Mobile Authentisierung mit dem neuen Personalausweis (MONA)

Mobile authentication using the new German identity card

Master-Thesis von Moritz Horsch

Juli 2011

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Kryptographie und Computeralgebra

Mobile Authentisierung mit dem neuen Personalausweis (MONA)
Mobile authentication using the new German identity card

Vorgelegte Master-Thesis von Moritz Horsch

1. Gutachten: Dr. Alexander Wiesmaier
2. Gutachten: Johannes Braun

Tag der Einreichung:

CASED

Center for Advanced Security Research Darmstadt

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 18. Juli 2011

(Moritz Horsch)

Danksagung

Ich bedanke mich herzlich bei Prof. Dr. Johannes Buchmann, Dr. Alexander Wiesmaier, Johannes Braun und Dr. Detlef Hühnlein für die Möglichkeit mich mit diesem sehr interessanten und innovativen Thema befassen zu können sowie für die hervorragende Unterstützung und Betreuung bei der Entwicklung und Verfassung dieser Arbeit.

Für die exzellente Zusammenarbeit und Unterstützung sowie sein großes Engagement möchte ich mich herzlich bei Herrn Dr. Joachim Schaaf, Deutsche Telekom Laboratories, bedanken. Für die hervorragende Unterstützung bezüglich des Personalausweises danke ich herzlich Frau Alexandra Wahler und Frau Dr. Annette G. Kersten von T-Systems.

Ein herzliches Dankeschön gilt auch Herrn Andreas Staab, media transfer AG, für die Unterstützung bei der Umsetzung der Kommunikation mit dem eID-Server und der Entwicklung der Web-Applikation.

Ich danke auch denen, dessen Namen nicht genannt wurden.

Konventionen

Für die Darstellung und Hervorhebung bestimmter Informationen gelten in diesem Dokument die folgenden typografischen Konventionen:

- *Kursive Schrift*
  Die kursive Auszeichnung wird zur Hervorhebung von neuen Begriffen und für Eigennamen, Zitate, Formeln und mathematische Konstrukte verwendet.
- Serifenlose Schrift
  Die serifenlose Schrift wird für Elemente einer graphischen Benutzeroberfläche wie Buttons oder Menüpunkte verwendet.
- `Typewriter Schrift`
  Diese Schrift wird im Text für die Bezeichnungen von Modulen, Paketen, Klassen, Methoden, Variablen und Sterotypen sowie Protokollnachricht, URLs und Dateinamen bzw. -pfade verwendet. Ebenfalls wird die Schrift für Quellcode bzw. Auflistungen verwenden.

Zusammenfassung

Eine Authentisierung im Internet beruht weitestgehend auf dem Nachweis eines Benutzernamens und dem dazugehörigen Passwort. Der neue Personalausweis soll mit Hilfe seiner Online-Ausweisfunkion im elektronischen Umfeld eine sicherere Alternative zu diesem Verfahren bieten. Für die Nutzung wird eine eID-Applikation und ein Kartenleser mit kontaktloser Schnittstelle benötigt. Die eID-Anwendung bildet das Bindeglied zwischen den einzelnen Kommunikationsparteien und stellt die Interaktion mit dem Benutzer bereit.

Anstatt des klassischen stationären eID-Szenarios mit Computer und Kartenleser, wird bei der mobilen Authentisierung mit dem neuen Personalausweis (MONA) ein Mobilfunkgerät mit *Near Field Communication* (NFC) Technologie und mobiler eID-Anwendung verwendet. Das Mobilfunkgerät vereint damit Computer und die Kartenleser-Funktionalität über die NFC-Schnittstelle. Die entwickelte eID-Applikation MONA wurde als Java MIDlet implementiert und erfolgreich, in Verbindung mit einer eigens entwickelten Web-Applikation, auf dem Nokia 6212 umgesetzt. Damit ist es möglich, sich gegenüber der, im Web-Browser des Nokia aufgerufenen, Web-Applikation mit seinem Personalausweis zu authentisieren. Die Anwendung weist eine Dateigröße von 424 KB auf, wobei ca. 50 KB der graphischen Benutzeroberfläche zuzusprechen sind. Durch eine Aufteilung der Implementierung in unterschiedliche Programmmodule, die von plattformabhängigen Schnittstellen separieren, ist ein Kernmodul entwickelt worden, dass die Basis für die mobile eID-Applikation MONA bildet, aber auch für eine stationäre Anwendung verwendet werden kann. Die entwickelte Web-Applikation unterstützt bereits beide Varianten und kann mit MONA und der AusweisApp als stationäres Pendant genutzt werden.

Im Wesentlichen fungiert MONA als Intermediär zwischen dem Ausweis und dem eID-Server, daher besteht ein Großteil der Operationen im Weiterleiten der Nachrichten. Zusätzlich müssen die Daten in die jeweiligen, vom Ausweis bzw. eID-Server erwarteten, Datenstrukturen transformiert werden. Das Design und die Architektur von MONA verwendet hier ein Ebenenmodell. Die einzelnen Schichten implementieren dabei ein ganz bestimmte Aufgabe und kommunizieren nur über einheitliche und fest definierte Schnittstellen. Dies ermöglicht ein komfortables Hinzufügen neuer Ebenen sowie das Ändern existierender Bestandteile. Insbesondere mit Blick auf eine Weiterentwicklung stand eine hohe Wiederverwendbarkeit und Flexibilität der entwickelten Anwendung im Vordergrund.

Die Laufzeit von MONA beträgt auf dem Nokia 6212 ca. 26,3 Sekunden für einen vollständigen Durchlauf. Die Laufzeitanalyse wurde dabei automatisiert und ohne Benutzerinteraktion durchgeführt, das heißt unter anderem ohne die Eingabe der PIN. Referenzmessungen auf einem Rechner mit stationärem Kartenleser haben 7,3 Sekunden ergeben. Als zeitkritische Operationen gelten die Berechnungen auf den elliptischen Kurven beim *Password Authenticated Connection Establishment* (PACE) Protokoll und

die Serialisierung der XML-Datenstrukturen bei den eCard-API-Nachrichten. Hinzu kommt die veraltete Technologie der Internetanbindung des Nokia Gerätes über das *EDGE* (Enhanced Data Rates for GSM Evolution) Mobilfunknetz. Die hohe Laufzeit ist daher insbesondere auf das im Jahre 2008 erschienene und sehr ressourcen-beschränkte Nokia 6212 zurückzuführen. Jedoch war dieses, zum Beginn des Projekts, das einige einsetzbare Gerät für die Umsetzung des Szenarios. Von der Gesamtlaufzeit von 26,3 Sekunden entfallen 5,4 Sekunden auf die Operationen des Personalausweises und 6,2 Sekunden auf die Übertragung über das Mobilfunknetz. Die entspricht allein ca. 44 % der Gesamtlaufzeit und wird durch externe Faktoren wie Ausweis und Endgerät verursacht. Der Verbindungsaufbau der TLS-Verbindung ist mit 2,5 Sekunden, PACE 7,8 Sekunden, EAC 4,3 Sekunden und das Auslesen der Daten mit 1,6 Sekunden zu bemessen. Bei ersten Tests und Messungen auf leistungsstarken und modernen Geräten zeigten sich jedoch erhebliche Effizienzsteigerungen, so dass mit aktueller Hardware eine Laufzeit von 15 Sekunden möglich erscheint.

Seit Anfang dieses Jahres sind neue NFC-fähige Smartphones auf Basis der Android-Plattform verfügbar. Bei der Entwicklung von MONA wurde bereits eine komfortable Portierbarkeit auf andere Plattformen berücksichtigt. Da MONA auf Basis der Testinfrastruktur und -ausweisen entwickelt wurde, sind Anpassungen vorzunehmen, um den jetzigen aktuellen Spezifikationen zu entsprechen.

Mit der Entwicklung von MONA konnte ein großer Schritt in Richtung mobiler Nutzung des neuen Personalausweises realisiert werden. Interessant sind jetzt die neuen Möglichkeiten durch aktuelle Endgeräte und die Umsetzung der verbleibenden Anforderungen für den Wirkbetrieb.

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

$\widetilde{PK}$	Ephemeral Public Key
$\widetilde{SK}$	Ephemeral Private Key
D	Domain-Parameter
PK_{PCD}	Terminal Authentication Public Key
PK_{PICC}	Chip Authentication Public Key
SK_{PCD}	Terminal Authentication Private Key
SK_{PICC}	Chip Authentication Private Key
AMS	Application Management Software
APDU	Application Protocol Data Unit
API	Application Programming Interfaces
APK	Android Package
ASN.1	Abstract Syntax Notation One
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Chip Authentication
CAN	Card Access Number
CAR	Certification Authority Reference
CDC	Connected Device Configuration
CHAT	Certificate Holder Authorization Template
CHR	Certificate Holder Reference
CLCD	Connected Limited Device Configuration
CMAC	Cipher-based Message Authentication Code
CSMA	Carrier Sense Multiple Access
CSP	Cryptographic Service Provider
CVCA	Country Verifying Certificate Authority
DES	Data Encryption Standard
DRM	Digital Rights Management
DV	Document Verifier

EAC	Extended Access Control
ECC	Elliptic Curve Cryptography
EDGE	Enhanced Data Rates for GSM Evolution
eID	Electronic Identity
FBZ	Fehlbedienungszähler
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HSDPA	High Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IDE	Integrated Development Environment
IdP	Identity Provider
IFD	Interface Device
JAD	Java Archive Descriptor
Java ME	Java Micro Edition
JAXB	Java API for XML Processing
JNLP	Java Network Launching Protocol
JSR	Java Specification Request
KDF	Key Derivation Function
L10N	Localization
LWUIT	Lightweight User Interface Toolkit
MIDP	Mobile Information Device Profile
MONA	Mobile Authentisierung mit dem neuen Personalausweis
MRZ	Machine Readable Zone
MTU	Maximum Transmission Unit
MVC	Model View Controller
NFC	Near Field Communication
nPA	Neuer Personalausweis
OMA	Open Mobile Alliance
PACE	Password Authenticated Connection Establishment
PAOS	Reverse HTTP binding for SOAP

PC/SC	Personal Computer/Smart Card
PCD	Proximity Coupling Device
PICC	Proximity Integrated Circuit Card
PIN	Personal Identification Number
PRNG	Pseudo Random Number Generator
PSK	Pre-shared Key
PUK	Personal Unblocking Key
QES	Qualifizierte elektronische Signatur
RFID	Radio-frequency Identification
RI	Restricted Identification
SDK	Software Development Kit
SigG	Signaturgesetz
SP	Service Provider
TA	Terminal Authentication
TAN	Transaktionsnummer
TLS	Transport Layer Security
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VfB	Vergabestelle für Berechtigungszertifikate
WLAN	Wireless Local Area Network
WSDL	Web Services Description Language
XML	Extensible Markup Language

Teil I.

Einführung

„The best way to predict the future is to implement it.“
— *David Heinemeier Hanaon*

1 Motivation

In den letzten Jahren wurden immer mehr Geschäftsprozesse in das Internet verlagert. Interaktive Online-Angebote im elektronischen Handel sowie im Service- und Dienstleistungsbereich werden weitläufig von Unternehmen aller Branchen angeboten. Überwiegend wird dabei auf eine Authentisierung der Kunden mittels Benutzername und Passwort gesetzt. Dass jedoch auch der Einsatz sicherer Authentisierungsmechanismen ein fortlaufender Prozess ist, zeigen Banken beim Online-Banking. Zu Beginn wurden klassische Listen mit Transaktionsnummern (TAN-Listen) zur Bestätigung einer Transaktion verwendet. Es folgte die *indizierte TAN* (iTAN) Liste und die heute präferierte *Mobile TAN* (mTAN).

Mit dem neuen Personalausweis steht neben der hoheitlichen Anwendung jetzt auch eine Online-Ausweisfunktion zur Verfügung, die die Sicherheit beim E-Business und E-Government deutlich erhöhen soll. Ziel ist es, den Ausweis ubiquitär zur Authentisierung im elektronischen Bereich einsetzen zu können und Benutzerkonten mit gleichen und unsicheren Passwörtern entgegenzuwirken. Das Szenario für die Nutzung des Personalausweises beschränkt sich jedoch auf das stationäre Umfeld. Das Internet und die damit verbundenen Anwendungen verlagern sich aber stetig in den mobilen Bereich. Auch die jüngsten Entwicklungen rund um die Tablet-Computer zeigen eine starke Verlagerung weg vom stationären Betätigungsfeld hin zur mobilen und portablen Nutzung, die das gewohnte Maß von Notebooks weit überschreitet. Stationäre Kartenleser mit Dimension eines Smartphones widersprechen dieser Entwicklung.

Mobile Endgeräte mit *Near Field Communication* (NFC) Technologie, als Kartenleser für den neuen Personalausweis, können als Initiator für eine mobile Nutzung fungieren und zu einer breiten Nutzung des Ausweises beisteuern. Trotz Subventionsprogramm ist die Verbreitung von Kartenlesern gering. Ein NFC-fähiges Smartphone als Kombination aus Computer und Kartenleser schafft vielleicht die erforderlichen Rahmenbedingungen und weckt das Interesse an der Nutzung des Ausweises. Das Ziel dieser Arbeit ist es, eine mobile Authentisierung mit dem neuen Personalausweis (MONA) zu ermöglichen. Die mobile Anwendung trägt ebenfalls den Namen MONA.

2 Ziel der Arbeit

Der *elektronische Identitätsnachweis* (eID-Funktion) des neuen Personalausweises ermöglicht einen Identitätsnachweis im elektronischen Umfeld. Der Ausweisinhaber kann seinen neuen Personalausweis für Registrierungs- und Anmeldeprozesse im Internet verwenden. Der Nutzer besucht per Web-Browser das Online-Angebot eines Dienstanbieters, das von einem Web-Server bereitgestellt wird. Der Anbieter wünscht eine Authentisierung per eID-Funktion, bevor er den Zugriff auf sein Angebot frei gibt. Als Beispiel sei ein Online-Shop oder Web-Mail Anbieter genannt. Die Authentisierung wird jedoch nicht von dem Dienstanbieter durchgeführt, sondern von einem separaten eID-Server.

Für das stationäre Szenario wird für die Kommunikation mit dem Ausweis ein, mit dem Computer verbundener, Kartenleser mit kontaktloser Schnittstelle benötigt. Zusätzlich muss auf dem Computer eine eID-Anwendung installiert sein. Die Anwendung verwaltet die Kommunikation mit dem Ausweis, implementiert Sicherheitsmechanismen und stellt die Benutzerinteraktion bereit. Bei der Nutzung der eID-Funktion startet die Anwendung und zeigt dem Ausweisinhaber Informationen über den Dienstanbieter wie beispielsweise Name, Internetadresse und Anschrift an. Zusätzlich werden dem Benutzer die erbetenen Zugriffsrechte auf seine einzelnen Daten angezeigt, die er noch weiter einschränken kann. Der Ausweisinhaber bestätigt den Auslesevorgang mit der Eingabe seiner PIN. Nach einer erfolgreichen Durchführung werden die ausgelesenen Daten vom eID-Server zum Web-Server des Dienstanbieters übertragen. Die Daten werden dann für einen Registrierungs- und Anmeldeprozess verwendet. Abschließend wird auf dem Computer wieder das Online-Angebot des Dienstanbieters im Web-Browser aufgerufen und der Ausweisinhaber kann das Angebot nutzen.

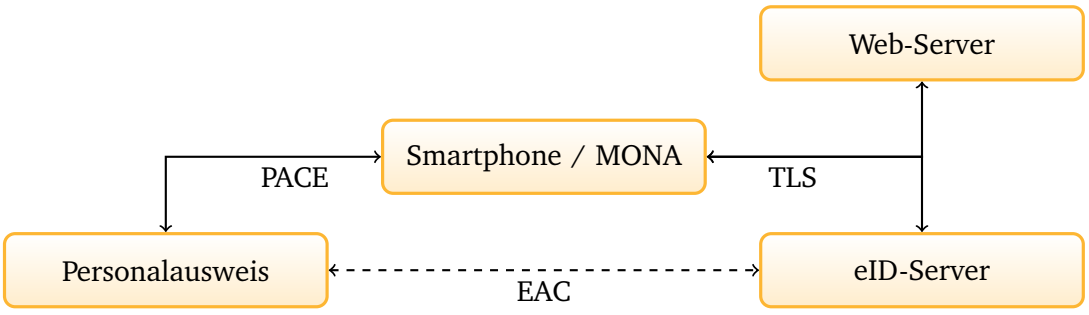


Abbildung 2.1.: eID-Szenario

Bei der mobilen Authentisierungen mit dem neuen Personalausweis wird anstatt des Computers mit Kartenleser ein NFC-fähiges mobiles Gerät verwendet. Der Aufbau des Szenarios ist in Abbildung 2.1 illustriert. Dabei bleibt das äußere Gerüst mit Ausweis, eID- und Web-Server bestehen. Das NFC-fähiges Smartphone vereinigt Kartenleser, Computer und eID-Anwendung. Die Kommunikation zwischen der

eID-Anwendung — stationär oder mobil — und dem eID- und Web-Server wird, wie in Abbildung 2.1 illustriert, per *Transport Layer Security* (TLS) Protokoll abgesichert. Eine Authentisierung des Ausweisinhabers anhand seiner PIN und die Absicherung der Übertragung zwischen der eID-Anwendung und dem Ausweis erfolgt über das *Password Authenticated Connection Establishment* (PACE) Protokoll. Das *Extended Access Control* (EAC) Protokoll beinhaltet eine Berechtigungsprüfung des eID-Servers und eine Echtheitsprüfung des Ausweises.

Ziel dieser Arbeit ist die Entwicklung einer mobilen eID-Applikation als Pendant zur stationären Anwendung im klassischen eID-Szenario. Dabei soll der vollständige Ablauf einer mobilen Authentisierung mit dem neuen Personalausweis und einem Smartphone gegenüber einer Web-Applikation realisiert werden. Dazu muss die mobile Anwendung alle erforderlichen Protokolle, Funktionen und Abläufe implementieren, die durch die Schnittstellen und Vorgaben der anderen Teilnehmer zu erfüllen sind. Maßgeblich für die Umsetzung sind die betreffenden internationalen Standards und die relevanten Spezifikationen des *Bundesamt für Sicherheit in der Informationstechnik* in Bezug auf den Personalausweis. Die Anwendung muss sich maßgeschneidert in das stationäre eID-Szenario integrieren lassen, da Anpassungen am Ausweis und eID-Server nicht umsetzbar sind. Für die mobile Authentisierung gilt es eine Web-Applikation zu entwickeln, die einen Anmeldeprozess mit dem Ausweis unterstützt und auf das mobile eID-Szenario und insbesondere das mobile Endgerät ausgelegt ist. Zusätzlich gilt es, eine flexible, wiederverwendbare und leicht zu erweiternde Implementierung zu entwickeln. Der Fokus liegt dabei auf einer einfachen Weiterentwicklung und Portierung auf andere mobile Plattformen, aber auch eine Einsatz im stationären Umfeld zu ermöglichen.

Zu Beginn dieses Dokuments werden ausführliche Hintergrundinformationen zum neuen Personalausweis und den verwendeten Technologien bereitgestellt sowie die einzelnen Bausteine für die Umsetzung des eID-Szenarios vorgestellt. Teil II dieses Dokuments geht dabei auf den neuen Personalausweis mit seinen Eigenschaften, Funktionen und Anwendungsgebieten ein. Zusätzlich werden die Sicherheitsmechanismen vorgestellt, die die Authentizität, Integrität und Vertraulichkeit des Ausweisdokumentes und der darauf gespeicherten Daten sicherstellen. In Teil III werden die Technologien und weitere technische Details erläutert, die für die Umsetzung benötigt werden. Die technische Umsetzung und Implementierung der eID-Applikation MONA wird in Teil IV dieses Dokuments vorgestellt. Dazu werden zuerst die Anforderungen und die Architektur erläutert. Darauf folgt die Beschreibung der Implementierung der einzelnen Komponenten. Teil IV enthält ebenfalls Informationen zu der für MONA entwickelten Web-Applikation. Eine Bewertung der Entwicklung bietet abschließend Teil V. Dazu zählt eine Analyse der Laufzeit und Vorschläge für Optimierungen sowie ein Ausblick auf zukünftige Arbeiten. Dabei werden Aspekte für eine Portierung von MONA auf die Android-Plattform erläutert und benötigte Anpassungen für den Wirkbetrieb der Anwendung aufgezählt.

Teil II.

Der neue Personalausweis

3 Eigenschaften und Funktionen

Das folgenden Kapitel beschreibt die Neuerungen und Anwendungsszenarien des neuen Personalausweises (nPA). Es werden zuerst die Funktionen betrachtet und dann die Anwendungsmöglichkeiten des Ausweises vorgestellt. Das Thema Sicherheit und dessen Gewährleistung der auf dem Ausweis elektronisch gespeicherten Daten sowie die dazugehörigen Protokolle werden in Kapitel 5 behandelt.

Äußerlich sticht insbesondere das neue Scheckkartenformat TD-1 nach ICAO Doc 9303-3 [64] hervor (siehe Abbildung 3.1). Die Bundesregierung sieht Vorteile im handlicheren Format und einer dadurch höheren Akzeptanz bei den Bürgerinnen und Bürgern [24]. Das Ausweisfoto und die personenbezogenen Daten des Inhabers sind weiterhin auf dem Ausweis aufgedruckt. Neben dem neuen Design verfügt der Personalausweis über einen integrierten kontaktlosen Chip zur Speicherung von personenbezogenen und biometrischen Daten. Dazu zählen beispielsweise die persönlichen Daten wie Name, Geburtsdatum, Anschrift und das Ausweisbild. Mit Zustimmung des Inhabers können die biometrischen Daten auch um Fingerabdrücke erweitert werden.

Abbildung 3.1.: Neuer Personalausweis
Bildmaterial [23]

Der **elektronische Identitätsnachweis** (eID-Funktion) bietet dem Ausweisinhaber die Möglichkeit, seine Identität auf elektronischem Wege – beispielsweise im Internet – nachzuweisen. Eine Adressverifikation, Altersüberprüfung, Registrierung sowie eine Authentifizierung bei Anmeldeprozessen können mit dieser Funktion abgebildet werden. Anbieter, die die eID-Funktion unterstützen und Daten vom Ausweis auslesen möchten, müssen ein Berechtigungszertifikat bei der *Vergabestelle für Berechtigungszertifikat* (VfB) beantragen. Dabei sind strikte Vorgaben zu erfüllen und die Gültigkeit der Zertifikate ist auf 48 Stunden beschränkt [50]. Der Zugriff auf biometrische Daten ist bei der eID-Funktion nicht möglich.

Die **elektronische Signatur** Funktion (eSign-Funktion) ermöglicht dem Ausweisinhaber das elektronische Unterschreiben von Dokumenten. Die Funktion gestattet das Erstellen einer *qualifizierten elektronischen Signatur* (QES) [28] gemäß dem Signaturgesetz (SigG) [11] und bietet damit bei rechtsverbindlichen Dokumenten eine gleichwertige Lösung zur klassischen handschriftlichen Unterschrift. Eine elektronische Signatur ist mit einem Anhang eines elektronischen Dokuments vergleichbar und erlaubt es die Identität des Unterzeichners nachzuweisen sowie das Dokument auf unautorisierte Veränderungen hin zu überprüfen. Der neue Personalausweis wird jedoch ohne Signaturzertifikat ausgeliefert. Der Inhaber kann dieses auf Wunsch bei einer Zertifizierungsstelle seiner Wahl beantragen[1]. Zur Nutzung der Signaturfunktion sind Komfort-Chipkartenleser (Cat-K) mit speziellen Berechtigungen notwendig [44].

Die **elektronische Pass** Funktion (ePass-Funktion) bzw. Biometrieanwendung des Ausweises stellt eine Authentisierung für den hoheitlichen Bereich bereit, wie sie vom elektronischen Reisepass bekannt ist. Der Zugriff auf diese Funktion bzw. das Auslesen der persönlichen und biometrischen Daten ist ausschließlich hoheitlichen Stellen mit speziellen Berechtigungen [43] und sogenannten *authentisierten Inspektionsterminals* vorbehalten. Die Funktion bzw. der Ausweis ist konform zur ICAO ePassport Anwendung [63] und damit als vollständiger Ersatz zum elektronischen Reisepass im Schengen-Raum zu betrachten.

Bei der eID- und eSign-Funktion ist eine Authentifizierung nur durch den Besitz der Karte nicht ausreichend. Analog zu EC- und Kreditkarten bedarf es eines geheimen Passwortes, das nur dem Inhaber bekannt ist und anhand dessen er den rechtmäßigen Besitz der Karte nachweisen kann. Der Ausweisinhaber verfügt dabei, sofern beantragt, über eine sechsstellige eID-PIN und eSign-PIN für die jeweilige Funktion. Die einzelnen Passwörter für eine Authentifizierung sind im Folgenden erläutert:

- **Personal Identification Number (PIN)**
  Durch die Eingabe der PIN authentisiert sich der Inhaber gegenüber einem Lesegerät. Für die eID- und eSign-Funktion wird jeweils eine sechsstellige PIN verwendet.

- **Personal Unblocking Key (PUK)**
  Der PUK dient zum Entsperren der PIN, falls diese mehrmals fehlerhaft eingegeben wurde. Nach dreimaliger Falscheingabe der PIN wird diese gesperrt und der PUK ist zum Zurücksetzen des Fehlbedienungszählers erforderlich. Der PUK kann jedoch nur zehnmal zum Zurücksetzen des Fehlbedienungszählers der PIN verwenden werden, danach ist die PIN unwiderruflich gesperrt.

- **Card Access Number (CAN)**
  Die Kartenzugriffsnummer ist auf dem neuen Personalausweis aufgedruckt und kann zur Überprüfung des optischen Zugriffs des Lesegerätes auf den Ausweis verwendet werden. Die CAN wird bei hoheitlichen Inspektionssystemen und Signaturterminals verwendet.

[1] Eine Liste von Zertifizierungsdiensteanbieter stellt die Bundesnetzagentur unter `http://www.bundesnetzagentur.de/cln_1932/DE/Sachgebiete/QES/QES_node.html` bereit.

- **Machine Readable Zone (MRZ)**
  Das MRZ-Passwort ist in einem geschützten Bereich des Chips gespeichert. Vom Lesegerät wird es aus der Dokumentennummer, dem Geburtsdatum und dem Ablaufdatum des Ausweises berechnet. Die MRZ wird ausschließlich im hoheitlich Bereich eingesetzt und stellt die Kompatibilität zur älteren Generation der ePass Anwendung sicher.

Der Chip des neuen Personalausweises ist ein kontaktloser *Radio Frequency Identification* (RFID) Chip nach ISO/IEC 14443 [73]. Im Gegensatz zur klassischen kontaktbehafteten Karten ist dieser in der Lage, die auf dem Chip befindlichen Daten über eine Entfernung von 10 bis 15 cm zu übertragen. Ein elektromagnetisches Feld, das von einem Lesegerät im 13,56 Mhz-Bereich erzeugt wird, dient zur Übertragung der Daten und zur Energieversorgung des neuen Personalausweises. Kontaktlose Chipkarten verfügen im Vergleich zu kontaktbehafteten Karten über eine höhere Haltbarkeit und sind unempfindlich gegen Feuchtigkeit und Verschmutzung. Durch den Einsatz einer kontaktlosen Chipkarte kann die Gültigkeitsdauer von zehn Jahren beibehalten werden. Kontaktbehaftete Karten wie EC- bzw. Kreditkarten verfügen auf Grund der Abnutzungserscheinungen nur über eine Gültigkeitsdauer von zwei Jahren.

Kontaktlose Chipkarten erfordern jedoch neue Anstrengungen in Bezug auf die Datensicherheit. Während bei der Verwendung einer kontaktbehafteten Chipkarte der Besitz oft ausreicht ist, ist diese Voraussetzung bei einer kontaktlosen Karten nicht gegeben. Für eine Kommunikation ist es ausreichend, wenn sich die kontaktlose Karte innerhalb des Übertragungsradius befindet, wodurch ein Abhören und eine unbemerkte Kommunikation möglich ist. Um das Sicherheitsniveau auch bei kontaktlosen Karten wie dem neuen Personalausweis aufrecht zu erhalten, werden Sicherheitsmechanismen eingesetzt, die in Kapitel 5 vorgestellen werden. Im folgenen Kapitel 4 wird eine Übersicht der Anwendungsszenarien des neuen Personalausweises bereitgestellt.

4 Anwendungsszenarien

Der neue Personalausweis geht in seinen Aufgaben und Einsatzmöglichkeiten weit über ein klassisches hoheitliches innerdeutsches Ausweisdokument hinaus. Einige neue Anwendungsmöglichkeiten werden in diesem Abschnitt vorgestellt:

E-Business

Die Authentifizierung bei Internetportalen geschieht im Wesentlichen durch die Angabe einer E-Mail-Adresse bzw. eines Benutzernamens und dem dazugehörigen Passwort. Der elektronische Identitätsnachweis kann diese Art der Authentifizierung sicherer gestalten. Der neue Ausweis kann beispielsweise für die Authentifizierung bei Webmail-Anbietern, Versandhäusern und weiteren Diensten im World Wide Web eingesetzt werden. Beispielsweise bieten die *HUK24* und *LVM Versicherung* auf ihren Internetportalen eine Anmeldung per Ausweis an und die *SCHUFA* ermöglicht eine Registrierung für ihren Dienst *Meine SCHUFA-Auskunft Online* mit dem neuen Personalausweis.
Zusätzlich kann die qualifizierte elektronische Signatur das digitale Unterzeichnen von Dokumenten und eine Verifikation der Echtheit durch den Empfänger ermöglichen. Als Einsatzgebiet ist die elektronische Rechnung denkbar.

E-Government

In der staatlichen Verwaltung werden Daten heute vorwiegend in elektronischer Form verarbeitet. Formulare, Anträge usw. werden jedoch zum größten Teil noch in Papierform eingereicht und müssen digital erfasst werden. Dieser Medienbruch ist zeitaufwendig und fehleranfällig. Der neue Personalausweis soll eine digitale Verwaltung ermöglichen, die es den Bürgerinnen und Bürgern gestattet, ihr Anliegen in elektronischer Form und unabhängig von Öffnungszeiten zu erledigen. Änderungen des Wohnsitzes könnten beispielsweise über ein Bürgerportal der Verbandsgemeinde realisiert werden. Als erste Anwendungen sind beispielsweise die Bürgerportale der Städte Hagen, Köln und Münster zu nennen. Ebenfalls bietet das Kraftfahrt-Bundesamt Einblick in das eigene Punktekonto im Verkehrszentralregister mittels Authentisierung per elektronischem Identitätsnachweis.

Abbildung 4.1 illustriert mögliche Anwendungsszenarien des neuen Personalausweises. Details sind unter anderem in [24] verfügbar sowie unter `http://www.personalausweisportal.de`. Auf dieser Website stellt das *Bundesministerium des Innern* (BMI) unter dem Titel *Neue Möglichkeiten mit dem neuen Personalausweis* eine Liste von Anbietern bereit, die den elektronische Identitätsnachweis auf ihren Internetportalen einsetzen.

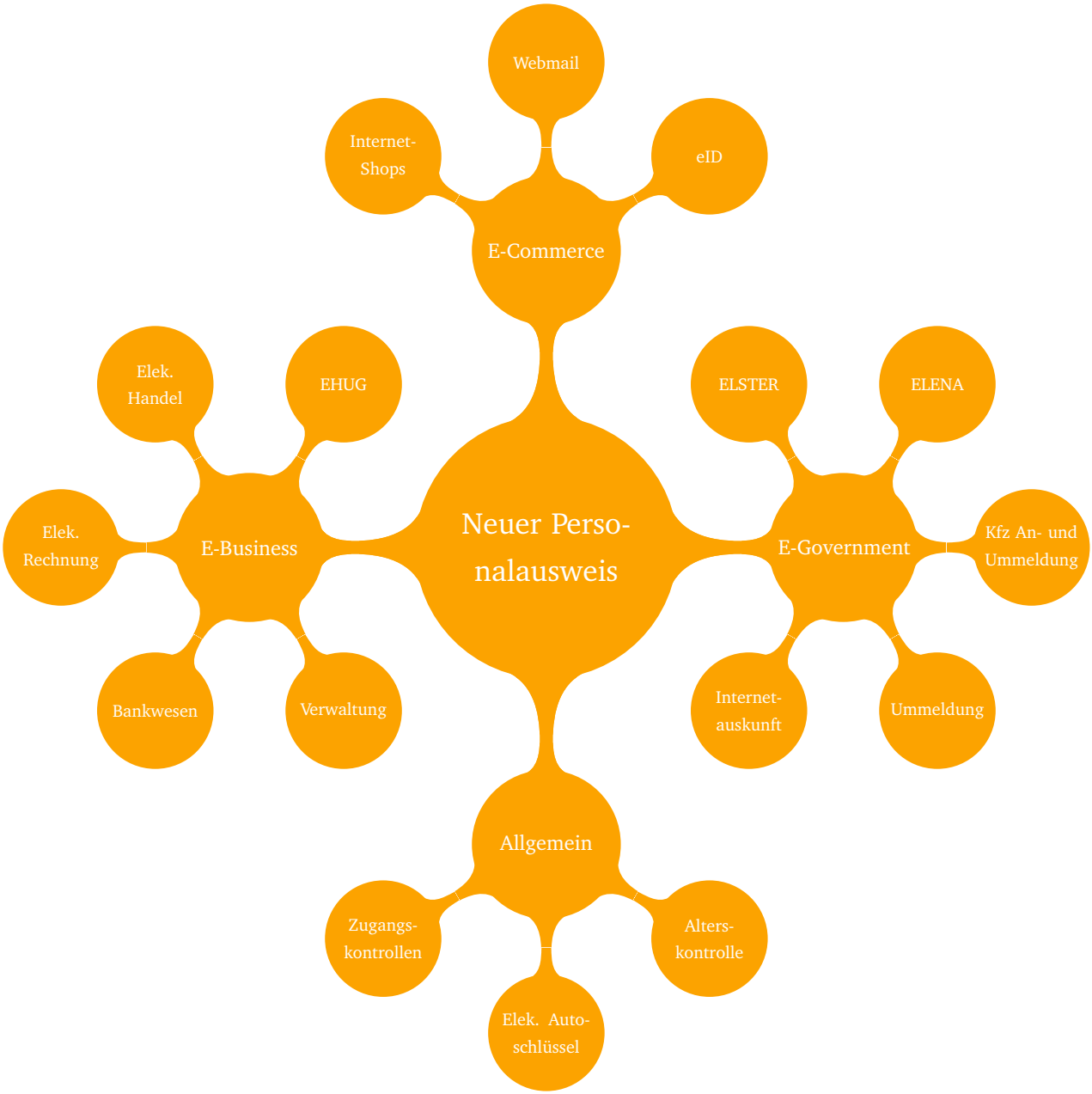


Abbildung 4.1.: Anwendungsgebiete des neuen Personalausweises

5 Sicherheitsmechanismen

Aufgrund der digitalen Speicherung von personenbezogenen Daten des Ausweisinhabers auf dem Chip des neuen Personalausweises bedarf es technischer Maßnahmen zur Sicherung und zum Schutz der Daten. Die Sicherheitsmechanismen müssen eine Authentifizierung des Ausweisinhabers, eine sichere und integere Datenübertragung sowie eine Verifikation des Ausweises und eine Zugriffskontrolle der Lesegeräte bereit stellen. Ziel ist es, sicherzustellen, dass der Ausweisinhaber dem Datenzugriff zugestimmt hat, das Terminal zu dem Zugriff berechtigt ist und dass es sich um einen authentischen Ausweis handelt.

