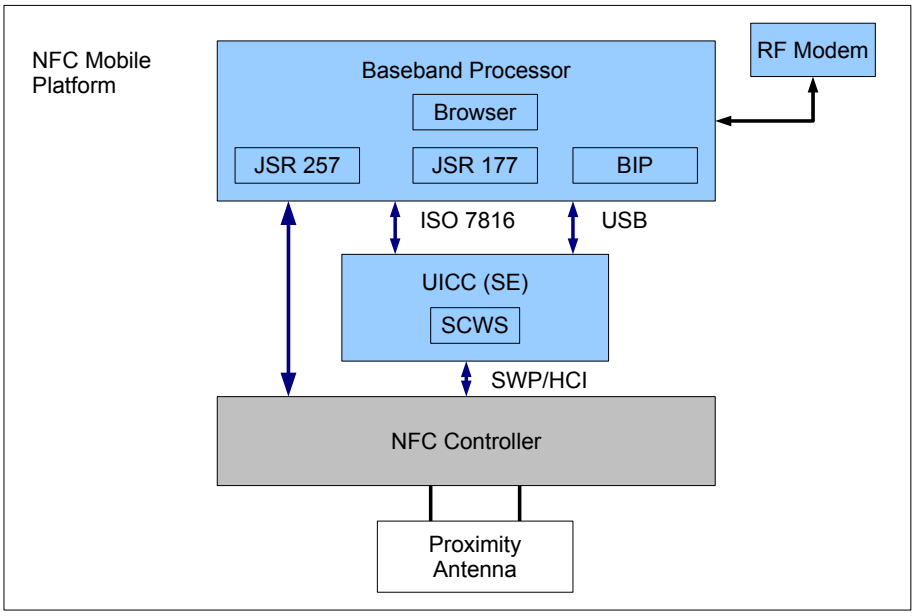

Abbildung 6.1: NFC Mobile Architektur

Java Crypto-API unterstützten Algorithmen. Damit bleibt es weiterhin notwendig, eine externe Kryptografie-Bibliothek wie zum Beispiel BouncyCastle [4] einzubinden.
Schließlich ist die Unterstützung von Extended Length APDUs eingeschränkt. Die Telefone sind nur in der Lage, Daten von einer Chipkarte in Form einer RAPDU bis maximal 255 Bytes inklusive Statuswort zu empfangen. In der gegenwärtigen Form der TR-03110 ist dies kein Problem, da Extended Length APDUs nur für den Versand von Daten an die Chipkarte benötigt werden und die zugehörigen Antworten zumindest bei der Nutzung der eID-Funktionalität immer in 255 Bytes passen, da ein Lesen von größeren Daten immer in kleineren Teilen geschieht. Der Versand von Daten mit mehr als 255 Bytes stellt für die Telefone überraschenderweise kein Problem da.

6.2.2 OpenMoko

Der OpenMoko [20] Freerunner A7 bietet zwar von der Software eine ideale Umgebung zur Implementierung einer mobilen Anwendung, doch fehlt die Leser Anbindung. Da diese Plattform neben einem offenen Software-Stack auch aus offener Hardware besteht, lässt sich ein Leser z.B. über USB mit dem Gerät verbinden. Hier besteht dann aber das Problem, dass kommerzielle Leser in der Regel über ein proprietäres Protokoll kommunizieren, so dass ein entsprechender Treiber notwendig ist. Diese Treiber stehen aber meist nur als Binärpaket zur Verfügung. In aller Regel aber nicht für die verwendete ARM-Architektur.

Sollte es ein offenes Protokoll geben, so dass ein generischer Treiber (CCID) benutzt werden kann oder es möglich ist, einen eigenen zu erstellen, dann stellt die Firmware auf dem Leser das nächste Problem dar. Diese muss in der Lage sein Extended Length APDUs an den Chip zu senden. Hier ist aber 256 Byte das übliche Maximum, da in der Firmware größere APDUs nicht berücksichtigt wurden.
Eine Alternative scheint sich durch die Verwendung von librfid [62] zu ergeben. Damit lassen sich RFID-Leser auf Basis des RC632 [5] ansteuern, ohne dass ein zusätzlicher Treiber notwendig wäre. Dazu gehört der OmniKey 6321 Leser, welcher die Abmessungen eines USB-Sticks besitzt. Prinzipiell ist damit eine Hardwarelösung möglich, die vollständig mit offener Software betrieben wird. Dennoch gibt es jedoch Nachteile wie den Stromverbrauch dieses nicht für den mobilen Einsatz entwickelten Lesers.

6.2.3 Android

Neben OpenMoko gibt es mittlerweile weitere auf Linux basierende mobile Plattformen. Die wichtigste ist dabei Android [1]. Wie schon auf dem OpenMoko ist es möglich Hardware mittels des Linux-Kernels anzusprechen, wenn auch mit einigen Einschränkungen, die meistens von den konkreten Hardwareherstellern abhängig sind. Derzeit sind noch keine Android-basierten Geräte mit integrierten NFC-Chips auf dem Markt verfügbar. Daher muss ein externer Leser angeschlossen werden. Obwohl eine USB-Schnittstelle mittlerweile an fast jedem Gerät vorhanden ist, ist diese in den seltensten Fällen in der Lage im Host-Mode betrieben zu werden. Dieses ist allerdings Voraussetzung, um USB-Geräte wie den oben beschriebenen RFID-Leser zu verwenden.

6.2.4 Zukünftige Hardware

Vor einigen Jahren schon hatte Nokia die Integration von NFC in die Mehrheit ihrer Mobiltelefonserien beabsichtigt, dennoch ist dies momentan noch immer nicht der Fall. Mehrere Hersteller haben zwar noch für dieses Jahr Modelle auf Basis von Android angekündigt, doch ist deren Verfügbarkeit noch keinesfalls gewiss. Apple scheint für ein zukünftiges iPhone eine Integration von NFC anzustreben, aber es gibt noch keinen festen Termin.
Absehbar ist, dass eine breite Verfügbarkeit wohl in den kommenden Jahren gegeben sein sollte. Dabei bleibt zu hoffen, dass als APIs für die Kommunikation mittels NFC nicht nur JSR-257 [34] mit ihren beschriebenen Unzulänglichkeiten zum Einsatz kommt, sondern Schnittstellen bereit stehen, über die ein Verhalten ähnlich zu PC/SC [25] zu realisieren ist wie z.B. JSR-268 [35]. Damit würde die Umsetzung des IFD-Layers des eCard-API-Frameworks sowie die Kommunikation mit ISO/IEC 14443 Smartcards im Allgemeinen erleichtert. Allerdings existiert im Rahmen der bestehenden NFC-Spezifikationen derzeit noch keine Alternative zu JSR-257 für die Kommunikation zwischen Applikationen und dem NFC-Controller. Daher beseht die Hoffnung darin, dass Plattformen wie Android es Applikationen ermöglichen, über eine Kernel- oder Treiber-Schnittstelle mit dem NFC-Modul zu kommunizieren. Ein weiterer Ansatz in diesem Kontext könnte es sein, von der Applikation über die SIM-Karte (UICC) mit dem NFC-Modul zu kommunizieren. Dabei könnte ein Teil der Logik auf der Client-Seite direkt in die UICC verlagert werden. Das

Problem dabei ist allerdings die Verbindung der UICC zum NFC-Modul. Wie in Abbildung 6.1 sichtbar, kommen das Single Wire Protocol (SWP) und das NFC Controller Interface (NCI) zum Einsatz. Leider ist eine Spezifikation zum NCI noch nicht verfügbar. Aus den Informationen zum SWP ergibt sich, dass es dabei um den Anwendungsfall geht, bei dem der Zugriff vom NFC-Modul auf die UICC per APDUs im Vordergrund steht und nicht in die andere Richtung, wie es für den Einsatz des Personalausweises erforderlich wäre.

Kapitel 7

Lösungsansätze

In diesem Kapitel werden die Ansätze für einen mobilen Einsatz der eCard-API beschrieben. Dabei lassen sich diese Ansätze in zwei wesentliche Kategorien einteilen. Die Erste erhält weitestgehend die Kompatibilität zur bestehenden Infrastruktur in Form des eID-Server und der eCard-API. Die Zweite verzichtet darauf und verlangt zwangsläufig Anpassungen der bestehenden Systeme. In Anbetracht des notwendigen Anpassungsaufwandes kann eine Lösung der zweiten Kategorie nur durch messbare Vorteile für den angestrebten Einsatzzweck legitimiert werden. Zusätzlich ist noch ein hybrider Ansatz in Form einer Proxy basierenden Lösung denkbar, welche einerseits die Kompatibilität in Richtung eID-Server erhält, dafür aber Spielraum in der Verbindung zum mobilen Endgerät für die Benutzung eines speziell angepassten Protokolls lässt.

7.1 Eingeschränkte eCard-API

Eine Anforderung an die eCard-API ist es, sämtliche Anwendungen auf allen eCards zu unterstützen. Ausgehend von der Annahme, dass für den Bürger der Einsatz der eID Funktion des Personalausweises mit einem mobilen Endgerät der Anwendungsfall ist, welcher das höchste Potenzial hat, tatsächlich in einer ökonomisch bedeutenden Größenordnung verwendet zu werden, lässt sich diese Anforderung aufweichen. Man kann sogar soweit gehen, dass man in einem ersten Schritt nur die eID-Funktionalität des Personalausweises unterstützt und damit die Unterstützung für andere eCards sowie andere Funktionen des Personalausweises verliert.
Mit diesen Einschränkungen kann man ein eCard-API-Framework erstellen, welches von den verfügbaren Schnittstellen nur die absolut notwendigen anbietet.
Es gibt keine Notwendigkeit mehr, die Verwaltung von mehreren Terminals oder Chipkarten zu unterstützen. Im Kontext von mobilen Geräten ist dies sinnvoll, da es keine redundante Komponenten gibt und nur ein einziger Leser vorhanden ist. Dieser wird wiederum nur eine kontaktlose Schnittstelle besitzen. Obwohl es nach ISO/IEC 14443 [31] und der NFC Spezifikation [18] möglich sein sollte mit einem Leser mit mehreren Karten oder Tags zu kommunizieren, wird dieses praktisch meist nicht unterstützt. Somit gibt es nur eine Karte an einem Leser.

Da bislang nur der Personalausweis die eID-Funktionalität anbietet, wird keine Unterstützung für andere Chipkarten benötigt. Damit besteht keine Notwendigkeit mehr mit CardInfo-Dateien zu arbeiten. Im Management-Interface kann auf die Funktionen zur Karten- und Terminalverwaltung verzichtet werden. Abhängig davon, wie die Anzeige der Zertifikatsberechtigungen implementiert ist, kann auf den Trusted-Viewer verzichtet werden. Dieses ermöglicht den Verzicht auf die Funktionen der Trusted-Viewer-Verwaltung. Das gleiche gilt für die Funktionen für die Identitäts- und Dienstverwaltung, welche für die eID-Funktionalität keine Bedeutung haben. Es bleiben also die Funktionen zur Verwaltung des Frameworks an sich. Allerdings sind diese für den eingeschränkten Anwendungsfall nicht von Bedeutung, sondern eher für lokal operierende Applikationen.

- *InitializeFramework*
- *TerminateFramework*
- *FrameworkUpdate*
- *GetDefaultParameters*
- *SetDefaultParameters*

Die Funktionen im eCard-Interface sind im Kontext der eID-Funktionalität unbedeutend. Damit kann auf dieses Interface verzichtet und der Identity-Layer weggelassen werden. Das Support-Interface enthält nur Funktionen, die nicht mit der Chipkarte kommunizieren. Daher hat dieses Interface keine Bedeutung und kann weggelassen werden.
Praktisch relevant für die eID-Funktionalität sind sieben verschiedene Aufrufe und entsprechende Antworten, die zwischen lokalem und entfernten eCard-API-Framework ausgetauscht werden. Diese Aufrufe befinden sich alle im ISO24727-3 und im IFD-Interface ergänzt um *StartPAOS*.

- *StartPAOS*
- *DIDAuthenticate*
- *Transmit*
- *CardApplicationEndSession*
- *CardApplicationStartSession*
- *CardApplicationPath*
- *CardApplicationConnect*

Dabei sind die letzten drei nicht absolut notwendig beziehungsweise sie müssen nicht von der eID-Server-eCard-API-Instanz gerufen werden, wenn die Client-eCard-API-Instanz dieses automatisch tut, so wie das in TR-03112-7 [44] beschrieben ist und von der gegenwärtigen Implementierung (AusweisApp [2]) realisiert wird.
Die damit verbleibenden Funktionen sind überschaubar komplex mit der Ausnahme von

DIDAuthenticate. Letztere hat wegen ihrer Abhängigkeit von dem durch den im *DIDName* Parameter identifizierten Protokoll den dynamischen IN/OUT-Parameter *authenticationProtocolData*, dessen Inhalt und Struktur durch das Protokoll festgelegt wird. Dahinter verbirgt sich der komplexeste Teil der Protokollabarbeitung von EAC auf der Client-Seite, der durch diese Aufrufe angestoßen wird. Natürlich lassen sich die Einschränkungen bezüglich der unterstützten Funktionalität auch auf diesen Funktionsaufruf übertragen. Von den in TR-03112-7 [44] beschriebenen Protokollen ist nur EAC relevant. Das heißt, es könnte auf die Unterstützung der weiteren Protokolle verzichtet werden. Namentlich sind das *PIN Compare*, *Mutual Authentication* und *RSA Authentication*.
Für die Umsetzung einer so eingeschränkten Lösung mit ihren vier notwendigen Funktionsaufrufen ist es nötig, dieselben Komponenten zu implementieren, aus denen die AusweisApp besteht. Es wird ein Browser-Plugin benötigt, welches die eingeschränkte eCard-API-Instanz lokal auf dem Client zur Verbindungsaufnahme mit dem eID-Server veranlasst. Diese lokale Instanz benötigt für ihre Kommunikation mittels PAOS [14, 13] über einen TLS/PSK [53] gesicherten Kanal entsprechende Bibliotheken. Für die kryptographischen Operationen innerhalb des EAC Protokolls sowie für das Secure Messaging ist Unterstützung ebenfalls notwendig. Die Integration der eCard-API-Funktionalität in das Browser-Plugin möglich, liefert aber, durch die Notwendigkeit des zweiten Kanals zum eID-Server, nur den Vorteil, dass auf IPC verzichtet werden kann.
Die tatsächlichen Einsparungen bei dieser Lösung liegen also zum größten Teil in der verringerten Komplexität durch das reduzierte Interface. Der größte Vorteil ist, dass der eID-Server nicht angepasst werden muss. Dennoch ist der Ressourcenbedarf für den Ablauf des Protokolls sowie für die Kommunikation, das Generieren und Parsen von Nachrichten, noch vergleichsweise hoch bzw. der AusweisApp gleichwertig. Das lässt sich durch den Anspruch der Kompatibilität zur eCard-API nicht ernsthaft verbessern.

7.2 ISO/IEC 24727 Konfigurationen und Proxies

Als Alternative oder Ergänzung zu einer eingeschränkten eCard-API bietet sich eine andere Verteilung der Komponenten im Vergleich zu der in der AusweisApp gewählten an. Dabei kann man hier wieder einige Unterscheidungen treffen. So kann die Verteilung durch eine andere ISO/IEC 24727 Stack-Konfiguration oder durch eine Proxy-Lösung ermöglicht werden. Die veränderte Stack-Konfiguration ändert grundsätzlich nichts am Protokoll, verlagert allerdings einige Schritte und Berechnungen vom Client auf die eID-Server-Seite. Diese Verschiebung von Funktionalität muss der Server-Seite vor Ablauf des EAC-Protokolls signalisiert werden und verlangt daher geringe Anpassungen an den ausgetauschten Nachrichten. Bei einer Proxy-Lösung kommt eine weitere Komponente ins Spiel. Diese vermittelt zwischen eCard-API konformen Aufrufen und einem speziell an die Zielplattform und den Einsatzzweck angepassten Protokoll.

7.2.1 Remote-ICC-Stack

Wie eingangs beschrieben ist das eCard-API-Framework eine Mischkonfiguration aus dem ISO/IEC 24727-4 Remote-Loyal-Stack und dem Remote-ICC-Stack. Der Remote-Loyal-

Stack wird benutzt, um EAC durchzuführen, wobei die einzelnen Subprotokolle im SAL des Clients ablaufen. Der Remote-ICC-Stack wird verwendet, um im Anschluss an EAC mit Secure Messaging geschützte APDUs direkt vom eID-Server an den IFD-Layer des Clients zu übergeben. Abbildungen 7.1 und 7.2 veranschaulichen dieses. Ein Ansatz zur Verringerung der Komplexität ist es, auf den SAL im Client zu verzichten, also eine reine Remote-ICC-Stack Konfiguration zu verwenden.

Bei der ISO/IEC 24727-4 Remote-ICC-Stack Konfiguration ist die Trennung der Kompo-

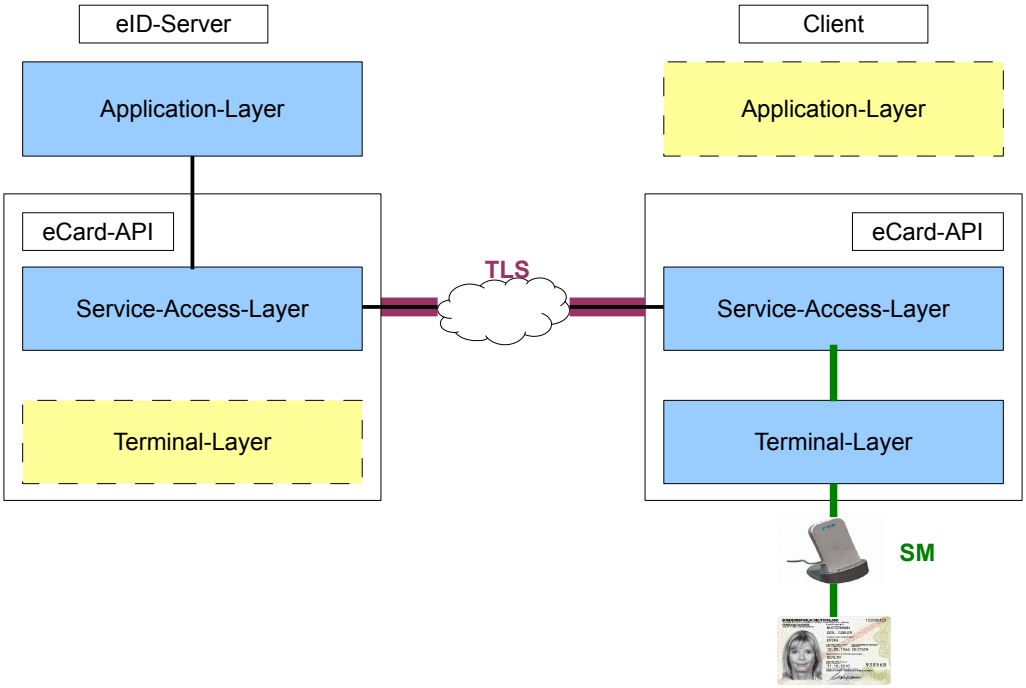

Abbildung 7.1: eCard-API-Framework Kommunikation vor EAC

nenten mittels der Trusted-Channel-API auf der Ebene des IFD-Interfaces. Das bedeutet, dass durch den Trusted-Channel während der relevanten Kommunikation nur noch kartenspezifische APDUs gesendet werden. Daneben bietet das IFD-Interface Funktionen zur Terminal- und Slot-Verwaltung sowie drei Funktionen, die sich auf den Benutzer beziehen. Die Umsetzung der zur Kartenkommunikation notwendigen Protokolle in APDUs muss schon auf dem entfernten System stattfinden. Dieses schließt die aufwendige Kryptooperationen mit ein. Bezüglich einer Reduzierung der notwendigen Rechenleistung auf dem Endgerät ist das eine Verbesserung. Allerdings hat diese Lösung auch Nachteile. Den wichtigsten stellt der erhöhte Kommunikationsaufwand dar. So werden im konkreten Fall von EAC aus einem SAL Aufruf zur Durchführung von PACE sechs einzelne APDUs zur Karte gesendet und die entsprechenden Antworten ausgewertet, wie in Tabelle 7.1 dargestellt.

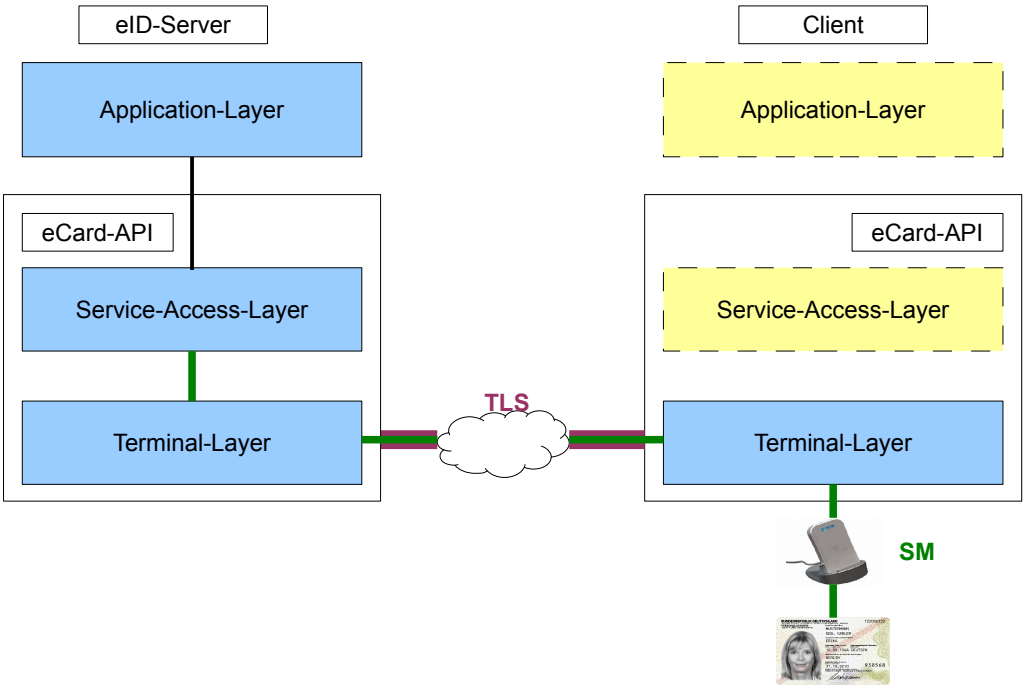

Abbildung 7.2: eCard-API-Framework Kommunikation nach PACE, TA und CA

Dabei wird die einzelne APDU für das Lesen von EF.CardAccess dem PACE-Vorgang zugeordnet. Ähnlich verhält es sich bei CA und TA. Praktisch würde dieses bei der Nutzung der eID-Funktionalität des Personalausweises bedeuten, dass aus den essentiellen drei *DIDAuthenticate* SAL-Aufrufen (mit Optimierung zwei) 16 einzelne APDU-Austauschvorgänge werden. Die Anzahl der zur Verifikation übermittelten Zertifikate ist in Tabelle 7.1 beispielhaft auf zwei festgelegt. Im Falle eines gewechselten CVCA-Zertifikates, müsste zusätzlich noch ein CVCA-Link-Zertifikat übermittelt werden, welches wiederum zwei zusätzliche APDUs bedeuten würde. Daher stellen die 16 Austauschvorgänge eine approximierte Anzahl dar.
Das Problem, dass die auszulesenden Daten dem Nutzer angezeigt und von ihm bestätigt werden müssen, sei hier vorläufig ignoriert. Es würde aber auf zusätzliche Komplexität hinauslaufen, die so im IFD-Layer nicht angedacht ist.
Zusätzlich wird in TR-03110 [48] die Benutzung von Extended Length APDUs eingeschränkt. Diese dürfen nur für *PSO:Verify Certificate*, *MSE:Set KAT* (deprecated), *General Authenticate* und *External Authenticate* benutzt werden. Das Lesen von Daten soll daher mit *Select* und darauf folgenden *Read Binary* APDUs erfolgen, die die Daten in mehreren Stücken lesen. Ausnahmen davon sind zugelassen, wenn der Chip im ATR/ATS oder im EF.ATR seine exakten Puffergrößen bekannt gibt. Diese Art des Lesens hat allerdings den

Operation	APDUs	Anzahl APDUs
Read EF.CardAccess	Read Binary	1
PACE	MSE:Set AT General Authenticate (Get Encrypted Nonce) General Authenticate (Map Nonce) General Authenticate (Perform Key Agreement) General Authenticate (Mutual Authentication)	5
TA	MSE:Set DST PSO:Verify Certificate (DV) MSE:Set DST PSO:Verify Certificate (Terminal) MSE:Set AT Get Challenge External Authenticate	7
Read EF.CardSecurity	Read Binary	1
CA	MSE:Set AT General Authenticate (Ephemeral Public Key)	2
Summe		16

Tabelle 7.1: EAC APDUs

Nachteil, dass die Länge der Daten benötigt wird, die in der Antwort auf die *Select* APDU enthalten sein kann, um die passende Sequenz von *Read Binary* APDUs zu erzeugen.
Diese hohe Anzahl von APDU-Austauschvorgängen hat in einem anvisierten mobilen Einsatzszenario, in dem vermutlich auf eine UMTS-Internet-Verbindung zurückgegriffen wird, negative Auswirkungen, da trotz genügend vorhandener Bandbreite durch die erhöhte Latenz die Zeit für einen Durchlauf signifikant ansteigen würde. Geht man allerdings von der Annahme aus, dass die Latenz in der Verbindung nur eine untergeordnete Rolle spielt, dafür aber Rechenleistung als primäres Optimierungsziel angesehen wird, bietet diese Konfiguration durchaus Potenzial.
Ein weiterer Nachteil ergibt sich aus den Sicherheitskriterien, unter denen die Lösung betrachtet wird. Prinzipiell kann in heutigen Standard-Computern kein vollständiges Vertrauen garantiert werden, da die benutzte Software einfach zu komplex ist. Die Lösung in Form der AusweisApp zur Nutzung des Personalausweises umgeht dieses Problem durch eine Ende-zu-Ende verschlüsselte Übertragung der Daten aus dem Personalausweis zum eID-Server. Der Transport der Daten vom eID-Server zum Dienstanbieter findet ebenfalls verschlüsselt statt. Selbst wenn die lokale Software nicht vertrauenswürdig ist, kann ein potenzieller Angreifer mit vollständiger Kontrolle über den Rechner mit der AusweisApp nicht an den Inhalt der Daten gelangen. Eine Ausnahme sind die vom Dienstanbieter dem Nutzer möglicherweise nachträglich im Browser angezeigten Daten, was außerhalb der eCard-API-Kommunikation liegt.
Allerdings besteht hier die Möglichkeit, dass Daten von irgendeinem eID-Service ausgelesen werden, ohne dass der Ausweisinhaber den Vorgang genau so wahrnimmt. Um den Aus-

lesevorgang zu authentisieren, ist eine Interaktion vom Inhaber vorgesehen. Dabei sollen die auszulesenden Datengruppen angezeigt und gegebenenfalls abgewählt werden können. Daneben müssen Informationen über den auslesenden Dienst beziehungsweise Zertifikatsinhaber präsentiert werden. Auf Basis dieser Daten trifft der Ausweisinhaber die Entscheidung, den Vorgang zu erlauben und bestätigt das durch Eingabe seiner PIN. Hier könnte ein Angreifer zum einen die Anzeige der auszulesenden Daten und des Berechtigungszertifikates manipulieren, um so den Nutzer zur Freigabe zu veranlassen. Zum anderen kann die PIN während der Eingabe mitgelesen werden.
Um so einen Angriff zu erschweren, ist der Einsatz von Kartenlesern nach TR-03119 [46], die als Standard- und Komfort-Leser bezeichnet werden, vorgesehen. Im Gegensatz zum so genannten Basis-Leser haben sie das PACE Protokoll direkt im Leser implementiert. Damit wird die PIN direkt auf dem Pinpad des Lesers eingegeben und ist nicht mehr so einfach abhörbar, da sie den Leser nicht verlässt. Zusätzlich muss der Komfort-Leser ein Display bieten, auf dem der Inhalt des Berechtigungszertifikates angezeigt werden soll. Auch die vorher angezeigten geforderten Berechtigungen, die vom Nutzer bestätigt wurden, sollen hier sichtbar sein. Da es nicht geplant ist an dieser Stelle noch einmal Einschränkungen vorzunehmen, muss der Nutzer hier die Rechte im Leser-Display mit den vorher bestätigten vergleichen. Die in PACE benutzen Berechtigungen müssen dann im Verlauf der TA mit den angezeigten abgeglichen werden. Ein Komfort-Leser sollte so eine wirksame Schutzmöglichkeit gegen die beschriebenen Angriffe trotz eines potenziell kompromittierten Rechners bieten.
Allerdings verändert sich hier nun die Ansteuerung. Das ursprüngliche *DIDAuthenticate* für PACE muss direkt an den Leser weitergegeben werden. Dazu gibt es den Funktionsaufruf *EstablishPACEChannel(InputData)*. Dieser leitet die in Abbildung 7.3 dargestellten Vorgänge ein. In TR-03119 [46] wird das Mapping der für PACE relevanten Funktionen *GetReadersPACECapabilities* und *EstablishPACEChannel* auf die PC/SC Funktion *SCardControl* beschrieben. Im Kontext des eCard-API-Frameworks ist die Ansteuerung dieser Funktionalität aber noch nicht final spezifiziert. Es gibt die Möglichkeit, dieses mittels der IFD-Interface Funktion *IFDControl* zu implementieren. Ebenfalls vorstellbar ist aber eine Implementation, bei der diese Funktionalität vom Leser direkt über PC/SC abgefragt wird. Ist ein Leser mit diesen erweiterten Funktionen verfügbar, wird in der Protokollausführung im Service-Access-Layer auf Karten-Seite eine direkte Kommunikation mit dem Leser über PC/SC begonnen und das IFD-Interface umgangen. Dabei ergibt sich allerdings das Problem, dass dann die restliche Kommunikation am IFD-Interface vorbei geführt werden müsste oder aber das *SlotHandle* zurückgegeben wird, damit IFD-Interface Funktionen wie *Transmit* aufgerufen werden können. Dies ist beim Aufruf von *EstablishPACEChannel* aber nicht vorgesehen.
Die tatsächliche Umsetzung ist daher über das IFD-Interface zu realisieren. Dazu fehlt aber noch die nötige Spezifikation. Zumindest ist in den bislang veröffentlichten Technischen Richtlinien die genaue Abbildung der Funktionen des IFD-Layers auf die PC/SC Funktionen nur mittels Empfehlungen angegeben, welche die Anbindung eines Komfort oder Standard-Lesers noch nicht berücksichtigen.
Unter der Annahme, das es einen IFD-Interface Aufruf gäbe, der *EstablishPACEChannel* aufruft, muss dies dem zugehörigen SAL bekannt sein, damit dort nicht die APDUs

Chip	*Kartenleser*	*Host-Rechner*
	EstablishPACEChannel(InputData)	
Extrahieren der für PACE notwendigen Parameter aus EF_CardAccess		
	PIN-ID aus InputData Prüfung auf Rolle Authentisierungsterminal	
MSE: Set AT General Authenticate Step 1+2		
	Extrahieren und Anzeigen Berechtigter aus InputData Anzeige der Berechtigungen aus CHAT Berechnung der Variablen StoreHash Eingabe der PIN Berechnung Kπ Entschlüsselung 'Encrypted Nonce' mit Kπ Bereinigen des Speichers	
General Authenticate Step 3+4 Berechnung des Schlüsselmaterials für SM		
	Bei ‚PSO: Verify Certificate' Vergleich HASH aus Certficate Extensions mit Variable StoreHASH	

Abbildung 7.3: Auszug [46] Abb. 8: Ablauf PACE mit Display

für PACE erstellt werden, sondern die entsprechende IFD-Layer Funktion gerufen wird. Dieses kann durch das Hinzufügen geeigneter Informationen in die *StartPAOS* Nachricht geschehen. Damit könnte eine Remote-ICC-Stack Konfiguration auf einem Endgerät realisiert werden, das ein Display, die Möglichkeit zur PIN-Eingabe und die Rechenleistung zur Durchführung von PACE hat. Hier muss die angedachte Funktion des Komfort-Lesers um die Fähigkeit zur Anzeige der zu lesenden Daten sowie deren Abwahl durch den Nutzer erweitert werden. Je nachdem, ob die Daten, die ausgelesen werden dürfen, schon vom Dienstanbieter selbst eingeschränkt werden sollen gegenüber den im Berechtigungszertifikat erlaubten, müssen die Parameter in *InputData* und die zugehörigen Rückgabewerte von *EstablishPACEChannel(InputData)* um eine Bitmaske erweitert werden, um die vom Nutzer bestätigten Attribute dem eID-Server mitzuteilen. Die aktuelle Version des eID-Servers unterscheidet bei der Eingabe zwischen *RequiredChat* und *OptionalChat*, um dem Endanwender optional abwählbare Attribute anzeigen zu können. Der im Zertifikat enthaltene *CHAT* kann wiederum noch mehr Rechte beinhalten, als die Kombination von *RequiredChat* und *OptionalChat*. Daher sind für die Eingabe zwei weitere Bitmasken erforderlich, welche den *RequiredChat* und *OptionalChat* enthalten.
Mit so einer Ansteuerung des PACE-Protokolls und den eingangs beschriebenen Voraussetzungen, dass das verwendete Endgerät in der Lage sein soll eigenständig PACE durchzuführen, muss also mit einen einzelnen Aufruf im IFD-Interface möglich sein, das komplette PACE-Protokoll inklusive PIN-Eingabe auf dem Endgerät durchzuführen. Dabei muss die