Die Technische Richtlinie TR-03110 [41, Kapitel 1.3] des *Bundesamts für Sicherheit in der Informationstechnik* (BSI) definiert die auf dem Ausweis gespeicherten Daten wie folgt:

- **Weniger vertrauliche Daten**
  Allgemeine Daten wie Name, Geburtsort, Geburtstag und Anschrift werden als *weniger vertrauliche* Daten eingestuft, weil diese leicht auf einem anderen Weg erfasst werden können. Zu diesen Daten wird auch das Passbild gezählt. Weniger vertrauliche Daten sind im Allgemeinen die Daten, die auf dem Ausweis aufgedruckt bzw. abgebildet sind.

- **Vertrauliche Daten**
  Insbesondere die Fingerabdrücke oder andere biometrische Daten sind als *vertraulich* klassifiziert.

Zum Schutz der Daten wurden vom BSI auf Basis des *ICAO Doc 9303* [63] mehrere Protokolle spezifiziert, die die wesentlichen Prinzipien der IT-Sicherheit Vertraulichkeit, Integrität, Verfügbarkeit und Authentizität gewährleisten sollen. Abbildung 5.1 illustriert den Ablauf der Protokolle zum Schutz und der Zugriffskontrolle der persönlichen Daten.

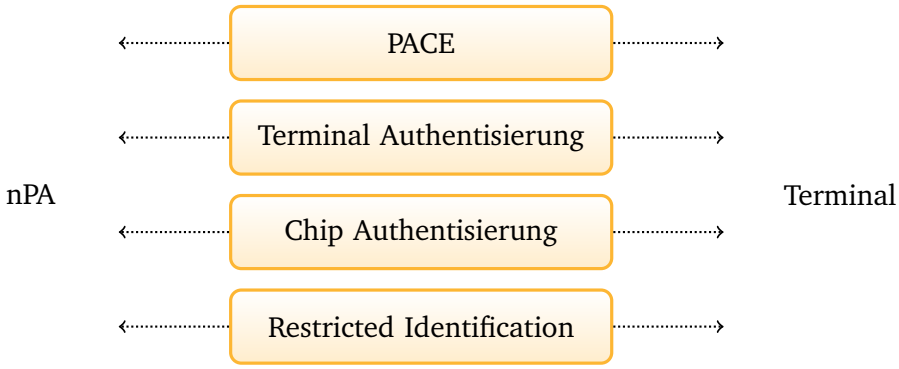


Abbildung 5.1.: Sicherheitsmechanismen

Die Sicherheitsprotokolle aus Abbildung 5.1 für den neuen Personalausweis sind im Folgenden zusammengefasst und werden in den Kapiteln 5.1 bis 5.3 erläutert:

- **Password Authenticated Connection Establishment (PACE)**
  PACE ist ein kryptographisches Passwort-basiertes Protokoll zur Absicherung der kontaktlosen Kommunikation zwischen Personalausweis und Kartenleser und Authentisierung des Karteninhabers.

- **Terminal Authentisierung (TA)**
  Bei der Terminal Authentisierung weist das Terminal bzw. der Kartenleser seine Berechtigung zum Datenzugriff gegenüber dem Ausweis nach.

- **Chip Authentisierung (CA)**
  Der Ausweis beglaubigt seine Echtheit gegenüber dem Terminal mit Hilfe der Chip Authentisierung.

- **Restricted Identification (RI)**
  Die Restricted Identification Funktion ermöglicht das Erstellen von Pseudonymen zum Wiedererkennen eines Ausweises ohne erneutes Auslesen der personenbezogenen Daten.

In den Erläuterungen zu den Protokollen wird im Folgenden der Ausweis mit *Proximity Integrated Circuit Card* (PICC) und das Terminal mit *Proximity Coupling Device* notiert.

5.1 Password Authenticated Connection Establishment

Um die Datenübertragung über die kontaktlose Schnittstelle zwischen dem Ausweis und einem Terminal bzw. Kartenleser abzusichern, bedarf es einer sicheren und verschlüsselten Kommunikation. Klassische Protokolle zum Aufbau einer sicheren Verbindung bzw. eines Schlüsselaustausches, wie beispielsweise dem *Diffie-Hellman Verfahren* [16], ermöglichen keine Authentifizierung der Teilnehmer. Dies ist aber im Umfeld des neuen Personalausweises unerlässlich. Passwort-basierte Protokolle [9] zum Schlüsselaustausch, wie das Password Authenticated Connection Establishment (PACE) Protokoll [41, 65], bieten die Möglichkeit, anhand eines gemeinsamen Passwortes eine Authentifizierung vorzunehmen. Allgemein betrachtet, ist PACE ein kryptografisches Protokoll zum Aufbau einer sicheren und verschlüsselten Verbindung zwischen zwei Teilnehmern. Zur Authentifizierung verwendet PACE ein schwaches Passwort bzw. ein Passwort mit einer geringen Entropie. Aus diesem nicht sicheren Passwort generiert das PACE-Protokoll starke und sicherere Schlüssel für eine vertrauliche Kommunikation.

Es existieren bereits mehrere solcher Passwort-basierten Protokolle (vgl. [9, Kapitel 7]), die aber meist patentiert und weniger für elliptische Kurven geeignet sind. Das vom BSI entwickelte PACE-Protokoll kompensiert diese Nachteile. Die Aufgaben und Ziele des PACE-Protokolls können im Allgemeinen auf folgende Punkte zusammengefasst werden:

- Authentisierung des Ausweisinhabers.
- Berechnung von starken und sicheren Sitzungsschlüsseln für die weitere Kommunikation.
- Kontrolle des optischen Zugriffs des Lesegerätes.

Um den Nachteil einer kontaktlosen Chipkarte in Form einer vom Ausweisinhaber unbemerkten und ungewollten Kommunikation, zu kompensieren muss sichergestellt werden, dass das Lesegerät einen optischen Zugriff auf den Ausweis hat. Durch die Eingabe der PIN wird sichergestellt, dass der Inhaber einer Kommunikation mit seinem Ausweis einwilligt. Das Passwort des Inhabers fließt direkt in die Berechnungen des PACE-Protokolls ein und stellt somit sicher, dass ein Lesegerät ohne korrektes Passwort das Protokoll nicht erfolgreich abschließen kann. Dadurch kann eine sichere Authentifizierung des Ausweisinhabers vorgenommen werden. Ohne korrektes Passwort werden von den Teilnehmern unterschiedliche Schlüssel berechnet und die Verifikation der Schlüssel, die das PACE-Protokoll abschließend durchführt, schlägt fehl.

Für die weitere Kommunikation zwischen dem Ausweis und dem Terminal ist es wichtig, dass diese verschlüsselt fortgesetzt wird. Das PACE-Protokoll berechnet daher Sitzungsschlüssel für die Verschlüsselung und Verifikation der übertragenden Daten. Eine Verschlüsselung allein auf Basis der PIN würde beispielsweise auf Grund der geringen Entropie nur eine unzureichende Sicherheit bieten. Durch die Möglichkeit einer Aufzeichnung der Kommunikation muss eine Sicherheit bzw. ein Schutz der übertragenen Daten für einen sehr langen Zeitraum gewährleistet werden.

PACE: Protokollablauf

Vor dem PACE-Protokoll müssen sich der Ausweis und das Terminal auf Parameter für den Schlüsselaustausch verständigen. Die sogenannten Domain-Parameter D enthalten beispielsweise Parameter der elliptischen Kurve, die unterstützten kryptografischen Protokolle, Hash-Algorithmen usw. Die Parameter sind auf dem Ausweis in der Datei `EF.CardAccess` abgelegt. Die Datei befindet sich im öffentlichen Speicher des Chips und ist ohne Einschränkungen lesbar. Dabei enthält die Datei entweder die vollständigen Domain-Parameter oder nur eine Referenz auf standardisierte Parameter [41, Kapitel A.2.1.1].

Der Ablauf des PACE-Protokolls ist in Abbildung 5.2 illustriert. Das Protokoll beginnt mit dem Austausch einer verschlüsselten Zufallszahl. Als Schlüssel wird K_π verwendet, der unter Verwendung einer *Key Derivation Function* (KDF) [41, Kapitel A.2.3] aus dem Passwort π (PIN, CAN usw.) abgeleitet wird.

1. Der Chip generiert eine Zufallszahl s, verschlüsselt diese mit dem gemeinsamen Schlüssel K_π und verschickt den Chiffretext z an das Terminal. Das Lesegerät entschlüsselt z mit dem Schlüssel K_π und erhält somit die Zufallszahl s.

2. Die Teilnehmer tauschen weitere Daten zur Berechnung der neuen Domain-Parameter D' bzw. des neuen Basispunktes G' aus (siehe dazu Kapitel 5.1 und 5.1).

3. Der Chip und das Terminal berechnen zufällige und flüchtige Diffie-Hellman Schlüssel, die sich aus einem privaten Schlüssel $\widetilde{SK}_{PICC}$ bzw. $\widetilde{SK}_{PCD}$ und einem öffentlichen Schlüssel $\widetilde{PK}_{PICC}$ bzw. $\widetilde{PK}_{PCD}$ zusammensetzen. Der öffentliche Schlüssel wird jeweils an den anderen Teilnehmer gesendet.

4. Beide Kommunikationspartner berechnen den gemeinsamen Schlüssel K aus den Schlüsseln $\widetilde{PK}_{PCD}, \widetilde{SK}_{PICC}$ bzw. $\widetilde{PK}_{PCD}, \widetilde{SK}_{PICC}$. Aus dem Schlüssel K leiten beide einen Sitzungsschlüssel $K_{ENC} = KDF_{ENC}(K, 1)$ zur Verschlüsselung und einen Schlüssel $K_{MAC} = KDF_{MAC}(K, 2)$ zur Authentifizierung ab.

5. Das Terminal und der Chip generieren jeweils einen Authentisierungs-Token T_{PICC} bzw. T_{PCD}, tauschen diesen aus und verifizieren diesen gegenseitig. T_{PICC} bzw. T_{PCD} ist eine Prüfsumme unter Verwendung des Schlüssel K_{MAC} über die neuen Domain-Parameter D'.

	Proximity Integrated Circuit Card		Proximity Coupling Device
(1)	$z = E\left(K_\pi, s\right)$	$\xrightarrow{z}$	$s = D\left(K_\pi, z\right)$
(2)	$G' = \mathrm{Map}()$	$\longleftrightarrow$	$G' = \mathrm{Map}()$
(3)	$\widetilde{PK}_{PICC} = \widetilde{SK}_{PICC} \cdot G'$	$\xleftarrow{\widetilde{PK}_{PCD}}$ $\xrightarrow{\widetilde{PK}_{PICC}}$	$\widetilde{PK}_{PCD} = \widetilde{SK}_{PCD} \cdot G'$
(4)	$K = \widetilde{PK}_{PCD} \cdot \widetilde{SK}_{PICC}$		$K = \widetilde{SK}_{PCD} \cdot \widetilde{PK}_{PICC}$
(5)	$T_{PICC} = MAC\left(K_{MAC}, \widetilde{PK}_{PCD}\right)$	$\xleftarrow{T_{PCD}}$ $\xrightarrow{T_{PICC}}$	$T_{PCD} = MAC\left(K_{MAC}, \widetilde{PK}_{PICC}\right)$

Abbildung 5.2.: Password Authenticated Connection Establishment

PACE: Version 1

Die erste Version des PACE-Protokolls (vgl. TR-03110 Version 2.0 [29]) verwendet für die Berechnung der neuen Domain-Parameter das *Generic Mapping*. Generic Mapping wird beim neuen Personalausweis verwendet. Die Funktion Map : $G \mapsto G'$ aus Abbildung 5.2 (siehe Schritt 2) ist im Allgemeinen ein Diffie-Hellman Verfahren [16] zur Berechnung des neuen Punktes G'. Das Terminal wählt eine Zufallszahl x, berechnet und sendet $X = x \cdot G$. Der Chip verfährt analog. Beide Teilnehmer berechnen dann H und G'. Ein Auszug des PACE-Protokolls unter Verwendung des Generic Mapping ist in Abbildung 5.3 verdeutlicht.

Proximity Integrated Circuit Card		Proximity Coupling Device
	$\vdots$	
$Y = y \cdot G$	$\xleftarrow{X}$ $\xrightarrow{Y}$	$X = x \cdot G$
$H = y \cdot X$		$H = x \cdot Y$
$G' = s \cdot G + H$		$G' = s \cdot G + H$
	$\vdots$	

Abbildung 5.3.: Generic Mapping

Ab der Version 2.01 der Protkoll-Spezifikation ist für PACE neben dem *Generic Mapping* auch das *Integrated Mapping* definiert. Die Funktion Map : $G \mapsto G'$ aus Abbildung 5.2 ist dabei wie folgt definiert: $G' = f\left(R(s \parallel t)\right)$. Die Zufallszahl t wird von Terminal bestimmt und verschlüsselt an den Chip gesendet. Die Funktionen R [109] und f [60, 114] bilden die Konkatenation von s und t auf einen neuen Punkt G' ab. PACE unter Verwendung von Integrated Mapping ist in Abbildung 5.4 illustriert.

Proximity Integrated Circuit Card		Proximity Coupling Device
$z_{PICC} = E\left(K_\pi, s\right)$	$\xrightarrow{z_{PICC}}$	$s = D\left(K_\pi, z_{PICC}\right)$
$t = D\left(K_\pi, z_{PCD}\right)$	$\xleftarrow{z_{PCD}}$	$z_{PCD} = E\left(K_\pi, t\right)$
$G' = f\left(R(s \parallel t)\right)$		$G' = f\left(R(s \parallel t)\right)$
	$\vdots$	

Abbildung 5.4.: Integrated Mapping

5.2 Extended Access Control

Das *Extended Access Control* (EAC) Protokoll [41] ist ein vom BSI entwickeltes System zum Zugriffsschutz der auf dem neuen Personalausweis gespeicherten Daten. Es handelt sich dabei um eine Weiterentwicklung des im ICAO Doc 9303 [63] vorgestellten Verfahrens. Wie in Abbildung 5.5 verdeutlicht, besteht Extended Access Control aus der *Terminal Authentisierung* und der *Chip Authentisierung*, die im Folgenden vorgestellt werden.

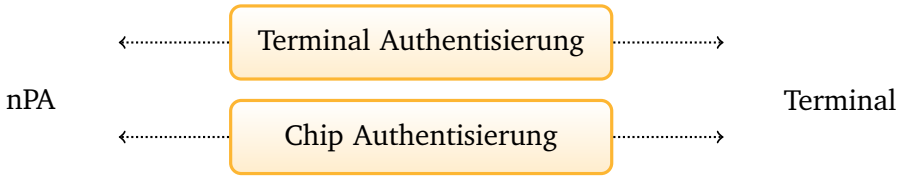


Abbildung 5.5.: Extended Access Control

Das Ziel des EAC-Protokolls, in Verbindung mit dem PACE-Protokoll, ist der Schutz und die Sicherheit der personenbezogenen Daten des Ausweisinhabers. Im ersten Schritt authentisiert das PACE-Protokoll den Ausweisinhaber. Das EAC-Protokoll führt dann eine gegenseitige Authentifizierung zwischen dem Ausweis und dem Terminal durch. Im Gegensatz zum Sicherheitsmechanismus des elektronischen Reisepasses wird beim neuen Personalausweis die Terminal Authentisierung vor der Chip Authentisierung durchgeführt. Dadurch ist sichergestellt, dass ein Ausweis keine Informationen an unautorisierte Terminals preis gibt womit beispielsweise das Erstellen von Bewegungsprofilen unterbunden wird.

Terminal Authentisierung

Der Zugriff auf die Daten des neuen Personalausweises ist nur autorisierten Terminals gestattet. Die Zugriffskontrolle erfolgt mit Zertifikaten, die von den jeweiligen staatlichen Behörden ausgestellt werden. Ist ein Lesegerät in der Lage, ein Zertifikat mit den nötigen Berechtigungen gegenüber dem Chip nachzuweisen und ist es dem Chip möglich, die Zertifikatskette zu verifizieren, erhält das Terminal Zugriff auf die Daten. Nach einer erfolgreichen Terminal Authentisierung muss der Ausweis die gesicherte Verbindung mit dem öffentlichen Schlüssel des Lesegerätes verknüpfen, damit eine Kopplung der Zugriffsrechte an das Terminal bestehen bleibt.

Als Wurzelinstanz für die Vergabe und Bereitstellung der Zertifikate gelten die jeweiligen nationalen Zertifizierungsstellen (Country Verifying Certificate Authority, CVCA). In Deutschland wird diese vom Bundesamt für Sicherheit in der Informationstechnik betrieben. Autorisierte Stellen (Document Verifier, DV) erhalten Berechtigungszertifikate mit individuellen Leserechten von der *Vergabestelle für Berechtigungszertifikate* (VfB), die festlegen, auf welche Daten des Ausweises zugegriffen werden darf.

	Proximity Integrated Circuit Card		Proximity Coupling Device
(1)		$\xleftarrow{C}$	$C = C_{CVCA} \parallel C_{DV_1} \parallel \cdots \parallel C_{DV_n} \parallel C_{PCD}$
(2)		$\xleftarrow{\mathbf{Comp}\left(\widetilde{PK}_{PCD,TA}\right),\ A_{PCD}}$	$\widetilde{PK}_{PCD,TA} = \widetilde{SK}_{PCD,TA} \cdot G'$
(3)		$\xrightarrow{ID_{PICC} \parallel r_{PICC,TA}}$	
(4)		$\xleftarrow{s_{PCD}}$	$s_{PCD} = \mathbf{Sign}\Big(SK_{PCD}, ID_{PICC} \parallel r_{PICC,TA} \parallel \mathbf{Comp}\left(\widetilde{PK}_{PCD,TA}\right) \parallel A_{PCD}\Big)$
(5)	$\mathbf{Verify}\Big(PK_{PCD}, s_{PCD}, ID_{PICC} \parallel r_{PICC,TA} \parallel \mathbf{Comp}\left(\widetilde{PK}_{PCD,TA}\right) \parallel A_{PCD}\Big)$		

Abbildung 5.6.: Terminal Authentisierung

Die einzelnen Schritte des Protokollablaufs aus Abbildung 5.6 werden im Folgenden stichpunktartig erläutert:

1. Das Terminal sendet eine Zertifikatskette C, die unter anderem das Zertifikat des Terminals (C_{PCD}) und der nationalen Zertifizierungsstelle (C_{CVCA}) enthält. Die Zertifikatskette muss es dem Chip ermöglichen, dass Zertifikat des Terminals zu verifizieren.

2. Das Terminal wählt ein zufälliges Schlüsselpaar und sendet den öffentlichen Schlüssel in komprimierter Form $\mathbf{Comp}\left(\widetilde{PK}_{PCD,TA}\right)$ an den Chip. Optional können weitere Daten A_{PCD} übermittelt werden.

3. Der Chip wählt eine Zufallszahl $r_{PICC,TA}$ und sendet diese zusammen mit einem Identifikator ID_{PICC} an das Terminal. Beim Einsatz von Basic Access Control (BAC) [63] wird die Dokumentennummer des Ausweises als ID_{PICC} verwendet. Im Falle von PACE berechnet sich dieser aus $ID_{PICC} = \mathbf{Comp}\left(\widetilde{PK}_{PICC}\right)$, wobei $\widetilde{PK}_{PICC}$ aus dem Protokollverlauf des PACE-Protokolls stammt (siehe Abbildung 5.2, Schritt 3).

4. Das Terminal erstellt die Signatur s_{PCD} und versendet sie an den Ausweis.

5. Abschließend verifiziert der Chip die Signatur.

Chip Authentisierung

Die Chip Authentisierung hat die Aufgabe die Echtheit des Ausweises bzw. dessen Chip nachzuweisen, um beispielsweise kopierte Ausweise zu erkennen. Der neue Personalausweis muss einem Lesegerät nachweisen, dass er einen einmaligen und individuellen privaten Schlüssel besitzt. Dieser Schlüssel ist im geschützten und nicht zugänglichen Bereich des Chips gespeichert. Das heißt, er kann nicht ausgelesen und demnach nicht kopiert werden. Beim Klonen eines Ausweises können ggf. Informationen einzelner Datengruppen kopiert werden, jedoch nicht der private Schlüssel des Chips.

Chip Authentisierung: Protokollablauf

Abbildung 5.7 illustriert den Protokollablauf. Die einzelnen Schritte werden im Folgenden erläutert:

1. Der Chip sendet seinen statischen öffentlichen Schlüssel PK_{PICC} an das Lesegerät.
2. Das Terminal berechnet den öffentlichen Schlüssel $\widetilde{PK}_{PCD,CA}$ aus dem privaten Schlüssel $\widetilde{SK}_{PCD,TA}$, der bereits während der Terminal Authentisierung gewählt wurde (siehe Abbildung 5.6, Schritt 2). Der Chip berechnet darauf hin die komprimierte Form des Schlüssels **Comp** $\left(\widetilde{PK}_{PCD,CA}\right)$ und vergleicht diesen mit dem während der Terminal Authentisierung empfangenen Schlüssel **Comp** $\left(\widetilde{PK}_{PCD,TA}\right)$. Die komprimierte Form beider Schlüssel muss identisch sein, sonst wird die Authentisierung abgebrochen.
3. Beide Teilnehmer berechnen den gemeinsamen Schlüssel K. In der Berechnung werden die statischen Schüssel SK_{PICC} und PK_{PICC} des Chips verwendet.
4. Der Chip berechnet die neuen gemeinsamen Sitzungsschlüssel $K_{ENC} = KDF_{ENC}(K, r_{PICC,CA}, 1)$ und $K_{MAC} = KDF_{MAC}(K, r_{PICC,CA}, 2)$. Der Authentisierungs-Token T_{PICC} und die Zufallszahl $r_{PICC,CA}$ werden an das Terminal gesendet. Unter Verwendung der Zufallszahl $r_{PICC,CA}$ berechnet nun das Terminal ebenfalls die gemeinsamen Sitzungsschlüssel und verifiziert den Authentisierungs-Token.

	Proximity Integrated Circuit Card		Proximity Coupling Device
(1)	$PK_{PICC} = SK_{PICC} \cdot G'$	$\xrightarrow{PK_{PICC}}$	
(2)		$\xleftarrow{\widetilde{PK}_{PCD,CA}}$	$\widetilde{PK}_{PCD,CA} = \widetilde{SK}_{PCD,TA} \cdot G'$
(3)	$K = SK_{PICC} \cdot \widetilde{PK}_{PCD,CA}$		$K = \widetilde{SK}_{PCD,TA} \cdot PK_{PICC}$
(4)	$T_{PICC} = MAC\left(K_{MAC}, \widetilde{PK}_{PCD,CA}\right)$	$\xrightarrow{T_{PICC}, r_{PICC,CA}}$	

Abbildung 5.7.: Chip Authentisierung

5.3 Restricted Identification

Das *Restricted Identification* (RI) Protokoll kann zum Erstellen von Pseudonymen verwendet werden. Ein Pseudonym kann dann für die Identifizierung bzw. Wiedererkennung verwendet werden. Ein Pseudonym ist an den Chip und an einen spezifischen Sektor gebunden, d. h. beispielsweise alle Terminals eines Dienstanbieters. Es besteht jedoch die Möglichkeit ein Pseudonym sektorübergreifend zu verwenden. Anhand eines Pseudonyms ist es möglich, dass autorisierte Terminals den neuen Personalausweis wiedererkennen, ohne (erneut) die personenbezogenen Daten auslesen zu müssen. Restricted Identification wird erst nach erfolgreicher Terminal und Chip Authentisierung durchgeführt. Die Restricted Identification wird auch für den Sperrlistenabgleich verwendet.

Restricted Identification: Protokollablauf

Den Protokollablauf illustriert Abbildung 5.6. Die einzelnen Schritte werden im Folgenden erläutert:

1. Das Terminal sendet seinen sektorspezifischen öffentlichen Schlüssel PK_{Sector}.
2. Der Chip berechnet und sendet eine Prüfsumme des öffentlichen Schlüssels des Terminals und dem eigenen privaten Schlüssel.
3. Das Terminal prüft den sektorspezifischen Identifikator I_{ID}^{Sector}, ob dieser vom Document Verifier (DV) annulliert bzw. für ungültig erklärt wurde.

	Proximity Integrated Circuit Card		Proximity Coupling Device
(1)		$\xleftarrow{PK_{Sector}}$	
(2)	$I_{ID}^{Sector} = \text{Hash}\left(SK_{ID}, PK_{Sector}\right)$	$\xrightarrow{I_{ID}^{Sector}}$	

Abbildung 5.8.: Restricted Identification

5.4 Zusammenfassung

In Abbildung 5.9 wird der Ablauf der Sicherheitsmechanismen nochmal verdeutlichen und auf die Abhängigkeiten bzw. Interaktionen zwischen den Protokollen eingegangen. Jegliche Kommunikation vor und während des PACE-Protokolls erfolgt unverschlüsselt, so zum Beispiel das Auslesen der Datei `EF.CardAccess` oder das Selektieren der Anwendung.
Für die Kommunikation während der Terminal und Chip Authentisierung wird *Secure Messaging* [41, 69] verwendet, das heißt eine verschlüsselte und integere Verbindung unter Verwendung der von PACE berechneten Sitzungsschlüssel K_{ENC} und K_{MAC}. In die Berechnungen der Terminal Authentisierung fließt der Schlüssel $\widetilde{PK}_{PICC}$ aus dem PACE-Protokoll mit ein. Aus diesem flüchtigen Schlüssel des Chips wird der Identifikator $ID_{PICC} = \textbf{Comp}\left(\widetilde{PK}_{PICC}\right)$ berechnet, der auch in die Signatur s_{PCD} einfließt (siehe Abbildung 5.6, Schritt 3 - 5). In den Berechnungen der Chip Authentisierung wird der Schlüssel $\widetilde{PK}_{PCD,CA}$ aus der Terminal Authentisierung verwendet und nach erfolgreicher Authentifizierung neue Sitzungsschlüssel K_{ENC} und K_{MAC} berechnet. Diese Schlüssel werden dann für das Secure Messaging während der Restricted Identification bzw. der weiteren Kommunikation verwendet.

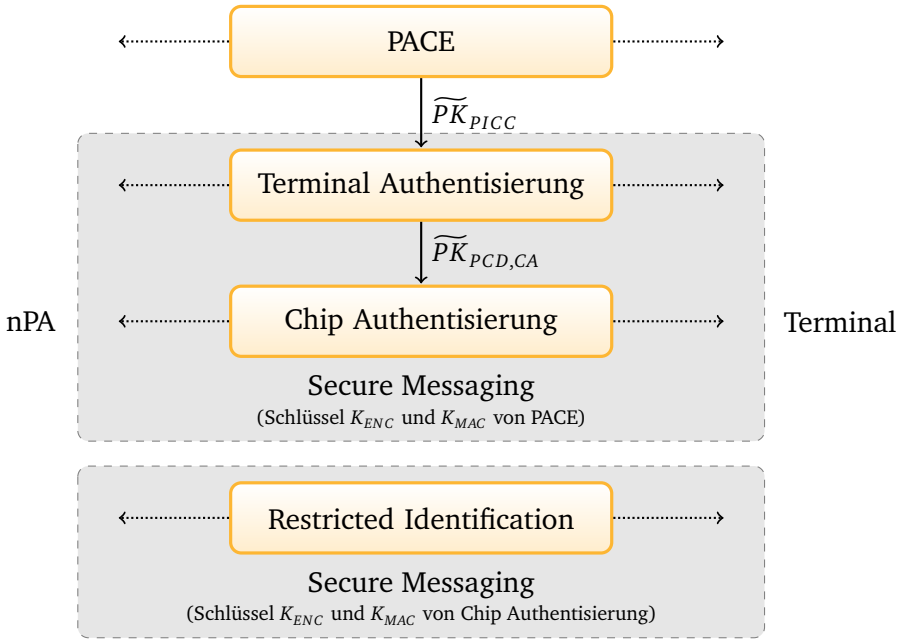


Abbildung 5.9.: Ablauf und Interaktion der Sicherheitsmechanismen

Diese Interaktion zwischen den Protokollen verdeutlicht, dass keines dieser Protokolle autonom abläuft, sondern diese aufeinander aufbauen. Dadurch entsteht eine hohe Abhängigkeit der einzelnen Berechnungen und ein geringeres Angriffspotenzial. Abbildung 5.9 verdeutlicht ebenfalls, dass die von PACE aufgebaute verschlüsselte Verbindung ausschließlich für die Terminal und Chip Authentisierung eingesetzt wird. Diese Eigenschaft stellt sicher, dass die Sitzungsschlüssel für den eigentlichen Datenaustausch bzw. die Restricted Identification nur einem authentifizierten Chip und Terminal bekannt sind.

Teil III.

Technologien

6 Near Field Communication

Die *Near Field Communication* (NFC) Technologie ist eine Erweiterung der RFID-Technologie und ermöglicht eine Kommunikation zwischen mobilen Geräten und physikalischen Objekten. Technologien zur Datenübertragung wie IrDA, Bluetooth oder WLAN benötigen aktive Komponenten mit eigener Energieversorgung. NFC ist dagegen in der Lage mit aktiven NFC-fähigen Geräten und aktiven oder passiven RFID-Tags, die ohne eigene Energieversorgung auskommen, zu kommunizieren. Dadurch ist es möglich Informationen von RFID-Tags zu lesen, die sich beispielsweise in Kaufhausprodukten oder Werbeplakaten befinden. Ein weiterer bedeutender Aspekt der NFC-Technologie ist die Kommunikation mit kontaktlosen Chipkarten nach ISO/IEC 14443 [73]. Damit lassen sich beispielsweise Anwendungen auf Basis der *Java Micro Edition* (Java ME) entwickeln, die mittels der JSR 257 [102] Spezifikation eine Datenübertragung zwischen Handy und kontaktloser Chipkarte ermöglicht. Insbesondere die einfache Handhabung zeichnet die NFC-Technologie aus. Im Gegensatz zu den klassischen Übertragungsarten in modernen mobilen Geräten, die eine aufwendige Konfiguration, einen hohen Energieverbrauch und einen komplexen Verbindungsaufbau benötigen, bietet NFC eine intuitive und benutzerfreundliche Bedienung. Eine Kommunikation zwischen zwei Komponenten wird bei NFC durch einfaches Zusammenführen der Komponenten hergestellt. Die NFC-Technologie verfügt über eine geringe Reichweite von ca. 10 cm und gestattet ausschließlich sogenannte Punkt-zu-Punkt Verbindungen. Das heißt, es ist nur eine Kommunikation zwischen zwei Komponenten möglich. Die Übertragungsraten von NFC liegen bei 106, 212 oder 424 KBit pro Sekunde. Im Vergleich zur weit verbreiteten Bluetooth-Technologie [82] sind sowohl die Übertragungsreichweite als auch die Übertragungsraten gering. NFC definiert zwei Typen von Protokollen:

- **Near Field Communication Interface Protocol-1 (NFCIP-1)**
  Standardisiert durch die ECMA-340 [20] und ISO/IEC 18092 [68] wird ein Transport-Protokoll, eine Kollisionsbehandlung, die Datencodierung und das Datenformat sowie die Verbindungstypen, die in Abbildung 6.1 abgebildet sind, definiert.

- **Near Field Communication Interface Protocol-2 (NFCIP-2)**
  NFCIP-2 stellt eine Schnittstelle zu bereits bestehenden Standards zur Verfügung und ist in ECMA-352 [19] und ISO/IEC 21481 [70] definiert. Eine Kommunikation ist dadurch mit ECMA-340 [20], ISO/IEC 14443 [73] und ISO/IEC 15396 [71] konformen Komponenten möglich. Als Kollisionsbehandlung ist *Carrier Sense Multiple Access* (CSMA) definiert, um sicherzustellen, dass NFCIP-2 konforme Komponenten keine Kommunikation anderer Geräte stören.

NFC-Geräte können zwei verschiedene Rollen annehmen: aktiv und passiv. In der aktiven Rolle baut das Gerät ein elektromagnetisches Feld auf, das zur Datenübertragung und ggf. zur Energieversorgung des zweiten Teilnehmers dient. Das aktive Gerät liest dann Daten vom Kommunikationspartner. Der Kommunikationspartner fungiert in der passiven Rolle und emulitert eine Smartcard. Die jeweiligen Konfigurationen der Teilnehmer sind in Tabelle 6.1 verdeutlicht.

Einheit A		Einheit B	Beschreibung
Aktiv (Initiator)	⟵	Passiv (Ziel)	Das elektromagnetische Feld wird von Einheit A erzeugt. Die Datenübertragung findet von Einheit B nach A statt.
Aktiv (Initiator/Ziel)	⟷	Aktiv (Initiator/Ziel)	Beim Senden von Daten aktiviert die jeweilige Einheit ein elektromagnetisches Feld. Es kann immer nur ein aktives elektromagnetisches Feld geben.
Passiv (Ziel)	⟶	Aktiv (Initiator)	Das elektromagnetische Feld wird von Einheit B erzeugt. Die Datenübertragung findet von Einheit A nach B statt.

Tabelle 6.1.: Verbindungstypen von NFC

Das erzeugte elektromagnetische Feld weitet sich in alle Richtungen aus, ist jedoch auf einen kleinen Radius begrenzt. Eine Verbindung zwischen zwei aktiven Geräten kommt zustande, wenn sich die elektromagnetischen Felder beider Kommunikationsteilnehmer überschneiden (siehe Abbildung 6.1). Im Falle einer Kommunikation mit einem passiven RFID-Chip ist es ausreichend, wenn sich der Chip innerhalb des Radius befindet.