Anzeige der zu lesenden Attribute sowie deren Abwahl durch den Nutzer in denselben Aufruf integriert werden, da im IFD-Layer sonst keine Funktionalität für diese Anzeige zur Verfügung steht. Es gibt zwar die Funktion *Output*, diese kann aber maximal einen String auf einem Display darstellen, was hier unzureichend ist.
Bei solch einer Umsetzung kommt es nur noch bei TA und CA zu zusätzlichen Kommunikationsvorgängen, wie aus Tabelle 7.1 zu entnehmen ist. Hier wird das Lesen des EF.CardSecurity mittels einer einzelnen *Read Binary* APDU angedeutet. Dieses ist aber praktisch so nicht möglich. Zum einen erlaubt TR-03110 [48] nicht die uneingeschränkte Verwendung von Extended Length APDUs, zum anderen ist EF.CardSecurity zu groß, um bei einer erwarteten Kartenpuffergröße zwischen 1199 und 1500 Bytes selbst mittels Extended Length APDUs mit weniger als zwei APDUs vollständig gelesen zu werden. Auf die Lösung dieses Problems wird im Folgenden noch genauer eingegangen. Im Vergleich zu dem optimierten EAC-Ablauf, kommt es zu neun zusätzlichen Kommunikationsvorgängen. Mit dem nicht optimierten sind es acht. Es ist also ungefähr eine verdreifachte Anzahl der Kommunikationsvorgänge zu erwarten.
Da aber mehrere APDUs innerhalb von TA keine Abhängigkeiten zu den Antworten ihrer Vorgänger haben, gibt es hier noch Optimierungspotenzial. So lassen sich die ersten sechs APDUs in einem einzigen *Transmit* zusammenfassen. Dabei spielt die Anzahl der zu verifizierenden Zertifikate keine Rolle. Die zwei APDUs je Zertifikat sind in jedem Fall unabhängig von den Ergebnissen der Berechnungen des Protokolls und damit im Voraus berechenbar. Für *MSE:Set AT* und das *Get Challenge* gilt das genauso. Kombiniert man nun noch die *External Authenticate* APDU aus TA mit dem folgenden *Read Binary* und die beiden Befehle in CA miteinander, ist eine Reduktion auf vier Kommunikationsvorgänge möglich. Da das ephemerale Schlüsselpaar für CA schon in oder sogar vor TA erstellt werden muss, lassen sich auch die CA APDUs an dieser Stelle vorausberechnen und mit den restlichen von TA zusammen übertragen. Es ist nicht notwendig den Inhalt von EF.CardSecurity vor der Ausführung von CA zu kennen, da die notwendigen Daten wie Domain Parameter schon in EF.CardAccess enthalten sind und der öffentliche Schlüssel des Chips erst für die Verifikation der Antwort des letzten *General Authenticate* benötigt wird. Dieser Vorgang, bei dem die einzelnen APDUs in maximal großen Transaktionen zusammengefasst sind, ist in Tabelle 7.2 dargestellt. Damit ist die Anzahl der Kommunikationsvorgänge im IFD-Layer identisch zu denen auf SAL-Ebene.
Auch die Optimierung durch den Einsatz des *Get Challenge* im direkten Anschluss an PACE, wodurch die verbleibenden TA APDUs komplett zusammengefasst werden können, ist in dieser Konfiguration nutzbar. Dabei kann auf die gleiche Art und Weise ein Kommunikationsvorgang eingespart werden. Die daraus resultierenden APDUs, zusammengefasst in den einzelnen Kommunikationsvorgängen, sind in Abbildung 7.3 dargestellt. Diese Optimierung setzt aber wiederum voraus, dass bei der Ansteuerung von PACE mittels *EstablishPACEChannel* das *Get Challenge* mit ausführt und das Ergebnis in ihren Rückgabeparametern enthält. Dieses erfordert eine zusätzliche Erweiterung an der Datenstruktur *OutputData*, um neben dem effektiven, vom Nutzer eingeschränkten CHAT ebenfalls die Challenge zurückgeben zu können. Im Grunde wird hier die Funktionalität aus TR-03119 [46], die ein höheres Sicherheitsniveau als ein Basis-Leser gewährleisten soll, dazu benutzt die Schnittstelle einzuschränken. Am Ende ist das Sicherheitsniveau beim

Operation	APDUs	Anzahl APDUs
PACE	Read Binary MSE:Set AT General Authenticate (Encrypted Nonce) General Authenticate (Map Nonce) General Authenticate (Perform Key Agreement) General Authenticate (Mutual Authentication)	6
TA	MSE:Set DST Verify Certificate (DV) MSE:Set DST Verify Certificate (Terminal) MSE:Set AT Get Challenge	6
TA Read EF.CardSecurity CA	External Authenticate Read Binary MSE:Set AT General Authenticate (Ephemeral Public Key)	4
Summe		16

Tabelle 7.2: EAC APDUs, zusammengefasst in Transaktionen

Operation	APDUs	Anzahl APDUs
PACE TA	Read Binary MSE:Set AT General Authenticate (Encrypted Nonce) General Authenticate (Map Nonce) General Authenticate (Perform Key Agreement) General Authenticate (Mutual Authentication) Get Challenge	7
TA Read EF.CardSecurity CA	MSE:Set DST Verify Certificate (DV) MSE:Set DST Verify Certificate (Terminal) MSE:Set AT External Authenticate Read Binary MSE:Set AT General Authenticate (Ephemeral Public Key)	9
Summe		16

Tabelle 7.3: optimierte EAC APDUs, zusammengefasst in Transaktionen

Einsatz einer mobilen Plattform, die nur die IFD-Schnittstelle bedient, wieder mit dem eines Basis-Lesers vergleichbar, da ein Benutzer prinzipiell der Software auf so einer mobilen Plattform kein höheres Vertrauen entgegenbringen kann als einem gewöhnlichen Standard-PC. Es sind also wieder dieselben Angriffe wie beim Einsatz des Basis-Lesers möglich.
Der praktische Unterschied dieser Lösung im Vergleich zu der eingeschränkten eCard-API besteht zum einen in einer weiteren Einschränkung des Interfaces auf nur noch zwei wesentliche Funktionen im IFD-Layer, *IFDControl* und *Transmit*, sowie das obligatorische *StartPAOS*. Zum anderen wird die Abarbeitung des Protokolls von TA und CA zum größten Teil vom Client zum eID-Server verlagert. Dabei ist die Logik, welche TA und CA im SAL zu APDUs umwandelt allerdings vergleichsweise simpel. Eine gravierende Verringerung der Applikationslogik wird dadurch nicht erreicht. Dieses wäre nur möglich, wenn auf die lokale Ausführung von PACE verzichtet werden würde. Dieses steht aber im Widerspruch zu den vorher formulierten minimalen funktionalen Anforderungen und würde das Sicherheitsniveau entschieden senken.
Abgesehen vom selbständig ausgeführten PACE-Protokoll, leitet der Client nur noch APDUs an den Personalausweis weiter. Nebenbei muss aber auf dem Client eine gewisse Funktionalität verbleiben, um die Leserechte aus dem Terminal-Berechtigungszertifikat dem Nutzer anzuzeigen sowie deren Übereinstimmung während TA zu überprüfen, wie dies ein Komfort-Leser tun muss.
Zusätzlich bleibt es damit immer noch notwendig auf dem Client einen Software-Stack zu haben, der den TLS/PSK-Kanal aufbauen sowie eCard-API-konforme PAOS Nachrichten versenden und empfangen kann. Die Realisierung von PACE sollte dabei kein größeres Problem sein, da es gegenwärtig schon Implementierungen gibt, die auf mobilen Plattformen funktionieren [58] oder unter einer Open Source Lizenz stehen [57]. Zusätzlich sind dem Autor noch bislang nicht veröffentlichte, eigenständige Implementation in Python und C/C++ bekannt.
Die Nachteile liegen im Wesentlichen in dem verringerten Funktionsumfang auf Client-Seite, die mit dem eingeschränkten Interface einhergeht. So lässt sich zwar EAC im IFD-Layer mit genauso vielen Funktionsaufrufen wie im SAL ausführen, für andere Protokolle gilt dieses aber nicht zwangsläufig. Daher ist diese Lösung unter Umständen nur schwer zu erweitern.
Natürlich bedeutet diese Konfiguration eine Anpassung beim eID-Server. Dieser benötigt nun in seinem SAL die Logik, welche vorher auf dem Client war. Da aber aktuell auf beiden Seiten eine eCard-API-Instanz existiert, die zudem noch vom selben Hersteller stammt, sollte dieser Logik-Transfer keine großen Komplikationen darstellen.
Verglichen mit der eCard-API-Lösung bekommt der eID-Server so potenziell Zugriff auf die Zwischenergebnisse der kryptographischen Berechnungen bei TA und CA. Allerdings sind alle diese Zwischenergebnisse Daten, die bei der eCard-API Lösung sowieso zwischen Client und eID-Server ausgetauscht werden. Das ephemerale Schlüsselpaar, welches in TA vom Terminal erstellt wird, wird direkt auf dem eID-Server generiert. Der zu signierende Zufallswert aus dem Chip wird an den eID-Server gesendet. Der öffentliche Schlüssel des statischen Chip-CA-Schlüsselpaares sowie die Domainparameter sind Teil von EF.CardSecurity, das genauso an den eID-Server gesendet wird. Als letztes wird der Zufallswert zur Erstellung der Secure Messaging Schlüssel in CA an den eID-Server übermittelt. Daraus folgt, dass

der eID-Server auf keinerlei zusätzliche Daten während der Durchführung von TA und CA Einfluss nehmen kann.
Nach dem Abschluss von CA gibt es keinen Unterschied mehr, da danach nur noch *Transmit* Aufrufe erfolgen, denn das Secure Messaging besteht dann zwischen Chip und eID-Server, sodass die APDUs schon direkt im eID-Server erstellt werden müssen.
Für einen externen Angreifer ändert sich durch diese Konfiguration nichts. Es ist weiterhin nur der TLS-Kanal, dessen Schlüssel aus dem PSK abgeleitet ist, sichtbar.

7.2.2 Weitere Einschränkung

Die Konfiguration des Remote-ICC-Stacks liefert eine Grundlage für ein eigenes Protokoll, bei dem die übertragenen Daten möglichst gering sein sollen. Nur APDUs zu übertragen ist hier ein sinnvoller Ansatz, da APDUs relativ wenig Overhead haben. Das sind jeweils nur wenige Bytes, die zu den Daten, welche sowieso übertragen werden müssen, addiert werden. Verzichtet man auf Webservices und damit die XML-Kodierung zugunsten einer möglicherweise proprietären aber im Kontext von APDUs platzsparende Kodierung, welche eine Umsetzung in konkrete Leser, PC/SC usw. Befehle einfach ermöglicht, kommt man zu einer Lösung, die unter den genannten Einschränkungen durchaus Potenzial bietet. Dabei könnte z.B. durch den Einsatz von ASN.1 eine effiziente Kodierung realisiert werden. Da die Daten der Zertifikate dem Nutzer anzuzeigen sind und diese selbst schon im ASN.1 Format vorliegen, müssen Bibliotheken für den Umgang mit ASN.1 sowieso zur Verfügung stehen.
Allerdings bietet eine Optimierung der übertragenen Datenmengen nicht wirklich viel Potenzial. Die gesamte eCard-API-Kommunikation eines eID-Protokolldurchganges ist ohne TLS/SSL ungefähr 37 KB groß. Mit TLS/SSL und dem Overhead durch TCP/IP inklusive den durch das SAML Protokoll verursachten Nachrichten, ist mit einem Datenvolumen von ca. 100 KB zu rechnen. Die tatsächlichen Daten, die für die Protokollausführung notwendig sind, lassen sich nicht reduzieren, egal ob diese auf SAL-Ebene in SOAP-Nachrichten oder schon in APDUs auf IFD-Layer-Ebene ausgetauscht werden. Durch eine optimale Kodierung, welche z.B. auf den XML- und SOAP-Header-Overhead verzichtet, ist zwar einen Reduzierung des e-Card-API Datenvolumens auf ungefähr ein Drittel möglich, insgesamt ergibt sich damit aber nur eine Veringerung von ca. 100 KB auf ca. 70 KB. Dieses erscheint als Kriterium zu unwesentlich.

7.2.3 Proxy

Die Idee einer Proxy-Lösung bietet sich an, wenn einerseits die Kompatibilität auf Seite des eID-Servers erhalten bleiben soll, andererseits aber auf den Endgeräten der Einsatz einer eCard-API konformen Software nicht möglich oder nicht gewünscht ist. Das grundlegende Konzept ist es, zwischen eID-Server und Endgerät einen Vermittler zu haben, der die eCard-API-konforme Kommunikation vom eID-Server auf ein Protokoll abbildet, welches das Endgerät versteht. Bei dieser Lösung sind besondere Sicherheitsaspekte zu betrachten. Eine Instanz dieses Proxys wird in der Regel seine Funktionalität für mehrere Endgeräte bereitstellen. Dabei besteht natürlich die Möglichkeit, sowohl Daten über

einzelne Benutzer als auch Daten über alle Benutzer eines Dienstes zu sammeln. Es muss evaluiert werden, welche Daten dabei gesammelt werden können, und darauf aufbauend, welche eID-Funktionalitäten unter diesen Voraussetzungen nutzbar sind.
Eine Motivation für ein Proxy-Modell kommt aus der Netzkonfiguration eines Mobilfunkbetreibers. Sämtlicher Internetverkehr kann über einen Zwangsproxy des Providers abgewickelt werden. Dabei kommt üblicherweise NAT zum Einsatz. Um die Netzauslastung beeinflussen zu können, kann eine Filterung von erlaubten Ports oder sogar Protokollen nicht ausgeschlossen werden. Obwohl Bandbreite für die eID-Funktionalität in heutigen Mobilfunknetzen eher keine Einschränkung darstellt, kann ein derartiger Proxy für die Authentifizierung durchaus allein deswegen zum Einsatz kommen, um die Funktionalität für die eigenen Kunden sicherzustellen. Dazu könnte z.B. eine eigene Applikation, welche auf den Endgeräten läuft, vom Provider zur Verfügung gestellt werden. In diesem Modell bietet der Provider die Nutzung der eID-Funktionalität als speziellen Dienst für seine Kunden an und behält durch den Einsatz seiner eigenen Software eine gewissen Kontrolle über die Nutzung. In Verbindung mit Endgeräten, die nur signierte Software vom Netzbetreiber ausführen, besteht hier ein denkbares Anwendungsszenario.
Ein weiterer Aspekt einer Proxy-Lösung ist die Wartung der Komponenten. Durch eine zentrale Kontrolle über den Proxy und die zugehörigen Client-Anwendungen ist ein problemloses Zusammenspiel und damit eine Funktionalität des gesamten Systems über nur eine zentrale Stelle zu gewährleisten. Dies kann im Vergleich mit unabhängigen Entwicklungen für verschiedene Endgeräte zu einem Vorteil werden. Insbesondere bei Anpassungen im Falle einer Schnittstellenveränderung des eID-Servers, kann ein Netzbetreiber/Proxy-Software-Dienstleister alle Clients aktuell halten. Eine Variante dieses Modelles könnte eine zum größten Teil im Browser ablaufenden Software sein. Dabei würde das Browser-Plugin auf dem Endgerät die Daten bezüglich der eCard-API Verbindung an den Proxy weiterleiten. Auf welche Art und Weise eine sichere Verbindung zwischen dem Endgerät und dem Proxy etabliert wird oder ob hier die schon zwischen Browser-Plugin und Proxy bestehende weiter benutzt wird, sei unwesentlich. Die notwendige Interaktion mit dem Nutzer kann dabei komplett im Browser stattfinden. Da die eingangs beschriebene minimale Funktionalität beibehalten werden soll, muss PACE lokal auf dem Endgerät ausgeführt werden. Der Proxy würde somit also die Daten an den Browser senden und das Browser-Plugin müsste dafür sorgen die Zertifikatsbeschreibung, Attribute und PIN-Eingabe dem Nutzer im Browser anzuzeigen. Bei Bestätigung des Nutzers läuft der restliche Vorgang wie bei der AusweisApp ohne weitere Interaktion ab. Am Ende muss nur der Aufruf der *RefreshAddress* oder der dort vorhandenen *SAMLResponse* vom Proxy an den Client-Browser delegiert werden, damit dieser die Daten an den Dienstanbieter weiterreichen kann.
In dieser Variante ist die benötigte Funktionalität des Browser-Plugins die gleiche wie die einer reduzierten AusweisApp. Nur bei der Anzeige der Daten für die Nutzerinteraktion ist durch die Nutzung des Browser keine eigene grafische Komponente notwendig. Ebenfalls kann bei Nutzung der TLS Kapazitäten des Browsers auf eine Unterstützung eigener verzichtet werden. Dennoch bleibt es notwendig, eine Komponente für die Ausführung von PACE und das Weiterleiten von APDUs zu besitzen.
Ausgehend davon, dass es eine API für die Kartenkommunikation gibt, über die APDUs versendet werden können, PACE als offene Implementierung verfügbar ist und eine Biblio-

thek für den Umgang mit Secure Messaging vorhanden ist, gibt es für eine Implementierung so eines Plugins keine Hindernisse. Zusammen mit der Tatsache, dass der Proxy die für TA und CA nötigen APDUs erstellt und damit den Client von möglichst viel Logik befreit, wie das schon für die ISO/IEC 24727-4 Remote-ICC-Stack Konfiguration gezeigt wurde, ist eine relativ einfach portierbare Komponente möglich. Praktisch ermöglicht dieses einen Verzicht auf Bibliotheken für TLS/PSK und PAOS im Client, setzt diese aber wiederum im Proxy voraus.

Angriffspotenzial

Ausgehend von der Netzkonfiguraton eines Mobilfunkbetreiber, wie sie schon beschrieben wurde, kann der Provider in jedem Fall feststellen, mit wem ein Endgerät kommuniziert. Dies liegt an der völligen Kontrolle über die Infrastruktur, was benutzte DNS-Server und Gateways betrifft. Zwar könnte ein Client immer noch einen verschlüsselten Tunnel oder eine VPN-Verbindung mit einem Endpunkt außerhalb der Kontrolle des Providers benutzen. Dies würde aber im betrachteten Szenario die Nutzung des Proxys vollständig umgehen und wird daher nicht betrachtet, da dieses eigentlich nicht der Anwendungsfall von mobilen bzw. ressourcenschwachen Endgeräten ist. Der Provider weiß daher trotz TSL/SSL Verschlüsselung mit welchem Dienst und zugehörigem eID-Server ein Endgerät Verbindungen aufnimmt. Allein daraus lassen sich schon gewisse Informationen gewinnen. Dies ist aber in jedem Szenario der Fall, und damit keine Einschränkung.
Durch den Proxy gewinnt der Netzbetreiber zusätzlichen Zugriff auf die Informationen, die im eCard-API-Framework innerhalb des TLS/PSK-geschützten Kanals übertragen werden. Konkret sind dies die Dateien EF.CardAccess und EF.CardSecurity, das Berechtigungszertifikat des Dienstanbieters sowie einige Zwischenergebnisse der kryptographischen Operationen. Aus letzteren lassen sich keine relevanten Informationen gewinnen oder die kryptographische Sicherheit des Protokolls gefährden. Problematisch kann allerdings der Zugriff auf den öffentlichen Schlüssel des Personalausweises in EF.CardSecurity sein. Dieser ist zwar kein eindeutiges Identifizierungsmerkmal, da sich eine bislang unbestimmt große Anzahl von Personalausweisen dasselbe Schlüsselpaar teilen, doch kann er dennoch gewisse Hinweise dazu liefern, einen konkreten Benutzer wieder zu erkennen. In diesem Kontext muss allerdings darauf hingewiesen werden, dass in den meisten Fällen eine eindeutige Zuordnung von Benutzern an Rufnummern besteht, die dem Netzbetreiber die Zuordnung eines Endgerätes zu Identifikationsdaten ohnehin ermöglicht. Da mobile Endgeräte in der Regel nur von einem Benutzer verwendet werden, ist die hinzugewonnene Information lediglich die, dass ein identifizierbarer Kunde einen Personalausweis mit einem bestimmten öffentlichen Schlüssel benutzt. Somit ließe sich eine Datenbank erstellen, die bestimmten öffentlichen Schlüsseln eine Menge von Kunden-Identitäten zuordnet. Wie groß diese Menge am Ende sein wird hängt maßgeblich von der Menge von Personalausweisen mit gleichen Schlüsselpaaren ab, aber auch von Faktoren wie Kundenzahl des Netzbetreibers und dem Anteil der Kunden, die den Proxy nutzen.
Ausgehend davon, das der Netzbetreiber nun feststellen kann, welcher Kunde welchen Dienst mit welchem Personalausweis nutzt, ergeben sich aber keine Einschränkungen. Beim normalen Auslesen der Datengruppen des Personalausweises ist Anonymität keine gefor-

derte Eigenschaft. Auch im Falle der Nutzung von Restricted Identification geht es nur darum, dass der Dienstanbieter einen Nutzer wiedererkennt, ohne das persönliche Daten übertragen werden müssen. Die Vorgabe, dass dies anonym geschieht, gibt es hier nicht. Einzig beim Einsatz der Altersverifikation oder dem Lesen der Wohnort-ID soll dem Nutzer eine anonyme Nutzung möglich sein. Dabei soll aber nur der Nutzer gegenüber dem Dienstanbieter anonym bleiben. Solange der Netzbetreiber seine Information nicht an den Dienstanbieter weitergibt, besteht kein Risiko. Die Information, welchen Dienst ein Ausweisinhaber nutzt, lässt sich in keinem Szenario vor dem ISP verbergen. Der einzig praktikable Ansatz dazu wäre die Nutzung von anonymisierenden Proxy-Servern oder eines VPNs. In so einer Konfiguration, die auf Anonymität ausgelegt ist, macht aber der Einsatz des Personalausweises zur Identifikation keinen Sinn.
Im ungünstigsten Fall kann also ein Dienstanbieter, nach der Nutzung der Altersverifikation oder dem Lesen der Wohnort-ID, mit Hilfe des Netzbetreibers Rückschlüsse auf die Identität des Nutzers ziehen. Gleichzeitig setzt ein Dienstanbieter diese Funktionalität gerade dann ein, wenn er weitere Identitätsdaten nicht benötigt. Um diesem Risiko zu begegnen, sollten für den Proxy-Betreiber die selben datenschutzrechtlichen Bedingungen wie für einen eID-Server-Betreiber gelten, sodass eine Weitergabe von entsprechenden Daten juristische Konsequenzen haben kann. Tatsächlich dürften die Anreize zum Missbrauchs der Daten in dieser Situation relativ gering sein.
Wenn bei einer Altersverifikation oder dem Lesen der Wohnort-ID dem Ausweisinhaber eine anonyme Nutzung wichtig ist, bleibt in dieser Konfiguration ein Risiko bestehen. Ist ein Nutzer sich dem beschriebenen Risiko bewusst, kann in einer Proxy-Konfiguration die Nutzung des Personalausweises durchaus erfolgen.

7.3 Entfernter Leser

Ein simpler und theoretisch teilweise schon einsetzbarer Ansatz ist es, ein mobiles Gerät einfach nur als Leser zu benutzen. Dabei wird auf die Funktionalität des eCard-API-Frameworks auf dem Gerät selbst verzichtet. Zwangsläufig setzt dieses aber eine eCard-API-Instanz voraus, die das mobile Gerät als Leser benutzt. Bei einer örtlichen Trennung von Gerät und eCard-API-Instanz ist zudem die eigenständige Durchführung von PACE auf dem Gerät erstrebenswert, da sonst die PIN bei der eCard-API-Instanz hinterlegt sein oder dahin übertragen werden müsste.
Obwohl diese Lösung im Grunde der Proxy-Lösung entspricht, ist sie in eingeschränkter Form gegenwärtig mit vorhandenen Mitteln realisierbar. Daher wird im Folgenden kurz auf eine mögliche Implementierung eingegangen. Ein weiterer Grund für die Betrachtung dieses Szenarios liegt in der Problematik der nicht vorhandenen Lesegeräte in der Bevölkerung. Ein mobiles Endgerät kann in dieser Konfiguration einfach den notwendigen Basis-Leser ersetzen und damit die nötige Infrastruktur schaffen.
Basierend auf den Projekten NFCBTPCSC [19] und MobilPACE [58] ließe sich ein mobiles kontaktlos-Lesegerät mit PACE Funktionalität auf Basis von derzeit verfügbaren Nokia NFC Mobiltelefonen bauen. NFCBTPCSC stellt unter Unix/Linux Betriebssystemen einen PC/SC Treiber bereit, der über Bluetooth (BlueZ [3]) mit einem Midlet auf einem NFC-

fähigen Mobiltelefon kommuniziert. Für den PC/SC Daemon stellt sich das Telefon als gewöhnlicher Kartenleser da. Mit der Integration von z.B. MobilPACE [58] in das Midlet, könnte die PACE Funktionalität direkt auf das Telefon gebracht werden. Die Ansteuerung des PACE-Protokolls würde dann, entsprechend TR-03119 [46], über *SCardControl* erfolgen, welches im IFD-Handler Teil ausgewertet und an das Midlet übermittelt wird.
Damit alleine ist unter Unix/Linux Systemen der Einsatz in Kombination mit der AusweisApp möglich.
Um die Anzeige des Berechtigungszertifikates direkt auf dem Endgerät zu erzwingen und so die eCard-API-Framework-Instanz von der Notwendigkeit einer Nutzerinteraktionen zu befreien, müsste das Gerät einige Eigenschaften eines Komfort-Leser emulieren und sich bei der Durchführung von PACE und TA auch so verhalten. Wie schon bei der Remote-ICC-Stack Lösung fehlt für die Anzeige und Auswahl der auszulesenden Daten (Attribute) aus dem Personalausweis noch eine Lösung.
Denkbar wären hier mehrere Ansätze. So könnte man immer alle bzw. alle notwendigen, also nicht abwählbaren Daten akzeptieren. Dieses vereinfacht allerdings Angriffe, bei denen versucht wird, unberechtigt Daten auszulesen. Eine elegantere Lösung wäre es, diese zu bestätigenden Datengruppen als Teil der *InputData* Parameter beim Aufruf *EstablishPACEChannel* zu übergeben. Ob diese dann nur angezeigt werden oder tatsächlich auswählbar sind, und damit in den Rückgabeparametern von *EstablischPACEChannel* vorhanden sein müssen, sei hier vernachlässigbar. Diese Anpassung entspricht genau der Forderung aus der Remote-ICC-Stack-Lösung.
Diese Integration von sämtlichen Nutzerinteraktionen in den Leser und damit auf das mobile Endgerät ermöglicht eine passive eCard-API-Framework-Instanz. Würde in diesem Szenario die verwendete Bluetooth-Verbindung durch eine IP-basierte Verbindung auf Basis von GSM/UMTS ersetzt, ist keine örtliche Nähe zur eCard-API-Instanz mehr notwendig. Dennoch stellt sich nun die Frage, wie denn der eigentliche Prozess der eID-Authentifikation angestoßen wird, da dieses durch das Anfragen einer Ressource im Browser bei einem Dienstanbieter geschieht. Vorstellbar wäre hier ein automatisierter Prozess, der entfernt durch eine Nutzerinteraktion ausgelöst werden könnte. Die dazugehörige eCard-API-Instanz würde dann entweder auf einem unter Kontrolle des Nutzer stehenden Rechner laufen oder aber von dem entsprechenden Dienstanbieter betrieben. Im letzteren Fall würden sich mehrere Nutzer diese Instanz teilen. Daher müsste bei dem auslösenden Vorgang so etwas wie ein Nutzer-Identifikator mit übergeben werden, der die Einbindung des mobilen Endgerätes des Nutzers als entfernten Leser durch eine konkrete Zuordnung der Verbindung zum Endgerät erlaubt.
Unter Sicherheitsaspekten betrachtet, hat diese Lösung den Nachteil, dass der Betreiber der Client-eCard-API-Instanz potenziell einige Daten über den Nutzer erfahren kann und die Kontrolle über die ihm angezeigten Informationen hat. Eine Implementierung eines Komfort-Lesers sollte aber die angezeigten Daten verifizieren und so eine Manipulation bei der Ausführung von TA erkennen. Die ausspähbaren Daten sind dieselben wie im Falle eines kompromittierten Rechners. Ist der Betreiber der Instanz zusätzlich noch der Dienstanbieter, dann kann er keine Informationen erhalten, die er nicht sowieso schon hat.
Dieselben Information können allerdings ebenfalls über die Verbindung zwischen eCard-API-Instanz und mobilem Endgerät mitgehört werden. Hier muss die Implementierung dies

durch den Einsatz von geeigneter Kryptographie, verhindern.

7.4 Alternativen zur eCard-API

Den vorherigen Lösungsansätzen ist gemein, dass sie mit der Kompatibilität zum eID-Servers nicht oder nur im geringen Maße brechen. Dadurch entsteht auf eID-Server-Seite kein oder nur geringer Anpassungsaufwand. Dafür müssen Änderungen an der eCard-API-Schnittstelle immer Client-seitig angepasst werden. Lässt man den Anspruch der Kompatibilität fallen, ergeben sich neue Möglichkeiten. Hier muss unterschieden werden, wie weit so ein Kompatibilitätsbruch gehen soll. Zum einen wäre es möglich, nur die eCard-API durch eine eigenständige Lösung zu ersetzen. Dabei würden für den Dienstanbieter keine Änderungen entstehen. Zum anderen kann man die Schnittstelle zum Dienstanbieter ändern. Dabei müssen allerdings die Vorgaben des BSIs zumindest aus TR-03130 [47] und TR-03110 [48] beachtet werden.
Der Vorteil so einer Lösung liegt hauptsächlich darin, dass es keine Abhängigkeiten zu Mechanismen geben muss, für die es schwierig ist geeignete Bibliotheken zu finden oder die den Einsatz von zu komplexen Software-Stacks nötig machen. Wie schon in den vorangegangenen Lösungsszenarien ermöglicht die Einschränkung der Nutzung auf nur die eID-Funktionalität des Personalausweises den Verzicht auf mehrere Komponenten des eCard-API-Frameworks. So besteht prinzipiell keine Notwendigkeit ISO/IEC 24727 zu benutzen, noch ist ein Einsatz von Web Services zwingend vorgeschrieben. Dennoch ist eine Implementation auf Basis von etablierten Standards immer zu empfehlen.
Unabhängig davon sind Optimierungen bezüglich Nachrichtenaustauschvorgängen und zu übertragenden Daten gegenüber der eCard-API nicht möglich, da diese unter den gegebenen Voraussetzungen und Rahmenbedingungen des Personalausweises die Kommunikation optimiert. Wie schon bei der Proxy-Lösung erläutert sind die Einsparmöglichkeiten bezüglich des Datenvolumens zwar gegeben, aber bei heutigen technischen Voraussetzungen eher irrelevant.
Der Nachteil ist der Bruch mit der Kompatibilität auf Seite des eID-Servers. Im Kontext dessen, dass die auf dem eID-Server laufende Software weitaus komplexer ist als auf dem Client, ist ein Herauslösen des eCard-API-implementierenden Teiles unwahrscheinlich. Der eID-Server als Produkt ist nicht ohne weiteres implementierbar, da die notwendigen Backend-Systeme, zum Beispiel zur Sperrlistenverwaltung, wiederum Schnittstellen zu anderen Systemen haben. Folglich müsste die eCard-API in einem bestehenden eID-Server-Produkt ausgewechselt oder ergänzt werden. Dem steht aber klar ein Kostenargument gegenüber. Ein neues Protokoll müsste bezüglich seiner Sicherheit evaluiert und vom BSI zum Einsatz freigegeben werden. Dieser Aufwand kann bei den anderen Lösungen zumindest reduziert werden, da diese nur geringe Änderungen an der vorhandenen Richtlinien und Standards benötigen.
Dennoch ist es aus akademischer Sicht durchaus sinnvoll Alternativen zu dem bestehenden System zu betrachten, da es durchaus Potenzial für Vereinfachungen gibt. In anderen Lösungen wurden dieselben Probleme auf andere Art gelöst. Dabei können einige Ansätze durchaus in ein mobiles Szenario übernommen werden.