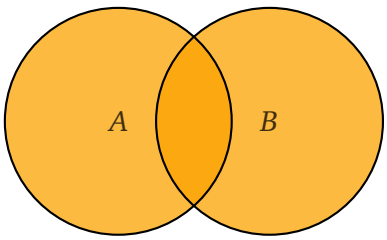


Abbildung 6.1.: Verbindungsradius von NFC

Weitere Informationen zur Near Field Communication Technologie sind unter anderem in dem Artikel *Mobile Services for Near Field Communication* [22] enthalten. Details zum Thema Sicherheit von NFC bieten die Artikel [54, 85, 81].

7 Java MIDlet

Bei der Entwicklung von Java-Applikationen als *Applet* [90] oder *Java Web Start* [91] kann auf eine vollständige *Java Standard Edition* (Java SE) zurückgegriffen werden. Bei Anwendungen für mobile Geräte steht jedoch nur die vom Funktionsumfang eingeschränkte *Java Micro Edition* (Java ME) zur Verfügung. Die Basis der Java ME bilden Profile und Konfigurationen, die eine *Java Virtual Machine* und der dazugehörigen Klassenbibliothek beinhalten. Als Konfigurationen steht die *Connected Device Configuration* (CDC) [97, 99] und die *Connected Limited Device Configuration* (CLDC) [93, 101] zur Verfügung sowie Erweiterungen für gerätespezifische Funktionalitäten. Die CLDC ist im Allgemeinen der De-facto-Standard für die mobilen Geräte und stellt eine Teilmenge der Java SE bereit, aber auch Erweiterungen, die auf mobile Geräte zugeschnitten sind. Profile lassen sich als Ergänzungen für die Konfigurationen beschreiben. Das wichtigste Profil für die CLDC ist das *Mobile Information Device Profile* (MIDP) [94, 98], das Funktionalitäten bzw. *Application Programming Interfaces* (API) bereitstellt. Dazu gehören unter anderem Komponenten für graphische Benutzeroberflächen sowie APIs für die Entwicklung von Spielen, Aufzeichnen und der Wiedergabe von Medien. Ein Anwendung für mobile Geräte, die auf dem MIDP basiert, wird als *MIDlet* bezeichnet.

Die Steuerung von MIDlets geschieht durch die *Application Management Software* (AMS). Dieser startet und informiert die installierten MIDlets über Ereignisse, wie beispielsweise eingehende Telefonanrufe. MIDlets unterstützen daher drei Zuständen: *Pausiert*, *Aktiv* und *Beendet*. Nach dem Starten eines MIDlets befindet sich dieses im Zustand Pausiert und wechselt durch Aufruf der Methode `startApp()` in den Zustand Aktiv. Bei Ereignissen wie einem Telefonanrufe wechselt ein MIDlet zurück in den Zustand Pausiert. Die Zustände und Übergänge sind in Abbildung 7.1 illustriert.

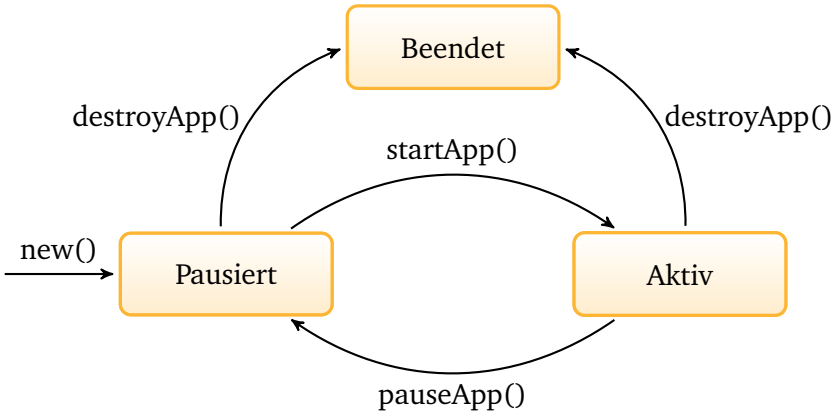


Abbildung 7.1.: Lebenszyklus eines MIDlets

Als Basis für alle MIDlets dient die Klasse `javax.microedition.midlet.MIDlet`. Ein MIDlet implementiert dabei unter anderem folgende Methoden:

- `startApp()`
  Mit Hilfe dieser Methode wechselt das MIDlet in den Zustand Aktiv und hat dann unter anderem Zugriff auf das Display. Die Initialisierung und das Erstellen weiterer Objekt findet im Allgemeinen in dieser Methode statt.

- `pauseApp()`
  Die `pauseApp()` Methode wird beispielsweise bei einem eingehenden Telefonanruf ausgeführt und sollte rechenintensive Prozesse des MIDlets anhalten.

- `destroyApp()`
  Diese Methode wird beim Beenden des MIDlets ausgeführt und sollte das etwaige Speichern von Dateien und Freigeben von Ressourcen beinhalten.

Die Zustandsübergänge können durch das MIDlet oder durch die AMS ausgelöst werden. Ausführliche Informationen zu Konfigurationen, Profilen und Grundlagen zu MIDlets sind in [115, 10, 77] verfügbar.

Eine MIDlet wird als JAR-Datei (JAR, Java Archive) bereitgestellt und benötigt zusätzlich eine JAD-Datei (JAD, Java Archive Descriptor). Diese Datei enthält unter anderem Informationen für die JAR-Datei sowie über die verwendete Konfiguration und das Profil (siehe Auflistung 7.1).

```
MIDlet-1: Main,,main.Application
MIDlet-Jar-Size: 123456
MIDlet-Jar-URL: http://www.example.com/Application.jar
MIDlet-Name: Application
MIDlet-Vendor: Vendor
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.1
```

Auflistung 7.1: JAD-Datei

Diese Eigenschaft ähnelt einer JNLP-Datei (JNLP, Java Network Launching Protocol [103]) zum Starten einer Java Web Start Anwendung. Auch die Installation eines MIDlets über eine Webseite ist vergleichbar. Durch einen Link auf eine JAD- oder die dazugehörige JAR-Datei wird das MIDlet direkt heruntergeladen und auf dem mobilen Gerät installiert.

8 eCard-API-Framework

Das *eCard-API-Framework* bietet Applikationen eine homogene und universelle Schnittstelle zum Zugriff auf verschiedene Chipkarten und deren Funktionen. Das Framework ist in der TR-03112 [41] spezifiziert und soll eine einheitliche Nutzung der Kartenprojekte der Bundesrepublik ermöglichen [59].

Im Folgenden werden einige Eigenschaften und die Architektur des Frameworks erläutert. Ein besonderes Augenmerk liegt auf der Web-Service Schnittstelle, da diese relevant für die Implementierung von MONA ist. Abschließend wird der Einsatz des eCard-API-Frameworks im Umfeld des neuen Personalausweises erläutert.

8.1 Eigenschaften

Im Umfeld des neuen Personalausweises ist beispielsweise die Kommunikation zwischen eID-Applikation und eID-Server durch das eCard-API-Framework definiert. Dabei muss auf dem Client im Allgemeinen nur mit einer Karte kommuniziert werden und der Umfang der Übertragungen hält sich in Grenzen. Auf dem Server hingegen müssen tausende von Anfragen gleichzeitig verarbeitet werden. Das Framework zeichnet sich hier durch eine hohe Skalierbarkeit aus und eignet sich sowohl für eine clientseitige als auch serverseitige Implementierung.

Eine weitere Stärke des Frameworks ist die Unabhängigkeit von einer konkreten Plattform und Programmiersprache. Durch die Modellierung und Strukturierung der Daten mittels XML (Extensible Markup Language) und dem Einsatz von Web Service Schnittstellen setzt das Framework auf etablierte Technologien. Die Web Service Schnittstellen der eCard-API sind mittels Web Services Description Language (WSDL) definiert. Für Java stehen beispielsweise Programme bereit, die die benötigten Datenstrukturen und Methoden in Java-Source bequem aus den WSDL-Dateien erstellen. Somit lässt sich das eCard-API-Framework problemlos als *High-Level-API* einbinden.

Um ein flexibles Framework bereitzustellen abstrahiert die eCard-API von der konkreten Hardware. Damit lassen sich beliebige Smartcards mit verschiedenen Protokollen und unterschiedlichen Hardwareausprägungen einbinden, wie beispielsweise kontaktlose und kontaktbehaftete Karten. Die eCard-API verwendet dafür sogenannte *CardInfo Files*, die die vollständige Funktionalität einer Karte beschreiben. Damit ist es möglich weitere Karten komfortabel zu integrieren, ohne Anpassungen am Framework vornehmen zu müssen. Gleiches gilt auch für die Vielfalt an Kartenlesern.

8.2 Architektur

Die Architektur des eCard-API-Frameworks ist in vier Ebenen eingeteilt. Auf der oberste Ebene befinden sich die Applikationen, die auf die in der untersten Ebene verbundenen Smartcards zugreifen.

Der *Application-Layer* [37] beinhaltet die Applikationen, die auf Smartcards zugreifen möchten. Die Applikationen nutzen die Schnittstellen der eCard-API, um auf Anwendungen, Dienste oder Services der Smartcards zuzugreifen. Zusätzlich können auf dieser Ebene noch weitere Schnittstellen definiert werden, die gleiche und wiederkehrende Aufgaben zusammenfassen und vereinfachen.

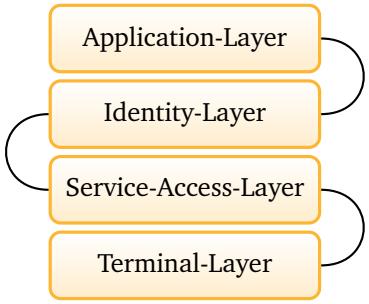


Abbildung 8.1.: Architektur des eCard-API-Frameworks

Der *Identity-Layer* [33, 36] besteht aus dem *Management-Interface* und dem *eCard-Interface*. Das Management-Interface bietet Methoden zum Steuern, Verwalten und Aktualisieren des Frameworks. Dabei stehen insbesondere Methoden zum Hinzufügen von neuen Konfigurationsdateien für Smartcards und Kartenterminals sowie vertrauenswürdige Instanzen bereit. Das eCard-Interface stellt Methoden für Verschlüsselung und Signaturerzeugung bereit. Dabei können sowohl Anfragen zur Signaturerstellung als auch zur Signaturverifikation verarbeitet werden. Als Format für die Daten wird beispielsweise *XML Signature* [18] unterstützt. Für das Ver- bzw. Entschlüsseln kann unter anderem *XML Encryption* [17] verwendet werden.

Der *Service-Access-Layer* [35, 39] beinhaltet das *Support-Interface* und das *ISO24727-3-Interface*. Das Support-Interface stellt Methoden bereit, die nicht von einer Karte ausgeführt werden können. So zum Beispiel das Kodieren und Komprimieren von Daten. Das ISO24727-3-Interface ist eine Web-basierte Implementierung der ISO/IEC 24727-3 [72]. Das Interface bietet Methoden für kryptographische Mechanismen sowie Funktionen zum Verwalten von Applikation und Schlüsselspeicher von Smartcards. Zusätzlich bietet das Interface Methoden zum Aufbau sicherer Verbindungen und Datenübertragung.

Der *Terminal-Layer* [34] stellt mit dem darin enthalten *IFD-Interface* die Kommunikation zu den angeschlossenen Hardware-Komponenten bereit. Dabei stehen Methoden zum Anfragen von angeschlossenen Kartenlesern und verbundenen Smartcards sowie deren Funktionalitäten und aktueller Status zur Verfügung. Die einzelne Terminals und Karten werden anhand einer eindeutigen Nummer identifiziert.

8.3 Web-Service Schnittstelle

Die Schnittstellen des eCard-API-Frameworks sind als Web-Service Schnittstellen definiert. Das Framework bietet damit eine offene und flexible Architektur. Mit offenen Standards wie WSDL für die Beschreibung der Schnittstellen und *SOAP* als Transportprotokoll ist eine hohe Interoperabilität gewährleistet. Die Protokolle, die insbesondere für die technische Umsetzung von MONA von Bedeutung waren, sind anschließend erläutert.