7.4.1 SAML-ECP-Profile

Eine Lösung zu Demonstrationszwecken wurde in der BDr entwickelt. Diese unterscheidet sich von der Middleware in Form der AusweisApp in einigen wesentlichen Punkten. Diese Unterschiede resultieren aus den eingeschränkten Anforderungen, die wie schon mehrfach beschrieben aus der Unterstützung nur der eID-Funktionalität des Personalausweises sowie aus der Ausrichtung auf mobile Endgeräte hervorgehen.
Die Kommunikation zwischen Dienstanbieter, eID-Server und Client wird dabei über ein abgewandeltes SAML ECP-Profil realisiert. Dieses setzt auf der Client-Seite einen erweiterten Browser voraus, da dieser die PAOS-Requests verarbeiten und PAOS-Responses senden können muss. Die offizielle Lösung nutzt an dieser Stelle das WebBrowser SingleSignOn-Profile, welches mit unmodifizierten Browsern funktioniert. Dafür ist aber im nächsten Schritt in der Antwort vom eID-Server ein Object-Tag enthalten, welches wiederum ein Browser-Plugin bzw. Browser Helper Object voraussetzt, um mit der lokalen Instanz des eCard-API-Frameworks zu kommunizieren. Daher ist in jedem Fall so etwas wie ein Plugin im Browser notwendig, um den eigentlichen Authentifikationsvorgang zu starten. Aus diesem Grund kann gleich das ECP-Profile verwendet werden, das ein Browser-Plugin voraussetzt. Die Kommunikation Client-eID-Server und der Authentifikationsvorgang, welcher außerhalb der SAML Profil-Spezifikation liegt, werden dabei in ein und derselben Verbindung abgehandelt. Dadurch ist es nicht notwendig so etwas wie eine Verschränkung von TLS-Kanälen zu erzwingen. Die in Abbildung 3.3 bestehenden Verbindung, welche die als Schritt drei und vier markierten SOAP-Nachrichten transportiert, wird für die Authentifikation weiterverwendet. Es werden also einfach weitere SOAP-Nachrichten ausgetauscht, bis der Vorgang abgeschlossen ist und die Nachricht, die die AuthnResponse enthält, beim Client eintrifft. Damit ist es möglich auf die PAOS-Kommunikation, wie sie das eCard-API-Framework benutzt, zu verzichten.
In dieser Lösung gibt es keine eigenständige Komponente, welche die Funktionalität des eCard-API-Frameworks übernimmt. Diese Funktionalität ist komplett in das Browser-Plugin integriert. Dieses ist insbesondere dann von Vorteil, wenn die Nebenläufigkeit von Anwendungen eingeschränkt ist. Als Nachteil stellt sich dar, dass der Browser so potenziell Zugriff auf alle Daten der Kommunikation hat. Wie schon in der Lösung auf Basis des ISO/IEC 24727-4 Remote-ICC-Stacks beschrieben, ist das Sicherheitsniveau auf mobilen Endgeräten nur mit dem eines Basis-Lesers vergleichbar. Ein kompromittierter Browser kann in diesem Szenario also wieder die Anzeige der auszulesenden Datengruppen und die Informationen des Berechtigungszertifikates manipulieren sowie die PIN abfangen. Dieses ist im Vergleich zu einem anderweitig kompromittierten System kein wirklicher Unterschied. Allerdings bieten sich Browser durch ihren Einsatz zur Kommunikation als Angriffspunkt an.
Aus der Verwendung der schon bestehenden Verbindung ergibt sich als weiterer wesentlicher Unterschied, dass die Kommunikation direkt vom Client gesteuert wird. Konkret manifestiert sich das darin, dass der Client Anfragen stellt, auf die der eID-Server antwortet. Die ausgetauschten Daten sind dabei aber die gleichen wie im Falle des eCard-API-Frameworks. Der Austauschvorgang von Daten für EAC ist durch TR-03110 [48] fest vorgegeben. Daher spielt es keine Rolle, welche Seite den aktiven Teil übernimmt. Durch

die Einbindung des Authentifikationsvorganges in den TLS-Kanal zwischen eID-Server und Client lassen sich so insgesamt die Anzahl der Kommunikationsvorgänge reduzieren und die Ressourcen für einen zweiten TLS-Kanal sparen. Dies ermöglicht gleichzeitig den Verzicht auf PAOS, da es durch den schon vorhandenen TLS-Kanal kein Problem mit Firewalls, Proxys oder NAT gibt, was im eCard-API-Framework die Motivation für PAOS darstellt. Somit hat eine Implementation zumindest auf der Client-Seite den Vorteil, auf die kritischen Bibliotheken für PAOS und TLS/PSK verzichten zu können. Eine Abhängigkeit besteht nur zu einer Kryptographie-Bibliothek, um PACE ausführen zu können, und einer Kartenleser-Abstraktion wie z.B. PC/SC.

Anpassung an TR-03112

Praktisch gehört zu diesem Client ein entsprechender Server. Das dieses System aber nur zu Demonstrationszwecken entwickelt wurde, ist die Anbindung der restlichen Systeme der Personalausweis-Infrastruktur sowie der notwendigen PKI natürlich nicht gegeben. Aus diesem Grund wurde der bestehende Client um eine Kompatibilität zum eCard-API-Framework erweitert, damit er ebenfalls an der tatsächlichen Infrastruktur funktioniert. Dabei sind einige Probleme an die Oberfläche getreten, welche sich aus der im Vergleich zum eCard-API-Framework veränderten Architektur ergeben.
Wie bereits erwähnt, beendet der aktuelle eID-Server die eCard-API Kommunikation erst nachdem der Client einen Aufruf an die *RefreshAddress* durchgeführt hat. Bei der Integration der eCard-API Funktionalität in das Browser-Plugin ist aber der Aufruf dieser URL mit der Abgabe der Kontrolle vom Plugin an den Browser verbunden, wodurch eine weitere eCard-API Kommunikation erschwert wird. Wie schon beschrieben ist dieses auf Client-Seite unproblematisch, kann aber auf dem eID-Server zu unnötig allozierten Ressourcen führen.
Als zusätzliche Abhängigkeit wird nun GnuTLS [23] verwendet, um den TLS/PSK-Kanal aufbauen zu können. Da der eID-Server gegenwärtig keine sinnvollen Fehlermeldungen liefert, wurde auf eine korrekte PAOS Implementation verzichtet und stattdessen ein einfaches String-Parsen und Ersetzen in vorgefertigten Nachrichten-Templates verwendet. Dieses ist zwar potenziell fehleranfällig, aber für eine Demonstrationslösung ausreichend.
Bei Versuchen mit verschiedenen Dienstanbietern wurde allerdings festgestellt, dass es eine eID-Server Implementation gibt, deren TSL/PSK Implementation sich von der des Anwendungstest-eID-Servers unterscheidet. So bricht dieser z.B. den SSL Handshake ab, wenn in der *Client Hello* Nachricht Extensions enthalten sind. So benutzt GnuTLS [23] eine Extension, um den Fehler in der TLS Renegotiation [60] zu umgehen. Die Nichtnutzung dieser Extension muss man daher explizit erzwingen. Ähnlich verhält es sich, wenn im *Client Hello* zu viele CipherSuites enthalten sind. Die *Client Hello* Nachricht wird auf TCP-Ebene mit einem Verbindungsabbruch beantwortet.

Kapitel 8

Empfehlungen

Auf Basis der erarbeiteten Lösungsansätze und den bei der Implementation einer eCard-API kompatiblen Demonstrationslösung gewonnenen Erfahrungen werden nachstehend die Änderungen erläutert, welche an der eCard-API (TR-03112 [44]), der EAC Spezifikation (TR-03110 [48]) und den Anforderungen an Chipkartenleser mit ePA Unterstützung (TR-03119 [46]) durchgeführt werden sollten. Diese Änderungen sollen dazu dienen, eine Einführung der benötigten Komponenten zu beschleunigen. Im Folgenden werden notwendige Angleichungen zwischen bestehenden Spezifikation und mögliche Erweiterung sowie deren Nutzen diskutiert. Schließlich wird für die einzelnen Lösungen aus dem vorherigen Kapitel kurz umrissen, welche dieser Anpassungen zur Umsetzung notwendig sind.

8.1 Abgleich von TR-03119 und TR-03112

In TR-03119 [46] wird für die Leser der Kategorien Standard und Komfort die Ausführung des PACE-Protokolls im Leser vorgeschrieben. Dazu wird über den Befehl *GetReadersPACECapabilities()* festgestellt, ob der Leser diese Funktion unterstützt. Im positiven Falle wird die Funktion *EstablishPACEChannel(InputData)* zur Erstellung des sicheren Kanals gerufen. Die Abbildung dieser Funktionen auf die PC/SC Funktion *SCardControl* wird durch eine Erweiterung der Funktion *GET_FEATURE_REQUEST* durch ein zusätzliches *#define FEATURE_EXECUTE_PACE 0x20* erreicht. Die genaue Beschreibung ist TR-03119 [46] A.11 zu entnehmen.
Mit dieser Anpassung ist die PC/SC Schnittstelle in der Lage, die PACE Funktionen des Lesers zu nutzen. Wird aber die eCard-API als Middleware zwischen einem Remote und Lokal-Terminal benutzt, ist die Kommunikation direkt über PC/SC nicht vorgesehen. Die systemnahen Funktionen der PC/SC Schnittstelle sind über den IFD-Layer des eCard-API-Framework abstrahiert. Dieser bietet allerdings keine direkte Umsetzung der PC/SC Funktion *SCardControl* in seinem Interface an. Von den vorhandenen Funktionen sind die aus der Gruppe der Karten-Terminal-Funktionen am ehesten für so einen Aufruf geeignet. Die Funktion *GetIFDCapabilities* gibt Information über die Fähigkeiten des Kartenterminals an den Aufrufer zurück und könnte damit den Aufruf von *GetReadersPACECapabilities* aus TR-03119 [46] abdecken. In der Funktion *GetIFDCapabilitiesRe-*

sponse gibt es die Datenstruktur *IFDCapabilities*, welche sich wiederum in eine Anzahl von weiteren Sub-Capabilities aufteilt. Darunter sind die optionalen *DisplayCapability* und *KeyPadCapability*, welche Informationen zu Anzahl und Eigenschaften von möglicherweise vorhandenen Anzeigen und Tastaturen von Lesegeräten enthalten. Aus diesen Informationen kann man aber nicht den Schluss ziehen, ob es sich um einen Leser vom Typ Standard oder Komfort handelt, da die notwendige Information, ob das PACE-Protokoll unterstützt wird, nicht enthalten ist. Die *GetReadersPACECapabilities* Funktion aus TR-03119 [46] liefert als Rückgabe nur eine Bitmaske, die essentiell in nur drei Bit kodiert, welche Fähigkeiten bezüglich PACE, eID und eSign im Leser vorhanden sind. Eine Abbildung dieser Funktionen aufeinander ist somit noch nicht möglich.
Folglich bleibt nur die der Betrachtung der IFD-Layer Funktion *ControlIFD*. Diese ist dazu gedacht ein (möglicherweise proprietäres) Kommando an das Kartenterminal zu senden, um spezielle Funktionen, für die es kein separates Kommando im IFD-Layer Interface gibt, aufrufen zu können, ohne das Änderungen am besagten Interface nötig wären. Diese Funktionalität deckt sich mit der benötigten. In den Hinweisen zu *ControlIFD* ist vermerkt, dass die Implementation mittels *FEATURE_MCT_READERDIRECT* aus PC/SC erfolgen kann. Dieses ist bereits ein *SCardControl* Aufruf, allerdings mit einer anderen Konstante als der in TR-03119 [46] definierten. An dieser Stelle müsste in der IFD-Layer Funktion *ControlIFD* das Datenfeld *Command* ausgewertet werden, um zwischen den verschiedenen *SCardControl*-Aufrufen zu unterscheiden und den passenden auszuwählen. Entsprechend muss das Ergebnis ausgewertet werden. Zusammen mit den Anpassungen des PC/SC/Stacks aus der TR-03119 [46] ermöglichen eine Interoperabilität des eCard-API-Frameworks mit Lesern, die eigenständig PACE ausführen sollen.
Da es aber mittlerweile den ersten Komfort-Leser gibt und dieser mit der AusweisApp interoperabel ist, sollten diese Anpassungen implizit schon existieren. Diese sind aber zum jetzigen Zeitpunkt noch nicht in den veröffentlichten Dokumenten enthalten.

8.2 Mobil-Leser

Um die Anzeige der zu lesenden Attribute gegen Manipulationen abzusichern, ist in TR-03119 [46] für den Komfort-Leser beschrieben, dass diese Daten auf dem Display des Lesers angezeigt werden sollen. Der Nutzer hat hier nun die Möglichkeit, diese mit dem am Rechner bestätigten zu vergleichen. Zusätzlich prüft der Komfort-Leser, ob diese so angezeigten Attribute mit denen im Terminal-Berechtigungszertifikat während TA übereinstimmen. Die eigentliche Bestätigung dieser Attribute vom Nutzer erfolgt aber davor über die eCard-API-Instanz auf dem Rechner. Dieser Vorgang kann aber in die Durchführung von PACE auf dem Leser integriert werden, indem die Parameter der Funktion *EstablishPACEChannel(InputData)* sowie deren Rückgabe angepasst werden.
Im Unterschied zu dem beschriebenen Komfort-Leser, der ein Display von mindestens 2x16 alphanumerischen Zeichen haben soll, kann bei einem mobilen Einsatz von einer graphischen Anzeige mit einer Auflösung von 320x240 Pixel (QVGA) oder sogar weitaus mehr ausgegangen werden. Zusätzlich steht oft eine Eingabemöglichkeit zur Verfügung, die einem PIN-Pad überlegen ist, was eine Interaktion an dieser Stelle umsetzbar macht.

	Cat-B	Cat-M	Cat-S	Cat-K
Pinpad	o	X	X	X
Pace	o	X	X	X
ePA-QES	o	o	o	X
Display (2x16 alphanumerische Zeichen)	o	X	o	X
Display QVGA	o	X	o	o
Zert. & Attribute auswählen	o	X	o	o
Sichere PIN Eingabe	o	o	X	X

Tabelle 8.1: Übersicht Chipkartenleser-Kategorien

In *InputData* würde Position 3 nicht der schon eingeschränkte CHAT sein, sondern der vom Dienstanbieter angefragte. Im Einklang mit den aktuellen verwendeten eCard-API Parametern wäre eine weiterer Parameter nötig, um zwischen *RequiredChat* und *OptionalChat* unterscheiden zu können. Somit gäbe es zwei weitere Positionen. In *OutputData* müssten zwei zusätzliche Positionen hinzugefügt werden: die Länge des eingeschränkten CHATs sowie der eingeschränkte CHAT selbst.
Da diese Anpassung gemacht wird, um eine andere Klasse von Geräten anzusteuern und nicht um eine höhere Sicherheitsstufe zu gewährleisten, wie dieses mit der Spezifikation von Standard- und Komfort-Lesern eigentlich bezweckt wurde, bietet es sich direkt an, einen weiteren Typ von Lesern zu definieren, der die Änderungsvorschläge an TR-03119 [46] aus diesem und dem vorangegangenen Abschnitt zusammenfasst, z.B. als Mobiler-Leser. Damit einher geht die Notwendigkeit, dass dieser Mobil-Leser identifiziert werden kann. Dazu wäre es sinnvoll, den Rückgabeparameter *BitMap* der Funktion *GetReadersPACECapabilities* um ein weiteres Bit (0x80) für die Verfügbarkeit eines großen Displays und Interaktionsmöglichkeiten zu erweitern.
Alternativ zum autonomen mobilen Einsatz kann dieses Profil für die Ansteuerung eines erweiterten Basis-Lesers dienen, der durch die Verwendung eines mobilen Endgerätes als Ersatz für ein dediziertes Lesegerät verwendet wird. Dabei steht primär die Verwendung von NFC-fähigen Mobiltelefonen im Vordergrund. So eine Realisierung ist als reiner Basis-Leser z.B. durch NFCBTPCSC [19] ermöglicht. Die erweiterten Fähigkeiten lassen sich aber erst durch diese veränderte Ansteuerung, wie sie so ein Mobil-Leser bietet, nutzen. Dieses könnte ein durchaus wichtiger Schritt zur Verbreitung der notwendigen Kartenleser-Infrastruktur sein. Das Sicherheitsniveau liegt zwar hier nicht über dem des Basis-Lesers, dennoch verlässt die PIN das mobile Endgerät nicht. Für einen Angriff, der die PIN ausspähen soll, muss also das Endgerät kompromittiert sein. Um eine Angriff auf den gesamten eID-Vorgang durchzuführen, muss eine kombinierte Kompromittierung des mobilen Endgerätes und des eCard-API-Framework ausführenden Rechners realisiert werden, was den Aufwand für einen Angreifer deutlich erhöht.

8.2.1 Implementierung

Ein Aspekt, der bis jetzt ignoriert wurde, ist die Implementierung so eines Mobil-Lesers. Die vorgeschlagenen Änderungen an der TR-03119 [46] betreffen nur die Ansteuerung so

eines Lesers über PC/SC. Zusammen mit einer Harmonisierung dieser Ansteuerung über den IFD-Layer des eCard-API-Frameworks ist es noch nicht klar, von welcher Komponente PACE durchgeführt wird und das Zertifikat sowie die Attribute dargestellt werden. Dabei ist auf mobilen Geräten davon auszugehen, dass grundsätzlich alles in Software umzusetzen ist, da von einer Integration spezieller Hardware oder Firmware zu diesem Zeitpunkt nicht ausgegangen werden kann. Somit stellt sich die Frage, welche dieser Funktionalitäten besser als Teil des mobilen eCard-API-Frameworks implementiert werden und welche möglicherweise als native Implementierung außerhalb des Frameworks auf dem mobilen Endgerät existieren. Dabei muss berücksichtigt werden, dass nicht unbedingt eine zu PC/SC kompatible Leser-Treiber-Abstraktion vorhanden ist.
Tatsächlich ist es die Aufgabe des IFD-Layers des eCard-API-Frameworks, unterschiedliche Leser-Ansteuerungen zu abstrahieren. Daraus ergibt sich die Möglichkeit eine plattformspezifische Implementation von PACE als Teil des mobilen eCard-API-Frameworks oder aber als native Komponente umzusetzen, die zwischen Leser-API und mobilem eCard-API-Framework sitzt. Im letzteren Fall könnte diese zwangsläufig plattformspezifische Komponente ein statisches Interface zum mobilen eCard-API-Framework anbieten, wodurch eine fast generische Implementierung für verschiedene Plattformen ermöglicht wird. Ein Vorteil dieser Idee ist es, dass dabei das PACE-Protokoll so effektiv wie möglich umsetzbar ist, da es den kryptographisch aufwändigsten Teil bei der EAC-Ausführung auf dem Client darstellt.
Welchen Sinn so eine Aufteilung in generische und native Komponenten hat, muss letztendlich eine Analyse der verfügbaren Hardware und der darauf vorhandenen APIs entscheiden. In diesem Bereich besteht noch Forschungsbedarf.

8.3 Extended Length APDUs und SFIs

In TR-03110 [48] Appendix B. ISO 7816 Mapping (Normative) B.9 wird die Verwendung von Extended Length APDUs von Terminals nur auf *PSO:Verify Certificate*, *MSE:Set KAT*, *General Authenticate* sowie *External Authenticate* eingeschränkt. Als Ausnahme werden bekannte Puffergrößen aus dem ATR/ATS der Karte angeführt. Die aktuell verfügbaren Anwendungstestkarten geben diese Puffergrößen noch nicht an und ein separates EF.ATR ist nicht vorhanden. Die ab November produzierten Ausweise werden vermutlich ein EF.ATR enthalten, in dem die Puffergröße voraussichtlich mit 1199 Bytes angegeben ist. Die eingesetzten Karten mit dem TCOS [6] Betriebssystem sollten theoretisch eine Puffergröße von ungefähr bis zu 1520 Bytes unterstützen.
Für den Personalausweis bei der Durchführung des EAC Protokolls ist dies tatsächlich nur für die Verifikation der Zertifikatskette in TA relevant, bei der diese Zertifikate an die Chipkarte gesendet werden.
Geht es aber um eine Optimierung der Anzahl der ausgetauschten APDUs, ist der Einsatz von Extended Length APDUs für das Lesen von Daten wünschenswert. Ohne diese Möglichkeit sieht der Vorgang des Lesens eines Datenobjekts in der Regel folgendermaßen aus, wie in TR-03110 B.8 [48] beschrieben. Zuerst wird die Datei mit einem *Select* Kommando selektiert. In dessen Response-APDU kann unter anderem die Größe des Objektes enthal-

ten sein. Damit lassen sich die notwendigen *Read Binary* APDUs mit passenden Offsets generieren, um die gesamte Datei zu lesen. Da die Daten aber immer im TLV-kodierten Format vorliegen, ist nach dem ersten *Read Binary* feststellbar, wie groß das Objekt ist. Die Notwendigkeit, zuerst die Größe der Daten zu kennen, bevor die *Read Binary* APDUs erstellt werden können, benötigt immer mindestens eine Interaktion mit der Gegenseite. Alternativ gibt es die Möglichkeit, ohne die Größe des Objektes zu kennen, solange *Read Binary* APDUs zu senden, bis das Statuswort (z.B. 0x6282) der R-APDU anzeigt, dass über das Ende des Objektes gelesen wurde. Dieses ist allerdings noch ineffizienter, wenn es über eine entfernte Verbindung ausgeführt wird, da jede einzelne Response-APDU auf das entsprechende Statuswort geprüft werden muss.
Im Vergleich dazu ist durch den Einsatz einer Extended Length APDU für *Read Binary* die Information über die Länge der Daten nicht notwendig. Natürlich gilt das nur unter der Annahme, dass die Puffergrößen auf Leser- und Karten-Seite beim Übertragen der Daten kein Problem darstellen, was aber in de eID-Anwendung und bei den EAC Datenobjekten unkritisch sein sollte. Ausgenommen davon ist das EF.CardSecurity Objekt. Dieses hat auf den Anwendungstest-Karten beispielsweise eine Länge von über 1900 Bytes. Hier wären mindestens zwei Extended Length *Read Binary* APDUs notwendig.
Zusätzlich lässt sich durch den Einsatz von SFIs das *Select* Kommando einsparen, wenn die Selektion der Datei explizit im *Read Binary* Kommando erfolgt. Allerdings sind SFIs von der ISO/IEC 24727 [40] im GCI nicht erlaubt und damit prinzipiell mit dem eCard-API-Framework nicht vereinbar. Da aber beim Einsatz von Extended Length APDUs die Längeninformation nicht relevant ist, stört ein vorher ausgeführtes *Select* nicht.
Ohne Extended Length APDUs gäbe es immer noch den Weg, einfach eine ausreichende Anzahl von *Read Binary* APDUs mit aufeinander folgenden Offsets zu erzeugen und einfach die zugehörigen Antworten, welche keine Objektdaten mehr enthalten, zu ignorieren. Dabei ist aber die ausreichende Anzahl eine Konstante, welche ein gewisses Fehlerpotential birgt.

8.4 Eingeschränkte eCard-API

Dieser Ansatz läuft auf eine Ergänzung von TR-03112 [44] hinaus, die mindestens die schon benannten Funktionen als Teil der mobilen oder reduzierten eCard-API benennt. Unter Umständen kann dabei durch einen weiteren Parameter in *StartPAOS* dem Server die Reduktion auf einen reduzierten Funktionsumfang auf dem Client übermittelt werden.

8.5 ISO-24727 Remote-ICC-Stack

Um eine Lösung wie die des Remote-ICC-Stacks umzusetzen, muss neben den Voraussetzungen durch den Mobil-Leser die eCard-API-Instanz diese Konfiguration sowohl auf der Server-Seite als auch auf der Client-Seite unterstützen. Auf der Client-Seite kann das durch eine Instanz, die nur ein eingeschränktes Interface anbietet, realisiert werden. Auf der Server-Seite muss die notwendige Logik, um aus den SAL-Funktionsaufrufen APDUs und IFD-Layer-Funktionsaufrufe zu generieren, vorhanden sein. Im konkreten Fall betrifft

dies nur die Logik zur Durchführung des EAC Protokolls, da nach dessen Durchführung die Kommunikation sowieso über das IFD-Layer-Interface stattfindet. Um der eCard-API-Instanz auf eID-Server-Seite diese veränderte Konfiguration anzuzeigen, ist es am einfachsten, einen entsprechenden Parameter in *StartPAOS* aufzunehmen.

Kapitel 9

Zusammenfassung

Zum Ende dieser Arbeit werden die entstandenen Ergebnisse noch einmal zusammengefasst und ein Überblicke über die entstandenen Artefakte gegeben. Den Abschluss bildet ein Ausblick auf zukünftige Entwicklungen.

9.1 Diskussion

Als erstes, vielleicht auch überraschendes Ergebnis ist festzustellen, dass das Optimierungspotenzial an dem durch die eCard-API vorgegebenen Ablauf und Format des Nachrichtenaustausches unter den gegebenen Zielen sehr gering ist. Es lässt sich zwar das Datenaufkommen durch geeignete Kodierungen deutlich verringern, doch erreicht man dadurch keine Verbesserung mit tatsächlicher Relevanz für den mobilen Einsatz. Die Komplexität der auf dem Client benötigten Software kann durch die beschriebenen Ansätze zwar leicht verringert werden, doch wird diese Verbesserung am Ende kaum messbar sein.
Ähnliches gilt für Proxy-Lösungen. Die Analyse hat gezeigt, dass diese durchaus machbar sind und, solange PACE lokal auf dem Endgerät ausgeführt wird, bei einer Betrachtung unter Sicherheitsaspekten bestehen können. Einen wirklichen Vorteil für den Nutzer können sie gegenüber einer Lösung ohne Proxy aber nicht bieten. Für den Entwickler bieten sie genau wie ein alternatives Protokoll die Möglichkeit, auf TLS/PSK und PAOS zu verzichten. Dieses ist aber nur dann ein Vorteil, solange es keine freien und weit verbreiteten Bibliotheken gibt. Überhaupt ist aufgrund des geringen Optimierungspotenziales kein Spielraum für ein alternatives Protokoll gegeben, da dieses Anpassungen an der bestehenden Infrastruktur voraussetzt.
Die Lösung der eingeschränkten eCard-API hat am meisten Potenzial für den mobilen Einsatz. Eine Standardisierung des reduzierten Funktionsumfanges erleichtert die Entwicklung von entsprechenden Clients. Der einfachste Ansatz ist es, ein Browser-Plugin zu entwickeln, das die gesamte, notwendige Funktionalität enthält. Dieses sollte mit dem eID-Server eine eCard-API konforme Kommunikationsverbindung aufbauen. Intern kann auf die tatsächliche Umsetzung nach ISO/IEC 24727 [37] vollständig verzichtet werden. Ein Nachteil besteht dabei in der zuvor beschriebenen Beendigung der Kommunikation vom Client, ohne die Sitzung sauber zu beenden. Dass dieser Ansatz umsetzbar ist, wurde durch eine

Anpassung eines bestehenden Demo-Clients gezeigt, der nun in der Lage ist, als alternative AusweisApp für die Nutzung der eID-Funktionalität zu dienen.
Obwohl der Einsatz einer schlanken Software weitaus attraktiver ist, sollte es praktisch sogar möglich sein, einen vollständig eCard-API konformen Client für den mobilen Einsatz zu entwickeln, der wie die AusweisApp nur initial von dem Browser-Plugin angesprochen wird. In Anbetracht des damit verbundenen Aufwandes bietet sich hier J2ME [12] als Umgebung an, sodass eine Implementation auf möglichst vielen Endgeräten lauffähig ist. Im Umfeld von eingebetteten Systemen ist sonst der Einsatz von nativem Code auf der Basis von C und C++ der übliche Weg. Dabei kann zumindest für TLS/PSK auf freie Bibliotheken zurückgegriffen werden.
Während der Arbeit an diesem Thema sind einige Werkzeuge entstanden, die bei der Beschäftigung mit der eCard-API hilfreich sind. So wurde der Netzwerk-Sniffer Wireshark [28] um die Fähigkeit erweitert, TLS/PSK geschützte Verbindung bei Kenntnis des PSKs zu entschlüsseln. Andererseits ist für Linux basierende Systeme ein PCSClite [61] IFDHandler entstanden, der die Ansteuerung der kontaktlosen Schnittstelle des OmniKey 5321 und 6321 auf der Basis von librfid [62] ohne die Nutzung von Binär-Treibern ermöglicht.

9.2 Ausblick

Die vorangegangenen Betrachtungen wurden alle unter der Einschränkung gemacht, dass nur auf die eID-Funktionalität des Personalausweises Rücksicht genommen wird. Würde man diese Einschränkung fallen lassen, ergeben sich an vielen Stellen weitere Ansatzpunkte zur Realisation einer eID-Funktionalität mit mobilen Geräten. So ist z.B. die Integration einer eID-Applikation in das Secure Element eines NFC-fähigen Mobiltelefons ein alternativer Ansatz, wenn auch nicht für den hoheitlichen Einsatz.
Generell lassen sich einige Anforderungen an das System aufweichen, wenn eine nicht hoheitliche Anwendung zum Einsatz kommt. Auf Basis der Erfahrungen, Protokolle und Infrastruktur aus dem Bereich des Personalausweises lassen sich durchaus weitere Anwendungen ableiten. Insbesondere mit Daten, die weniger sensibel als hoheitliche sind, können Dienste realisiert werden, deren Zweck nur in der Übermittlung der ausgelesenen Daten besteht, was einen Einsatz in föderierten Umgebungen ermöglicht.
Ein Aspekt, der in dieser Arbeit nicht behandelt wurde, ist die Sicherheit der Softwarekomponenten. So bieten das Browser-Plugin und die eCard-API-Nachrichten durchaus einige Ansatzpunkte, um nach Sicherheitslücken zu suchen.
Leider ist es bis jetzt noch nicht gelungen, das Hardwareproblem für den mobilen Einsatz zufriedenstellend zu lösen. In naher Zukunft sollten aber die ersten Geräte auf den Markt kommen, die von der Hardware her die nötige Ausstattung besitzen. Wenn die Hersteller den Anwendungsfall mobile eID wahrnehmen, was durch diese Arbeit unterstützt werden soll, dann besteht die Chance, dass die notwendigen Software-Schnittstellen vorhanden sind. Damit könnte der mobile Einsatz der eID-Funktion des Personalausweises ein weiterer Schritt zur digitalen Gesellschaft sein.

Anhang A

EAC Version 2 PAOS Kommunikation

Eine beispielhafte Aufzeichnung der ausgetauschten Nachrichten beim Nutzen der eID-Funktionalität mittels EAC Version 2. Die Kombination von TA und CA in nur einem Schritt wird nicht genutzt.