SOAP

SOAP ist ein Protokoll zum Austausch von Daten zwischen Computersystemen und ist auf der Anwendungsschicht des *OSI-Modells* (Open Systems Interconnection Reference Model) [67] einzuordnen. Die Repräsentation der Daten erfolgt mittels XML. Eine SOAP-Nachricht besteht aus einem äußeren *Envelope* genannten Element (siehe Auflistung 8.1). Darin befindet sich ein optionales Element *Header* und ein Element *Body*, das die Daten enthält.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>
      ...
    </S:Header>
    <S:Body>
      ...
    </S:Body>
</S:Envelope>
```

Auflistung 8.1: SOAP-Nachrichten

Für den Transport definiert die SOAP-Spezifikation sogenannte *Bingings*. Im Allgemeinen wird das *HTTP-Protokoll* (Hypertext Transfer Protocol) [51] als Transportprotokoll für die Übertragung von SOAP-Nachrichten genutzt. Neben diesem *SOAP HTTP Binding* [8, Kapitel 7] existiert noch das *PAOS-Binding*, was im Folgenden erläutert wird.

PAOS

Bei klassischen Client-Server-Architekturen fordert der Client Informationen von dem Server an. Dabei besteht eine klare Trennung zwischen dem passiven Web-Server, der auf Anfragen wartet und die Daten bereitstellt, und dem aktiven Client, der die Anfragen stellt und auf die Informationen zugreifen möchte. Im Internet wird zum laden von Websites das HTTP-Protokoll verwendet. Der Client sendet ein HTTP-GET Befehl (HTTP-Request) und der Server übermittelt die Daten innerhalb einer HTTP-Response. Das HTTP-GET Kommando enthält dabei die URL (Uniform Resource Locator) der Angeforderten Daten. Diese Trennung zwischen dem Bereitstellen und Anfordern von Informationen kann aber nicht in allen Szenarien aufrecht erhalten werden. Bei dem eID-Szenario des Personalausweises muss der eID-Server beispielsweise auch mit der eID-Applikation bzw. dem Ausweis kommunizieren. Hier kann eher von

einer gleichwertigen Rolle gesprochen werden. Das heißt, sowohl Server als auch Client stellen Informationen bereit und fordern Informationen an. Jedoch ist es oft technisch nicht möglich einen Web-Server auf dem Client zu betreiben und insbesondere aus Sicherheitsgründen auch nicht gewollt.

An dieser Stelle kommt das *Reverse HTTP Binding for SOAP* (PAOS) [1] zum Einsatz. Im Gegensatz um klassischen SOAP-HTTP-Binding wird bei PAOS ein SOAP-Request mittels HTTP-Response übertragen und ein SOAP-Response per HTTP-Request Befehl. Bei dem SOAP-HTTP-Binding kann der Client per HTTP-GET Informationen anfragen, die dann als SOAP-Nachricht vom Server übermittelt werden, oder per HTTP-POST SOAP-Nachrichten an den Server schicken. Durch das PAOS-Binding kann der Server nun auch SOAP-Requests an den Client schicken. Gemäß des *Request-Response Message Exchange Pattern* [1, Kapitel 4] sendet der Client zuerst eine Initialisierungs-Nachricht per HTTP-POST an den Server (siehe Abbildung 8.2). Damit signalisiert der Client welchen Service oder welche Informationen er nutzen möchte und dass PAOS für die Kommunikation verwendet werden soll. Beispielsweise könnte der Client Informationen zum Wetter einer Stadt anfordern. Der Server antwortet darauf, innerhalb einer HTTP-Response, mit einem SOAP-Request. In dieser Nachricht verlangt der Server dann beispielsweise den Ort oder die PLZ. Der Client wiederum antwortet mit einem SOAP-Response per HTTP-POST (HTTP-Request). Abschließend wird das Ergebnis vom Server an den Client geschickt. Damit ist die Anfrage des Clients (Wettervorhersage für den Ort) aus der ersten Nachricht abgeschlossen.

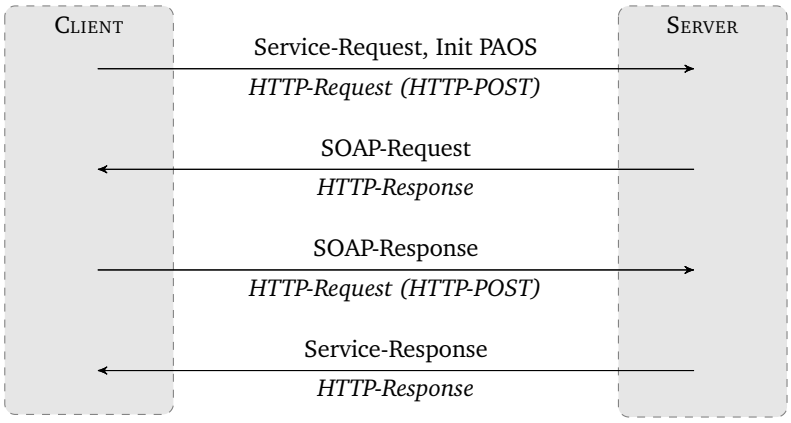


Abbildung 8.2.: Ablaufdiagramm des PAOS-Bindings

Werden jetzt die Nachrichten der eCard-API betrachtet (siehe Abbildung 8.3), dann entspricht die `StartPAOS` Nachricht der Initialisierungs-Nachricht und die `StartPAOSResponse` der letzten abschließenden Nachricht aus dem vorherigen Beispiel. Die Beschreibung der Nachrichten (z. B. `InitializeFramework` und `InitializeFrameworkResponse`) macht den auf von PAOS noch einmal deutlich: Die `InitializeFramework` Nachricht ist aus Sicht der HTTP-Ebene eine Response, aber auf der SOAP-Ebene ein SOAP-Request. Bei der Antwort des Client, in Form einer `InitializeFrameworkResponse` Nachricht wird dann ein SOAP-Response per HTTP-POST übertragen.

8.4 Anwendungsbeispiel für die eID-Applikation MONA

Im folgenden Kapitel wird ein Anwendungsbeispiel des eCard-API-Frameworks vorgestellt. Dabei wird das eID-Szenario erläutert und damit eine technische Präzisierung der Beschreibung aus Kapitel 2. Der Ablauf, die Protokolle und die Kommunikation entsprechen den Aufgaben, die die eID-Applikation MONA umsetzen muss. Die Beschreibungen beziehen sich auf die folgenden Richtlinien:

- **BSI TR-03110 – Advanced Security Mechanisms for Machine Readable Travel Documents**
  Die Richtlinie [41] beschreibt die Sicherheitsmechanismen für die Absicherung und Authentifizierung der Kommunikation zwischen (Remote) Terminal und dem Personalausweis. Die Richtlinie spezifiziert das PACE- und EAC-Protokoll sowie Zertifikate, Infrastruktur und Datenstrukturen .
- **BSI TR-03112 – eCard-API-Framework**
  Die Spezifikation [32] definiert das eCard-API-Framework als Kommunikationsschnittstelle zwischen eID-Anwendung und dem eID-Server.
- **BSI TR-03130 – eID-Server**
  Die Richtlinie [47] spezifiziert die Infrastruktur und Schnittstellen des eID-Servers als Web-Service zur Nutzung der elektronischen Ausweisfunktion in Web-Applikationen.

Die in diesem Abschnitt genannten Funktionen und Nachrichten der eCard-API beschränken sich auf Teil 4 und 5 der TR-03110. Die weiteren Bestandteile werden aus technischer Sicht nicht benötigt. Zu Beginn wird der Verbindungsaufbau erläutert. Die einzelnen Schritt sind in Abbildung 8.3 illustriert.

Verbindungsaufbau

1. Der Web-Browser stellt eine TLS-Verbindung zum Web-Server des Dienstanbieters her und sendet innerhalb dieses TLS-Tunnels ein Serviceanfrage in Form eines HTTP-Request.
2. Die Web-Applikation verwendet die `useIDRequest` Funktion der eID-Schnittstelle um eine Authentisierung für den Client bei dem zuständigen eID-Server vorzubereiten. Die `useIDRequest` Nachricht enthält eine Liste mit den angeforderten Datengruppen (z. B. Name und Anschrift). Der eID-Server generiert einen Pre-shared Key (`PSK`) und einen `SessionIdentifier`, die per `useIDResponse` Nachricht zurück an die Web-Applikation gegeben werden.
3. Die Web-Applikation beantwortet die Anfrage aus Schritt 1 mit einer HTTP-Response, die den `PSK`, `SessionIdentifier`, die `ServerAddress` des eID-Servers und eine `RefreshAddress` enthält. Die eID-Applikation wird gestartet und die Parameter verarbeitet.
4. Die eID-Applikation baut eine PSK-basierte TLS-Verbindung zum eID-Server auf und verwendet dabei den von der Web-Applikation übermittelten `PSK` und `SessionIdentifier`.
5. Die Anwendung initialisiert mit einer `StartPAOS` Nachricht die Verbindung zum eID-Server.

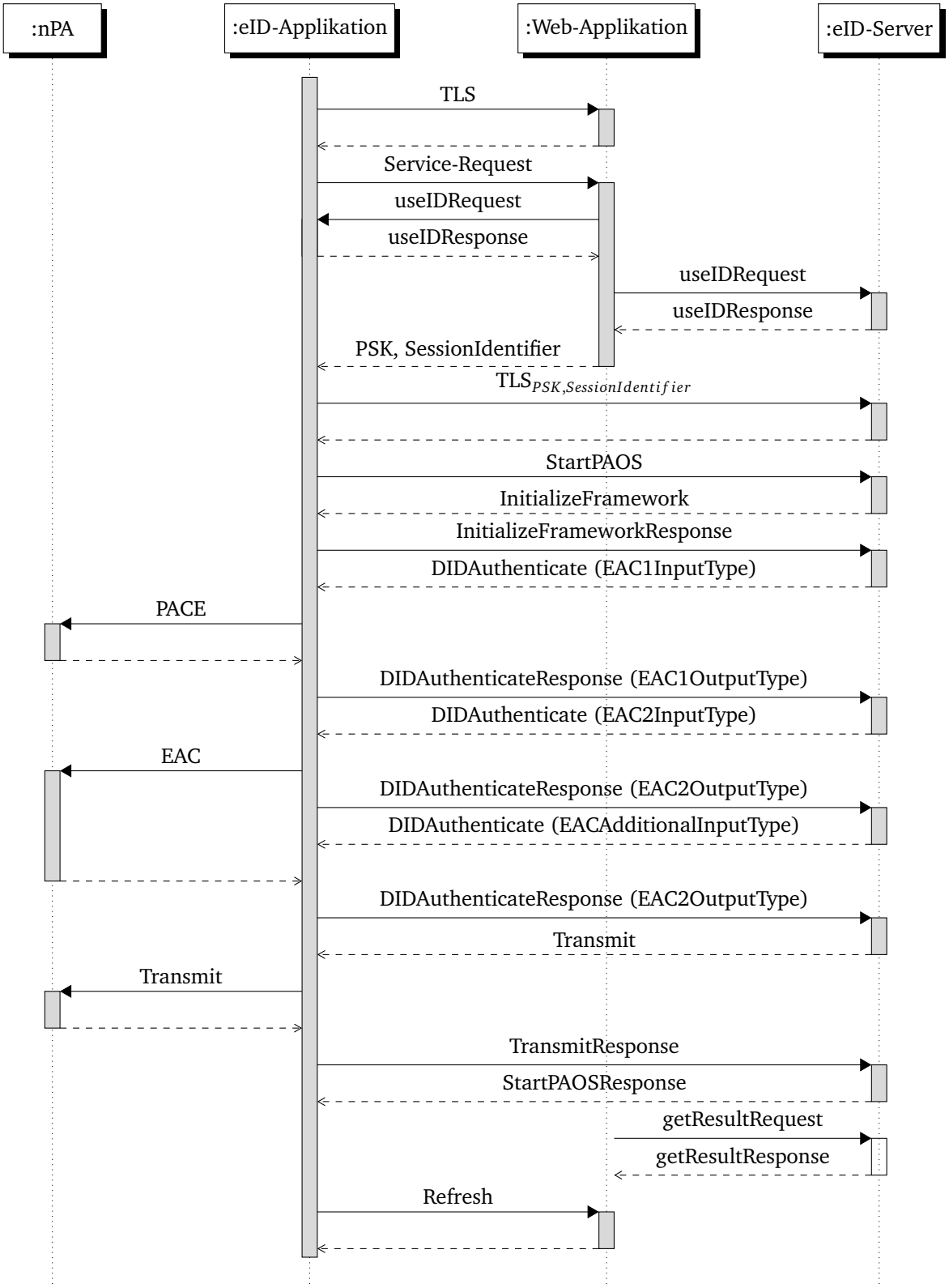


Abbildung 8.3.: Sequenzdiagramm des eID-Szenarios

Der Verbindungsaufbau zwischen der eID-Applikation und dem eID-Server ist damit abgeschlossen. Der weitere Kommunikationaustausch erfolgt jetzt über eCard-API-Nachrichten. Es erfolgt die Authentisierung der Teilnehmer und der Aufbau eines sicheren Kanals zwischen Ausweis und eID-Server:

6. Zu Beginn wird das eCard-API-Framework mit einer `InitializeFramework` Nachricht vom eID-Server initialisiert. Der Client sendet eine `InitializeFrameworkResponse` mit der Version des lokalen eCard-API-Frameworks. Die nachfolgende `DIDAuthenticate` Nachricht des Servers enthält das Berechtigungszertifikat des Dienstanbieters, das *Certificate Holder Authorization Template* (CHAT) und weitere Daten für das EAC-Protokoll.

7. Die eID-Applikation extrahiert die Informationen über den Dienstanbieter der Web-Applikation und zeigt dem Benutzer unter anderem den Namen und URL des Dienstanbieters sowie die ausstellende Zertifizierungsstelle an. Des Weiteren wird das CHAT mit den angeforderten Datengruppen angezeigt. Nach Bestätigung erfolgt die Aufforderung für die Eingabe der PIN. Das PACE-Protokoll wird durchgeführt und Informationen für die Durchführung des EAC-Protokolls, ein ggf. modifiziertes CHAT usw. werden an den Server mittels `DIDAuthenticateReponse` Nachricht übermittelt.

8. Im folgenden Schritt erfolgt die gegenseitige Authentisierung zwischen Server und Personalausweis mittels EAC-Protokoll. Die gesicherte Verbindung wird dabei direkt zwischen beiden Teilnehmern aufgebaut. Während des Protokolls ist die Verbindung zwischen eID-Applikation und Ausweis weiterhin per PACE und zwischen eID-Applikation und eID-Server per TLS abgesichert. Der erforderliche Datenaustausch während des Protokolls erfolgt in den zwei `DIDAuthenticate` bzw. `DIDAuthenticateReponse` Nachrichten.

Nach Schritt 8 haben sich alle Teilnehmer gegenseitig authentisiert. Der Ausweisinhaber hat den rechtmäßigen Besitz der Karte mit der PIN und dem Durchführen des PACE-Protokolls nachgewiesen. In Zusammenhang mit dem EAC-Protokoll hat der eID-Server seine Berechtigung und der Ausweis seine Echtheit bestätigt.

Im letzten Abschnitt folgt jetzt das Auslesen der Daten und das Ausführen weiterer Funktionen des Ausweises:

9. In diesem Schritt erfolgt der eigentliche Datentransfer der personenbezogenen Daten und das Ausführen von Funktionen wie der Sperrlisten-Abgleich oder der Restricted Identification. Der eID-Server sendet eine `Transmit` Nachricht mit einer Liste von APDUs, die an den Personalausweis weitergeleitet werden. Die Antworten werden in der `TransmitReponse` Nachricht zurück an den Server übergeben.

10. Die Authentisierung und Verbindung zu der eID-Applikation wird mit einer `StartPAOSResponse` Nachricht vom Server beendet.

11. Die Web-Applikation fragt mittels einer `getResultRequest` Nachricht nach dem Status bzw. Ergebnis der Authentisierung. Die ausgelesenen Daten vom Ausweis werden in einer `getResultResponse` Nachricht an die Web-Applikation übergeben und auf dem eID-Server gelöscht.

12. Die Website des Dienstanbieters wird aktualisiert und die eID-Applikation beendet. Der Benutzer ist jetzt erfolgreich authentifiziert.

Mit dem letzten Schritt ist die eID-Funktion abgeschlossen und die Aufgabe von MONA beendet. Der Benutzer kann jetzt ganz gewohnt das Online-Angebot nutzen.

9 Transport Layer Security

Das *Transport Layer Security* (TLS) Protokoll [15] bietet eine gesicherte Verbindung zwischen Anwendungen, die eine Ende-zu-Ende-Verbindung auf Basis des *Transmission Control Protocols* (TCP) [61] verwenden. TLS gewährleistet Vertraulichkeit durch symmetrische Verschlüsselung sowie Integrität und Authentizität der Daten durch den Einsatz von kryptografischen Prüfsummen. Zusätzlich findet eine Authentifizierung der Kommunikationsteilnehmer statt [119].

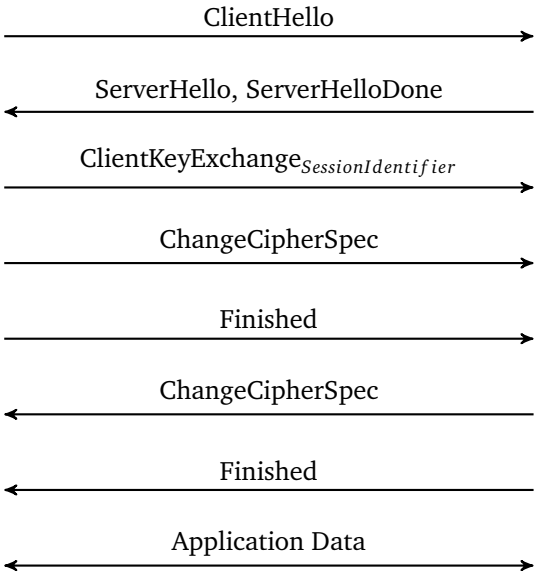


Abbildung 9.1.: Transport Layer Security mit Pre-Shared Keys

Die TR-03112 fordert für die Kommunikation zwischen eID-Server und eID-Anwendung eine *Pre-shared key* (PSK) Variante des TLS-Protokolls gemäß RFC 4279 [21]. Dabei erfolgt der Schlüsselaustausch nicht durch das TLS Protokoll, sondern vorher auf einem anderen Weg zwischen den Teilnehmern. Beim Starten der eID-Anwendung übergibt die Web-Applikation mehrere Parameter an die Anwendung. Darunter befindet sich ein `SessionIdentifer` und ein `PSK`. Diese zwei Parameter sind dann sowohl dem Client als auch dem eID-Server und der Web-Applikation bekannt (vgl. Kapitel 8.4). Beim Verbindungsaufbau der eID-Anwendung zum eID-Server kann dieser folglich die Authentisierungsanfrage zuordnen. Damit der eID-Server auch den passenden `PSK` für die Verbindung verwendet, sendet der Client in der `ClientKeyExchange` Nachricht den `SessionIdentifier` an den Server.

Anmerkung: Die beschriebene Variante entspricht den Anforderungen der Testumgebung. Gemäß der aktuellen Spezifikation und Infrastruktur erfolgt zusätzlich eine Verifikation des TLS-Serverzertifikats und eine auf RSA [111] basierenden Berechnung des TLS-Schlüssels (siehe Kapitel 20.2).

Teil IV.

Implementierung

10 Anforderungen

Zu Beginn werden die Anforderung betrachtet, die an eine eID-Applikation für den neuen Personalausweis gestellt werden. Für die Nutzung der eID-Funktion sieht das BSI ausschließlich zertifizierte Komponenten vor. Für Kartenleser gilt die Richtlinie TR-03119 und für eine eID-Applikation die TR-03112 sowie die Konformitätstests gemäß TR-03105 [45]. Die eID-Applikation muss zusätzlich nach Common Criteria [13] und dem Protection Profile BSI-PP-0066 [26] evaluiert werden (siehe TR-03127 Anhang A). In Anbetracht dessen, dass ein NFC-fähiges Smartphone mit MONA sowohl als Kartenleser als auch als eID-Applikation fungiert, sind dementsprechend die Anforderungen für beide Komponenten zu berücksichtigen.

Die in diesem Kapitel aufgeführten Anforderungen sind in *funktionale* und *nichtfunktionale* aufgeteilt. Die funktionalen Anforderungen gehen im Wesentlichen auf die Richtlinien des BSIs zurück, die für Kartenleser und eID-Applikation zur Verwendung mit dem neuen Personalausweis umzusetzen sind. Die funktionalen Anforderungen beschreiben demnach wie MONA zu realisieren ist bzw. was MONA implementieren muss. Die nichtfunktionalen Anforderungen definieren dazu weitere Eigenschaften.

Funktionale Anforderungen

Im Folgenden sind die Technischen Richtlinien des BSI aufgeführt, dessen Vorgaben maßgeblich für die Implementierung von MONA sind.

BSI TR-03110 – Advanced Security Mechanisms for Machine Readable Travel Documents

Die Technische Richtlinie [41] definiert die Protokolle PACE, EAC und RI als Sicherheitsmechanismen für den neuen Personalausweis. Zusätzlich konkretisiert die Richtlinie auf Basis der ISO/IEC 7816 die benötigten *Application Protocol Data Units* (APDU) und Strukturen der *Card Verifiable* Zertifikate (CV-Zertifikate) und definiert die *ASN.1* (Abstract Syntax Notation One) Datenstrukturen und Dateipfade. In Bezug auf die Zertifikate ist zusätzlich die *Certificate Policy* für die eID-Anwendung [50] hinzuzuziehen.

BSI TR-03112 – eCard-API-Framework

Die Richtlinie [32] definiert das eCard-API-Framework, das für die Kommunikation zwischen eID-Anwendung und eID-Server verwendet wird. Zusätzlich zu dem Framework verlangt die Richtlinie den Einsatz des SOAP-Protokolls mit PAOS-Binding sowie die Verwendung von TLS mit Pre-shared Keys als Sicherheits- bzw. Transportprotokoll.

BSI TR-03119 – Anforderungen an Chipkartenleser mit nPA Unterstützung

Die Spezifikation [48] definiert drei Klassen von Kartenlesern und Anforderungen, die die jeweilige Klasse von Lesern erfüllen muss. Die Richtlinie fordert beispielsweise für Kartenleser, auf denen das PACE-Protokoll ausgeführt wird, ein sicheres Erzeugen der Zufallszahlen. Der Pseudozufallszahlengenerator muss mindestens die Klasse K3 gemäß *AIS 20* [27] erfüllen und ein sicheres Seed von mindestens 100 Bit Entropie erzeugen. Zusätzliche Anforderungen sind unter anderem Fehlertoleranz und Eigenschaften nach ISO/IEC 7816, ISO/IEC 14443 und dem ICAO Standard.

BSI TR-03127 – Architektur elektronischer Personalausweis und elektronischer Aufenthaltstitel

Die Richtlinie [49] konkretisiert beim Personalausweis verwendete Verfahren und Eigenschaften. Dazu zählen beispielsweise die Ziffernanzahl der PIN und insbesondere die für MONA relevanten Interaktionen mit dem Ausweisinhaber bei der Online-Authentisierung. Gemäß Spezifikation sind dem Benutzer der Name, die Anschrift und die Email-Adresse des Dienstanbieters sowie die erwünschten Zugriffsrechte und der Gültigkeitszeitraum des Berechtigungszertifikates anzuzeigen. Zusätzlich muss dem Benutzer der Zweck der Datenübermittlung und die zuständige Datenschutzbehörde mitgeteilt werden. Bei der Entwicklung der graphischen Oberfläche von MONA gilt es, diese Vorgaben umzusetzen.

BSI TR-03130 – eID-Server

Die Technische Richtlinie [47] spezifiziert die eID-Schnittstelle des eID-Servers zur Anbindung von Dienstanbietern, die die eID-Funktion für ihren Dienst zur Authentisierung oder Registrierung verwenden möchten. Die Spezifikation betrifft daher die für MONA entwickelte Web-Applikation.

Nichtfunktionale Anforderungen

Im Folgenden werden nichtfunktionale Anforderungen bzw. Eigenschaften definiert, die die Implementierung von MONA aufweisen soll:

Bedienbarkeit

Die Bedienbarkeit bzw. Benutzerfreundlichkeit ist maßgeblich für die Akzeptanz der Software bei den Benutzern verantwortlich. Die für den Benutzer darzustellenden Informationen sind zwar durch die TR-03127 vorgegeben, jedoch sind beim mobilen Szenario weitere Aspekte zu berücksichtigten, weil auf den Displays der mobilen Geräte nur ein geringer Teil der Daten strukturiert und übersichtlich dargestellt werden kann.

Wartbarkeit

MONA ist auf Basis der Testausweise und -infrastruktur implementiert. Daher sind für den Wirkbetrieb Anpassungen an der Implementierung vorzunehmen. Diese Aspekte müssen bei der Entwicklung bereits berücksichtigt werden, um nachträgliche Anpassungen zu erleichtern und zukünftige Weiterentwicklungen zu vereinfachen.

Effizienz

Die Effizienz einer Anwendung hat neben der Bedienbarkeit für den Benutzer durch kurze Ausführungszeiten auch Auswirkungen auf den Ressourcenverbrauch bzw. die Ressourcenanforderungen. Eine effiziente Implementierung ist insbesondere für mobile Geräte ein wichtiger Aspekt, da sich jeglicher Ressourcenverbrauch unmittelbar in der Akkulaufzeit des Gerätes widerspiegelt. Des Weiteren trägt die Ausführungszeit der Anwendung einen erheblichen Beitrag zur Bedienbarkeit und Akzeptanz der Applikation bei. Für MONA ist eine kurze Ausführungszeit insbesondere wichtig, weil der Benutzer seinen Ausweis mit dem Gerät zusammenführen bzw. beide Komponenten in der Hand halten muss.

Portabilität

Die Portabilität bzw. Plattformunabhängigkeit ist für MONA insbesondere wünschenswert, damit die entwickelten Komponenten bzw. der Quellcode auch als Basis für eine stationäre eID-Applikation auf einem Rechner oder eine Portierung auf andere Plattform wie beispielsweise Android verwendet werden können.

Sicherheit

MONA erfasst das Passwort des Ausweisinhabers und verwaltet das Schlüsselmaterial für die Absicherung der kontaktlosen Schnittstelle. Diese Informationen gilt es zu schützen und beispielsweise nur solange zu speichern, wie sie für den Programmablauf benötigt werden.

Einschränkungen

Die Richtlinien des BSIs TR-03110, TR-03112 und TR-03127 legen die Anforderungen für eine eID-Applikation für den Ausweis fest. Für ein NFC-fähiges Smartphones, als Kartenleser für den neuen Personalausweis, gelten aber auch die Anforderungen der TR-03119. Das BSI wünscht beispielsweise für die Tastatur der Kartenleser *eine barrierefreie Ausführung, z. B. durch ein fühlbares Tastenfeld oder durch Brailleschrift* [48, Kapitel 4.3.3]. Des Weiteren werden die in der Richtlinie definierten Klassen von Lesern (Basis-, Standard- und Komfort-Chipkartenleser) einem mobilen Kartenleser nicht gerecht und die hardwarespezifischen Anforderungen sind aus Anwendungssicht nicht umzusetzen. Es bedarf daher einer Kooperation zwischen Entwicklern, BSI und den Geräteherstellern der NFC-fähiges Smartphones, um die Anforderungen einer mobiler Nutzung des Personalausweis umsetzen zu können.

11 Entwicklungsumgebung

Das folgende Kapitel beschreibt die bei der Implementierung von MONA eingesetzten Programme und die verwendete Entwicklungsumgebung.

Als Programmiersprache für die Anwendung MONA wurde Java gewählt. Diese Entscheidung resultiert zum einen aus dem Ziel einer möglichst plattformunabhängigen Implementierung und den Einschränkungen der Geräte. Auf dem verwendeten Nokia 6212 [87] ist die Entwicklung einer Anwendung nur auf Basis der Java Micro Edition bzw. als MIDlet realisierbar.

Als *integrierte Entwicklungsumgebung* (Integrated Development Environment, IDE) für die Implementierung von MONA wurde Netbeans verwendet. Des Weiteren wurde das *Software Development Kit* (SDK) für das Nokia 6212 verwendet, um Anwendungen mit NFC-Funktionalität zu entwickeln. Das SDK bietet einen Emulator und enthält die benötigten Bibliotheken wie die JSR 257 [102] für die NFC-Kommunikation. Die Web-Applikation wurde mit dem *Web Application Framework* Grails entwickelt und die *SpringSource Tool Suite* (STS) als IDE genutzt. Die Entwicklungswerkzeuge sind in Tabelle 11.1 zusammengefasst.

Software	Version	Website
Java Platform Micro Edition SDK	3.0	`http://www.oracle.com/technetwork/java/javame/`
Java SE Development Kit	6u25	`http://www.oracle.com/technetwork/java/javase/`
Netbeans IDE	7.0	`http://www.netbeans.org`
Series 40 Nokia 6212 NFC SDK	1.0	`http://www.forum.nokia.com/devices/6212_classic`
SpringSource Tool Suite	3.6.2	`http://www.springsource.com/developer/sts`
Grails	1.3.7	`http://www.grails.org`
Apache CXF	2.3.3	`http://cxf.apache.org`

Tabelle 11.1.: Entwicklungswerkzeuge

Die Entwicklung wurde auf einem per VirtualBox (4.0.8) [92] virtualisiertem Windows 7 (32-Bit) durchgeführt. Als Hostsystem (Intel Core 2 Quad Q8300, 4 GB RAM) wurde *Arch Linux* (64-Bit, 2.6.38 Kernel) [125] verwendet. Zusätzlich wurde ein Kartenleser *SDI 010* [116] für stationäre Tests eingesetzt.

12 Modul- und Paketübersicht

Zu Beginn wird eine Übersicht der Module und Pakete gegeben sowie deren Aufgaben und Funktionalitäten, die die darin enthaltenen Klassen implementieren. Die Implementierung wurde in drei Module aufgeteilt. Die Kernfunktionalität der Anwendung wurde in dem Modul `MONACore` als Java ME Klassenbibliothek zusammengefasst. In den Modulen `MONAClient` und `MONALocalClient` ist das Kernmodul um die handy- bzw. rechnerspezifischen Eigenschaften und Funktionen ergänzt (siehe Abbildung 12.1).

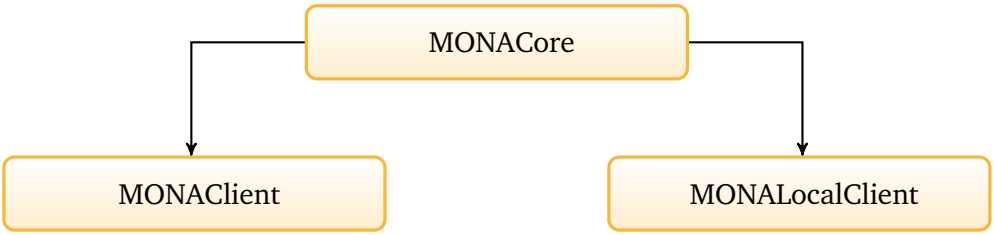


Abbildung 12.1.: Module von MONA

Die Einteilung in verschiedene Module verfolgt das Ziel eine möglichst plattformunabhängige Lösung zu entwickeln. Plattformunabhängig bedeutet in diesem Zusammenhang eine Java Bibliothek bereitzustellen, die den größtmöglichen Funktionsumfang beinhaltet und gleichzeitig als Basis für eine mobile und stationäre eID-Anwendung verwendet werden kann. Das heißt, die Bibliothek bzw. das Modul `MONACore` beinhaltet die Implementierung der Kernelemente wie die Protokolle, APDUs, eCard-API-Nachrichten usw. Jedoch verzichten bzw. abstrahieren die Klassen in dem Modul `MONACore` von der konkreten Verwendung einer Schnittstelle. Das bedeutet, dass die Klassen keine Abhängigkeiten an eine bestimmte Schnittstelle, wie die JSR 257 für die NFC-Verbindung, haben und somit ohne Anpassungen als Grundlage für eine Java ME und SE Applikation verwendet werden können. Die konkrete Anbindung an eine Schnittstelle erfolgt in den Modulen `MONAClient` und `MONALocalClient`, die dann beispielsweise auch eine plattformabhängige Implementierung einer graphischen Benutzeroberfläche beinhalten. Folglich bietet das Modul `MONACore` alle Funktionalitäten für eine eID-Anwendung, beinhaltet jedoch keine Anbindung an eine konkrete Schnittstelle oder graphische Benutzerinteraktion. Damit ist eine sehr hohe Wiederverwendbarkeit und Wartbarkeit des Quellcodes realisiert worden.

Im Folgenden sind die einzelnen Module und die darin enthaltenen Pakete und Klassen erläutert:

MONACore

Das Modul beinhaltet die Pakete und Klassen der Hauptkomponenten und Kernfunktionalitäten der eID-Applikation MONA.

- `de.tud.cdc.mona.apdu`
  Das Paket beinhaltet eine Implementierung der erforderlichen *Application Protocol Data Units* (APDU) nach ISO/IEC 7816-4 [69] und TR-03110 [41]. Dabei werden universelle Klassen mit Datenstrukturen für *Command* und *Response-APDUs* bereitgestellt sowie die benötigten APDUs für das PACE- und EAC-Protokoll und allgemeine Kommandos zur Interaktion mit einer Chipkarte.

- `de.tud.cdc.mona.asn1`
  In diesem Paket sind Klassen für die ASN.1-Funktionalität zusammengefasst. Dazu zählt das Analysieren, Verarbeiten und Abbilden der ASN.1-Datenstrukturen für die Domain-Parameter, die beim PACE- und EAC-Protokoll benötigt werden. Des Weiteren enthält das Paket Klassen mit Definitionen der *Object Identifier*, Domain-Parameter und sonstigen Datenstrukturen.

- `de.tud.cdc.mona.card`
  Das Paket beinhaltet Hilfsklassen für die Kommunikation mit einer Smartcard, die insbesondere das Selektieren und Auslesen von Dateien erleichtert. Zusätzlich beinhaltet das Paket eine Klasse mit den Definitionen der *File Identifier* des Personalausweises.

- `de.tud.cdc.mona.crypto`
  In diesem Paket sind Klassen zusammengefasst, die die benötigten kryptographischen Funktionen für das PACE- und EAC-Protokoll sowie Secure Messaging bereitstellen.

- `de.tud.cdc.mona.layer`
  Das Paket bündelt die Klassen für die Modellierung und Definition der Schnittstellen sowie die Interaktion der Layer. In mehreren Unterpaketen sind die Klassen der einzelnen Layer enthalten.

- `de.tud.cdc.mona.protocols`
  Das Paket enthält die Klassen der Implementierung der Sicherheitsmechanismen und -protokolle von PACE, EAC und Secure Messaging.

- `de.tud.cdc.mona.ui`
  Das Paket enthält ein Interface, das Schnittstellen für die User Interaktion definiert.

- `de.tud.cdc.mona.utils`
  Enthält Hilfsklassen für diverse Operationen auf Datentypen sowie allgemeine Funktionalitäten.

MONAClient

Das `MONAClient` Modul erweitert `MONACore` um eine graphische Benutzeroberfläche und die NFC-Anbindung sowie weitere handy-spezifischer Funktionalitäten zu einer vollständigen mobilen Anwendung.

- `de.tud.cdc.mona.conf`
  Enthält die Klasse `Configuration`, die Einstellungen für die graphische Benutzeroberfläche und weiterer Komponenten definiert.
- `de.tud.cdc.mona.layer`
  Beinhaltet die Implementierung der Interfaces aus dem Paket des `MONACore` Moduls, um die Kommunikation per NFC-Schnittstelle bereitzustellen.
- `de.tud.cdc.mona.gui`
  Beinhaltet eine Vielzahl von Klassen, die graphische Anzeigekomponenten und Benutzerinteraktionen bereitstellen sowie Klassen für eine Lokalisierung- und Theme-Funktionalität.
- `de.tud.cdc.mona.main`
  Enthält das MIDlet (Hauptprogramm).

MONALocalClient

Das `MONALocalClient` Modul ergänzt `MONACore`, um MONA als lokale Applikation zu verwenden und die Kommunikation per stationärem Kartenleser zu realisieren.

- `de.tud.cdc.mona.layer`
  Beinhaltet die Implementierungen der Interfaces aus dem Paket des `MONACore` Moduls, um die Kommunikation zum Ausweis per Kartenleser und *Java Smart Card I/O* [100] herzustellen.
- `de.tud.cdc.mona.main`
  Enthält die Java-Applikation (Hauptprogramm).

13 Kommunikation

Die wesentlichen Aufgaben einer eID-Anwendung sind die Interaktion mit dem Benutzer sowie die Absicherung, Vermittlung und Abwicklung der Kommunikation zwischen *Identity Provider* und *Credential*. Im Umfeld des Personalausweis und der Nutzung der eID-Funktion fungiert der Ausweis als Credential und ein eID-Server als Identity Provider. Die eID-Anwendung MONA ist daher im Wesentlichen ein Intermediär zwischen dem Ausweis und dem eID-Server und stellt die Benutzerinteraktion bereit.

Werden die Endpunkte der Kommunikation betrachtet, wie in Abbildung 13.1 verdeutlicht, dann müssen die Daten beim Ausweis in Form von *Application Protocol Data Units* (APDU) vorliegen, spezifiziert nach ISO/IEC 7816 und TR-03110, und zum eID-Server als SOAP-Nachrichten konform zum eCard-API-Framework gemäß TR-03112 übertragen werden. Zu den Aufgaben von MONA gehört folglich das Konvertieren der Daten in die jeweilige gewünschte Struktur. Neben den Datenstrukturen der APDUs und SOAP-Nachrichten müssen zusätzlich der Aufbau der eCard-API-Nachrichten und die Anpassungen durch das PAOS-Binding berücksichtigt werden. Die unterschiedlichen Datenstrukturen und Protokolle erfordern darüber hinaus weitere Informationen, die für den Transport bzw. für die Abwicklung der Verbindung zu den reinen Daten hinzukommen. Diese zusätzlichen Informationen für das *Message Handling* müssen bei der Konvertierung in andere Strukturen entfernt bzw. ergänzt werden.

Für die Implementierung der einzelnen Komponenten des Kommunikationsweges wurde für das Design und die Architektur ein Ebenenmodell gewählt. Wie in Abbildung 13.1 gezeigt, bestehen die beiden Endpunkte der Kommunikation aus dem Personalausweis und dem eID-Server. Zusammengefasst kann der Ablauf wie folgt erläutert werden: Ankommende Nachrichten des eID-Servers werden auf der TLS-Ebene entschlüsselt und an die PAOS-Ebene weitergeben. Dort erfolgt das Entfernen des PAOS- bzw. HTTP-Headers. Auf der SOAP-Ebene erfolgt das Extrahieren der eCard-API-Nachricht aus dem SOAP-Body. Die eCard-API-Ebene verarbeitet die Nachrichten und leitet diese ggf. an die EAC-Ebene zur Übertragung oder Verarbeitung durch die Sicherheitsprotokolle weiter. Die IFD-Ebene kapselt die Übertragungstechnologie zur Chipkarte und gestattet die Übertragung von APDUs. Der Versand von Nachrichten erfolgt analog in umgekehrter Reihenfolge.

Die einzelnen Schritte der Verarbeitung, Aufbereitung und Steuerung der Kommunikation und Daten werden durch die jeweiligen Ebenen repräsentiert. Die TLS-Ebene ist beispielsweise für den Aufbau der TLS-Verbindung zum eID-Server sowie deren Steuerung bei abgehenden bzw. ankommenden Nachrichten zuständig. Für die jeweilige Ebene ist dabei ausschließlich die syntaktische Korrektheit der Daten von Bedeutung, die Semantik der Daten jedoch nicht. Dazu zählen beispielsweise das Zusammenführen und die Überprüfung der Längenangaben bei Paketfragmenten. Die Inhalte der Daten werden jedoch ignoriert.

Die einzelnen Ebenen implementieren dabei die hinreichenden Funktionalitäten, um die gewünschten Operationen ausführen zu können. Die Einteilung in Ebenen bietet die Möglichkeit von der konkreten Implementierung einer Funktionalität zu abstrahieren und jede Ebene als *Black Box* zu betrachten. Das heißt, dass beispielsweise die eCard-API-Ebene ausschließlich dafür Sorge zu tragen hat, dass korrekte Nachrichten erstellt und versandt werden. Allerdings ist der Transport der Daten, d. h. der Einsatz des PAOS- oder HTTP-Bindings bei SOAP-Nachrichten, für diese Ebene nicht von Bedeutung. Die Architektur erlaubt es ebenfalls eine Ebene bzw. Funktionalität bequem zu aktivieren oder zu deaktivieren. Bei der eCard-API-Ebene ist beispielsweise festzustellen, dass die Absicherung des Transportmediums nicht zu den Aufgaben des eCard-API-Frameworks zählt. Dies gilt sowohl für die Kommunikation mit der Chipkarte per Secure Messaging, als auch mit dem eID-Server mittels TLS. Die Architektur erlaubt aus diesem Grund das einfache Austauschen der Sicherheitsprotokolle wie TLS und Secure Messaging. Das Beispiel Secure Messaging zeigt auch, dass Funktionen, wie die Verschlüsselung von Secure Messaging auf der IFD-Ebene, bequem aktiviert bzw. deaktiviert werden können, ohne Kenntnis der anderen Ebenen oder Änderungen an den Schnittstellen.

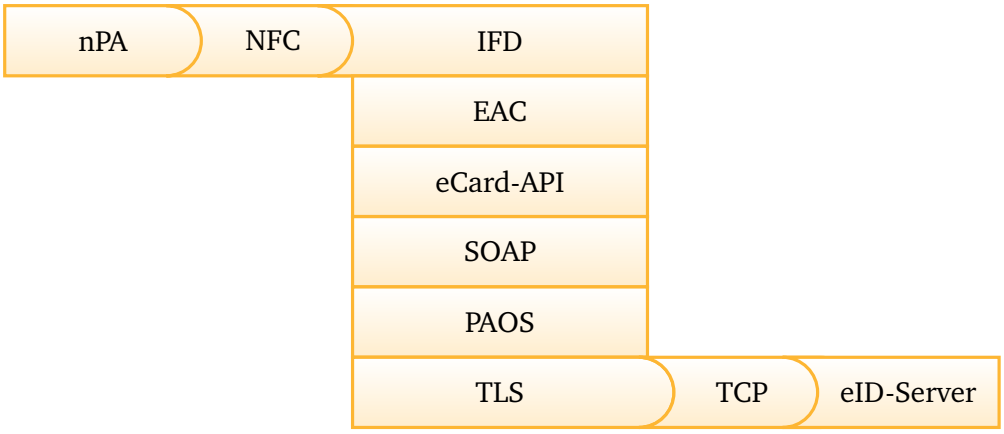


Abbildung 13.1.: Architektur der Kommunikationsebenen

Das gewählte Design orientiert sich an dem *Apache MINA* [4] Framework und zeichnet sich durch die hohe Flexibilität, Erweiterbarkeit und geringe Kopplung der Komponenten aus. Da die Schnittstellen zwischen den Ebenen klar, einheitlich und universell definiert sind, lassen sich bequem weitere Ebenen hinzufügen, austauschen oder entfernen. Damit konnte eine hohe Interoperabilität der Bestandteile des Kommunikationsfluss realisiert werden.

Im Folgenden sind die Interfaces und Klassen beschrieben, die die Struktur, Funktionalität und Basis der beschriebenen Architektur definieren und implementieren:

Paket: `de.tud.cdc.mona.layer`

- `de.tud.cdc.mona.layer.ILayer`
  Das Interface `ILayer` definiert Methoden, die von jeder Ebene implementiert werden müssen. Dazu zählen die Methoden `sessionOpened` und `sessionClosed` zum Öffnen bzw. Schließen einer Sitzung. Die Methoden erwarten jeweils ein `LayerChain` und `Session` Objekt. Des Weiteren definiert das Interface die Methoden `messageReceived` und `messageSent`, die jeweils eine `Session` und eine `Message` vom Typ `Object` erwarten und das Empfangen und Versenden einer Nachricht auf jeder Ebene repräsentieren.

- `de.tud.cdc.mona.layer.LayerAdapter`
  Die Klasse `LayerAdapter` implementiert das Interface `ILayer` und verbindet eine `LayerChain` mit den Methoden `messageReceived` und `messageSent`. Dadurch ist es möglich innerhalb einer Ebene durch `super.messageReceived` bzw. `super.messageSend` eine Nachricht an die darüberliegende bzw. darunterliegende Ebene weiterzugeben.

- `de.tud.cdc.mona.layer.LayerChain`
  Die Klasse `LayerChain` repräsentiert eine Kette von Ebenen, die jeweils eine Sitzung auf der darunterliegenden Ebene öffnen oder schließen sowie Nachrichten an die darüberliegende bzw. darunterliegende Ebene weitergeben können. Die Klasse verfügt über die Methoden `addLayer` zum Hinzufügen einer Ebene zu einer Kette sowie Methoden `getNextLayer` und `getPreviousLayer` zum Zugriff auf die darüberliegenden bzw. darunterliegenden Layer.

Die einzelnen Ebenen kommunizieren ausschließlich über die definierten Schnittstellen des Interfaces `ILayer` und erben von der Klasse `LayerAdapter`. Um die Ebenen zusammenzuführen, wird eine Instanz der Klasse `LayerChain` erzeugt und mittels der Methode `addLayer` die einzelnen Ebenen zu einem Verband verknüpft. Eine Sitzung wird dann mit dem Methodenaufruf `sessionOpened` auf einem Layer gestartet. Die Methode erwartet ein `LayerChain` und `Session` Objekt. Der Layer führt dann die eigene Implementierung der `sessionOpened` Methode aus und beendet die gewünschten Operationen mit dem Aufruf von `super.sessionOpened`. Damit wird durch die Vererbungshierarchie die Implementierung der Klasse `LayerAdapter` aufgerufen, die wiederum die Methode `sessionOpened` auf dem nächsten Layer der Kette bzw. dem nächsten Element der `LayerChain` ausführt. Die gleiche Funktionsweise findet bei den Methoden `sessionClosed`, `messageReceived` und `messageSent` statt. Dabei wird bei `super.messageReceived` immer die Methode der darüberliegenden Ebene und bei `super.messageSent` die der darunterliegenden Ebene aufgerufen.

Die Architektur gestattet es, auf jegliche Arten von Abhängigkeiten zwischen den Ebenen zu verzichten. Durch den Aufruf der super-Referenz beendet eine Ebene die eigenen Operationen und übergibt die Daten an die nächste Ebene. Dabei ist der Umfang der Operationen unerheblich. Eine Ebene kann beispielsweise Operationen auf ankommenden Nachrichten ausführen, abgehende Nachrichten bzw. Daten jedoch ohne Änderungen weiterreichen. Gleiches gilt für das Öffnen oder auch Schließen einer Sitzung.

Das Design und die Vererbungshierarchie gestatten mit Hilfe der super-Referenz ein sukzessives Verarbeiten der einzelnen Ebenen. Die Abfolge der Ebenen ist dabei durch die `LayerChain` definiert. Für eine Ebene ist dabei nicht ersichtlich, welche Ebenen ober- bzw. unterhalb liegen. Durch die einheitliche und universelle Schnittstellendefinition ist dies auch nicht erforderlich. Um neben den reinen Daten auch zusätzliche Informationen zwischen den Ebenen austauschen zu können sowie die Operationen der Ebenen an eine Sitzung bzw. bestimmte Kommunikationsverbindung zu koppeln, erwarten alle Methoden, definiert durch das `ILayer` Interface, ein `Session` Objekt. Die Klasse `Session` ist als Datencontainer zu betrachten, der neben diversen Sitzungsvariablen auch für Zustandsinformationen verwendet werden kann. Beispielsweise können Ebenen Informationen über ihren Status in dem `Session` Objekt ablegen, falls dies Operationen oder die Abfolge anderer Ebenen betrifft. In MONA wird die Klasse `Session` jedoch ausschließlich als Container für Sitzungsinformationen verwendet. Beispielsweise werden die Parameter wie die URL des eID-Servers, der PSK und der `SessionIdentifier` mittels `Session` Objekt beim Aufruf von `sessionOpened` an die TLS-Ebene übergeben, die dann mit diesen Parametern die gesicherte Verbindung zum Server aufbaut.

Paket: `de.tud.cdc.mona.layer.session`

- `de.tud.cdc.mona.layer.session.Session`
  Die Klasse repräsentiert eine Sitzung und verwaltet alle dazugehörigen Informationen. Dazu stehen die Methoden `setProperty`, `getProperty` und `removeProperty` für das Hinzufügen, Entfernen und den Zugriff auf eine Sitzungsvariable sowie die Methode `destroy` zum Beenden einer Sitzung zur Verfügung.

Insbesondere in Hinblick auf den Parameter `SessionIdentifier` werden die Vorteile dieser Architektur deutlich. Der `SessionIdentifier` wird sowohl auf der TLS-Ebene für den Verbindungsaufbau als auch auf der PAOS-Ebene für den HTTP-Header der Nachrichten benötigt. Würde auf die entwickelte Architektur verzichtet, hätte das beispielsweise zur Folge, dass die beiden Ebenen spezielle Methoden zum Austausch der Informationen verwenden müssten. Damit würde aber gleichzeitig eine Abhängigkeit entstehen, die Austausch, Veränderung oder Ersatz einer Ebene deutlich erschwert. Werden die Operationen der PAOS-Ebene betrachtet, dann hängen diese ausschließlich von den zu verarbeitenden Nachrichten und dem `SessionIdentifier` ab. Dies ermöglicht beispielsweise die PAOS-Ebene parallel zu nutzen. Das heißt, die gleiche Instanz der Ebene kann mehrere Verbindungen und Nachrichten gleichzeitig verarbeiten.

13.1 IFD-Layer

Der IFD (Interface Device) Layer ist einer der Endpunkte der Kommunikation und stellt die Verbindung zum Ausweis bereit. Dabei ist die Schnittstelle so konstruiert, dass von der korrekten Übertragungstechnologie abstrahiert wird. Das heißt, dass innerhalb von MONA keine Abhängigkeiten an eine bestimmte Übertragungstechnologie oder Schnittstelle existieren. Damit ist ein einfacher Austausch der korrekten Kommunikationsschnittstelle möglich, wie am Beispiel der JSR 257 [102] für das mobile Szenario und der Java Smart Card I/O [100] bei der Verwendung eines lokalen Kartenlesers deutlich wird.

Der IFD-Layer erweitert nicht die Klasse `LayerAdapter` und ist nicht in der `LayerChain` als Ebene eingetragen. Diese Abweichung in dem Ebenenmodel ist mit Optimierungen der Laufzeit zu begründen. Die Implementierung des IFD-Layers in der Klasse `IFDConnection` erlaubt das Senden von APDUs auf zwei verschiedene Arten: asynchroner und synchroner Versand. Bei der asynchronen Variante wird eine APDU an den IFD übergeben und mit dem weiteren Programmverlauf fortgefahren. Die APDU wird parallel in einem eigenen Prozess versendet und empfangen. Innerhalb des Programms kann die Antwort bzw. die Response-APDU dann zu einem späteren Zeitpunkt vom IFD abgefragt werden. Bei dem synchronen Versand wird die APDU an den IFD übergeben und auf die Antwort gewartet. Der sonstige Programmablauf blockiert während der Übertragung. Der asynchrone Versand wird nur bei der Implementierung des PACE-Protokolls verwendet und erlaubt Berechnungen parallel zum Versand und Empfang der APDUs sowie den Berechnungen des Ausweischips durchzuführen. Auf diese Weise lässt sich die Laufzeit optimieren. Die synchrone Variante wird insbesondere dann eingesetzt, wenn die Inhalte einer Response-APDU unmittelbar für den weiteren Programmverlauf benötigt werden. So ist beispielsweise der asynchrone Versand beim Auswählen oder Auslesen einer Datei nicht sinnvoll. Intern wird in der Klasse `IFDConnection` jedoch jeder Versand in einem eigenen Prozess bzw. Thread verarbeitet. Bei einem synchronen Versand werden die Prozesse jedoch direkt innerhalb der Klasse `IFDConnection` synchronisiert. Damit muss die Synchronisation nicht innerhalb der Komponenten erfolgen und die Anforderungen des *Open NFC Framework* [62] konnten erfüllt werden. Das Framework fordert, dass einen separater Thread für die I/O Operationen wie bei der NFC-Schnittstelle verwendet wird.

Für die EAC-Ebene ist ausschließlich die Funktionalität der `IFDConnection` sichtbar. Dabei wird nicht über die Schnittstellen des Interfaces `ILayer` kommuniziert, sondern über die Methoden des Interfaces `IIFDConnection`. Optional kann auf dem IFD das Secure Messaging aktiviert werden, welches die APDUs vor dem Versand verschlüsselt. In diesem Fall werden aber ausschließlich die zu verwendenden Schlüssel an den IFD übergeben. Die Implementierung und Operationen von Secure Messaging sind nach außen nicht sichtbar. Die Klasse `IFDConnectionAdapter` definiert abstrakte Methoden, die von einem Adapter für die konkrete Übertragungsschnittstelle implementiert werden müssen, wie beispielsweise die JSR 257 oder die Java Smart Card I/O. Damit lassen sich beliebige Schnittstellen bequem einbinden.

Paket: de.tud.cdc.mona.layer.ifd

- de.tud.cdc.mona.apdu.layer.ifd.IIFDConnection
  Definiert ein Interface für die Kommunikationsschnittstelle zwischen Anwendung und Übertragungsschnittstelle. Dazu gehören unter anderem die Methoden zum Senden einer APDU und dem Aktivieren bzw. Deaktivieren eines zusätzlichen Secure Messaging Kanals.

- de.tud.cdc.mona.apdu.layer.ifd.IFDConnection
  Die Klasse implementiert das Interface IIFDConnection und setzt die definierten Methoden um. Dabei werden APDUs per sendAPDU nebenläufig übertragen und auf die dazugehörige Response-APDU kann mittels receiveAPDU zugegriffen werden. APDUs per sendAPDUSynchronized werden synchron übertragen. Eine empfangene Response-APDU wird als Instanz der Klasse ResponseNormalOperation oder ResponseError zurückgegeben, abhängig vom Statuscode der APDU. Zusätzlich übernimmt die Klasse das Instanziieren und die Initialisieren eines neuen SecureMessaging Objekts.

- de.tud.cdc.mona.apdu.layer.ifd.IFDConnectionAdapter
  Die abstrakte Klasse definiert einen Adapter für die (kontaktlose) Kommunikationsschnittstelle zwischen Gerät und Personalausweis. Eine konkrete Implementierung der Klasse (siehe NFCAdapter) muss die Methoden transmitData und isCardConnected implementieren.

- de.tud.cdc.mona.apdu.layer.ifd.IFDConnectionThread
  Erweitert die Klasse java.lang.Thread und erlaubt so ein nebenläufiges Ausführen. Die Klasse besitzt eine Warteschlange (Queue), der beliebige APDUs übergeben werden können und die sequenziell übertragen werden. Bei einer Instanziierung der Klasse wird ein Objekt der Klasse IFDConnectionAdapter übergeben. Auf der Instanz des IFDConnectionAdapter wird die Methode transmitData aufgerufen und so die APDU über die jeweilige Übertragungsschnittstelle versendet.

- de.tud.cdc.mona.apdu.layer.ifd.NFCAdapter[1]
  Erweitert die Klasse IFDConnectionAdapter und implementiert die Methode transmitData zur Übertragung der Daten per NFC-Schnittstelle. Dabei wird die Methode exchangeData der in der JSR 257 enthaltenen Klasse javax.microedition.contactless.sc.ISO14443Connection verwendet.

- de.tud.cdc.mona.apdu.layer.ifd.SmartCardIOAdapter[2]
  Die Klasse erweitert den IFDConnectionAdapter und implementiert die abstrakten Methoden zur Anbindung von stationären Kartenlesern per Java Smart Card I/O [100].

[1] Die Klasse befindet sich im Modul MONAClient.
[2] Die Klasse befindet sich im Modul MONALocalClient.

Paket: `de.tud.cdc.mona.layer.ifd.common`

- `de.tud.cdc.mona.apdu.layer.ifd.common.IFDUtils`
  Die Klasse implementiert die zwei Methoden `createContextHandle` und `createSlotHandle` zum Erzeugen eines ContextHandle bzw. SlotHandle (siehe [38, Kapitel 3.1.3]). Dabei wird eine Zufallszahl von der Klasse `java.util.Random` generiert und als Initialwert für einen SHA-1 Hashalgorithmus verwendet. Der berechnete Hash wird dann für ein `ContextHandle` auf 20 Bytes und für ein `SlotHandle` auf 16 Bytes zugeschnitten.

Der IFD-Layer nimmt ausschließlich Daten in Form von APDUs entgegen. Die implementierten Datenstrukturen für APDUs sind im folgenden Abschnitt erläutert:

Application Protocol Data Unit

Die Kommunikation auf der Anwendungsebene zwischen Kartenleser und dem neuen Personalausweis erfolgt mittels Application Protocol Data Units (APDU), spezifiziert nach ISO/IEC 7816 und TR-03110. Dabei werden sowohl allgemeine APDUs wie beispielsweise zur Auswahl und zum Auslesen von Datenelementen, als auch protokollspezifische APDUs für die Sicherheitsmechanismen des Ausweises benötigt.

Auf der Java ME Plattform steht im Gegensatz zur Java SE Plattform aber beispielsweise keine Java Smart Card I/O zur Verfügung, die eine APDU-Funktionalität bzw. Datenstrukturen für APDUs bereitstellt. Die erforderliche Funktionalität könnte zwar auf die Java ME Plattform portiert werden, jedoch hätte dies einen Konflikt mit der Konformität der Lizenz zur Folge. MONA verwendet die in *MobilePACE* [58] entwickelten Datenstrukturen für die APDU-Funktionalität, setzt jedoch einige Optimierungen um. Bei MobilePACE wurde jede Command-APDU in einer separaten Klasse implementiert, jetzt sind alle Command-APDUs als statische Methoden in der Klasse `CardCommands` zusammengefasst. Dadurch lassen sich Ressourcen und Laufzeit einsparen, die durch die Klassen und deren Instanziierung benötigt würden. Da die APDUs von den Methoden der Klasse `CardCommands` bereits ISO/IEC 7816 konform erstellt werden, müssen diese nicht zusätzlich geparst werden und die Felder mit den Längenangaben berechnet werden. Das heißt, die APDUs können direkt an den IFD übergeben werden und Ressourcen für das Parsen der Daten entfallen. An der Architektur der Response-APDUs wurde jedoch festgehalten, da hier eine einfachere und bequemere Handhabung Vorrang hat. Insbesondere ist der Ressourcengewinn als deutlich geringer zu beziffern, weil der Aufwand und die Komplexität beim Parsen einer Response-APDU kleiner ist, als bei einer Command-APDU. Des Weiteren reduziert das Arbeiten mit den Objekten `ResponseError` und `ResponseNormalOpertation` die Komplexität außerhalb des IFD-Layers. Würde beispielsweise eine Response-APDU auf Basis eines Byte-Arrays im EAC-Layer verarbeitet, müsste an dieser Stelle zuerst Header und Trailer der APDU bestimmt und der Statuscode des Trailer überprüft werden. Die in MobilePACE entwickelten und in MONA verwendeten Datenstrukturen erlauben es hingegen im Programmablauf das referenziertes Objekt auf die Kompatibilität zu den Klassen `ResponseError`

und `ResponseNormalOpertation` zu prüfen. Das heißt, die Prüfung, ob eine Command-APDU erfolgreich ausgeführt wurde, kann durch eine Zeile Code realisiert werden. Die notwendigen Operationen auf einem Byte-Array würden jedoch mehrere Zeilen benötigen und sich insbesondere an vielen Stellen im Programmablauf wiederholen.

Im Folgenden sind die Pakete für die APDU-Funktionalität inklusive Datenstrukturen, Konstanten und Implementierung der erforderlichen APDUs erläutert:

Paket: `de.tud.cdc.mona.apdu`

Das Paket beinhaltet allgemeine Datenstrukturen, Hilfsklassen und Konstanten für APDUs.

- `de.tud.cdc.mona.apdu.IAPDU`
  Definiert ein Interface für ISO/IEC 7816-4 APDUs mit den Methoden `getData`, `toByteArray` und `toString`.
- `de.tud.cdc.mona.apdu.APDU`
  Repräsentiert eine ISO/IEC 7816 APDU und implementiert Methoden des `IAPDU` Interfaces zum Zugriff auf den Datenteil einer APDU.
- `de.tud.cdc.mona.apdu.APDUConstants`
  Definiert eine Vielzahl von Konstanten für Status- und Headerfelder einer APDU, die in ISO/IEC 7816-4 und TR-03110 spezifiziert sind. Dazu zählen beispielsweise das *Instruction*- (INS), *Class*- (CLA) und *Parameter*- (P1, P2) Feld einer `MSE:Set AT` APDU.
- `de.tud.cdc.mona.apdu.APDUUtils`
  Die Klasse beinhaltet Methoden zum Konkatenieren von Byte-Arrays und Bytes die auf das Zusammenfügen der einzelnen Felder einer APDU zugeschnitten sind.

Paket: `de.tud.cdc.mona.apdu.command`

Das Paket beinhaltet die Umsetzung von Datenstrukturen für Command-APDUs sowie die Implementierungen der erforderlichen Command-APDUs für die Kommunikation mit dem neuen Personalausweis.

- `de.tud.cdc.mona.apdu.command.ICommandAPDU`
  Definiert eine Schnittstelle für Command-APDUs und stellt mit der Methode `getHeader` den Zugriff auf den Header einer APDU zur Verfügung.
- `de.tud.cdc.mona.apdu.command.CommandAPDU`
  Die Klasse stellt eine Datenstruktur für Command-APDUs bereit, erweitert die Klasse `APDU` und implementiert das Interface `ICommandAPDU`. Ein mittels Konstruktor übergebenes Byte-Array wird geparst und in Header und Body unterteilt. Bei dem Body werden darüber hinaus die Felder *Length Command* (Lc), *Data* und *Length Expected* (Le) bestimmt.

- `de.tud.cdc.mona.apdu.command.CardCommands`
  Die Klasse implementiert eine Vielzahl von Methoden zum Erzeugen von APDUs. Dazu zählen ISO/IEC 7816 APDUs wie `SELECT`, `READ BINARY` und `GET CHALLENGE` sowie TR-03110 APDUs wie `MSE:Set AT`, `MSE:Set DST`, `General Authenticate` und `External Authenticate` usw. Die Methoden sind dabei als `static` definiert und liefern die erzeugten APDUs in Form eines Byte-Arrays zurück.

Paket: `de.tud.cdc.mona.apdu.response`

Das Paket beinhaltet die Implementierung von Response-APDUs.

- `de.tud.cdc.mona.apdu.response.IResponseAPDU`
  Definiert ein Interface für Response-APDUs. Dazu zählen die Methode wie `getTrailer` zum Zugriff auf den Trailer bzw. Datenteil sowie die Methoden `getStatusCode` und `getStatusBytes` zum Zugriff auf den Statuscode einer Response-APDU.

- `de.tud.cdc.mona.apdu.response.ResponseAPDU`
  Erweitert die Klasse `APDU` und implementiert das Interface `IResponseAPDU`. Die Klasse stellt damit eine allgemeine Datenstruktur für eine Response-APDU bereit. Ein im Konstruktor übergebener Byte-Array wird geparst und in Trailer und Header aufgeteilt.

- `de.tud.cdc.mona.apdu.response.ResponseError`
  Die Klasse implementiert eine Response-APDU, die einen Fehlercode im Statusfeld aufweist und damit ein *Warning Processing*, *Execution Error* oder *Checking Error* nach ISO/IEC 7816-4 repräsentiert.

- `de.tud.cdc.mona.apdu.response.ResponseNormalOperation`
  Die Klasse implementiert eine Response-APDU, die im Statusfeld eine 0x9000 aufweist und damit ein *Normal Processing* nach ISO/IEC 7816-4 darstellt. Das heißt, die vorherige Command-APDU wurde erfolgreich ausgeführt. Im Datenteil der APDU können sich ggf. Daten befinden.

Paket: `de.tud.cdc.mona.card`

- `de.tud.cdc.mona.card.CardConstants`
  Definiert Konstanten wie die *File Identifier* der einzelnen Dateien des neuen Personalausweises.

- `de.tud.cdc.mona.card.CardUtils`
  Stellt Methoden zum einfachen Ausführen von Dateioperationen, wie das Auswählen und Auslesen von Dateien, für Smartcards bereit.

13.2 EAC-Layer

Der EAC-Layer übernimmt im Kommunikationsablauf insbesondere die Steuerung der Protokolle PACE, Terminal und Chip Authentisierung sowie Secure Messaging. Das Design orientiert sich dabei vereinfacht an der *EAC-Box* [46]. Die EAC-Ebene erwartet eCard-API-Nachrichten vom Typ `DIDAuthenticate` und `Transmit`. Die `DIDAuthenticate` Nachrichten werden dabei zum Transport von Daten für die Sicherheitsprotokolle PACE und EAC verwendet. Bei einer `Transmit` Nachricht extrahiert die Ebene die einzelnen darin enthaltenen APDUs und leitet diese sequentiell an den Ausweis weiter. Ebenfalls wird auf dieser Ebene die `TransmitResponse` Nachricht erstellt.

Im Folgenden sind die Pakete und wichtigsten Klassen der EAC-Ebene erläutert:

Paket: `de.tud.cdc.mona.layer.eac`

- `de.tud.cdc.mona.layer.eac.EACEngine`
  Der EAC-Layer ist in der Klasse `EACEngine` umgesetzt und übernimmt insbesondere das Steuern der Protokolle PACE, Terminal und Chip Authentisierung und Secure Messaging sowie das Verarbeiten von eCard-API-Nachrichten, die Daten für den Protokollablauf bzw. für den Ausweis enthalten.

Paket: `de.tud.cdc.mona.layer.eac.common`

- `de.tud.cdc.mona.layer.eac.common.Certificate`
  Die Klasse repräsentiert selbst-beschreibende *Card Verifiable* Zertifikate (CV-Zertifikate) spezifiziert durch den ISO 7816 Standard (Teil 4, 6 und 8). Die Zertifikate enthalten eine *Certificate Holder Reference* (CHR), ein *Certificate Holder Authorization Template* (CHAT), *Certificate Effective Date*, *Certificate Expiration Date*, einen *Certificate Profile Identifier*, eine *Certification Authority Reference* (CAR) sowie optionale Zertifikats-Erweiterungen (vgl. TR-03110 Anhang C.1).

- `de.tud.cdc.mona.layer.eac.common.CertificateDescription`
  Beinhaltet zusätzliche Informationen für ein Zertifikat, die für die Benutzerinteraktion verwendet werden. Dazu zählen unter anderem der Name, die Anschrift und URL des Dienstanbieters und die zuständigen Zertifizierungsstelle, die das Zertifikat ausgegeben hat (siehe TR-03110 Anhang C.3.1).

- `de.tud.cdc.mona.layer.eac.common.CHAT`
  Die Klasse implementiert ein *Certificate Holder Authorization Template* (CHAT). Dazu stehen Methoden zum Parsen eines CHATs sowie zum Aufbereiten und Anzeigen der Daten für die Benutzerinteraktionen bereit.

13.3 eCard-API-Layer

Der eCard-API-Layer implementiert in der Klasse `eCardAPIEngine` die für das eID-Szenario benötigte Funktionalität des eCard-API-Frameworks. Der Layer übernimmt die wesentlichen Aufgaben zur Verwaltung und Steuerung des Kommunikationsablaufes. Das heißt, die ankommenden Nachrichten werden in diesem Layer analysiert, verarbeitet, beantwortet und/oder weitergeleitet. Der Klasse kommt daher eine große Bedeutung beim Steuern und Verwalten des Kommunikations- und Informationsflusses zu. Des Weiteren besitzt, im Vergleich zu den anderen Layern, die Klasse `eCardAPIEngine` einen separaten Konstruktor und erwartet eine Instanz des Interfaces `IUIController`. Das Interface definiert Methoden zur Benutzerinteraktion, wie das Anzeigen von Daten, eine Eingabeaufforderung usw. Die Schnittstelle ist bewusst universell definiert und besitzt darüber hinaus keine Abhängigkeiten an bestimmten Datenstrukturen oder graphischen Komponenten. Das heißt, wenn der eCard-API-Layer den *UIController* auffordert beispielsweise die PIN des Benutzers abzufragen, dann ist für den eCard-API-Layer nicht ersichtlich, wie diese Abfrage realisiert wird. Dementsprechend kann der UIController eine graphische Benutzeroberfläche auf Basis der für Java ME zur Verfügung stehenden Bibliotheken sein, aber auch auf Java SE Bibliotheken wie Swing oder AWT beruhen. Diese Abstraktion gestattet es aber auch, dass die PIN beispielsweise aus einer Konfigurationsdatei gelesen wird und vollständig auf Benutzerinteraktionen verzichtet wird, was insbesondere für die Entwicklung von Vorteil ist.

Paket: `de.tud.cdc.mona.layer.ecardapi`

- `de.tud.cdc.mona.layer.ecardapi.eCardAPIConstants`
  Die Klasse enthält statische Informationen wie die URIs der Status- und Fehlermeldungen, die aus Teil 1 des eCard-API-Frameworks hervorgehen. Des Weiteren beinhaltet die Klasse Definitionen der benötigten XML-Namensräume als Konstanten in Form von URIs und den dazugehörigen Präfixen.

- `de.tud.cdc.mona.layer.ecardapi.eCardAPIEngine`
  Die Klasse erweitert den `LayerAdapter` und implementiert die eCard-API-Ebene. Beim Öffnen einer Sitzung wird eine `StartPAOS` Nachricht erzeugt und versendet. Bei ankommenden Nachrichten erfolgt auf dieser Ebene ein Event-Handling. Nachrichten wie `InitializeFramework` und `StartPAOSResponse` werden auf dieser Ebene verarbeitet. `DIDAuthenticate` und `Transmit` Nachrichten werden verarbeitet und ggf. an die EAC-Ebene weitergeleitet. Beim Eintreffen der Nachrichten werden unter Umständen Events auf dem UIController ausgelöst.

Paket: `de.tud.cdc.mona.layer.ecardapi.wsdl`

Das Paket enthält die Java Klassen der durch die Web Services Description Language (WSDL) definierten Datenstrukturen und Nachrichten (siehe [37, Kapitel 2]).

Für die Realisierung des eID-Szenarios wird nur eine Teilmenge des eCard-API-Frameworks benötigt. Die Nachrichten und die zusätzlich erforderliche Datenstrukturen sind in separaten Klassen implementiert und in dem Paket `de.tud.cdc.mona.layer.ecardapi.wsdl` zusammengefasst. Zu beachten ist, dass die einzelnen Nachrichten zusätzliche Elemente enthalten können, die zwar im eCard-API-Framework definiert sind, jedoch in dem Szenario bzw. vom eID-Server und der eID-Applikation nicht verwendet und benötigt werden. Die Abfolge und die Verwendung im Kommunikationsverlauf ist in Abbildung 13.2 illustriert. Im Folgenden sind die benötigten Nachrichten des eCard-API-Frameworks aufgezählt und erläutert:

- **InitializeFramework**
  Die Nachricht initialisiert das eCard-API-Framework.
- **InitializeFrameworkResponse**
  Die Nachricht enthält Statusinformationen bzw. Fehlermeldungen, die beim Initialisieren des eCard-API-Frameworks aufgetreten sind. Zusätzlich beinhaltet die Nachricht Angaben zur Version des lokal installierten bzw. verwendeten Frameworks.
- **StartPAOS**
  Die `StartPAOS` Nachricht wird zum Aufbau einer PAOS-Verbindung zwischen eID-Server und eID-Applikation verwendet. Die Nachricht enthält einen `SessionIdentifier` und ein `ConnectionHandle`, der einen Kartenslot im Terminal und eine IFD-Layer Instanz adressiert.
- **StartPAOSResponse**
  Beendet die PAOS-Verbindung und beinhaltet Status- bzw. Fehlermeldungen vom eID-Server.
- **DIDAuthenticate**
  Eine `DIDAuthenticate` Nachricht setzt sich aus einem Element für Status- bzw. Fehlerinformationen und einem Element `AuthenticationProtocolData` zusammen. Dabei wird zwischen verschiedenen Typen des `AuthenticationProtocolData` Elements unterschieden, die im Folgenden aufgezählt werden:
  - **EAC1InputType**
    Besteht aus dem Certificate Holder Authorization Template (CHAT), dem Berechtigungszertifikat inklusive Beschreibung und der Authenticated Auxiliary Data A_{PCD}, die bei Initialisierung der Terminal Authentisierung (`MSE:Set AT`) übertragen wird.
  - **EAC2InputType**
    Beinhaltet die einzelnen Elemente der Zertifikatskette (*Country Verifying Certification Authority* (CVCA), *Document Verifier* (DV) und Terminal-Zertifikat) sowie den flüchtigen *Public Key* des Remote-Terminals. Dabei wird der vollständige Schlüssel übertragen, das Berechnen der komprimierten Form erfolgt beim Client.

- **EACAdditionalInputType**
  Eine Nachricht mit einem Element `AuthenticationProtocolData` enthält die bei der Terminal Authentisierung berechnete Signatur des Remote-Terminals.

- **DIDAuthenticateResponse**
  Eine `DIDAuthenticateResponse` Nachricht symbolisiert eine Antwort auf eine `DIDAuthenticate` Nachricht und unterscheidet zwischen verschiedenen Typen des `AuthenticationProtocolData` Elements, die im Folgenden erläutert werden:

  - **EAC1OutputType**
    Überträgt nach Durchführung des PACE-Protokolls den *Retry Counter*, das (eingeschränkte) Certificate Holder Authorization Template (CHAT), die Certification Authority Reference (CAR), den Inhalt der `EF.CardAccess` sowie den Card Identifier ID_{PICC}.

  - **EAC2OutputType**
    Das Element `AuthenticationProtocolData` vom Typ `EAC2OutputType` wird bei der Durchführung der Terminal und Chip Authentisierung verwendet. Im ersten Fall enthält das Element die Zufallszahl $r_{PICC,TA}$ und nach Abschluss der Chip Authentisierung den Inhalt der Datei `EF.CardSecurity`, den Authentisierungstoken und die Zufallszahl $r_{PICC,CA}$.

- **Transmit**
  Die Nachricht besteht aus einem Stapel von APDUs, die an die Chipkarte geschickt werden. Der Stapel wird dabei sukzessiv abgearbeitet und die Antworten zu jeder einzelnen APDU gespeichert. Die Zieladresse (Smartcard) ist diesbezüglich mit einem beigefügten `SlotHandle` adressiert. Da die APDUs durch den Secure Messaging Kanal zwischen Remote-Terminal und Ausweis abgesichert sind, müssen keine Operationen auf den APDUs ausgeführt werden. Die APDUs werden ohne Änderungen direkt an den Ausweis gesendet.

- **TransmitResponse**
  Die `TransmitResponse` Nachricht enthält Status- bzw. Fehlermeldungen sowie die Antworten auf die einzelnen APDUs einer `Transmit` Nachricht. Dabei umfasst der Stapel einer `TransmitResponse` Nachricht die gleiche Anzahl von Einträgen und zu jeder Command-APDU aus der `Transmit` Nachricht die dazugehörige bzw. resultierende Response-APDU.

Die durch das eCard-API-Framework in Form von WSDL definierten Datenstrukturen und Nachrichten sind von einer Vielzahl von Java Klassen im Paket `de.tud.cdc.mona.layer.ecardapi.wsdl` abgebildet. Dabei ist ausschließlich die hinreichende Funktionalität für das eID-Szenario des eCard-API-Frameworks implementiert. Die Abbildungen der WSDL-Definitionen wurden manuell erstellt, wodurch eine schlanke Implementierung möglich war. Für die Modellierung und Verarbeitung der XML-Strukturen wird die Bibliothek *kXML 2* [55] verwendet. Die JSR 172 kann für die Web-Service Funktionalität nicht verwendet werden.

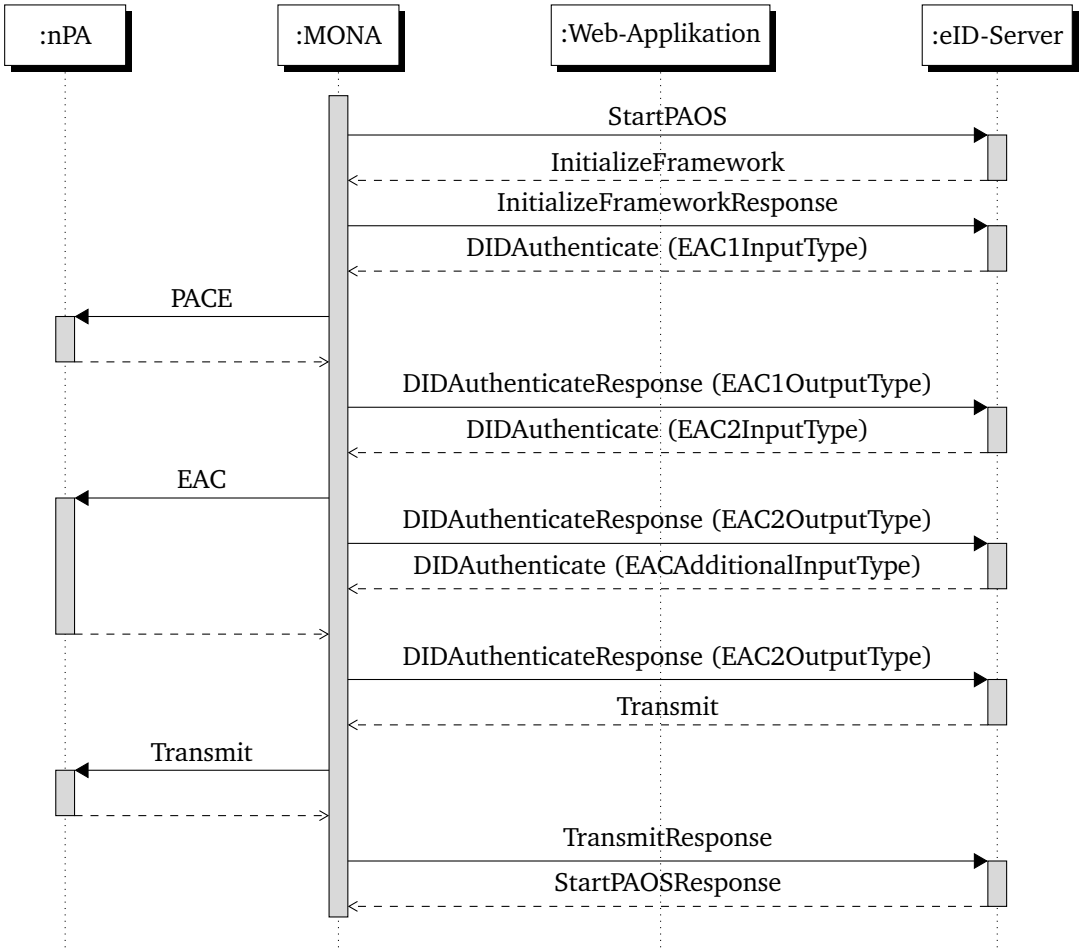


Abbildung 13.2.: Sequenzdiagramm des eID-Szenarios (Ausschnitt)

Die JSR 172 (J2ME Web Services Specification) [96] stellt eine Spezifikation für die Nutzung von Web-Services auf mobilen Geräten bzw. auf Basis der Java ME Plattform bereit. Mit einem *Stub Generator* lassen sich aus den WSDL-Dateien automatisch die notwendigen Java Klassen erzeugen. Die JSR 172 setzt eine Teilmenge der für die Java SE Plattform spezifizierten JSR 63 (Java API for XML Processing) [95] um. Für eine Umsetzung der eCard-API bietet die JSR 172 jedoch nicht die notwendige Funktionalität. So wird beispielsweise der XML-Typ `Integer` in Java intern auf den Datentyp `BigInteger` abgebildet. Dieser Datentyp steht bei der Java ME Plattform jedoch nicht zur Verfügung. Die Datenstrukturen können daher nicht mit Hilfe der JSR 172 abgebildet werden, ohne Änderungen an der eCard-API vorzunehmen. In MONA ist die eCard-API daher mit Hilfe der kXML2 Bibliothek manuell implementiert worden.

13.4 SOAP-Layer

Die SOAP-Funktionalität ist in der Klasse `SOAPEngine` implementiert. Beim Versand von Nachrichten nimmt die Ebene die XML-Nachrichten der eCard-API-Ebene entgegen und konstruiert den SOAP-Envelope mit SOAP-Header und der eCard-API-Nachricht als SOAP-Body. Der Header enthält neben den Standardelementen auch die zusätzlichen Bestandteile, die durch die Verwendung von PAOS erforderlich sind [1]. Beim Empfang einer Nachricht entfernt die Ebene jegliche SOAP-spezifischen Teile wie Envelope und Header und übergibt nur die Nachricht im SOAP-Body an die darüberliegende eCard-API-Ebene.

Zusätzlich zum Ein- und Auspacken von Nachrichten verwaltet die Ebene die *Message Identifier* der Nachrichten in Form von *Universally Unique Identifier* (UUID) [80]. Jede SOAP-Nachricht besitzt einen Message Identifier im Header der Nachricht, der in dem `<MessageID>` Element angegeben ist. Jede Antwort auf die Nachricht muss neben einem neuen Message Identifier auch ein `<RelatesTo>` Element enthalten, das den vorherige Message Identifier enthält und damit eine Verknüpfung zwischen Anfrage und Antwort herstellt.

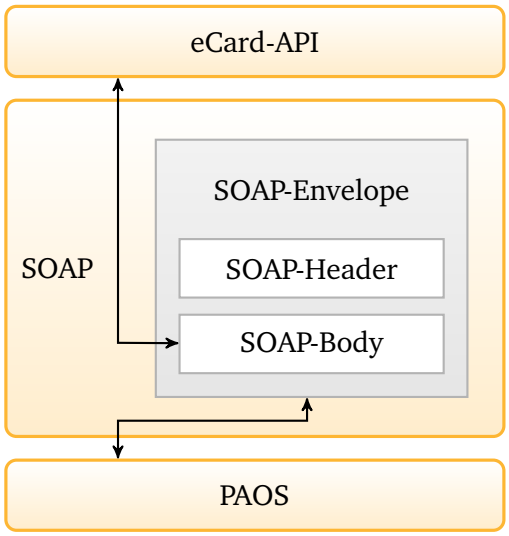


Abbildung 13.3.: SOAP-Layer

Abbildung 13.3 verdeutlicht die wesentliche Funktionalität der SOAP-Ebene. Nachrichten der eCard-API werden in einer SOAP-Nachricht als Body verpackt. Bei ankommenden Nachrichten wird den SOAP-Envelope und der dazugehörige Header entfernt und der Body an die eCard-API-Ebene weitergegeben.

Paket: `de.tud.cdc.mona.layer.soap`

- `de.tud.cdc.mona.layer.soap.SOAPConstants`
  Definiert Konstanten für die Implementierung des SOAP-Protokolls.

- `de.tud.cdc.mona.layer.soap.SOAPEngine`
  Die Klasse erweitert die Klasse `LayerAdapter` und implementiert die erforderliche SOAP-Funktionalität. Dazu zählt das Hinzufügen von SOAP-Envelope und SOAP-Header zu einer eCard-API-Nachricht bzw. das Extrahieren der Nachricht aus dem SOAP-Body.

- `de.tud.cdc.mona.layer.SOAP.SOAPHeader`
  Die Klasse implementiert den SOAP-Header und setzt Funktionalitäten wie das Zusammenführen des SOAP-Headers und *PAOS-SOAP-Headers* [1] sowie das Verwalten der <`MessageID`>- und <`RelatesTo`>-Elemente nach der *WS-Addressing* Spezifikation [128] um.

- `de.tud.cdc.mona.layer.SOAP.UUIDGenerator`
  Die Klasse stellt einen Generator für *Universally Unique Identifier* (UUID) nach RFC 4122 [80] zur Verfügung.

Zu beachten ist, dass mit dem Hinzufügen des PAOS-SOAP-Headers auf der SOAP-Ebene bereits „PAOS-Funktionalität“ integriert ist. Dadurch beschränken sich die Operationen der PAOS-Ebene ausschließlich auf den HTTP-Header und nicht auf die SOAP-Nachricht bzw. den SOAP-Header. Dies reduziert die Anzahl der Operationen der PAOS-Ebene und die Zugriffe auf die Objekte. Für Geräte mit hinreichenden Ressourcen ist diese Anpassung nicht notwendig.

13.5 PAOS-Layer

Für den Transport der SOAP-Nachrichten definiert das eCard-API-Framework für das eID-Szenario das PAOS-Binding [1]. PAOS verwendet im Gegensatz zum gängigen SOAP-HTTP-Binding ein HTTP-Response bzw. -Request zum Versenden von SOAP-Requests bzw. -Responses. Der PAOS-Layer nimmt diese Anpassungen bei Versand und Empfang der Nachrichten vor.

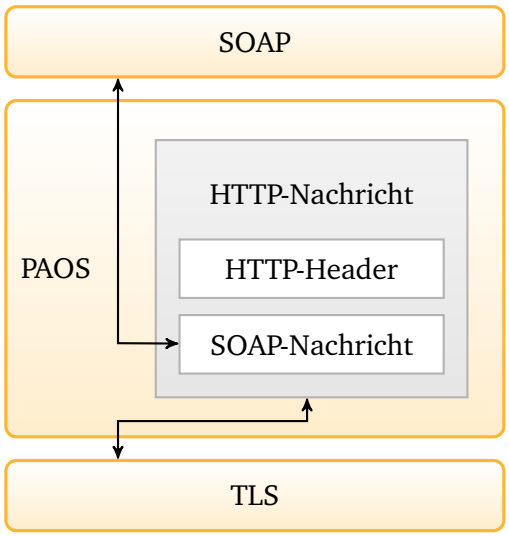


Abbildung 13.4.: PAOS-Layer

Die Klasse `PAOSEngine` implementiert das PAOS-Binding und fügt einen HTTP- bzw. PAOS-Header den SOAP-Nachrichten hinzu bzw. entfernt die Header bei ankommenden Nachrichten. Dabei wird gemäß des *Request-Response Message Exchange Pattern* [1, Kapitel 4] ein *HTTP-POST* [51] verwendet und als *Uniform Resource Identifier* (URI) [6] der `SessionIdentifier` übertragen.

Paket: `de.tud.cdc.mona.layer.paos`

- `de.tud.cdc.mona.layer.paos.PAOSConstants`
  Definiert Konstanten wie beispielsweise Präfix, Namesraum und *MINE Type* des PAOS-Bindings.
- `de.tud.cdc.mona.layer.paos.PAOSEngine`
  Die Klasse implementiert die PAOS-Funktionalität und fügt bzw. entfernt den PAOS-HTTP-Header von eingehenden bzw. abgehenden Nachrichten. Dabei werden Längenangaben, *Media Type*, *Binding Name*, *Version* usw. hinzufügt.
- `de.tud.cdc.mona.layer.paos.PAOSSOAPHeader`
  Die Klasse definiert den PAOS-SOAP-Header mit den Elementen `Version`, `EndpointReference` usw. (siehe [1, Kapitel 8]).

13.6 TLS-Layer

Das eCard-API-Framework spezifiziert für die Kommunikation zwischen eID-Applikation und eID-Server den Einsatz von Transport Layer Security (TLS) mit Pre-Shared Keys nach RFC 4346 [15] und RFC 4279 [21]. Die TLS-Implementierung der Java ME Plattform unterstützt jedoch keine Pre-Shared Keys. Anpassungen an dieser Implementierung wären schwer umzusetzen und hätten insbesondere Probleme mit der Lizenz zur Folge. Daher wurde auf die *MicroTLS* [123] Implementierung von E. Tews bzw. einer weiterentwickelten Variante von F. Kiefer zurückgegriffen und diese um die benötigte Pre-Shared Keys Funktionalität erweitert. Bei MicroTLS sind keine Einschränkungen durch die Lizenz gegeben. Die Bibliothek ist beispielsweise Bestandteil der mobilen Variante des Crypto-Providers *Bouncy Castle* [124].

MicroTLS und dessen Modifikationen sind als externe Bibliothek implementiert. Diese benötigt ausschließlich einen externen CSP für die benötigten kryptographischen Funktionen. Abhängigkeiten zu Funktionen oder Klassen von MONA bestehen nicht.

Innerhalb von der eID-Applikation MONA wird die TLS-Verbindung und -Kommunikation in der Klasse `TLSEngine` gesteuert. Durch das Aufrufen der Methode `sessionOpened wird` eine TLS-Verbindung zum Ziel aufgebaut. Dabei wird ausschließlich die Adresse des Ziels, der `Session Identifier` und der Pre-Shared Key (PSK) an die MicroTLS Bibliothek übergeben. Die einzelnen Protokollschritte und Berechnungen von TLS erfolgen durch die externe MicroTLS Bibliothek und sind nicht Bestandteil der TLS-Ebene in MONA.

Neben dem Weiterleiten der Nachrichten an MicroTLS muss die TLS-Ebene die eingehenden Nachrichten zusammenfügen. Da TCP-Pakete bei Ethernet Netzwerken eine Größe von 1500 Bytes [57, 110, 84] haben, müssen größere Nachrichten in mehreren Paketen übertragen werden. Eine ankommende Nachricht kann demnach nicht sofort weitergeleitet bzw. an die nächste Ebene übergeben werden, sondern muss ggf. erst zusammengefügt werden. Die TLS-Ebene extrahiert dabei den `Content-Length` Parameter aus dem HTTP-Header der ersten Nachricht und wartet bis die angegebene Länge bzw. Anzahl an Bytes übertragen wurde. Ist eine Nachricht vollständig zusammengesetzt wird diese an die darüberliegende Ebene weitergegeben.

Paket: `de.tud.cdc.mona.layer.tls`

- `de.tud.cdc.mona.layer.tls.TLSEngine`
  Die Klasse implementiert die TLS-Ebene. Dazu zählt das Einbinden der MicroTLS Bibliothek, der Verbindungsaufbau und das Versenden und Empfangen der Nachrichten.

14 Sicherheitsmechanismen

In diesem Kapitel wird die Implementierung der Sicherheitsmechanismen und -protokolle betrachten. Teile der Implementierung basieren auf Entwicklungen von *MobilePACE* [58] und *MobileEAC* [76].

14.1 Kryptographische Funktionen

Für die Berechnungen des PACE-Protokolls sind mehrere kryptographische Funktionen bzw. Operationen notwendig, die in separate Klassen ausgelagert sind. Damit konnte bei der Implementierung von PACE der Protokollablauf fokussiert und die Umsetzung übersichtlich und kompakt realisiert werden. Als *Cryptographic Service Provider* (CSP) wurde der *FlexiProvider* [122] verwendet.

Paket: `de.tud.cdc.mona.crypto`

- `de.tud.cdc.mona.crypto.CipherSuite`
  Die Klasse `CipherSuite` implementiert die benötigten kryptographischen Funktionen für das PACE-Protokoll. Dazu zählen Methoden zum Entschlüsseln der Zufallszahl sowie zum Generieren von Schlüsseln und das *Generic Mapping*.

- `de.tud.cdc.mona.crypto.KDF`
  Die Klasse `KDF` implementiert die *Key Derivation Function* [41, Kapitel A.2.3]. Dabei stehen die folgenden Methoden zur Verfügung: Methode `derivePI` zum Ableiten des Schlüssels K_π, `deriveMAC` für den Schlüssel K_{MAC} und `deriveENC` zum Ableiten des Schlüssels K_{ENC}.

14.2 Protokolle

Die Implementierung der Sicherheitsprotokolle für den Personalausweis sind im Folgenden erläutert:

Paket: `de.tud.cdc.mona.layer.eac.protocols`

Das Paket enthält eine abstrakte Klasse, die eine Basisklasse für die Implementierungen der Protokolle bildet und den Zugriff auf eine Instanz der Klasse `IIFDConnection` bereitstellt. Zusätzlich beinhaltet das Paket die Implementierung des PACE- und EAC-Protokolls sowie des Secure Messaging in den gleichnamigen Klassen.

Das PACE-Protokoll ist in der Klasse `PACE` implementiert und die einzelnen Protokollschritte sind als eigenständige und private Methode umgesetzt, die sukzessiv aufgerufen werden. Als Abbruchkriterium des Programmablaufs gelten eine fehlerhaft Authentisierung sowie Übertragungsfehler und Sicherheitsaspekte in Bezug auf die erstellen Schlüssel. Die Initialisierung des Protokolls per `MSE:Set AT` APDU im ersten Protokollschritt wird nur bei einer deaktivierten PIN abgebrochen. Da die Implementierung ebenfalls für das PIN-Management verwendet wird, gilt eine blockierte oder suspendierte PIN nicht als Abbruchkriterium. Die Klasse `PACE` erwarten im Konstruktor eine Instanz der Klasse `IIFDConnection` für die Kommunikation und ein Objekt der Klasse `PACESecurityInfos`, das die *SecurityInfos* für PACE repräsentiert und die Domain-Parameter enthält. Die Methode `start` führt zum Ausführen des Protokolls und erwartet das Passwort, den Passworttyp und das zu verwendende CHAT. Zusätzlich stellt die Klasse Methoden für die berechneten Sitzungsschlüssel K_{MAC} und K_{ENC} und weitere Parameter bereit.

Die Klassen `CA` und `TA` stellen die Implementierung der Terminal und Chip Authentisierung bereit. Anders als bei der Implementierung von PACE findet bei EAC die logische Kommunikation zwischen Ausweis und dem eID-Server statt. Daher sind die Protokollschritte in einzelnen Methoden implementiert, die nur den jeweiligen Schritt ausführen. Sobald eine Methode abgearbeitet ist, geht der Programmfluss zurück zur aufrufenden Instanz (EAC-Ebene).

Die Klasse `Secure Messaging` stellt das Secure Messaging gemäß ISO/IEC 7816 bereit. Bei der Instantiierung werden die Schlüssel K_{ENC} und K_{MAC} übergeben. Dann steht jeweils eine Methode zum Ent- und Verschlüsseln einer APDU zur Verfügung.

14.3 PIN-Management

MONA implementiert ebenfalls das PIN-Management gemäß TR-03127. Der Ausweis verfügt über einen Fehlbedienungszähler (FBZ), der das Erraten der PIN durch Ausprobieren verhindert. Nach zweimaliger Falscheingabe muss der Benutzer den dritten Versuch zusätzlich mit der CAN freischalten. Dies verhindert sogenannte *Denial of Service-Angriffe*, bei denen ohne Kenntnis des Ausweisinhabers versucht wird, über die kontaktlose Schnittstelle, die PIN zu erraten bzw. bewusst durch mehrmaliges Falscheingabe zu sperren. Sollte auch der dritte Versuch scheitern, muss der Benutzer den FBZ mit der PUK zurücksetzten.

Paket: `de.tud.cdc.mona.card.npa`

- `de.tud.cdc.mona.card.npa.PINManagement`
  Die Klasse implementiert die Methode `getStatus` zum Abfragen des FBZ bzw. Status der eID-Funktion. Das heißt, damit kann geprüft werden, ob die PIN blockiert ist oder wie viele Versuche noch zur Verfügung stehen. Die weiteren Interaktionen sind mit den Methoden `resumePIN` und `unblockPIN` implementiert. Zusätzlich stellt die Klasse mit der Methode `changePIN` die Möglichkeit zum Ändern der PIN zur Verfügung.

14.4 Abstract Syntax Notation One

Die Daten in der auf dem Ausweis gespeicherten Datei `EF.CardAccess` sind im ASN.1 (Abstract Syntax Notation One) Format kodiert. Die Datei enthält die Domain-Parameter für die Protokolle und Informationen über die verwendeten bzw. unterstützten kryptographischen Mechanismen. Die Implementierung basiert auf den in [58, Kapitel 5.3.3] entwickelten Datenstrukturen und verwendet die *CoDec* [121] ASN.1 Bibliothek . Die wesentlichen Elemente sind im Folgenden zusammengefasst:

Paket: `de.tud.cdc.mona.layer.asn1`

Das Paket enthält Klassen um die ASN.1-kodierten Daten zu verarbeiten. Dazu zählt unter anderem die Klasse `TLV`, die das Verarbeiten von *Type-Length-Value* kodierten Datenströmen ermöglicht.

Paket: `de.tud.cdc.mona.layer.asn1.eac`

In Paket sind Klassen für die Modellierung der *SecurityInfo*-Datenstruktur, bestehend aus dem *Object Identifier* des Protokolls und den dazugehörigen erforderlichen bzw. optionalen Daten gemäß TR-03110 [41, Kapitel A.1] implementiert. Das Paket enthält ebenfalls die jeweiligen Implementierung der *SecurityInfos* für PACE, Chip und Terminal Authentisierung.

Paket: `de.tud.cdc.mona.layer.asn1.eac.oid`

Das Paket enthält mehrere Klassen, die die Object Identifier für die Protokolle PACE, Terminal und Chip Authentisierung sowie Restricted Identification definieren.

15 Benutzerschnittstelle und -interaktion

15.1 Graphische Benutzeroberfläche

Die graphischen Komponenten der Benutzeroberfläche von MONA wurden nach dem *Model View Controller* (MVC) Architekturmuster umgesetzt. Das heißt, es folgt eine strikte Trennung zwischen den Daten, der Anzeige und der Steuerung. Die einzelnen Komponenten der graphischen Benutzeroberfläche (Graphical user interface, GUI), wie beispielsweise die Eingabemaske für die PIN und die Statusanzeige sind weitestgehend autonom implementiert, um keine Abhängigkeiten oder eine feste Abfolge zu erzwingen. Die Steuerung der Abläufe und die Verarbeitung der Events erfolgt über einen Controller (siehe Abbildung 15.1). In MONA ist ein solcher Controller in der Klasse `FormController` implementiert. Die Klasse implementiert das Interface `IUIController` und ist somit für die vollständige Anbindung, Verwaltung und Verarbeitung der Benutzerinteraktion zuständig.

Die einzelnen Ansichten der graphischen Benutzeroberfläche erweitern jeweils die Klasse `MyForm`. Diese Komponenten stellen daher nur die Anzeige bzw. Benutzerinteraktionen bereit. Die Steuerung der einzelnen graphischen Elemente erfolgt durch eine Klasse `FormController`, bei dem sich die Elemente registrieren und die über Änderungen, Eingaben und Aktionen informiert wird. Auf Basis dieser Informationen erfolgt dann die Steuerung der graphischen Elemente oder sonstiger Programmkomponenten.

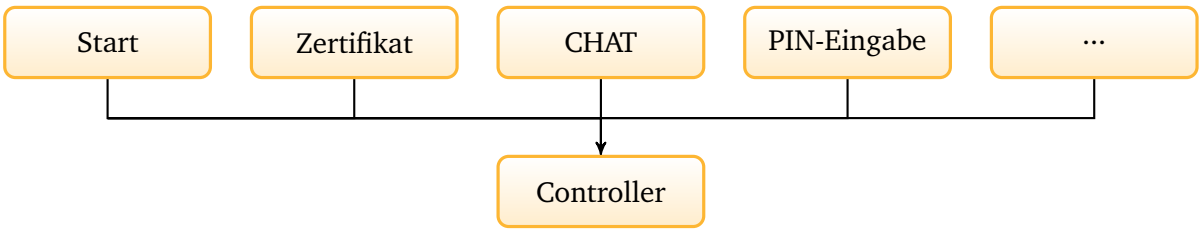
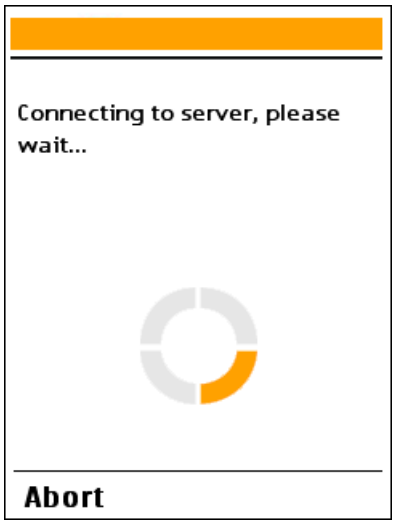


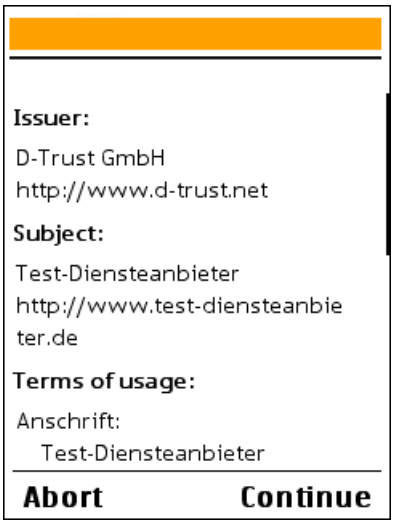
Abbildung 15.1.: Architektur der graphischen Benutzeroberfläche

Abbildung 15.1 illustriert die Architektur der Elemente der graphischen Oberfläche von MONA. Die Elemente werden von einem Controller gesteuert, bei dem sie sich registrieren müssen. Die einzelnen Elemente bzw. Ansichten senden Eingaben und Aktionen an den Controller. Der Controller erhält darüber hinaus weitere Informationen wie Fehlermeldungen oder Events von anderen Komponenten. Zusätzlich ist der Controller an ein `Session` Objekt gebunden, das als Speicherplatz für Datenobjekte, Benutzereingaben und Fehlermeldungen benutzt wird.

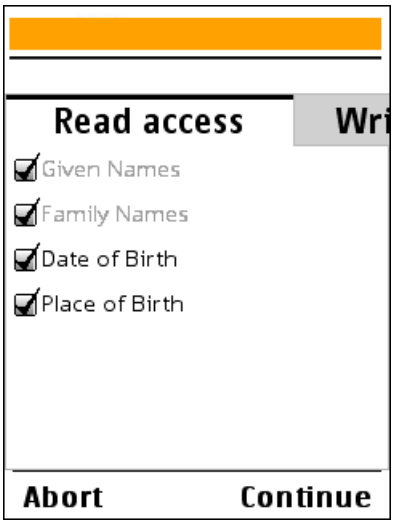
In Abbildung 15.2 sind die einzelnen Ansichten der GUI ersichtlich. Der erste Bildschirm wird während des Verbindungsaufbaus angezeigt und wechselt automatisch zur zweiten Anzeige sobald alle Informationen vom Server vorliegen. Es folgt die Anzeige des Berechtigungszertifikates, der Datengruppen und die PIN-Eingabe. In der fünften Anzeige wird der Benutzer aufgefordert den Personalausweis mit dem Gerät zusammen zu führen. In der letzten Anzeige signalisiert eine Statusleiste die Ausführung der Authentisierung. Nach erfolgreicher Ausführung wird MONA beendet und wieder die Website aufgerufen.



(a) Startbildschirm und Ladeanzeige während dem Verbindungsaufbau.

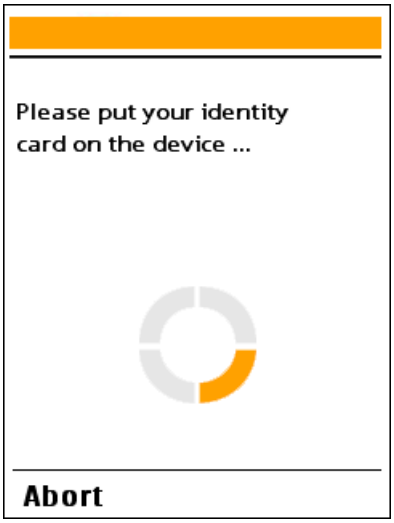


(b) Anzeige der Informationen des Berechtigungszertifikates.

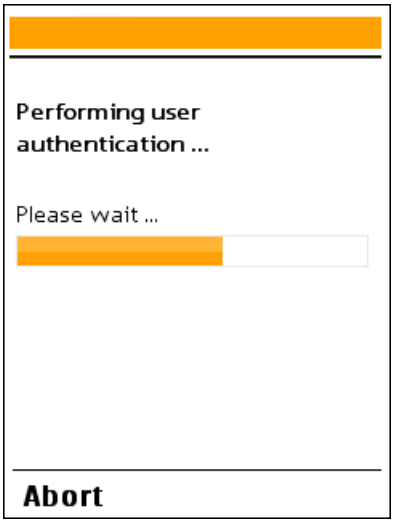


(c) Anzeige der angeforderten Datengruppen und An/Abwahl einzelner Datenfelder.

(d) Eingabefeld für die PIN.



(e) Warteanzeige bsi Ausweis erkannt wurde.



(f) Fortschrittsanzeige während dem Ausführen der Protokolle und Auslesen der Daten.

Abbildung 15.2.: Graphische Benutzeroberfläche von MONA (Screenshots)

Die graphische Benutzeroberfläche von MONA verwendet das *Light Weight User Interface Toolkit* (LWUIT) [104] Framework anstelle des standardmäßig bei Java ME verwendeten LCDUI-Framework der JSR 118 [98]. Das LWUIT-Framework ähnelt mehr dem von Java SE bekannten SWING-Framework für graphische Benutzeroberflächen und bietet deutlich mehr Funktionalität.

Das MIDlet inklusive der graphischen Oberfläche ist separat in dem Modul `MONAClient` implementiert. Die folgenden Angaben zu Paketen und Klassen beziehen sich daher auf dieses Modul und nicht auf das Modul `MONACore`.

Paket: `de.tud.cdc.mona.gui`

- `de.tud.cdc.mona.gui.FormController`
  Die Klasse implementiert das Interface `IUIController` und setzt die definierten Funktionen in graphische Interaktionen um. Die Klasse stellt damit das Bindeglied zwischen Programmablauf in der eCard-API-Ebene und den Benutzerinteraktionen dar. Die Klasse ist insbesondere für die Events der GUI zuständig und verarbeitet die einzelnen Kommandos und Aktionen der Buttons.

Paket: `de.tud.cdc.mona.gui.components`

- `de.tud.cdc.mona.gui.components.CommandFactory`
  Stellt eine *Factory* für Commands bereit.
- `de.tud.cdc.mona.gui.components.MyButton`
  Erweitert die Klasse `com.sun.lwuit.Button` um Konstruktoren für Objekte der Klasse `MyCommand`.
- `de.tud.cdc.mona.gui.components.MyCommand`
  Erweitert die Klasse `com.sun.lwuit.Command` um zusätzliche Funktionalitäten.
- `de.tud.cdc.mona.gui.components.MyForm`
  Die Klasse `MyForm` erweitert die Klasse `com.sun.lwuit.Form`. Beim Instanziieren der Klasse werden verschiedene Einstellungen für die graphische Oberfläche geladen. Dementsprechend gelten hier vorgenommene Anpassungen für alle erbenden Klassen bzw. Forms.

Paket: `de.tud.cdc.mona.gui.components.form`

Das Paket enthält eine Vielzahl von Klassen, die alle die Klasse `MyForm` erweitern und eine jeweilige Ansicht implementieren. Hierzu gehören allgemeine Ansichten, wie der Startbildschirm und die Abfrage beim Schließen der Anwendung sowie Ansichten die explizit auf das eID-Szenario zugeschnitten sind. Dazu gehört beispielsweise eine Ansicht für die Zertifikatinformationen, das CHAT oder eine Eingabemaske für die PIN.

Paket: `de.tud.cdc.mona.conf`

- `Configuration`
  Definiert eine Vielzahl von Einstellungen der graphischen Oberfläche, wie beispielsweise die Position der LWUIT-Ressource-Datei, der Name des Themes, der Speicherort der Logdatei.

Themes

Das LWUIT-Framework unterstützt *Themes* (engl. für Thema) zur Definition der graphischen Komponenten in Bezug auf Farben, Schriftarten, Bilder, Seitenränder und Rahmen (siehe [105, Kapitel 11]). Das Programm *LWUIT Theme Creator* stellt einen Editor für LWUIT-Themes bereit. Damit lässt sich das Design bzw. Erscheinungsbild einer Vielzahl der LWUIT-Komponenten wie Buttons, Textfelder, Listen usw. definieren. MONA verwendet ein LWUIT-Theme für die Einstellungen der Elemente der graphischen Oberfläche und verzichtet auf Definitionen des Erscheinungsbildes wie Farb- und Schrifteinstellungen innerhalb des Quellcodes.

Paket: `de.tud.cdc.mona.gui.theme`

- `de.tud.cdc.mona.gui.theme.ResourceSupport`
  Die Klasse stellt Methoden zum Laden von Ressourcen-Dateien (LWUIT-Konfigurationsdateien) bereit. Dabei kann ein Dateipfad angegeben werden, standardmäßig wird jedoch der Dateipfad aus der Klasse `Configuration` verwendet.

- `de.tud.cdc.mona.gui.theme.IconSupport`
  Die Klasse erwartet bei der Instanziierung eine LWUIT-Ressourcen-Datei und bietet ein einfaches Laden von Bildern aus der Ressourcen-Datei.

- `de.tud.cdc.mona.gui.theme.LocalizationSupport`
  Der Konstruktor der Klasse erwartet eine LWUIT-Ressourcen-Datei und bietet eine Methode zum Zugriff auf die einzelnen Attribute einer Lokalisierung.

- `de.tud.cdc.mona.gui.theme.ThemeSupport`
  Die Klasse erwartet eine LWUIT-Ressourcen-Datei und implementiert ein einfaches Laden und Initialisieren eines LWUIT-Themes. Standardeinstellungen können über die Klasse `Configuration` definiert werden.

Paket: `resources`
Das Paket enthält Konfigurationsdateien und Grafiken, die für die graphische Oberfläche benutzt werden.

- `theme.res`
  Ressourcen- bzw. Konfigurationsdatei für das in MONA verwendete LWUIT-Theme.

Die Datei beinhaltet Bilder und Spracheinstellungen (siehe nächstes Kapitel Lokalisierung) sowie alle Einstellungen des Erscheinungsbildes. Die Einstellungen bzw. das Design der Anwendung lässt sich damit bequem über diese Datei bzw. den LWUIT-Editor verändern oder austauschen.

Lokalisierung

MONA unterstützt eine *Lokalisierung* (Localization, L10N) der Elemente der graphischen Benutzeroberfläche. Entsprechend können jegliche Texte und Beschriftungen der Buttons für beliebige Sprachen definiert werden. Die Konfiguration der Lokalisierung erfolgt über den LWUIT-Editor (siehe [105, Kapitel 12.2.3]) und die Daten werden zusammen mit den Einstellungen zum Theme in einer Konfigurationsdatei gespeichert. Dabei wird einem bestimmten Element wie beispielsweise Quit eine deutsche Beschreibung Schließen und eine englische Beschreibung Exit zugeordnet. Innerhalb der Implementierung von MONA wird zum Beispiel dem Button zum Beenden der Anwendung jetzt als Label bzw. Beschriftung der Platzhalter Quit zugewiesen. In Abhängigkeit der geladenen Lokalisierung wird der Button dann mit Schließen oder Exit beschriftet. In MONA sind neben den Texten und Beschriftungen der Buttons auch beispielsweise die Beschriftungen der Elemente der Zertifikatsbeschreibung und der Datengruppen des CHATs über die Konfiguration definiert. Änderungen an der Beschriftung können daher bequem über den LWUIT-Editor realisiert werden und müssen nicht an der konkreten oder sogar an mehreren Stellen im Quellcode geändert werden.

Die Funktionalität einer Lokalisierung kann auch über die Codegenerierung der *Netbeans IDE* realisiert werden. Netbeans erstellt automatisch die benötigten Funktionen zum Laden einer Konfigurationsdatei (Java-Properties-Datei) in einer Klasse `LocalizationSupport`. In der Konfigurationsdatei können dann analog zum LWUIT-Editor Platzhalter und Beschriftungen definiert werden. Diese Variante verwendet jedoch für jede Sprache eine eigene Konfigurationsdatei wie beispielsweise `messages_de_DE.properties` und `messages_en_US.properties`. Der LWUIT-Editor erlaubt das Importieren und Exportieren solcher Konfigurationsdateien. Dementsprechend ist es möglich diese Sprachdateien bequem zwischen beiden Varianten auszutauschen bzw. zu integrieren. Änderungen am Format oder der Notation sind nicht erforderlich, lediglich auf eine *ACSII-Kodierung* der Datei ist zu achten.

15.2 Observer Pattern

Eine geringe Kopplung zwischen Komponenten bzw. Klassen ermöglicht ein flexibles sowie leicht erweiterbares Design und verhindert starke Abhängigkeiten zwischen den jeweiligen Komponenten. Im Gegenzug erschwert eine solche Architektur die Interaktion zwischen den Komponenten. Für die Layer der Kommunikation aus Kapitel 13 wurden daher einheitliche Schnittstellen definiert. Jedoch ist nicht in allen Fällen eine solche inhaltliche und aufgabenspezifische Bündelung wie bei den Layern vorhanden. Um eine Interaktion auch zwischen vollkommen verschiedenen Komponenten herzustellen, kann das *Observer Pattern* [52] verwendet werden. Die Architektur des Design-Patterns erlaubt, dass sich ein Objekt bei einem oder mehreren Objekten registriert und über Ereignisse informiert wird. Das Pattern definiert dabei Schnittstellen für *beobachtende* (Observer) und *beobachtete* Objekte (Observable).

Paket: `de.tud.cdc.mona.util.observer`

- `de.tud.cdc.mona.util.observer.IObservable`
  Das Interface definiert Methoden zum Hinzufügen, Entfernen und Benachrichtigen von Observern. Bei Java Klassen, die das Interface `IObservable` implementieren, können sich andere Objekte mit der Methode `addObserver` registrieren, um über Ereignisse benachrichtigt zu werden sowie Informationen oder Daten übermittelt zu bekommen. Analog ermöglicht der Aufruf der Methode `removeObserver` das Abmelden bei einer Klasse. Die Methode `notifyObserver` benachrichtigt registrierte Observer und erwartet als Parameter eine Nachricht bzw. Daten sowie einen Parameter `type`, der den Typ der Nachricht genauer spezifiziert.

- `de.tud.cdc.mona.util.observer.IObserver`
  Die Schnittstelle definiert die Methode update mit den Parametern `source`, `message` und `type`. Dabei sind die Parameter `message` und `type` vom Typ `Object` und `source` ist vom Typ `IObservable`. Durch die universellen Typen bei den Parametern `message` und `type` können Informationen von einem beliebigen Typ ausgetauscht werden. Dies erhöht zwar die Komplexität, macht die Schnittstelle jedoch sehr flexibel. Die Referenz auf das Objekt `source` gestattet es festzustellen, von welchem Objekt die Informationen stammen. Somit besteht die Möglichkeit, Statusmeldungen von verschiedenen Quellen differenziert verwenden bzw. auswerten zu können.

Das Observer Pattern wird beispielsweise bei dem `IUIController` bzw. `FormController` verwendet. Der Controller registriert sich bei dem eCard-API-Layer, um Informationen über den Protokollablauf zu erhalten. Zusätzlich wird diese Schnittstelle genutzt, um Events auf der graphischen Oberfläche auszulösen. Diese Events können beispielsweise das Umschalten der Anzeige oder das Aufrufen von Eingabemasken bedeuten. Dabei sorgt der eCard-API-Layer ausschließlich für das Erzeugen der Events. Ob ein Controller registriert ist oder die Events verarbeitet werden, ist nicht Aufgabe des Layers. Diese Abstraktion ermöglicht es, beliebige Controller für die Events zu registrieren oder diese auch bequem auszutauschen.

15.3 Logging

MONA verwendet *Microlog* [75] als Framework zum Loggen des Informationsaustauschs und Programmablaufs. Microlog basiert auf dem etablierten *log4j* [3] Framework und ermöglicht damit einen schnellen Einstieg sowie vertraute Einstellungen und Konfigurationen. Microlog stellt verschiedene *Logging-Levels* und *Appender* für die Ausgabe bereit. Die Ausgabe kann auf dem Display und in einer Log-Datei erfolgen, aber auch eine Weiterleitung per SMS, MMS und Bluetooth ist möglich. Das Format der Ausgabe kann des Weiteren mittels *Formatter* beliebig konfiguriert werden.

Paket: `resources`

- `microlog.properties`
  Die Konfigurationsdatei definiert die Einstellungen für Microlog. Unter anderem werden das Logging-Level, der Speicherort und das Format der Log-Datei definiert.

Paket: `de.tud.cdc.mona.util.logger`

- `LayerLogger`
  Die Klasse implementiert das Interface `de.tud.cdc.mona.layer.ILayer` und kann damit einer `LayerChain` hinzugefügt werden (vgl. Kapitel 13). Nachrichten die diese Ebene passieren, werden an Microlog unter dem Level *TRACE* weitergeleitet und in Abhängigkeit von den Einstellungen angezeigt oder abgespeichert.

Der `LayerLogger` entspricht einer weiteren Ebene im Sinne der in Kapitel 13 vorgestellten Ebenenarchitektur. Das Hinzufügen einer Instanz der Klasse `LayerLogger` in einer `LayerChain` gestattet das Aufzeichnen aller ausgetauschten Nachrichten, die diese Ebene passieren. In Abhängigkeit von der Position in der `LayerChain` kann die Granularität der Aufzeichnung gewählt werden. Wird der `LayerLogger` zwischen dem eCard-API und SOAP-Layer eingefügt, dann werden alle eCard-API-Nachrichten aufgezeichnet. Wird der Logger zwischen der PAOS- und SOAP-Ebene positioniert, so wird das vollständige SOAP-Envelope aufgezeichnet.