A.1 Browser object

```
<HTML>
  <HEAD><TITLE> eCard Client Initiator</TITLE></HEAD>
  <BODY>
    <object type="application/vnd.ecard-client">
      <param name="ServerAddress" value="live.eid-service.de:443"/>
      <param name="SessionIdentifier"
value="e29270f1b8e745edcc9fe0b9caa9"/>
      <param name="Binding" value="urn:liberty:paos:2003-08"/>
      <param name="PathSecurity-Protocol" value="uri:iso:PAOS"/>
      <param name="PathSecurity-Parameters"
value="61e78caedbe49a0d9d414ba5b42b5c95"/>
      <param name="RefreshAddress"
value="https://live.eid-service.de:443/epa/plugin?UEsDBBQACAAIABBm3DwAAAAA
AAAAAAAAAALAAAAZWludHJhZy50eHRNj0FrgzAcxb9KQPC0WBPNTFq8rHXFQteCHnqTNPk7RR
clxm5%2B%2Bzl26e3Be7%2FHezd8Hb7BgsZvi1eAffTgNjRg%2Fp8G6xWzQcdeTtN7OzUoMw7s
aNsJ0L%2BPHjQgAVnTDu%2BHoWvBq1o9VnJ2jal6VXWwpJVgmiUQRVqQJBIaOGWvNedSRjThmt
MdukrXpJuVfG46FVlR5JeP%2FJACFTQJa3LnkMQMtFKihvAulJTimd4P60LjcLmM4Dn4cZvGff
U71Ug7gUvz4oI5ZwITv7TSTDVYnBk16NZ8eqqZTQfaP0gH3nkwL4hydFr%2F05CEiITbmG1JjI
7n8hdQSwcIJBtO7vYAAAA1AQAAUEsBAhQAFAAIAAgAEGbcPCQbTu72AAAANQEAAAsAAAAAAAAA
AAAAAAAAAAAAAAGVpbnRyYWcudHh0UEsFBgAAAAABAAEAOQAAAC8BAAAAA%3D%3D"/>
    </object>
  </BODY>
```

```
</HTML>
```

A.2 StartPAOS

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 2115
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08" xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
        <ns3:MetaData>
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:MessageID>urn:uuide6b3c26a914f256a6d2d8c9bfee7bb6d19bc659c
    </ns5:MessageID>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
  </ns2:Header>
  <ns2:Body>
    <iso:StartPAOS xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
```

```
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:SessionIdentifier>e29270f1b8e745edcc9fe0b9caa9
      </iso:SessionIdentifier>
      <iso:ConnectionHandle xsi:type="iso:ConnectionHandleType">
        <iso:ContextHandle>428684103954799E4D6712D0239081FD74BBBA24
        </iso:ContextHandle>
        <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
        </iso:SlotHandle>
      </iso:ConnectionHandle>
    </iso:StartPAOS>
  </ns2:Body>
</ns2:Envelope>
```

A.3 PACE

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=609004BD2A01077841A3623CF4DAA14D; Path=/; Secure
Content-Type: application/vnd.paos+xml
Content-Length: 3692
Date: Mon, 28 Jun 2010 10:48:36 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:MessageID>urn:uuide81d0abe56095d4b8f24612585ab2fa2ce0bc6d3
    </ns5:MessageID>
    <ns5:ReplyTo>
      <ns5:Address>https://live.eid-service.de:443</ns5:Address>
    </ns5:ReplyTo>
```

```
    <ns5:Action>http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
    </ns5:Action>
  </ns1:Header>
  <ns1:Body>
    <iso:DIDAuthenticate xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:ConnectionHandle xsi:type="iso:ConnectionHandleType">
        <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
        </iso:SlotHandle>
      </iso:ConnectionHandle>
      <iso:DIDName>PIN</iso:DIDName>
      <iso:AuthenticationProtocolData xsi:type="iso:EAC1InputType">
        <iso:Certificate>
7F218201427F4E81FB5F290100420E5A5A4456434141544130303030357F494F060A04007F
000702020202038641047 0C07FAA329E927D961F490F5430B395EECF3D2A538194D8B637DE
0F8ACF60A9031816AC51B594097EB211FB8F55FAA8507D5800EF7B94E024F9630314116C75
5F200B5A5A444B423230303033557F4C12060904007F000703010202530500030 1DF045F25
060100000601085F2406010000070001655E732D060904007F0007030103028020 7C190193
2DB75D08539F2D4A27C938F79E69E083C442C068B299D185BC8AFA78732D060904007F0007
030103018020BFD2A6A2E4237948D7DCCF7975D71D40F15307AA59F580A48777CBEED093F5
4B5F3740618F584E4293F75DDE8977311694B69A3ED73BBE43FDAFEC11B7ECF054F84ACB12
31615338CE8D6EC332480883E14E0664950F85134290DD716B7C153232BC96
        </iso:Certificate>
        <iso:CertificateDescription>
30820178060A04007F00070301030101A1160C1442756E646573647275636B657265692047
6D6248A2241322687474703A2F2F7777772E62756E646573647275636B657265692E64652F
```

```
64766361A3180C164465757473636865204B726564697462616E6B204147A4131311687474
703A2F2F7777772E646B622E6465A581FC0C81F9EFBBBF54617562656E7374722E20372D39
0D0A313031313137204265726C696E0D0A696E666F40646B622E64650D0A45726F6566666E75
6E672065696E6573204B6F6E746F730D0A4265726C696E65722042656175667474726167465
722066C3BC7220446174656E73636875747A20756E6420496E666F726D6174696F6E736672
6569686569742C20416E2064657220557261 6E696120342D31302C20313037383720426572
6C696E2C203033302F3133203838392D302C206D61696C626F7840646174656E7363687574
7A2D6265726C696E2E64652C20687474703A2F2F7777772E646174656E73636875747A2D62
65726C696E2E64650D0A
        </iso:CertificateDescription>
        <iso:RequiredCHAT>7F4C12060904007F000703010202530500010098 00
        </iso:RequiredCHAT>
        <iso:OptionalCHAT>7F4C12060904007F000703010202530500000000 00
        </iso:OptionalCHAT>
        <iso:AuthenticatedAuxiliaryData>
67177315060904007F000703010402530832303130303632 38
        </iso:AuthenticatedAuxiliaryData>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticate>
  </ns1:Body>
</ns1:Envelope>
```

A.4 PACE Response

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 3896
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
```

```
        <ns3:MetaData>
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:RelatesTo>urn:uuide81d0abe56095d4b8f24612585ab2fa2ce0bc6d3
    </ns5:RelatesTo>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:MessageID>urn:uuid8c1f43be885675878167e6886fd838bd1b63b511
    </ns5:MessageID>
  </ns2:Header>
  <ns2:Body>
    <iso:DIDAuthenticateResponse
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
      <iso:AuthenticationProtocolData xsi:type="iso:EAC1OutputType">
        <iso:RetryCounter>3</iso:RetryCounter>
```

```
        <iso:CertificateHolderAuthorizationTemplate>
7F4C12060904007F0007030102025305000100980 0
        </iso:CertificateHolderAuthorizationTemplate>
        <iso:CertificationAuthorityReference>ZZCVCAATA0001
        </iso:CertificationAuthorityReference>
        <iso:EFCardAccess>
31820264300D060804007F000702020202010230 0F060A04007F000702020302020201023 0
0F060A04007F000702020402020201013 02F060804007F000702020616234177542065504 1
202D2042447220476D6248202D20546573746B617274652076312E303081FE060904007F00
07020203023081F0060B04007F000701010502020230 81E0020101302C06072A8648CE3D01
01022100A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E537730
440420A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E53740420
662C61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04044104A3E8
EB3CC1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F42D996C823439C5
6D7F7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE022100A9FB57DBA1EEA9BC3E
660A909D838D718C397AA3B561A6F7901E0E82974856A70201013081FE060904007F000702
0204023081F0060B04007F000701010502020230 81E0020101302C06072A8648CE3D010102
2100A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377304404
20A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E53740420662C
61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04044104A3E8EB3C
C1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F42D996C823439C56D7F
7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE022100A9FB57DBA1EEA9BC3E660A
909D838D718C397AA3B561A6F7901E0E82974856A7020101
        </iso:EFCardAccess>
        <iso:IDPICC>
4F5311EC8F92D60040EA63365E2B06C832856CDE1CE5F8B3C7E7696DAD7628BD
        </iso:IDPICC>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticateResponse>
  </ns2:Body>
</ns2:Envelope>
```

A.5 TA

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=341FA8890EE9FDEE11384F354B48E21D; Path=/; Secure
Content-Type: application/vnd.paos+xml
Content-Length: 3357
Date: Mon, 28 Jun 2010 10:48:53 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:MessageID>urn:uuid971902ad07b243cb3680a5f5b1d3027141e73206
    </ns5:MessageID>
    <ns5:ReplyTo>
      <ns5:Address>https://live.eid-service.de:443</ns5:Address>
    </ns5:ReplyTo>
    <ns5:RelatesTo>urn:uuid8c1f43be885675878167e6886fd838bd1b63b511
    </ns5:RelatesTo>
    <ns5:Action>http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
    </ns5:Action>
  </ns1:Header>
  <ns1:Body>
    <iso:DIDAuthenticate
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:ConnectionHandle xsi:type="iso:ConnectionHandleType">
        <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
        </iso:SlotHandle>
      </iso:ConnectionHandle>
      <iso:DIDName>PIN</iso:DIDName>
      <iso:AuthenticationProtocolData xsi:type="iso:EAC2InputType">
        <iso:EphemeralPublicKey>
8D44E99377DA28436D2F7E8620347D7C08B186B179633E3654842E940AB179B498F974970D
990D47C61FE5D4D91EBB10831E824EC6F2600D89D6661CDF47F734
```

```
        </iso:EphemeralPublicKey>
        <iso:Certificate>
7F2181E47F4E819D5F290100420D5A5A435643414154413030303 17F494F060A04007F0007
02020202038641045 2DD32EAFE1FBBB4000CD9CE75F66636CFCF1EDD44F7B1EDAE25B84193
DA04A91C77EE87F5C8F959ED276200DE33AB574CE9801135FF4497A37162B7C8548A0C5F20
0E5A5A4456434141544130303030357F4C12060904007F000703010202530570030 1FFB75F
2506010000060101 5F2406010001000 3015F37406F13AE9A6F4EDDB7839FF3F04D71E0DC37
7BC4B08FAD295EED241B524328AD0730EB553497B4FB66E9BB7AB90815F04273F09E751D7F
D4B861439B4EE65381C3
        </iso:Certificate>
        <iso:Certificate>
7F218201427F4E81FB5F290100420E5A5A4456434141544130303030357F494F060A04007F
000702020202038641047 0C07FAA329E927D961F490F5430B395EECF3D2A538194D8B637DE
0F8ACF60A9031816AC51B594097EB211FB8F55FAA8507D5800EF7B94E024F9630314116C75
5F200B5A5A444B423230303033557F4C12060904007F000703010202530500030 1DF045F25
060100000601085F2406010000070001655E732D060904007F000703010302802 07C190193
2DB75D08539F2D4A27C938F79E69E083C442C068B299D185BC8AFA78732D060904007F0007
030103018020BFD2A6A2E4237948D7DCCF7975D71D40F15307AA59F580A48777CBEED093F5
4B5F3740618F584E4293F75DDE8977311694B69A3ED73BBE43FDAFEC11B7ECF054F84ACB12
31615338CE8D6EC332480883E14E0664950F85134290DD716B7C153232BC96
        </iso:Certificate>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticate>
  </ns1:Body>
</ns1:Envelope>
```

A.6 TA Response

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 2328
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
```

```
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
        <ns3:MetaData>
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:RelatesTo>urn:uuid971902ad07b243cb3680a5f5b1d3027141e73206
    </ns5:RelatesTo>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:MessageID>urn:uuid9a2157b6c211d5d21be26605cc446f15512997e3
    </ns5:MessageID>
  </ns2:Header>
  <ns2:Body>
    <iso:DIDAuthenticateResponse
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
```

```
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
      <iso:AuthenticationProtocolData xsi:type="iso:EAC2OutputType">
        <iso:Challenge>1331F2B1571E6DC2</iso:Challenge>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticateResponse>
  </ns2:Body>
</ns2:Envelope>
```

A.7 CA

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=77444A9D88892CFF90A803EA20D94A40; Path=/; Secure
Content-Type: application/vnd.paos+xml
Content-Length: 2160
Date: Mon, 28 Jun 2010 10:48:55 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:MessageID>urn:uuidb77981dee98ef0544b58d4da423275c267380c47
    </ns5:MessageID>
    <ns5:ReplyTo>
      <ns5:Address>
https://live.eid-service.de:443
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:RelatesTo>urn:uuid9a2157b6c211d5d21be26605cc446f15512997e3
    </ns5:RelatesTo>
    <ns5:Action>http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
    </ns5:Action>
  </ns1:Header>
  <ns1:Body>
    <iso:DIDAuthenticate
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
```

```
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:ConnectionHandle xsi:type="iso:ConnectionHandleType">
        <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
        </iso:SlotHandle>
      </iso:ConnectionHandle>
      <iso:DIDName>PIN</iso:DIDName>
      <iso:AuthenticationProtocolData xsi:type="iso:EACAdditionalInputType">
        <iso:Signature>
7117D7BF95D8D6BD437A0D43DE48F42528273A98F2605758D6A3A2BFC38141E7577CABB4F8
FBC8DF152E3A097D1B3A703597331842425FE4A9D0F1C9067AC4A9
        </iso:Signature>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticate>
  </ns1:Body>
</ns1:Envelope>
```

A.8 CA Response

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 6306
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
```

```
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>
http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
        <ns3:MetaData>
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:RelatesTo>urn:uuidb77981dee98ef0544b58d4da423275c267380c47
    </ns5:RelatesTo>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:MessageID>urn:uuid46e562f84619a07feaf80c5ce368fd8ea7df69dc
    </ns5:MessageID>
  </ns2:Header>
  <ns2:Body>
    <iso:DIDAuthenticateResponse
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
```

```
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
      <iso:AuthenticationProtocolData xsi:type="iso:EAC2OutputType">
        <iso:EFCardSecurity>
3082078F06092A864886F70D010702A08207803082077C020103310F300D06096086480165
030402010500308203120608040 07F0007030201A0820304048203003182 02FC300D060804
007F000702020202010230 0F060A04007F000702020302020201023 00F060A04007F000702
020402020201013 01B060A04007F000702020502033009020101020106010100020201403 02F060804007F0007
020206162341775420655041202D2042447220476D6248202D20546573746B617274652076
312E30305C060904007F000702020102304F300906072A8648CE3D020103420004565C5705
D40C1B3B8B8E7CA91DB919060FC8C58CDA0A5D568635D7D7AEEC91B163AC43107347551275
5A57C5BBFA3398244CDA41DF8B5C8F33200302A7DF83BF3081FE060904007F000702020302
3081F0060B04007F000701010502020 23081E0020101302C06072A8648CE3D0101022100A9
FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E537730440420A9FB
57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E53740420662C61C430
D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04044104A3E8EB3CC1CFE7
B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F42D996C823439C56D7F7B22E1
4644417E69BCB6DE39D027001DABE8F35B25C9BE022100A9FB57DBA1EEA9BC3E660A909D83
8D718C397AA3B561A6F7901E0E82974856A70201013081FE060904007F0007020204023081
F0060B04007F000701010502020 23081E0020101302C06072A8648CE3D0101022100A9FB57
DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E537730440420A9FB57DB
A1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E53740420662C61C430D84E
A4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04044104A3E8EB3CC1CFE7B773
2213B23A656149AFA142C47AAFBC2B79A191562E1305F42D996C823439C56D7F7B22E14644
417E69BCB6DE39D027001DABE8F35B25C9BE022100A9FB57DBA1EEA9BC3E660A909D838D71
8C397AA3B561A6F7901E0E82974856A7020101A0820327308203233082 02C9A00302010202
0104300A06082A8648CE3D040302306331 0B300906035504061302444531 1D301B06035504
0A0C1442756E646573647275636B6572656920476D62483111300F060355040B0C08546573
7420655041311430120603550403 0C0B546573742043412065504131 0C300A060355040513
03303031301E170D303931303232323232303030305A170D313031303330323230303030 5A30
66310B300906035504061302444531 1D301B060355040A0C1442756E646573647275636B65
72656920476D62483111300F060355040B0C085465737420655041311730150603550403 0C
0E546573742044532065504120303431 0C300A060355040513033030343082013330 81EC06
072A8648CE3D02013081E0020101302C06072A8648CE3D0101022100A9FB57DBA1EEA9BC3E
660A909D838D726E3BF623D52620282013481D1F6E5377304404207D5A0975FC2C3057EEF6
7530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9042026DC5C6CE94A4B44F330B5D9BB
D77CBF958416295CF7E1CE6BCCDC18FF8C07B60441048BD2AEB9CB7E57CB2C4B482FFC81B7
AFB9DE27E1E3BD23C23A4453BD9ACE3262547EF835C3DAC4FD97F8461A14611DC9C2774513
2DED8E545C1D54C72F046997022100A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561
```

```
A6F7901E0E82974856A702010103420004538E0F9A735FE29E31865D2E7532228C6ADA2AC7
1A3280E34414F56757AECB5B490C5441636B4163E709051D37A0F74AD6EB029D8E90196652
9522E922E6AC0AA3818E30818B301F0603551D230418301680144D29BD40BEA355A664B653
B297AA11EAC8D14F2C302B06092A864886F70D010915041E041C312E322E3237362E302E38
302E312E31322E302E32302E352E312E30302B0603551D1004243022800F32303039313032
333030303030305A810F323031303130333033233353935395A300E0603551D0F0101FF0404
03020780300A06082A8648CE3D040302034800304502210 0A829E7A8D58FE267D601D72B9B
FDCCAEBEFA9B0248D923FD146001AF33C45103022063F53FFE30DE44D521401D5416486E19
17DB7A2CFBA4965F0D26F6F72E6325BC318201233082011F02010130683063310B30090603
55040613024445311D301B060355040A0C1442756E6465736472756 36B6572656920476D62
483111300F060355040B0C0854657374206550413114301206035504030C0B546573742043
4120655041310C300A06035504051303303031020104300D06096086480165030402010500
A04A301706092A864886F70D010903310A060804007F0007030201302F06092A864886F70D
0109043122042073 91C2A90E67C6C10BD0CDA73059ADD93DB09CF2B49802D3A2FBEA3A3257
E5DD300C06082A8648CE3D040302050004473045022100A964F46134B5BFA25BAB8A7DF2D6
32807FC5198FDB549F81688701B57DAF8559022064A16D655F685E2A93A769F55DFBCD4F64
635BDED361C8111984914E56D1C7F8
        </iso:EFCardSecurity>
        <iso:AuthenticationToken>9C8042733AA7CEB5
        </iso:AuthenticationToken>
        <iso:Nonce>87D35E0797F0B7F5</iso:Nonce>
      </iso:AuthenticationProtocolData>
    </iso:DIDAuthenticateResponse>
  </ns2:Body>
</ns2:Envelope>
```

A.9 Transmit

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=ED4F13CA27A4E4F8390E822339BABDD4; Path=/; Secure
Content-Type: application/vnd.paos+xml
Content-Length: 2683
Date: Mon, 28 Jun 2010 10:48:58 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:MessageID>urn:uuid68892c74badee1c6809b84b8159bf2c4852f8da7
    </ns5:MessageID>
    <ns5:ReplyTo>
```

```
      <ns5:Address>https://live.eid-service.de:443
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:RelatesTo>urn:uuid46e562f84619a07feaf80c5ce368fd8ea7df69dc
    </ns5:RelatesTo>
    <ns5:Action>http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
    </ns5:Action>
  </ns1:Header>
  <ns1:Body>
    <iso:Transmit xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
      </iso:SlotHandle>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
0CA4040C1D871101C1452A63D6480171126816A70BFD0AC18E083A24FC6D6823AE9700
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
0CB084000D9701FA8E080A359D5CA02F4CCC00
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
0CB085000D9701FA8E08860C91732AAAEE1E00
```

```
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
0CB088000D9701FA8E082158931E5F80715000
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
0CB091000D9701FA8E0873A5E34880683B3700
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
      <iso:InputAPDUInfo xsi:type="iso:InputAPDUInfoType">
        <iso:InputAPDU>
8C2080001D871101647B9E37D94FE8D62CE60D12D7EA19E08E083D5E13C3C34E63F900
        </iso:InputAPDU>
      </iso:InputAPDUInfo>
    </iso:Transmit>
  </ns1:Body>
</ns1:Envelope>
```