```
microlog.level=TRACE
microlog.appender=FileAppender
microlog.appender.FileAppender.filename=e:/MONALog.txt
microlog.formatter=PatternFormatter
microlog.formatter.PatternFormatter.pattern=%r [%P] %m %T
```

Auflistung 15.1: Microlog Konfigurationsdatei

Die in Auflistung 15.1 aufgeführte Konfigurationsdatei für Microlog speichert, durch das definierte Logging-Level *TRACE*, jegliche Kommunikation und Ereignisse in der Datei `MONALog.txt`. Die Pfadangabe `e:/` repräsentiert bei dem Nokia 6212 die Micro SD Karte. Der definierte `PatternFormatter` sorgt für eine Ausgabe in Form von Zeitangabe, Logging-Level, Nachricht und, falls vorhanden, dem Namen des übergebenen `java.lang.Throwable` Objekts beim Auftreten einer Java Exception.

16 Sicherheitskonzept

Die Sicherheitsprotokolle zur Authentisierung, Verschlüsselung und Zugriffskontrolle des neuen Personalausweises können mathematisch analysiert und als sicher verifiziert werden [14, 5]. Der Umsetzung bzw. die Nutzung der eID-Applikation, mit PIN-Eingabe, Berechtigungsanzeige und Kommunikation mit dem eID-Server und dem Ausweis, müssen die Ausweisinhaber jedoch im Allgemeinen vertrauen. Um die Integrität von MONA sicherzustellen, ist das MIDlet mit einem *VeriSign Class 3 Code Signing* Zertifikat signiert. Durch Verifikation der Signatur durch die *Java ME Virtual Machine* können nachträgliche Veränderungen registriert werden. Die Signatur stellt damit sicher, dass das MIDlet nach der Auslieferung und damit die einzelnen Komponenten von MONA wie Protokollimplementierungen, Benutzerinteraktion und Kommunikation nicht manipuliert oder verändert wurden. Die Signatur ist unter dem Parameter `MIDlet-Jar-RSA-SHA1` in der JAD-Datei gespeichert. Die notwendigen Zertifikate für eine Verifizierung sind unter den Parametern `MIDlet-Certificate` in der JAD-Datei abgelegt. Die Signatur garantiert jedoch nicht, dass keine Schadsoftware auf dem Gerät installiert ist oder keine Manipulationen am Betriebssystem bzw. der Virtual Machine vorgenommen wurden, wodurch die Sicherheit der Anwendung gefährdet sein kann. Informationen zu einem Integritätsschutz auf Basis eines von der *Open Mobile Alliance* (OMA) spezifizierten *Digital Rights Management* (DRM) [89] Systems für Nokia Geräte sind in [88] verfügbar. Das MIDlet ist darüber hinaus mit dem Obfuscator ProGuard [79] komprimiert und optimiert worden. Das Obfuskieren erschwert zusätzlich die Dekompilierung des Quellcodes und damit Einblicke in die Implementierung bzw. *Reverse Engineering*.

Die sicherheitskritischen Daten wie die PIN des Ausweisinhabers und die von PACE generierten Sitzungsschlüssel werden von MONA nur solange gespeichert, wie sie für den Programmablauf benötigt werden. Die PIN wird nach der Eingabe in einem Sitzungsobjekt gespeichert und die Instanz des Eingabefeldes der GUI wird überschrieben. Nachdem die PIN zum Entschlüsseln der Zufallszahl beim PACE-Protokoll verwendet wurde, wird diese auch aus dem Sitzungsobjekt entfernt und sonstige Speicherstellen in Form von Variablen überschrieben. Die PIN ist danach in keinem Objekt oder Variable mehr gespeichert. Das Gleiche erfolgt mit den temporären Schlüsseln von PACE. Nach dem Protokolldurchlauf werden nur die Sitzungsschlüssel für das Secure Messaging und die neuen Domain-Parameter behalten. Die temporären privaten und öffentlichen Schlüssel, die während des PACE-Protokolls gewählt werden, werden ebenfalls überschrieben. Die Sitzungsschlüssel werden nach dem Durchlauf der Terminal Authentisierung und Abschalten des *clientseitigen* Secure Messaging Kanals ebenfalls gelöscht. Zu beachten ist jedoch, dass diese Maßnahmen ausschließlich dazu dienen, die sicherheitskritischen Daten nur solange zu speichern, wie sie benötigt werden. Zugriffsbeschränkungen auf den Speicherbereich und Schutz des reservierten Speichers sind Aufgaben der Java ME Virtual Machine und nicht der Anwendung. Das heißt, die Sicherheit der Daten, die während des Programmablaufs verarbeitet werden, ist im Wesentlichen abhängig von der Sicherheit der Virtual Machine und des Betriebssystems.

17 Web-Applikation

Die elektronische Identitätsfunktion gestattet es, ein Registrierungs- bzw. Anmeldeprozess mit dem neuen Personalausweis durchzuführen. Damit ist das Ausfüllen von Formularen und das Anmelden bzw. Registrieren auf einer Website im Internet möglich. Die Kommunikation mit der Karte, das Durchführen der notwendigen Sicherheitsmechanismen und die Interaktion mit dem Benutzer werden durch eine lokale eID-Applikation realisiert. Zum Starten der eID-Anwendung und zur Weitergabe sitzungsabhängiger Parameter an die Anwendung wird in der TR-03112 ein HTML <object>-Tag [127, 129] vorgeschlagen (siehe [38, Kapitel 2]). Ein solcher <object>-Tag, eingebettet in die Website, wird vom Browser erfasst und die dazu registrierte eID-Anwendung gestartet. Von mobilen Browsern werden solche <object>-Tags jedoch nicht verarbeitet. Dementsprechend sind für den mobilen Einsatz bzw. für MONA andere Wege zum Starten der Anwendung und der Parameterübergabe zu wählen.

Das Starten von Java ME MIDlets von einer Website (per HTML-Link) wird jedoch nicht von allen Betriebssystemen für mobile Endgeräte unterstützt. Diese Möglichkeit findet sich beispielsweise bei der Symbian und Android-Plattform, jedoch nicht bei den herstellerspezifischen Betriebssystemen, wie eins auf dem Nokia 6212 installiert ist. Es besteht jedoch die Möglichkeit, ein MIDlet per Link zu installieren und danach zu starten. Dieser Installationsmechanismus für Java ME MIDlets bietet ein Pendant zum <object>-Tag bei lokalen eID-Applikationen und ermöglicht MONA zu starten. Bei einem MIDlet besteht die Möglichkeit, beliebige Parameter in der dazugehörigen JAD-Datei zu übergeben. Diese Datei kann von einer Web-Applikation bereitgestellt werden, d. h. sie wird auf der Website verlinkt. Die Parameter können dann beim Starten des MIDlets aus der JAD-Datei ausgelesen werden. Alternativ könnte auch die *Push Registry* der MIDP 2.0 [98] Spezifikation verwendet werden. Dies würde jedoch ein Verbindungsaufbau (Push) von der Web-Applikation zum mobilen Endgerät bedeuten. In dieser Push-Nachricht können bereits die Sitzungsparameter übergeben werden oder in einem zweiten Schritt vom MIDlet beispielsweise per *Web-Service Request* angefordert werden. Diese Konstellation funktioniert natürlich nur, wenn das MIDlet bereits auf dem Gerät installiert ist und erfordert weitere Anstrengung in Punkto Integrität, Authentizität und Vertraulichkeit der zu schützenden Sitzungsparameter.

Um das Starten von MONA per Website zu realisieren, wurde eine Web-Applikation entwickelt, die sowohl lokale eID-Applikationen per <object>-Tag unterstützt als auch mobile eID-Anwendungen wie MONA, die als MIDlet implementiert sind. Die Web-Applikation implementiert die Anbindung zum eID-Server gemäß TR-03130 [47] und gestattet das Starten der eID-Anwendungen. Nach Erhalt der bei der Authentisierung durch den eID-Server ausgelesenen Daten, erfolgt der Login und die Daten werden dem Benutzer zur Verifikation angezeigt. Da es sich um eine Demoanwendung handelt um die Umsetzbarkeit einer mobilen Verwendung der eID-Funktion zu zeigen, wurden keine weiteren Funktionalitäten in die Web-Applikation integriert. Die Web-Applikation wurde auf Basis des Web-Application Frameworks

Grails [117] und des Web-Service Frameworks *Apache CXF* [2] implementiert. Grails verwendet als Programmiersprache *Groovy* [12] und basiert auf den etablierten Frameworks *Hibernate*, *Spring*, *Quartz* und *SiteMesh*. Weitere Details zur Entwicklung mit dem Grails-Frameworks sind in [112] zu finden.

Im folgenden Abschnitt wird die Interaktion zwischen Client, Web-Applikation und eID-Server beschrieben. Der Ablauf ist inklusive der Komponenten der Web-Applikation in Abbildung 17.1 illustriert. Die Kommunikation zwischen Web-Applikation und eID-Server ist durch die TR-03130 definiert. Web-Anwendung und MONA bzw. der Client kommunizieren für das klassische HTTP-Protokoll. Ausführliche Erläuterungen zu den Komponenten sind im weiteren Verlauf dieses Kapitels zu finden.

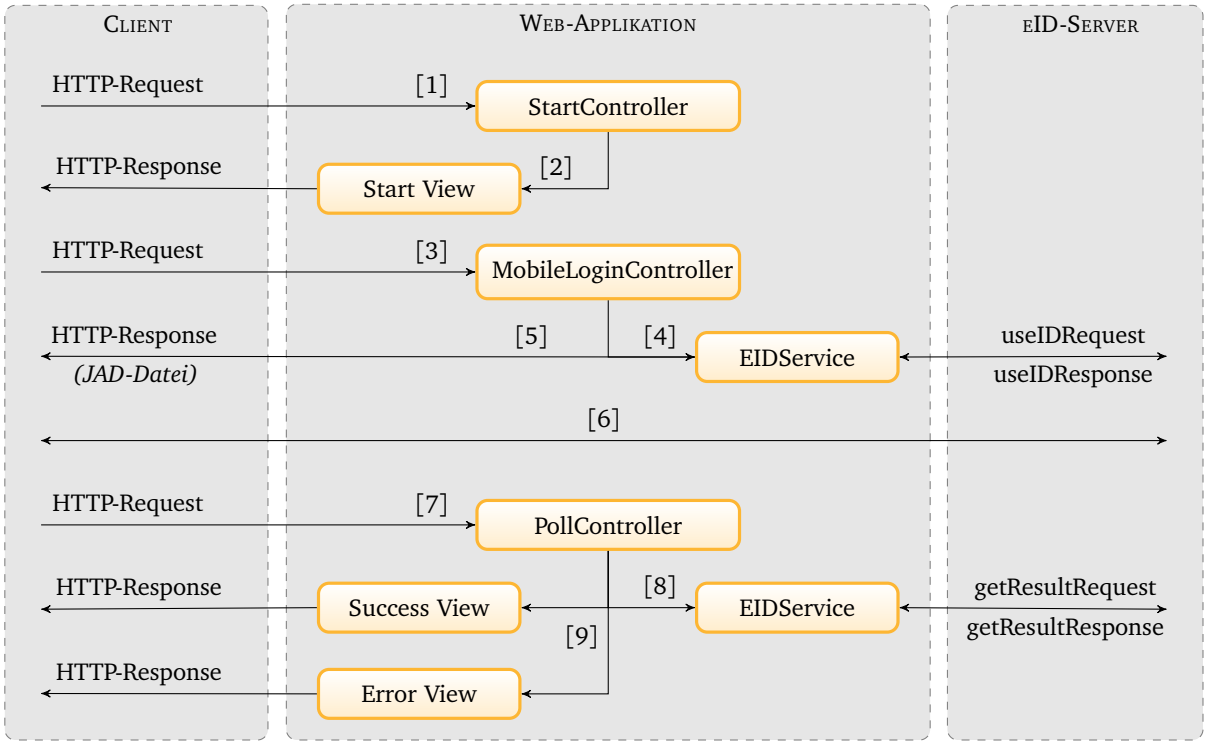


Abbildung 17.1.: Ablaufdiagramm der Web-Applikation

Im Folgenden ist der Ablauf der Interaktion zwischen MONA, Web-Applikation und eID-Server aus Abbildung 17.1 erläutert:

1. Der Benutzer ruft die Website auf.
2. Der `StartController` nimmt die Anfrage entgegen und zeigt die `Start` View (Startseite) an.
3. Der Benutzer klickt auf den Link Mobile Login. Die Anfrage wird vom `MobileLoginController` entgegen genommen.
4. Der `MobileLoginController` ruft den `EIDService` auf, um eine Authentisierung einzuleiten. Der `EIDService` kontaktiert den eID-Server und erhält die notwendigen Parameter für die Authentisierung. Diese werden vom eID-Server über den `EIDService` an den Controller übermittelt.
5. Die JAD-Datei wird vom `MobileLoginController` mit den Parametern für die eID-Funktion vervollständigt und an den Client übergeben.
6. Die eID-Anwendung MONA wird auf dem Client gestartet und die Authentisierung mit dem eID-Server gemäß TR-03112 durchgeführt.
7. Nach Abschluss der Interaktion mit dem eID-Server wird die `RefreshAddress` auf dem Client geöffnet. Die Anfrage der Adresse wird vom `PollController` bearbeitet.
8. Der `PollController` informiert sich beim `EIDService` über den Status der Authentisierung.
9. Sollte zum Zeitpunkt der Anfrage kein Ergebnis der Authentisierung vorliegen, wird die Antwort zurückgehalten, andernfalls, je nach Ergebnis, eine Fehlerseite oder Benutzerseite angezeigt.

Die Wiedererkennung des Clients in Schritt 7 erfolgt durch Übergabe des `SessionIdentifiers` als `<searchpart>` der URL gemäß [7]. Alternativ kann dies auch über ein Cookie erfolgen. Die Authentisierung bzw. Nutzung der Web-Applikation mit einer lokalen eID-Applikation unterscheidet sich nur in Schritt 3 und 5. Anstatt des `MobileLoginController` wird in Schritt 3 der `LoginController` aufgerufen und in Schritt 5 keine JAD-Datei an den Client zurückgegeben, sondern die `Login` View aufgerufen. Diese enthält dann den `<object>`-Tag zum Ausführen der lokalen eID-Applikation.

Im nächsten Abschnitt sind die einzelnen *Services*, *Controller* und *Views* der Web-Applikation beschrieben. Zu beachten ist, dass das Grails-Framework durch den strikten Einsatz des *Model View Controller* Designmodells für jeden Controller ein View bzw. für jedes View einen Controller vorsieht, auch wenn diese keine Funktionalität, Anzeige oder Inhalt implementieren.

Services

Der *Service Layer* von Grails verfolgt das Ziel, den Controllern und Views auf dem *Web Layer* gleiche und wiederkehrende Funktionalitäten als API bereitzustellen. Die Business Logik einer Web-Applikation wird daher in Services zentralisiert, gekapselt und implementiert. Controller und Views können die Schnittstellen der Services daher bequem nutzen, ohne Details der Implementierung zu kennen (z. B. Struktur, Aufbau der Web Service Nachrichten). Für die für MONA entwickelte Web-Applikation ist die eID-Schnittstelle entsprechend der TR-03130 [47] als Grails-Service implementiert. Der Ablauf der Kommunikation ist in Kapitel 4 der TR-03130 beschrieben.

- **EIDService**
  Die Anbindung der eID-Schnittstelle ist als Grails-Service in der Klasse `EIDService` implementiert. Die Klasse stellt die Methoden `connect` und `poll` bereit. Die Methode `connect` erwartet einen Parameter vom Typ `OperationsSelectorType`, mit dem die Funktionen und verpflichtenden bzw. optionalen Datenfelder festgelegt werden, die bei der eID-Funktion verwendet werden sollen. Diese Informationen werden bei der `useIDRequest` Nachricht an den eID-Server übertragen. Die Methode `poll` ruft in zyklischer Abfolge die `getResult` Methode der eID-Schnittstelle auf und fragt nach dem Status der Authentisierung des Benutzers. Die Antwort des eID-Servers enthält etwa (1) eine Nachricht, dass die Authentisierung noch nicht abgeschlossen ist oder (2) das Fehler aufgetreten sind oder (3) die ausgelesenen Datengruppen des Ausweises.

Controller

Die Aufgaben der Controller einer Grails-Anwendung liegen im Verarbeiten von Anfragen, Steuern des Anwendungsablaufs, Generieren von Antworten oder im Weiterleiten an Views. Für jede Anfrage wird dabei eine neue Instanz des Controllers erstellt, der den Request verarbeitet.

- `LoginMobileController`
  Der Controller ruft die `connect` Methode des `EIDServices` auf und erhält einen `SessionIdentifier` und einen `PSK` für die Authentisierung des Clients. Im nächsten Schritt öffnet der Controller die zum MIDlet korrespondierende JAD-Datei und fügt die Parameter `ServerAddress`, `SessionIdentifier`, `PathSecurity-Protocol`, `PathSecurity-Parameters`, `RefreshAddress` und `Binding` hinzu (siehe [38, Kapitel 2]). Die modifizierte JAD-Datei wird dann per HTTP-Response an den Client gesendet.

- `LoginController`
  Der Controller implementiert die gleiche Funktionalität wie der `LoginMobileController`, fügt die Parameter `SessionIdentifier`, `PSK` usw. jedoch nicht in eine Datei ein, sondern speichert diese in einem Sitzungsobjekt. Die Parameter werden dann von der `Login` View aus dem Sitzungsobjekt gelesen und als HTML `<object>`-Tag in die HTML Seite eingefügt.

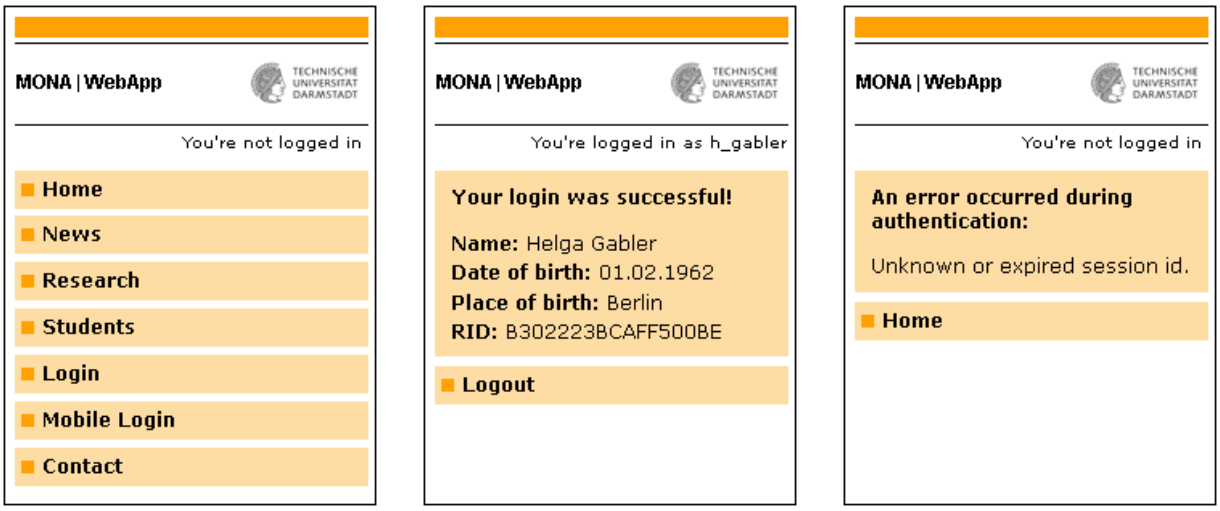
- `PollController`
  Der PollController ruft stetig die Polling-Methode der `EIDService` Klasse auf, bis ein Ergebnis der Authentisierung oder eine Fehlermeldung vorliegt.
- `StartController`
  Implementiert keine Funktionalität.
- `SuccessController`
  Implementiert keine Funktionalität.
- `ErrorController`
  Implementiert keine Funktionalität.

Views

Views definieren die einzelnen Webseiten einer Web-Applikation und erzeugen den dazugehörigen HTML Code. Folglich repräsentieren die Views das Erscheinungsbild einer Web-Applikation (siehe Abbildung 17.2). Die einzelnen Seiten bzw. Views sind als *Groovy Server Pages* [12] implementiert.

- `LoginMobile`
  Implementiert keine Anzeige.
- `Login`
  Erstellt eine HTML Seite, die den HTML `<object>`-Tag für lokale eID-Applikationen enthält (z.B. lokale MONA Applikation oder AusweisApp).
- `Poll`
  Implementiert keine Anzeige.
- `Start`
  Definiert die Startseite der Web-Applikation. Die Seite enthält zwei Buttons: Der Mobile Login ruft den `LoginMobileController` auf, während der Login Button mit dem `LoginController` verknüpft ist.
- `Success`
  Die Seite zeigt die Benutzerseite mit den ausgelesenen Datengruppen an.
- `Error`
  Die Webseite zeigt Fehlermeldungen an, die bei der Authentisierung aufgetreten sind und von der Web-Applikation stammen oder vom eID-Server übermittelt wurden.

Im jetzigen Entwicklungsstand der Web-Applikation muss der Benutzer noch zwischen der *mobilen* oder *stationären* eID-Applikation auf der Startseite der Web-Applikation wählen. Erweiterungen wie *Spring Mobile* [118] ermöglichen es, dass die Web-Applikation automatisch den Client bzw. das Gerät erkennt und so die passende Variante wählt. Auch Eigenschaften der Darstellung wie die Schriftgröße können so auf jedem Gerät bzw. Display optimal angepasst werden.



(a) Startseite der Web-Applikation. MONA wird durch klicken auf Login gestartet.

(b) Benutzerseite nach einem erfolgreichem Login. Ausgelesene Daten werden angezeigt.

(c) Seite mit Fehlermeldung bei fehlgeschlagender Authentisierung.

Abbildung 17.2.: Graphische Benutzeroberfläche der Web-Applikation (Screenshots)

Die Konfiguration der Web-Applikation und insbesondere der Anbindung zum eID-Server erfolgt zentral über die Spring-Konfigurationsdatei. Die Datei definiert den *Web Service Client* (JAX-WS) [106] und das Transportprotokoll sowie die Parameter für die zertifikat-basierte Authentisierung zwischen Web-Applikation und eID-Server per TLS-Protokoll. Zusätzlich beinhaltet die Konfigurationsdatei die Einstellungen für *WS-Security* [86], was zur Absicherung der Web Service Kommunikation eingesetzt wird. Dabei wird ein X.509-Zertifikat und Signaturen per *XML-Signature* [18] zur Validierung des Nachrichtenaustauschs verwendet. Weitere Details sind aus [113] und der Dokumentation von Apache CXF [2] zu entnehmen. Weitere Einstellungen der Web-Applikation wie die URL der SOAP-Schnittstelle des eID-Servers und Parameter für die eID-Applikation (`ServerAddress`, `RefreshAddress`) werden ebenfalls über die projektspezifische Konfigurationsdatei verwaltet.

Teil V.

Bewertung

18 Benchmarks

Im folgenden Kapitel wird die Laufzeit von MONA analysiert und zeitkritische Operationen benannt. Als Vergleichswerte zu den Messungen mit dem Nokia 6212 wurden zusätzliche Messungen mit einem Rechner und einem stationärem Kartenleser durchgeführt. Bei den Zeitangaben in Form von α/β ms in Tabelle 18.1 repräsentiert α den Startzeitpunkt der Operation und β den Endzeitpunkt. Bei Messwerten, die eine Verarbeitung auf den verschiedenen Kommunikationsebenen (vgl. Kapitel 13) implizieren, repräsentiert der Startzeitpunkt den Beginn der Verarbeitung auf dem obersten Layer und der Endzeitpunkt den Abschluss der Verarbeitung auf dem niedrigsten Layer. Die Angaben in den Tabellen 18.1, 18.2 und 18.3 sind in Millisekunden notiert. Die Messwerte in der Spalte *Zeitverlauf* in den Tabellen 18.1 und 18.2 spiegeln den zeitlichen Verlauf der Ausführung nach dem Programmstart wieder.

Im ersten Schritt (siehe Tabelle 18.1 – Initialisierung) wird das Sitzungsobjekt für den weiteren Programmablauf initialisiert. Die Sitzungsparameter, wie `SessionIdentifer` und `PSK`, werden aus der JAD-Datei extrahiert und die einzelnen Layer instanziiert und verknüpft. Darüber hinaus werden Einstellungen für das Logging und ggf. Objekte und Ressourcen für eine graphische Benutzeroberfläche geladen. Die Messungen der Laufzeiten wurden jedoch automatisiert und ohne Benutzereingaben erstellt. Jegliche oszillierenden Zeitfaktoren, ausgelöst durch Interaktionen mit dem Benutzer in Form von PIN-Eingabe und Auswahl der Datengruppen sind daher nicht in den Zeitangaben von Tabelle 18.1 enthalten. Dieser zusätzliche Zeitaufwand würde vor der Durchführung des PACE-Protokolls entstehen und ist in Tabelle 18.1 zwischen Punkt 6 und 7 einzuordnen. Es ist jedoch anzumerken, dass eine graphische Benutzeroberfläche mit signifikantem Ressourcenverbrauch, in Form von Statusanzeigen, Fortschrittsleisten oder Animationen, insgesamt zu einer höheren Ausführungszeit führen kann. Eine schlechte Übertragungsqualität zum Ausweis bzw. zum eID-Server haben ebenfalls negative Auswirkungen auf die Gesamtlaufzeit.

In Schritt 2 erfolgt der Aufbau der TLS-Verbindung zwischen MONA und dem eID-Server. Das in diesem Schritt ablaufende *TLS Handshake* Protokoll impliziert einen weiteren Austausch von Nachrichten und kryptographischen Operationen, was sich in den Messwerten widerspiegelt. Nach erfolgreichem Verbindungsaufbau folgt dann in Schritt 3 der Aufbau der PAOS-Verbindung und der Austausch der ersten eCard-API-Nachrichten in den Schritten 4 – 6. Der Empfang der `DIDAuthenticate` (EAC1InputType) Nachricht in Schritt 6 löst auf dem eCard-API-Layer ein Event zur Benutzerinteraktion aus, welches ggf. von einem `IUIController` verarbeitet wird (vgl. Kapitel 15.1).

Im den Schritten 7 – 15 folgt die Durchführung des PACE-Protokolls und damit die Absicherung der Luftschnittstelle zwischen Smartphone und dem Personalausweis. Zu Beginn wird die Datei `EF.CardAccess` ausgelesen und die Domain-Parameter D werden extrahiert und verarbeitet. Es folgt die Initialisierung

von PACE und die Durchführung der einzelnen Protokollschritte. In Schritt 12 findet das Erzeugen der beiden flüchtigen PACE-Schlüssel und eine Multiplikation mit dem Basispunkt der elliptischen Kurve statt. Dieser Schritt kann, wie in [126] beschrieben, durch Vorberechnungen direkt nach dem Programmstart aus Basis von zwischengespeicherten Domain-Parametern optimiert werden. Die Vorberechnungen der Schlüssel sind jedoch bei den in Tabelle 18.1 gemessenen Durchläufen nicht aktiviert, da sich diese zusätzlichen Berechnungen negativ auf andere Operationen auswirken, sofern keine freien Ressourcen zur Verfügung stehen. Sollten zwischen Schritt 6 und 7 jedoch Interaktionen mit dem Nutzer erfolgen, dann können die Vorberechnungen zu diesem Zeitpunkt ausgeführt und die Laufzeit von PACE um bis zu eine Sekunde reduziert werden. In Schritt 13 finden die Berechnungen der neuen Domain-Parameter D' statt. Dabei wird eine optimierte Berechnung, wie in [126] vorgestellt, verwendet. Auch wenn damit alle ressourcen-intensiven ECC-Operationen durchgeführt sind, ist zwischen Schritt 14 und 15 ein hohes Zeitaufkommen zu verzeichnen. In Schritt 14 werden zuerst – teilweise parallel ausgeführt – die Sitzungsschlüssel K_{MAC} und K_{ENC} für das Secure Messaging sowie das Authentisierungs-Token T_{PCD} berechnet und an den Ausweis übertragen. Bei den in Tabelle 18.1 aufgeführten Messwerten erfolgt die Antwort des Ausweises nach 245 ms bei den Messungen mit dem Nokia bzw. 234 ms auf dem Rechner. Die Verzögerung ist zwar in den Messwerten enthalten, wird jedoch durch parallele Berechnungen von MONA produktiv genutzt. Nach dem Empfang des vom Ausweis stammenden Tokens T_{PICC} erfolgen dessen Verifikation und das Extrahieren der Certificate Authority Reference (CAR). Im Anschluss an eine erfolgreiche Authentisierung aktiviert der EAC-Layer das Secure Messaging auf dem IFD-Layer mit den von PACE ausgehandelten Sitzungsschlüsseln und versendet eine `DIDAuthenticateResponse` (EAC1OutputType) Nachricht an den eID-Server.

Der Empfang der `DIDAuthenticate` (EAC2InputType) Nachricht in Schritt 17 leitet die Terminal Authentisierung ein. Bei der Übermittlung der Zertifikatskette in Schritt 18 werden zwei Zertifikate und dementsprechend vier APDUs an den Ausweis übertragen, da jedes Zertifikat mit einer `MSE:Set DST` APDU initialisiert und mit einer `PSO:Verify Certificate` APDU übertragen wird. Das Auslesen der Datei `EF.CardAccess` benötigt 9 APDUs, was in den Laufzeitmessungen deutlich zu erkennen ist. Die Chip Authentisierung wird direkt nach dem Auslesevorgang durchgeführt und in Schritt 27 mit dem Versand des Authentisierungs-Tokens in der `DIDAuthenticateResponse` (EAC2OutputType) Nachricht abgeschlossen. Der EAC-Layer deaktiviert zu diesem Zeitpunkt das Secure Messaging für die Kommunikation mit dem Ausweis, weil die gesicherte Verbindung bzw. Verschlüsselung dann direkt zwischen eID-Server und Ausweis erfolgt.

Nr.	Operation	Zeitverlauf	
		Nokia 6212	Rechner
1	Initialisierung	839	110
2	TLS-Verbindung hergestellt	3355	719
3	StartPAOS gesendet	3358/3590	719/750
4	InitializeFramework empfangen	4158/4239	860/860
5	InitializeFrameworkResponse gesendet	4301/4439	875/875
6	DIDAuthenticate (EAC1InputType) empfangen	5055/5104	1000/1000
	PACE		
7	EF.CardAccess ausgelesen	6882/7201	1188/1594
8	Domain-Parameter extrahiert	7508/8178	1641/1782
9	PACE Initialisiert	8188	1782
10	MSE: Set AT gesendet	8304	1797
11	General Authenticate Encrypted Nonce	8461	1953
12	General Authenticate Map Nonce	8790	2250
13	General Authenticate Key Agreement	10331	2688
14	General Authenticate Mutual Authentication	12969	2985
15	PACE abgeschlossen	14383	3250
	Terminal Authentisierung		
16	DIDAuthenticateResponse (EAC1OutputType) gesendet	14430/14689	3266/3266
17	DIDAuthenticate (EAC2InputType) empfangen	16651/16729	3500/3500
18	Verify Certificates	16789	3516
19	MSE:Set AT	17947	4344
20	Get Challenge	18055	4407
21	DIDAuthenticateResponse (EAC2OutputType) gesendet	18145/18307	4453/4453
22	DIDAuthenticate (EACAdditionalInputType) empfangen	18766/18803	4578/4578
23	External Authentication	18806	4578
24	EF.CardSecurity ausgelesen	19127/20611	4828/5625
	Chip Authentiserung		
25	MSE:Set AT	20660	5625
26	General Authenticate	20777	5719
27	DIDAuthenticateResponse (EAC2OutputType) gesendet	21135/21957	6000/6032
	Restricted Identification, Datenzugriff		
28	Transmit empfangen	24029/24156	6297/6313
29	TransmitResponse gesendet	25461/25677	7141/7157
30	StartPAOSResponse empfangen	26204/26237	7282/7282
31	Verbindungsabbau abgeschlossen	26287	7297
32	Authentisierung abgeschlossen	26294	7297

Tabelle 18.1.: Laufzeitmessung

Bei den in Tabelle 18.1 aufgeführten Zeiten wurden in Punkt 28 – 29 nur die Datengruppen DG04, DG05, DG08 und DG09 (Vorname(n), Familienname, Geburtsdatum, Geburtsort) und die Restricted Identifcation durch den eID-Server angefordert bzw. durchgeführt. Das korrespondierende CHAT lautet: 0x0000019804. Im Detail betrachtet, benötigt jeder Auslesevorgang einer Datengruppe per `READ BINARY` APDU beim Nokia ca. 80 ms. Die `Verify` APDU für die *Document Validity Verification* (siehe [41, Anhang A.6.5.3, B.5]) ist ebenfalls mit ca. 80 ms zu bemessen. Das Ausführen der Restricted Identifcation benötigt ca. 480 ms. Hierbei ist zu beachten, dass dies zwei APDUs (`MSE: Set AT` und `General Authenticate`) und eine Multiplikation auf dem Ausweises beinhaltet. Gleiches gilt für den Sperrlistenabgleich, der technisch ebenfalls ein Durchführen der Restricted Identifcation Funktion ist. Die gleichen Operationen sind auf dem Rechner per Kartenleser mit 30 ms für die Datengruppen und die Document Validity Verification sowie 350 ms für die Restricted Identifcation zu beziffern. Die Ausführungszeit variiert daher auch mit der Anzahl der abgefragten Daten und Funktionen. In Folge dessen ist in Abhängigkeit vom Umfang der angeforderten Informationen mit Schwankungen von ca. einer Sekunde zu rechnen.

Mit dem Empfang der Nachricht `StartPAOSResponse` in Schritt 30 ist die Authentisierung beendet und die Verbindung zum Server kann getrennt werden. Eingeleitet durch den eCard-API-Layer beenden die übrigen Layer die Verbindung zum Server und geben die verwendeten Ressourcen frei. Das heißt, der EAC-Layer meldet sich bei dem *Discovery Manager* [102] der JSR 257 bzw. NFC-Schnittstelle ab, die Layer löschen ihre Sitzungsvariablen und abschließend sendet der TLS-Layer ein `Close notify` per *TLS Alert* Protokoll an den eID-Server. Im Schritt 32 sind alle Operationen abgeschlossen und MONA versucht die `RefreshAddress` zu öffnen.

Die Angaben in Tabelle 18.1 zeigen nur die Messwerte der obersten und untersten Ebene beim Versand bzw. Empfang einer Nachricht, in Tabelle 18.2 sind zusätzlich genauere Messungen der Ebenen aufgeführt. Die Angaben in der Tabelle beziehen sich auf den Versand der `StartPAOS` bzw. den Empfang der `InitializeFramework` Nachricht. Zu beachten ist, dass „Nachricht versendet“ die Verarbeitung der Nachricht auf der jeweiligen Ebene impliziert, ein „Nachricht empfangen“ jedoch nur das Eintreffen einer Nachricht auf dem Layer bedeutet. Für die in Tabelle 18.2 aufgeführten Werte ergeben sich daher auf dem Nokia eine Verarbeitungszeit von 169 ms für die `StartPAOS` Nachricht auf der SOAP-Ebene. Auch bei der Verarbeitung einer ankommenden Nachricht ist für den SOAP-Layer der höchste Ressourcenaufwand zu beziffern. Zeitintensiv ist insbesondere die Serialisierung der XML-Datenstrukturen. Beispielsweise beansprucht die `DIDAuthenticateResponse` (EAC2OutputType) Nachricht mit 5283 Byte ca. 530 ms, was zu einer Gesamtzeit von 822 ms für das Versenden der Nachricht führt.

Nr.	Operation	Zeitverlauf	
		Nokia 6212	Rechner
3	StartPAOS versendet (eCard-API-Layer)	3358	734
3.1	Nachricht versendet (SOAP-Layer)	3527	734
3.2	Nachricht versendet (PAOS-Layer)	3539	734
3.3	Nachricht versendet (TLS-Layer)	3584	750
3.4	Warte auf Response	3590	765
4	Nachricht empfangen (TLS-Layer)	4158	870
4.1	Nachricht empfangen (PAOS-Layer)	4171	875
4.2	Nachricht empfangen (SOAP-Layer)	4173	875
4.3	InitializeFramework empfangen (eCard-API-Layer)	4239	875

Tabelle 18.2.: Laufzeitmessung (Layer)

Mit größeren ankommenden Nachrichten steigt auch die Verarbeitungszeit auf dem TLS-Layer, weil hier ggf. durch Fragmentierung in mehrere TCP-Pakete die Nachricht erst wieder zusammengefügt werden muss, bevor sie an den PAOS-Layer weitergereicht werden kann. Die Vorbereitungszeit des PAOS-Layer bleibt bei allen Nachrichten ungefähr konstant. Allgemein liegt die Verarbeitungszeit einer Nachricht durch die Layer auf dem Nokia beim Versand von Nachrichten insgesamt zwischen 138/305/822 ms (Minimum, Durchschnitt, Maximum) und beim Empfang bei 33/68/127 ms (Minimum, Durchschnitt, Maximum). Bei dem Rechner sind die Verarbeitungszeiten der Layer zu vernachlässigen und sind auch nur bei der `DIDAuthenticateResponse` (EAC2OutputType) mit 32 ms signifikant zu messen.

In Tabelle 18.3 sind die Laufzeiten der einzelnen Abschnitte aus Tabelle 18.1 zusammengefasst. Zu beachten ist, dass die Angaben teilweise den Versand und Empfang der eCard-API-Nachrichten und auch das Einlesen der Datei `EF.CardAccess` und `EF.CardSecurity` beinhaltet. Das heißt, die Werte enthalten auch sekundäre Faktoren wie die Bandbreite der Internetanbindung. Eine ausführliche Analyse dieses Aspektes folgt im nächsten Abschnitt „Mobilfunknetz“. Die Angaben in der Zeile „bereinigt“ geben die tatsächliche Ausführungszeit des Protokolls an, d. h. ohne die genannten zusätzlichen Faktoren. So beträgt die Laufzeit für das PACE-Protokoll beispielsweise nur 6195 ms (1468 ms Rechner) ohne Auslesen der `EF.CardAccess`, dem Parsen der Domain-Parameter und dem logischen und erforderlichen Protokollabschluss per `DIDAuthenticateResponse` (EAC1OutputType) Nachricht.

	Init	TLS	PACE	TA	CA	RI, Daten
Nokia 6212	839	2516	7807	3958	1320	1648
(bereinigt)			6195	1699	496	1279
Rechner	110	562	2078	2125	407	860
(bereinigt)			1468	1187	375	844

Tabelle 18.3.: Laufzeitmessung (Zusammenfassung)

In Tabelle 18.4 sind die Größen der jeweiligen Protokollnachrichten angegeben. Das Datenvolumen der Nachrichten beträgt insgesamt 28761 Bytes. Hinzu kommt für jede Nachricht der HTTP-Header mit 195 Bytes. Das Datenvolumen zwischen eID-Server und MONA beträgt daher 31101 Byte (≈ 30 KB). Durch die TLS-Verschlüsselung und Fragmentierung durch TCP sollte sich der tatsächliche Datenaustausch auf maximal 35 KB beziffern lassen.

Nachricht	Nachrichtengrösse	
	MONA	eID-Server
StartPAOS	1239	
InitializeFramework		1510
InitializeFrameworkResponse	1162	
DIDAuthenticate (EAC1InputType)		3818
DIDAuthenticateResponse (EAC1OutputType)	2875	
DIDAuthenticate (EAC2InputType)		2840
DIDAuthenticateResponse (EAC2OutputType)	1307	
DIDAuthenticate (EACAdditionalInputType)		1631
DIDAuthenticateResponse (EAC2OutputType)	5283	
Transmit		3686
TransmitResponse	2143	
StartPAOSResponse		1266
	14010 Bytes	**14751 Bytes**

Tabelle 18.4.: Größenangaben der eCard-API-Nachrichten

Zusammenfassend lässt sich feststellen, dass auf dem sehr ressourcen-beschränkten Nokia 6212 die Zeitaufwände für die einzelnen Operationen überwiegend konstant verteilt sind. Zeitaufwendige Operationen sind die ECC-Berechnungen bei PACE und die Serialisierung bei der XML-Verarbeitung. Hinzu kommen die Faktoren der Internetanbindung und auch die nicht zu vernachlässigenden Operationen des Ausweises. Bei der Messung mit dem Nokia 6212 entfallen 5474 ms auf den Ausweis, was ca. 21 % der Gesamtlaufzeit entspricht. Bei dem Rechner beträgt der Anteil sogar 62 % (4515 ms). Die TR-03105-2 [40] definiert auf Basis der ISO/IEC 14443-2 [74] eine magnetische Feldstärke von 1,5 A/m – 7,5 A/m für die Systemtests mit dem Ausweis. Es ist unwahrscheinlich, dass das Nokia diese Feldstärke bereitstellt und auch zukünftige NFC-fähige Geräte diese Leistung zur Verfügung stellen können. Es ist daher davon auszugehen, dass die Übertragungsdauer zwischen NFC-fähigem Gerät und Ausweis weiterhin höher liegen wird, als bei einem stationären Kartenleser.

Die Kommunikation über das Mobilfunknetz ist bei dem Nokia mit 6204 ms und bei dem Rechner mit 984 ms zu beziffern. Den Aspekt der Internetanbindung des Handys wird im nächsten Abschnitt analysiert.

Mobilfunknetz

Während die Zeitspanne zwischen Versand und Empfang einer eCard-API-Nachricht auf dem Rechner mit ca. 150 ms sehr gering ist, wurden auf dem Nokia 6212 teilweise 2000 ms gemessen. Da die Messungen mit dem Nokia 6212 unter Verwendung des Datendienstes *EDGE* (Enhanced Data Rates for GSM Evolution) durchgeführt wurden, galt es zu prüfen in wie weit das Kommunikationsnetz Auswirkungen auf die Übertragungsdauer hat und welche Übertragungszeiten bei anderen Netzen zu erwarten sind. EDGE bietet nur ungefähr 50 % der Datenübertragungsrate von *UMTS* (Universal Mobile Telecommunications System). Das Nokia 6212 unterstützt zwar UMTS, jedoch ist für den Datendienst bzw. für die Datenübertragung nur GPRS oder EDGE verfügbar. Da elegante Lösungen zur Messungen auf dem Nokia leider nicht zur Verfügung stehen, wurden die Paketumlaufzeiten (Round Trip Time, RTT [108]) auf einem Rechner und einem HTC Desire mit dem Programm ping[1] gemessen. Die Messungen repräsentieren 100 ICMP Pakete zum Ziel-Host `http://www.tu-darmstadt.de`. Die Messungen der Bandbreite erfolgten über den Benchmark-Dienst von Ookla unter `http://www.speedtest.net` bzw. die ebenfalls von diesem Anbieter bereitgestellte Applikation für Android.

	Bandbreite		Paketumlaufzeit				
	Download	Upload	Minimum	Durchschnitt	Maximum	Abweichung	Zeit
Rechner Alice DSL 16000	10382 kbit/s	933 kbit/s	22,63 ms	23,65 ms	25,58 ms	0,52 ms	99123 ms
HTC Desire T-Mobile HDSPA	4548 kbit/s	1392 kbit/s	49,85 ms	69,94 ms	98,05 ms	8,15 ms	99151 ms
HTC Desire T-Mobile UMTS	180 kbit/s	188 kbit/s	158,25 ms	198,60 ms	263,99 ms	21,05 ms	99135 ms

Tabelle 18.5.: Benchmarks der Kommunikationsnetze

Wie Tabelle 18.5 zeigt, bestehen erhebliche Leistungsunterschiede bei der zur Verfügung stehenden Bandbreite und der Paketumlaufzeit der getesteten Kommunikationsnetze. Die zur Verfügung stehende Bandbreite der Netze ist für MONA aber eher zweitrangig, da die zu übertragende Datenmenge der Nachrichten nur wenige Kilobytes betragen. Auffällig sind jedoch die großen Unterschiede bei der Paketumlaufzeit. Werden die durchschnittlichen Messwerte der einzelnen Netze verglichen, dann liegt der Wert bei UMTS fast zehnmal höher als bei einer DSL Verbindung. Für das Nokia 6212 mit dem Datendienst EDGE sind noch höhere Werte anzunehmen. Die hohe Übertragungsdauer beim Nokia ist daher insbesondere auf das EDGE Netz zurückzuführen. Damit ist davon auszugehen, dass die Gesamtlaufzeit von MONA durch UMTS oder *HSDPA* (High Speed Downlink Packet Access) signifikant reduziert werden kann. Werden die Messwerte aus Tabelle 18.1 betrachtet, dann ist die Übertragungsdauer für die zwölf Nachrichten auf ca. 6000 ms (950 ms Rechner) zu beziffern. Mit den gemessenen Werten in Tabelle 18.5 ist für moderne Smartphones mit UMTS oder HSDPA davon auszugehen, dass sich die Übertragungsdauer halbiert.

[1] `http://www.debianadmin.com/manpages/pingmanpage.htm`

19 Optimierungen

Die gemessene Laufzeit von MONA mit 26,3 Sekunden ist für einen produktiven und benutzerfreundlichen Einsatz deutlich zu hoch. Eine wesentliche Beschleunigung werden modernere und performante Geräte und Mobilfunknetze bringen. Jedoch können auch noch mehrere Aspekte von MONA optimiert werden, die in den nächsten Abschnitten vorgestellt werden.

19.1 eCard-API-Nachrichten

Die Nachrichten zwischen eID-Anwendung und eID-Server sind durch das SOAP-Protokoll und das eCard-API-Framework strikt definiert. Jedoch können die Nachrichten durch den Verzicht auf nicht benötigte Definitionen der *XML-Namespaces* komprimiert werden. Wie in Auflistung 19.1 ersichtlich, definiert das `InitializeFramework` Element im SOAP-Body eine Vielzahl von Namespaces. Für ein valides XML-Dokument ist jedoch ausschließlich die Definition mit dem Präfix `ns6` erforderlich, auf die weiteren Definitionen kann, wie in Auflistung 19.2 dargestellt, verzichtet werden.

```
<S:Envelope xmlns:S=http://schemas.xmlsoap.org/soap/envelope/ ... >
   <S:Header>
      ...
   </S:Header>
   <S:Body>
      <ns6:InitializeFramework
         xmlns="urn:oasis:names:tc:dss:1.0:core:schema" xmlns:ns2="http://www.w3.org/2000/09/xmldsig#"
         xmlns:ns3="urn:iso:std:iso-iec:24727:tech:schema" xmlns:ns4="http://uri.etsi.org/01903/v1.3.2#"
         xmlns:ns5="http://www.w3.org/2001/04/xmlenc#" xmlns:ns6="http://www.bsi.bund.de/ecard/api/1.1"
         xmlns:ns7="http://uri.etsi.org/02231/v2#" xmlns:ns8="http://www.setcce.org/schemas/ers"
         xmlns:ns9="urn:oasis:names:tc:SAML:1.0:protocol" xmlns:ns10="urn:oasis:names:tc:SAML:1.0:assertion"
         xmlns:ns11="urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#"
         xmlns:ns12="http://www.w3.org/2001/04/xmldsig-more#"
         xmlns:ns13="urn:oasis:names:tc:SAML:2.0:assertion"/>
   </S:Body>
</S:Envelope>
```

Auflistung 19.1: InitializeFramework Nachricht

Die `InitializeFramework` Nachricht aus Auflistung 19.1 kann damit von 1525 Bytes auf die Nachricht in Auflistung 19.2 mit 919 Bytes reduziert werden. Die ca. 600 Bytes Mehraufkommen haben bei einer UMTS-Verbindung mit den gemessen Geschwindigkeiten aus Tabelle 18.5 rechnerisch eine ca. 25 ms höhere Übertragungsdauer zur Folge. Zu beachten ist außerdem, dass die größere Nachricht die oft verwendete MTU (Maximum Transmission Unit) von 1492 Bytes bei TCP Paketen [57, 110, 84] überschreitet. Demnach erfordert der Transport der größeren Nachricht zwei TCP-Pakete während die komprimierte Nachricht in einem Paket versendet werden kann. Weitere Informationen zur Kommunikation via TCP sind [120] zu entnehmen.