A.10 Transmit Response

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 2810
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
        <ns3:MetaData>
```

```
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:RelatesTo>urn:uuid68892c74badee1c6809b84b8159bf2c4852f8da7
    </ns5:RelatesTo>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:MessageID>urn:uuid92cff41d8575fd6efa4523cd52e8af005fcc5368
    </ns5:MessageID>
  </ns2:Header>
  <ns2:Body>
    <iso:TransmitResponse
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
      <iso:OutputAPDU>
990290008E08630A38437449D4689000
      </iso:OutputAPDU>
```

```
      <iso:OutputAPDU>
871101C5C1A3C365072229163DBC9115DFE136990262828E08044D5F992E09D17C6282
      </iso:OutputAPDU>
      <iso:OutputAPDU>
871101FC28D39D706A40204EE439E706449655990262828E086F198583A77188AF6282
      </iso:OutputAPDU>
      <iso:OutputAPDU>
87110120775AB074FB804D4C41D3919672B3A4990262828E088ECBFEA1699F9F646282
      </iso:OutputAPDU>
      <iso:OutputAPDU>
874101260F37DF1B6049F9D384E48A88C5005C08941587E4234CF33FEF7341A82D7A62B9A0
8E6C0A366F71CD34B96D3CC2845314FE774B7484847F505282FD301B2544990262828E08B1
3FC4E54BA081B86282
      </iso:OutputAPDU>
      <iso:OutputAPDU>990290008E086F99EFCC92DAEB229000
      </iso:OutputAPDU>
    </iso:TransmitResponse>
  </ns2:Body>
</ns2:Envelope>
```

A.11 CardApplicationEndSession

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=199C9C73B94E68651575BD7AA991652B; Path=/; Secure
Content-Type: application/vnd.paos+xml
Content-Length: 1888
Date: Mon, 28 Jun 2010 10:49:00 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:MessageID>urn:uuid40d9518035f2258184cf0cb6c50dbb065af3842a
    </ns5:MessageID>
    <ns5:ReplyTo>
      <ns5:Address>https://live.eid-service.de:443
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:RelatesTo>urn:uuid92cff41d8575fd6efa4523cd52e8af005fcc5368
    </ns5:RelatesTo>
    <ns5:Action>http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
```

```
    </ns5:Action>
  </ns1:Header>
  <ns1:Body>
    <iso:CardApplicationEndSession
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <iso:ConnectionHandle xsi:type="iso:ConnectionHandleType">
        <iso:SlotHandle>05D4F40AEBD9919383C22216055EA3DB15056C51
        </iso:SlotHandle>
      </iso:ConnectionHandle>
    </iso:CardApplicationEndSession>
  </ns1:Body>
</ns1:Envelope>
```

A.12 CardApplicationEndSession Response

```
POST /?sessionid=e29270f1b8e745edcc9fe0b9caa9 HTTP/1.1
Content-Length: 2234
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:2003-08","urn:liberty:2006-08";
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
Host: live.eid-service.de:443

<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns4="urn:liberty:paos:2003-08"
```

```
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns2:Header>
    <ns3:PAOS ns2:actor="http://schemas.xmlsoap.org/soap/actor/next"
ns2:mustUnderstand="1">
      <ns3:Version>urn:liberty:2006-08</ns3:Version>
      <ns3:Version>urn:liberty:2003-08</ns3:Version>
      <ns3:EndpointReference>
        <ns3:Address>http://www.projectliberty.org/2006/01/role/paos
        </ns3:Address>
        <ns3:MetaData>
          <ns3:ServiceType>
http://www.bsi.bund.de/ecard/api/1.0/PAOS/GetNextCommand
          </ns3:ServiceType>
          <ns3:Options/>
        </ns3:MetaData>
      </ns3:EndpointReference>
    </ns3:PAOS>
    <ns5:RelatesTo>urn:uuid40d9518035f2258184cf0cb6c50dbb065af3842a
    </ns5:RelatesTo>
    <ns5:ReplyTo>
      <ns5:Address>http://www.projectliberty.org/2006/02/role/paos
      </ns5:Address>
    </ns5:ReplyTo>
    <ns5:MessageID>urn:uuid0f0484e72ab8a375b6cf6c3435a4c44d16277e92
    </ns5:MessageID>
  </ns2:Header>
  <ns2:Body>
    <iso:CardApplicationEndSessionResponse xsi:type="iso:ResponseType"
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:olsc="http://www.openlimit.com/ecard/api/ext/acbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:dssades="urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:dssx="urn:oasis:names:tc:dss-x:1.0:profiles:SignaturePolicy:schema#"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
    </iso:CardApplicationEndSessionResponse>
  </ns2:Body>
</ns2:Envelope>
```

A.13 StartPAOS Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=B3002AC26DC48543FB702C4F5A4DC156; Path=/; Secure
Content-Type: text/html
Content-Length: 1379
Date: Mon, 28 Jun 2010 10:49:02 GMT

<ns1:Envelope xmlns:ns2="urn:liberty:paos:2003-08"
xmlns:ns1="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="urn:liberty:paos:2006-08"
xmlns:ns5="http://www.w3.org/2005/03/addressing">
  <ns1:Header>
    <ns5:RelatesTo>urn:uuid0f0484e72ab8a375b6cf6c3435a4c44d16277e92
    </ns5:RelatesTo>
  </ns1:Header>
  <ns1:Body>
    <iso:StartPAOSResponse xsi:type="iso:ResponseType"
xmlns:ecdsa="http://www.w3.org/2001/04/xmldsig-more#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:vr="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:dsse="urn:oasis:names:tc:dss-x:1.0:profiles:encryption:schema#"
xmlns:ec="http://www.bsi.bund.de/ecard/api/1.1"
xmlns:tsl="http://uri.etsi.org/02231/v2#"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:XAdES="http://uri.etsi.org/01903/v1.3.2#"
```

```
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ers="http://www.setcce.org/schemas/ers"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <dss:Result>
        <dss:ResultMajor>/resultmajor#ok</dss:ResultMajor>
      </dss:Result>
    </iso:StartPAOSResponse>
  </ns1:Body>
</ns1:Envelope>
```

Literaturverzeichnis

[1] Android. http://www.android.com/ (Abruf am 16. August 2010). Open Handset Alliance.

[2] AusweisApp. https://buergerclient.siemens.com (Abruf am 16. August 2010). http://www.openlimit.com/.

[3] BlueZ - Official Linux Bluetooth protocol stack. http://www.bluez.org/ (Abruf am 23. August 2010). BlueZ Project.

[4] Bouncy Castle Crypto APIs. http://www.bouncycastle.org/ (Abruf am 16. August 2010). The Legion Of The Bouncy Castle.

[5] CL RC632 Multiple Protocol Contactless Reader IC. http://www.nxp.com/acrobat/other/identification/SFS070632.pdf (Abruf am 30. August 2010). NXP Semiconductors.

[6] Deutsche Telekoms Smart Card Operating System. T-Systems International GmbH.

[7] iPhone. http://www.apple.com/iphone/ (Abruf am 16. August 2010). Apple Inc.

[8] ISO 7816 - Identification cards – Integrated circuit cards. ISO/IEC.

[9] ISO 7816 - Identification cards – Integrated circuit cards – Part 15: Cryptographic information application. ISO/IEC.

[10] ISO 7816 - Identification cards – Integrated circuit cards – Part 9: Commands for card management. ISO/IEC.

[11] Java Card Technology. http://www.oracle.com/technetwork/java/javacard/overview/index.html (Abruf am 16. August 2010). Oracle Corporation.

[12] Java Platform, Micro Edition. http://www.oracle.com/technetwork/java/javame/index.html (Abruf am 16. August 2010). Oracle / SUN.

[13] Liberty Reverse HTTP Binding for SOAP Specification, Version v1.1. http://www.projectliberty.org/liberty/content/download/1219/7957/file/liberty-paos-v1.1.pdf (Abruf am 16. August 2010). Liberty Alliance Project.

[14] Liberty Reverse HTTP Binding for SOAP Specification, Version v2.0. http://www.projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf (Abruf am 16. August 2010). Liberty Alliance Project.

[15] Liberty Specifications. http://www.projectliberty.org/ (Abruf am 16. August 2010). Liberty Alliance Project.

[16] Maemo Development Platform. http://maemo.org/ (Abruf am 16. August 2010). Nokia Group.

[17] mobile Firefox. https://www.mozilla.com/en-US/mobile/ (Abruf am 16. August 2010). Mozilla Corporation.

[18] NFC Forum Specifications. http://www.nfc-forum.org/specs (Abruf am 16. August 2010). NFC Forum.

[19] Nokia NFC Bluetooth PCSC Reader. http://code.google.com/p/nfcbtpcsc/ (Abruf am 16. August 2010).

[20] OpenMoko. http://www.openmoko.com/ (Abruf am 16. August 2010). Openmoko, Inc.

[21] Safari. http://www.apple.com/safari/ (Abruf am 16. August 2010). Apple Inc.

[22] Technische Richtlinien. https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/index_htm.html (Abruf am 16. August 2010). Bundesamt für Sicherheit in der Informationstechnik.

[23] The GNU Transport Layer Security Library. http://www.gnu.org/software/gnutls/ (Abruf am 16. August 2010). Free Software Foundation, Inc.

[24] The OpenSSL Project. http://www.openssl.org/ (Abruf am 16. August 2010). OpenSSL Software Foundation.

[25] The PC/SC Specification. http://www.pcscworkgroup.com/ (Abruf am 16. August 2010). PC/SC Workgroup.

[26] The WebKit Open Source Project. http://webkit.org/ (Abruf am 16. August 2010).

[27] TS 15480 - Identification card systems – European Citizen Card. CEN/TC 224/WG 15.

[28] Wireshark. http://www.wireshark.org/ (Abruf am 16. August 2010).

[29] ISO 7498 - Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, Juni 1996. ISO/IEC.

[30] PKCS #15: Cryptographic Token Information Format Standard Version 1.1. http://www.rsa.com/rsalabs/node.asp?id=2141 (Abruf am 16. August 2010), Juni 2000. RSA Laboratories.

[31] ISO 14443 - Identification cards — Contactless integrated circuit(s) cards – Proximity cards, Februar 2001. ISO/IEC.

[32] ISO 7816 - Identification cards – Integrated circuit cards – Part 8: Commands for security operations, 2004. ISO/IEC.

[33] ISO 7816 - Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange, 2005. ISO/IEC.

[34] JSR-000257 Contactless Communication API. `http://jcp.org/en/jsr/detail?id=257` (Abruf am 16. August 2010), Oktober 2006. Nokia Corporation.

[35] JSR-000268 Java Smart Card I/O API. `http://jcp.org/en/jsr/detail?id=268` (Abruf am 16. August 2010), Dezember 2006. Sun Microsystems, Inc.

[36] ISO 24727 - Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 1: Architecture, Februar 2007. ISO/IEC.

[37] ISO 24727 - Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 3: Application Interface, 2007. ISO/IEC.

[38] ISO 24727 - Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 4: API Administration, 2007. ISO/IEC.

[39] Secure Interoperable ChipCard Terminal V 1.20. `http://www.teletrust.de/projekte/sicct/` (Abruf am 16. August 2010), November 2007. Teletrust e.V.

[40] ISO 24727 - Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 2: Generic card interface, 2008. ISO/IEC.

[41] Security Assertion Markup Language (SAML) V2.0 Technical Overview. `http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf` (Abruf am 16. August 2010), March 2008. OASIS Security Services TC.

[42] Identification card systems — European Citizen Card — Part 3: European Citizen Card Interoperability using an application interface, WD 1.56, 2009. CEN/TC 224/WG15.

[43] ISO 24727 - Identification cards - Integrated circuit card programming interfaces – Part 6: Registration authority procedures for the authentication protocols for interoperability, 2009. ISO/IEC.

[44] Technische Richtlinie TR-03112, eCard-API-Framework Version 1.1. `https://www.bsi.bund.de/cln_165/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html` (Abruf am 16. August 2010), July 2009. Bundesamt für Sicherheit in der Informationstechnik.

[45] Technische Richtlinie TR-03112, eCard-API-Framework – Part 7 – Protocols Version 1.1. https://www.bsi.bund.de/cln_165/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html (Abruf am 16. August 2010), July 2009. Bundesamt für Sicherheit in der Informationstechnik.

[46] Technische Richtlinie TR-03119, Anforderungen an Chipkartenleser mit ePA-Unterstützung Version 1.1. https://www.bsi.bund.de/cln_165/ContentBSI/Publikationen/TechnischeRichtlinien/tr03119/index_htm.html (Abruf am 16. August 2010), 2009. Bundesamt für Sicherheit in der Informationstechnik.

[47] Technische Richtlinie TR-03130, eID-Server Version 1.1. https://www.bsi.bund.de/cln_165/ContentBSI/Publikationen/TechnischeRichtlinien/tr03130/tr-03130.html (Abruf am 16. Mai 2010), 2009. Bundesamt für Sicherheit in der Informationstechnik.

[48] Technische Richtlinie TR-03110, Advanced Security Mechanisms for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE) and Restricted Identification (RI), Version 2.03. https://www.bsi.bund.de/cln_165/ContentBSI/Publikationen/TechnischeRichtlinien/tr03110/index_htm.html (Abruf am 16. August 2010), März 2010. Bundesamt für Sicherheit in der Informationstechnik.

[49] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Updated by RFC 5246.

[50] Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik, Frystyk, Winer, and Dave. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508 (Abruf am 16. August 2010), Mai 2000. W3C.

[51] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[52] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746.

[53] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.

[54] J. Kemp et al. Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf (Abruf am 16. August 2010), March 2005. OASIS SSTC.

[55] S. Cantor et al. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf` (Abruf am 16. August 2010), March 2005. OASIS Security Services TC.

[56] S. Cantor et al. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. `http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf` (Abruf am 16. August 2010), March 2005. OASIS SSTC.

[57] W. Müller F. Morgner, D. Oepen. OpenPACE. `http://sourceforge.net/projects/openpace/` (Abruf am 16. August 2010). Systems Architecture Group, Institut für Informatik, Humboldt-Universität zu Berlin.

[58] Moritz Horsch. MobilePACE - Password Authenticated Connection Establishment implementation on mobile devices. `http://www.cdc.informatik.tu-darmstadt.de/reports/reports/Moritz_Horsch.bachelor.pdf` (Abruf am 16. August 2010), September 2009. TU Darmstadt.

[59] D. Hühnlein and M. Bach. How to use ISO/IEC 24727-3 with arbitrary smart cards. `http://www.ecsec.de/pub/2007_TrustBus.pdf` (Abruf am 16. August 2010), 2007. Springer.

[60] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010.

[61] L. Rousseau. Muscle PCSC lite. `http://pcsclite.alioth.debian.org/` (Abruf am 16. August 2010).

[62] Harald Welte. librfid - A Free Software RFID stack. `http://openmrtd.org/projects/librfid/` (Abruf am 16. August 2010).

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel *"Mobile eCard-API"* selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 4. Oktober 2010

Kristian Beilke

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Institutes für Informatik der Humboldt-Universität ausgestellt werden darf.

Berlin, den 4. Oktober 2010

Kristian Beilke