```
<S:Envelope xmlns:S=http://schemas.xmlsoap.org/soap/envelope/ ... >
   <S:Header>
      ...
   </S:Header>
   <S:Body>
      <ns6:InitializeFramework
         xmlns:ns6="http://www.bsi.bund.de/ecard/api/1.1"/>
   </S:Body>
</S:Envelope>
```

Auflistung 19.2: InitializeFramework Nachricht (komprimiert)

Eine geringe Datengröße hat neben der schnelleren Übertragung auch eine schnellere Verarbeitung zur Folge. Das heißt weniger Ressourcen bzw. eine geringere CPU Auslastung und damit einen geringeren Stromverbrauch. Auch wenn die Transportzeit mit wenigen Millisekunden sehr gering ausfällt, müssen die zusätzlichen Daten entschlüsselt und vom XML-Parser verarbeitet werden. Die Optimierung bzw. Anpassung muss jedoch am eID-Server vorgenommen werden. MONA verwendet bereits diese komprimierte Variante für ausgehende SOAP- bzw. eCard-API-Nachrichten zum Server.

19.2 Extended APDUs

Das Nokia 6212 unterstützt „offiziell“ keine *Extended APDUs* gemäß ISO/IEC 7816-4, d. h. keine APDUs mit mehr als 255 Bytes Daten. Trotzdem können mit dem Nokia Command-APDUs mit mehr als 255 Bytes versendet, Response-APDUs jedoch nur mit bis zu 253 Bytes empfangen werden. Dieses Phänomen konnte auch bei jeweils einem Gerät von Samsung und MobiWire festgestellt werden.

Ohne Unterstützung von Extended APDUs muss die Übertragung der Dateien `EF.CardAccess` und `EF.CardSecurity` mittels mehrerer `READ BINARY` APDUs erfolgen, die durch einen unterschiedlichen Offset im Header die Dateien in mehreren Blöcken anfordert. Wie in Tabelle 19.1 aufgeführt, sind dadurch, anstatt einer Extended APDU für eine Datei, drei bzw. neun APDUs notwendig, um die Datei auszulesen. Die verhältnismäßig höhere Laufzeit beim Auslesen der Datei `EF.CardSecurity` resultiert durch den Secure Messaging Kanal.

Datei	Dateigröße	Anzahl APDUs	Zeit
`EF.CardAccess`	616 Bytes	3	250 ms
`EF.CardSecurity`	1938 Bytes	9	1200 ms

Tabelle 19.1.: Details zur Übertragung der Systemdaten des Ausweises

Insofern das verwendete mobile Gerät Extended APDUs unterstützt, sollten für das Auslesen der Dateien `EF.CardAccess` und `EF.CardSecurity` Extended `READ BINARY` APDUs verwendet werden. Damit reduziert sich die Anzahl von zwölf auf zwei APDUs. Diese Änderung sollte die Laufzeit für das Auslesen der Dateien von insgesamt ca. 1450 ms auf ca. 200 - 300 ms reduzieren.

19.3 Graphische Benutzeroberfläche

Bei der Interaktion mit dem Benutzer ist eine übersichtliche Darstellung der Informationen des Berechtigungszertifikats und die Auswahl der Datengruppen auf kleinen Displays und geringer Auflösung wie beim Nokia 6212 nur eingeschränkt möglich. Wird der Umfang der darzustellenden Informationen betrachtet, wie in Abbildung 19.1 bei der AusweisApp [25] dargestellt, dann ist ebenfalls fraglich, ob diese auf großen Displays adäquat dargestellt werden können. Es sollte offen diskutiert werden, welche Informationen tatsächlich für den Benutzer relevant und welche ggf. im Umfeld der mobilen Nutzung weniger von Bedeutung sind. Zum Beispiel ist zu diskutieren, ob Angaben wie Gültigkeit des Berechtigungszertifikates oder die ausstellende Zertifizierungsstelle einen Mehrwert an Sicherheit für den Benutzer darstellen. Da ein Berechtigungszertifikat nur 48 Stunden gültig ist, ist der Informationsgehalt diese Angabe wohl eher gering. Es sollte Aufgabe der eID-Applikation sein, die Verbindung nach dem Empfang eines ungültigen bzw. abgelaufenen Zertifikates abzubrechen. Fraglich ist auch, ob die Angabe der Zertifizierungsstelle einen Sicherheitsgewinn für den Benutzer mit sich bringt. Es ist davon auszugehen, dass der Name der Zertifizierungsstelle für eine Vielzahl der Nutzer keine Aussage auf einen vertrauenswürdigen oder bekannten Anbieter beinhaltet.

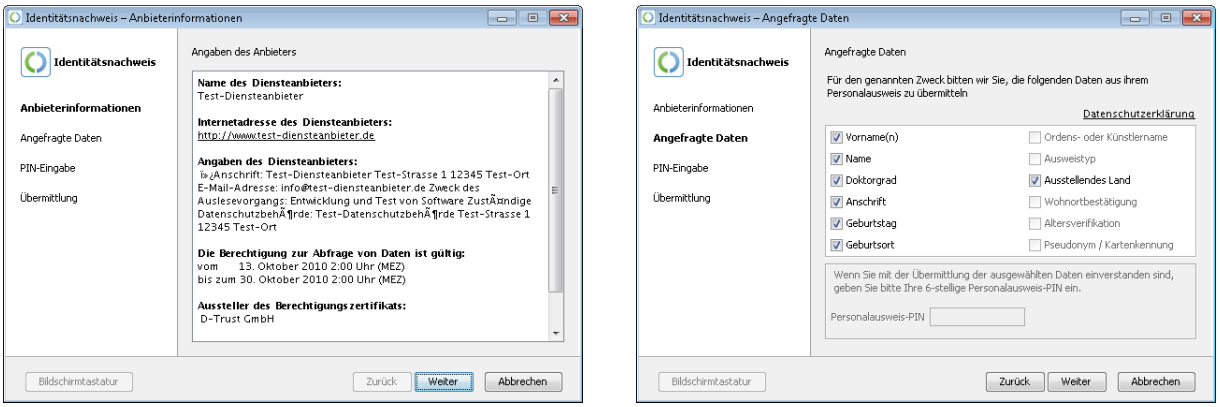


Abbildung 19.1.: Graphische Benutzeroberfläche der AusweisApp (Screenshots)

Wie in Abbildung 19.1 ersichtlich, lassen sich die Informationen auf einem Computer praktikabel darstellen. Eine Liste mit zwölf Datengruppen bzw. Funktionen oder gar den Informationen zum Berechtigungszertifikat sind auf einem Smartphone leider weder übersichtlich noch benutzerfreundlich darstellbar. Eine Darstellung im mobilen Bereich sollte sich somit auf die wesentlichen Informationen beschränken: Die auszulesenden Datengruppen und Name sowie Internetadresse des Dienstanbieters, der die Daten erhält.

Ein weiterer Aspekt wäre Berechtigungsprofile für bestimmte Anbieter einzurichten. Es ist anzunehmen, dass beispielsweise für eine Registrierung bei einem Online-Shop immer der Name und die Anschrift erforderlich sind. Für spätere bzw. wiederholte Anmeldung jedoch nur die Restricted Identification. Denkbar ist, in MONA Profile für bestimmte Gruppen von Anbietern einzurichten und diesen bereits definierte Rechte auf Datengruppen aus Sicht des Benutzers zuzuordnen. Bei einer zweiten Anmeldung

wird dann automatisch nur Zugriff auf die Restricted Identification gewährt. Ziel dieser Profile sollte eine automatische Sparsamkeit des Zugriffs auf die Datengruppen sein. Gegebenenfalls könnte damit verhindert werden, dass Benutzer gutgläubig zusätzliche Datengruppen freigeben und informiert werden, wenn der Anbieter andere Datengruppen anfordert als bei der letzten Anmeldung notwendig waren.

Das Einführen solcher Berechtigungsprofile kollidiert jedoch mit den Anforderungen der TR-03127 des BSI. Ebenfalls müsste das Wiedererkennen eines Dienstanbieters bzw. das Zuordnen zu einer Anbietergruppe durch zusätzliche Informationen im Zertifikat realisiert werden. Da ein Berechtigungszertifikat nur 48 Stunden gültig ist, kann eine Wiedererkennung nicht anhand einer Prüfsumme des Zertifikats erfolgen. Möglich wäre dies beispielsweise anhand des *Issuer Name* Attributes des Zertifikats, jedoch wäre eine eindeutige Identifikationsnummer vorzuziehen.

20 Ausblick

Das folgende Kapitel präsentiert einen Ausblick auf mögliche Weiterentwicklungen von MONA. Dabei wird eine Portierung auf die Android Plattform und Aspekte für den Wirkbetrieb betrachtet.

20.1 Portierung auf Android

Die Anzahl der in den letzten Jahren verfügbaren NFC-fähigen Geräte ist sehr überschaubar. Ab März 2011 stehen mit dem *Google Nexus S* und dem *Samsung Galaxy S II* zwei neue Oberklasse-Smartphones auf Basis der Android Plattform mit NFC-Funktionalität zur Verfügung. Im Verhältnis zum Nokia 6212, das im Jahre 2008 auf den Markt kam, setzen diese Geräte natürlich ganz neue Maßstäbe in Bezug auf die zu Verfügung stehende Leistung und das Nutzererlebnis.

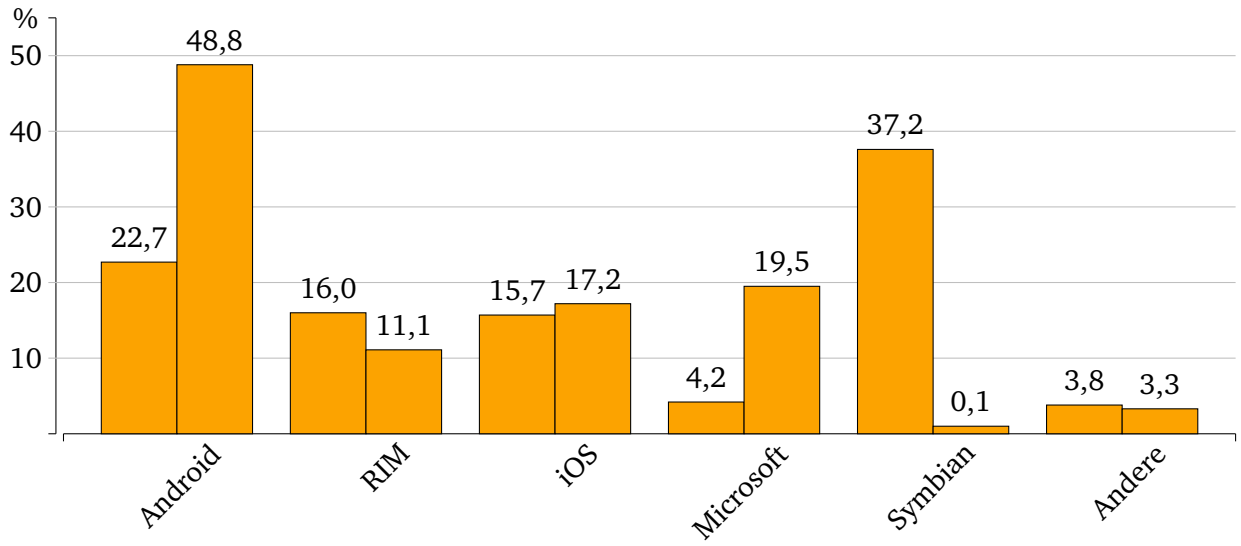


Abbildung 20.1.: Marktanteile der Smartphone-Betriebssysteme für 2010 und Prognose 2015

In Abbildung 20.1 sind die vom Marktforschungsunternehmen Gartner [53] für das Jahr 2010 veröffentlichten und für 2015 prognostizierten Marktanteile der Smartphone-Betriebssysteme illustriert. Die Grafik zeigt, dass Android sehr starke Wachstumsraten zugesprochen werden. Da Anwendungen für Android ebenfalls in Java entwickelt werden und die neuen NFC-fähigen Smartphones überwiegend diese Plattform nutzen bzw. wahrscheinlich nutzen werden, ist insbesondere Android als Ziel-Plattform für eine Portierung von MONA interessant.

Im Folgenden sind die Komponenten und Aspekte beschrieben, die bei einer Portierung angepasst und berücksichtigt werden müssen. Allgemein ist festzuhalten, dass Programmteile von MONA, die „Standardfunktionen“ von Java verwenden, weitestgehend unverändert übernommen werden können. Hingegen sind die Java ME spezifischen Schnittstellen auszutauschen.

Kryptographische Funktionen

Für die Sicherheitsmechanismen des neuen Personalausweises PACE, EAC und Secure Messaging sowie TLS mit Pre-shared Keys werden folgende kryptographischen Funktionen benötigt:

- Hashwert-Berechnung (MD5, SHA-1, zukünftig auch SHA-256)
- Verschlüsselung (AES-128, AES-224 und AES-256 im CBC Mode)
- Elliptische Kurven
- Prüfsummen-Berechnung (AES im CMAC Mode, HMAC)
- RSA

Für die Bereitstellung dieser Funktionen wird zurzeit der FlexiProvider [122] als Cryptographic Service Provider (CSP) verwendet. Ein CSP ist erforderlich, weil die Java ME Plattform nur eine sehr begrenzte Funktionalität an kryptographischen Funktionen bietet. Gleiches gilt auch für die Android Plattform. Die mobile Variante des FlexiProviders sollte mit akzeptablem Aufwand auf Android portiert werden können. Bevorzugt sollte jedoch geprüft werden, ob eine Portierung auch mit der Java SE Variante des Providers möglich ist und damit ein größerer Funktionsumfang an kryptographischen Funktionen zur Verfügung stehen würde.

Transport Layer Security

MicroTLS und die Erweiterung um die Pre-shared Keys Funktionalität benötigen die kryptographischen Mechanismen AES-256, RSA, SHA-1, MD5 und HMAC. Die Hash-Funktionen SHA-1 und MD5 könnten über die Standardfunktionalität von Android abgedeckt werden. Um die Anforderungen von PACE umsetzen zu können, wird jedoch bereits ein externer CSP benötigt, so dass alle erforderlichen kryptographischen Funktionen für TLS von dem CSP abgedeckt sein sollten. Anpassungen an der Implementierung sind aber bei der Schnittstelle zum Internet erforderlich. MONA verwendet hier die HTTP-Schnittstelle bzw. `Connector` Klasse der JSR 118 [98], bei Android muss eine äquivalente Schnittstelle für die TCP- bzw. HTTP-Verbindung verwendet werden. MicroTLS arbeitet intern jedoch nur auf Objekten der Klassen `java.io.InputStream` und `java.io.OutputStream`. Demnach sind hier keine Abhängigkeiten an Java ME Schnittstellen vorhanden und die Kommunikation sollte über die Klasse `java.net.Socket` der Android Plattform hergestellt werden können.

XML

MONA verwendet für die XML-Funktionalität den XML-Parser und Serializer der *XML Pull API* [56] und die darauf aufbauende *kXML 2* [55] Bibliothek für die Datenstrukturen. Die XML Pull API ist bereits in der Klassenbibliothek der Android Plattform im Paket `org.xmlpull.v1` integriert. Bevorzugt wäre daher zu prüfen, ob die kXML Bibliothek auf Android portiert werden kann und die Abhängigkeit an die XML Pull API mit der bereits in Android integrierten Variante aufgelöst werden kann.

Starten von MONA

MONA wird zurzeit durch das Verlinken der JAR- bzw. JAD-Datei gestartet und die Parameter (`SessionIdentifier`, `PSK`, `ServerAddress`, `RefreshAddress` usw.) werden mittels der JAD-Datei übergeben. Android Applikationen verwenden das *Android Package* (APK) Format, was auf dem JAR-Dateiformat basiert. Ein Pendant zur JAD-Datei findet sich bei Android jedoch nicht. Für MONA als Android Applikation ist daher eine Alternative zu finden, wie die Anwendung gestartet wird und wie die Parameter übergeben werden. Zum Starten der Anwendung könnten beispielsweise Links wie `mona://xyz` definiert werden, bei denen die Parameter im *Query* und *Fragment* Teil[1] der URI [6] übergeben werden. Applikationen im Android Market werden beispielsweise mittels `market://search?q=pname:MyApp` verlinkt. Da der Scheme Teil der URI bzw. URL [7] (`market` bzw. `mona`) jedoch durch die Internet Assigned Numbers Authority (IANA) [66] festgelegt wird, ist von dieser Lösung abzuraten. Besser scheint die Verwendung eines `<intent-filter>`[2] zu sein, um eine Applikation für Links, Dienste usw. zu registrieren.

Die RFC 2616 [51] definiert für das HTTP-Protokoll keine Längenbegrenzungen für URLs bzw. URIs, jedoch wird beschrieben, dass ein Server einen Request mit einer zu langen Adresse auch mit dem *Fehlercode 414* (Request-URI Too Long) ablehnen kann. Auch wenn der *Searchpart* Teil der URL nur den `SessionIdentifier`, `PSK` und die `ServerAddress` beinhalten würde, hätte die resultierende Adresse schon mehr als 150 Zeichen. Die Parameter per URL an MONA zu übergeben, könnte daher zu Problemen bei den Browsern oder auch bei den Servern führen, wenn die URLs unbekannte Ausmaße annehmen und nicht mehr verarbeitet werden können. Daher sollte beispielsweise nur der `SessionIdentifier` und ein Adresse zum Empfang der weiteren Parameter in der URL zum starten von MONA kodiert werden. MONA würde dann die Adresse aufrufen und die Parameter aus dem HTML-Code parsen oder die Parameter mit Hilfe eines Web Service Request anfordern.

[1] Vereinfacht ist vom folgenden Aufbau auszugehen: `scheme://<host>:<port>/<path>/?<query>#<fragment>`

[2] `http://developer.android.com/guide/topics/manifest/intent-filter-element.html`

Effizienz

Zum jetzigen Entwicklungsstand beträgt die Laufzeit einer Authentisierung per eID-Funktion auf dem Nokia 6212 ca. 26,3 Sekunden. Wie in [126] ersichtlich, sind die kryptographischen Funktionen und insbesondere die ECC Operationen im Wesentlichen für die Laufzeit verantwortlich. Beim Ausführen des PACE-Protokolls sind beispielsweise fünf Punktmultiplikationen erforderlich, die auf dem Nokia 6212 ca. 5000 ms benötigen. Wie in Tabelle 20.1 aufgeführt, können modernere Smartphones, wie das HTC Desire mit einem 1 GHz Prozessor, jedoch eine Punktmultiplikation in einem Drittel der Zeit ausführen. Allein für die fünf Punktmultiplikationen könnte der Aufwand von ca. 5000 ms auf 1500 ms reduziert werden. Die angegebenen Werte in Tabelle 20.1 entsprechen dem Durchschnitt von zehn Operationen unter Verwendung von 256 Bit Zahlen.

	Nokia 6212	Nokia 6220c	Samsung S8000	HTC Desire
Punktmultiplikation	1051,3 ms	624,6 ms	763,7 ms	305,2 ms
Punktverdopplung	2,3 ms	2,0 ms	1,8 ms	0,7 ms
Punktaddition	3,8 ms	1,9 ms	3,0 ms	1,5 ms

Tabelle 20.1.: Benchmarks von Operationen auf elliptischen Kurven

Zusätzlich zu den ECC-Operationen wirkt sich die Rechenleistung moderner Smartphones auch positiv auf die XML-Verarbeitung aus, die bei dem Nokia 6212 teilweise mehrere hundert Millisekunden in Anspruch genommen hat. Wird beispielsweise das neue Google Nexus S[3] betrachtet, das ebenfalls mit einem 1 GHz Prozessor ausgerüstet ist, oder zukünftige Smartphones mit Dual-Core Prozessoren, dann ist davon auszugehen, dass die Laufzeit für die rechenintensiven Operationen noch weiter sinken wird.

Zusammenfassung

Anwendungen für die Android Plattform werden ebenfalls in Java entwickelt. Dies erleichtert eine Portierung von MONA und den verwendeten externen Bibliotheken. Zusätzlich bietet die Java Klassenbibliothek der Android Plattform einen größeren Funktionsumfang als die Java ME Plattform. Die Klasse `java.lang.String` ist beispielsweise für die SE und ME Plattform verfügbar. Bei der ME Variante stehen jedoch nicht alle Methoden zur Verfügung. Android bietet hier die modernere Klassenbibliothek im Vergleich zur Java ME Plattform. Die zur Verfügung stehenden Klassen entsprechen mehr der von der Java SE bereitgestellten Funktionalität. In Tabelle 20.2 sind die Komponenten und die dazugehörigen Abhängigkeiten zusammengefasst.

[3] `http://www.google.com/nexus/`

Komponente	Abhängigkeiten	Anmerkungen
GUI	-	Vollständige Neuimplementierung erforderlich.
IFD	NFC	API für NFC-Schnittstelle ähnlich der JSR 257.
PACE, EAC, SM, TLS	CSP	Kryptographische Funktionen müssen durch einen CSP bereitgestellt werden.
TLS	HTTP-Schnittstelle	API für HTTP bzw. Socket ähnlich Connector Klasse der JSR 118.
eCard-API-Framework, SOAP, PAOS	XML	XML-Parser und -Serializer sind in Android enthalten. Zusätzliche Datenstrukturen müssen Implementiert oder durch externe Bibliotheken bereit gestellt werden.

Tabelle 20.2.: Komponenten und Abhängigkeiten in Bezug auf eine Portierung

Da die Android Plattform einen breiteren Funktionsumfang in Bezug auf die Java Klassen als die Java ME Plattform bietet, sollten die Herausforderungen insbesondere in den Änderungen bzw. Anpassungen der NFC-Schnittstelle liegen. Zusätzlich ist der FlexiProvider zu portieren und die graphische Oberfläche vollständig neu zu implementieren.

In Bezug auf die Bedienbarkeit und Verwendbarkeit ist mit modernen Smartphones mit 1 GHz Prozessoren und aufwärts sowie schnellen Mobilfunknetzen wie HSDPA eine Laufzeit von 15 Sekunden zu erwarten.

20.2 Wirkbetrieb

Die eID-Anwendung MONA ist auf Basis der Testinfrastruktur und -ausweise entwickelt worden. Für den Einsatz mit aktuellen Ausweisen sind Änderungen vorzunehmen, um den neu definierten Anforderungen und den aktuellen Spezifikationen zu entsprechen. Die Anpassungen für einen Wirkbetrieb bzw. produktiven Einsatz von MONA werden im Folgenden aufgeführt.

Zufallszahlen

Für die Zufallszahlen, die bei dem PACE-Protokoll verwendet werden, fordert das BSI ein Sicherheitsniveau von 100 Bit [48, Anhang A.4.3]. Für Linux empfiehlt das BSI für die *Seedgenerierung* die Funktion `/dev/random` und bei Windows eine Kombination aus den seit Systemstart durchlaufenden Prozessorzyklen und der aktuellen Systemzeit (siehe [30, Kapitel 9.3]). Auf dem mobilen Gerät ist das Sicherstellen der Anforderung an den Pseudozufallszahlengenerator jedoch eine Herausforderung, da das sichere Erzeugen von Zufallszahlen sehr viel Zeit in Anspruch nimmt. Es bietet sich daher an, eine Hardware-Sicherheitsmodul in Form einer (U)SIM oder Micro SD Karte zu verwenden. Alternativ könnte auch, wie in [78] beschrieben, Daten von der Kamera, Mikrofon, Bewegungssensor, GPS-Position, usw. als Quelle für einen *Seedpool* verwendet werden.

Transport Layer Security

MONA verwendet im jetzigen Entwicklungsstand nur die Pre-Shared Key (PSK) *Cipher Suite* `TLS_PSK_WITH_AES_256_CBC_SHA`. Die TR-03112 fordert in der aktuellen Version 1.1 jedoch die PSK Cipher Suite `TLS_RSA_PSK_WITH_AES_256_CBC_SHA` für die TLS-Verbindung zwischen eID-Applikation und eID-Server (siehe [32, Kapitel 2.3.6.1]).

Bei der RSA Variante der PSK Cipher Suite sendet der Server zusätzlich eine `Certificate` Nachricht mit dessen Zertifikat. Der Client wählt eine 46 Bytes Zufallszahl, konkateniert diese mit dem `PSK` und bildet damit in weiteren Operationen das *Premaster Secret*. Die Zufallszahl wird mit dem öffentlichen Schlüssel des Servers aus dem Zertifikat verschlüsselt und innerhalb der `ClientKeyExchange` Nachricht mit dem `SessionIdentifier` an den Server gesendet. Weitere Details sind der RFC 4279 [21, Kapitel 4] zu entnehmen.

PACE und EAC

Die Implementierungen des PACE- und EAC-Protokolls sind auf die Testausweise und -infrastruktur in der Version 2.01 der TR-03110 [31] ausgelegt. Für den Wirkbetrieb sind Änderungen der Implementierung erforderlich, um der aktuellen Version 2.05 der TR-03110 [41] bzw. den an die Bürgerinnen und Bürger ausgegebenen Ausweisen in der Version 2.03 [42] gerecht zu werden. Dazu zählen unter anderem Änderungen an den Domain-Parametern und die Verwendung des CBC-Modes für die AES Verschlüsselung der Zufallszahl beim PACE-Protokoll anstatt des bei den Testausweisen verwendeten ECB-Modes.

SOAP und PAOS-Binding

Das SOAP-Protokoll ist in der Version 1.1 [8] und das PAOS-Binding in der Version 2.0 [1] implementiert. Folglich werden die Vorgaben der aktuellen TR-03110 in der Version 1.1 [32] erfüllt. Änderungen für den Wirkbetrieb sind daher nicht erforderlich.

eCard-API-Framework

Das eCard-API-Framework ist in der aktuellen Version 1.1 [32] implementiert. Daher sind keine Anpassungen für den Wirkbetrieb notwendig. Die Implementierung des Frameworks beschränkt sich jedoch nur auf die für das eID-Szenario erforderlichen Nachrichten und Datenstrukturen.

Extended APDUs

Extended APDUs werden für die Übertragung der Zertifikatskette bei der Termial Authentisierung und für den sektorspezifischen öffentlichen Schlüssel bei der Restricted Identification und dem Sperrlistenabgleich verwendet. Auch wenn MONA ohne eine Unterstützung von Extended APDUs realisiert werden konnte, weil das Endgerät die Übertragung von großen APDUs gestattet, müssen für den Wirkbetrieb die Anforderungen der TR-03110, TR-03119 und der *Certificate Policy für die eID-Anwendung* [50] berücksichtigt werden. Für Endgeräte ist daher eine Unterstützung von Extended APDUs gemäß ISO/IEC 7816 erforderlich.

Web-Applikation

Die Implementierung der eID-Schnittstelle der Web-Applikation zum eID-Server entspricht der aktuellen Version 1.4.1 der TR-03130 [47]. Die Authentisierung der Anwendung gegenüber dem Server ist jedoch auf die Anforderungen des eID-Servers der *media transfer AG* [83] ausgelegt. Es ist daher nicht auszuschließen, dass Änderungen am Authentisierungsmechanismus vorgenommen werden müssen, wenn ein eID-Server eines anderen Anbieters verwendet wird.

21 Fazit

Die mobile eID-Applikation MONA bietet ein Pendant zur stationären AusweisApp. MONA wurde als Java ME MIDlet implementiert und erfolgreich, in Verbindung mit einer eigens entwickelten Web-Applikation, auf dem Nokia 6212 umgesetzt. Die Applikation bemisst 424 KBytes und ca. 20000 Zeilen Quellcode. Dabei entfallen ca. 50 KBytes auf die graphische Benutzeroberfläche. Das Nokia 6212 ist im Jahre 2008 auf den Markt gekommen und mit heutigen Geräten in Bezug auf die Leistung, Bedienbarkeit, Benutzererlebnis und Haptik nicht vergleichbar. Daher ist auch die hohe Laufzeit von MONA mit 26,3 Sekunden insbesondere auf das ressourcen-beschränkte Gerät zurückzuführen. Mit moderneren Geräten und schnellen Mobilfunkverbindungen ist mit einer Laufzeit von ca. 15 Sekunden zu rechnen. Die modernen und aktuellen NFC-fähigen Geräte, die seit Anfang dieses Jahres auf dem Markt sind, basieren hauptsächlich auf der Android Plattform. Die Implementierung von MONA wurde darauf ausgelegt, dass möglichst viele Komponenten bei einer Portierung wiederverwendet werden können. Neben einer Portierung auf Android ist MONA aber auch als Basis für eine stationäre eID-Applikation geeignet.

Neben der Implementierung bleibt jedoch auch die vom BSI geforderte Zertifizierung eine Herausforderung. Während eine Zertifizierung von MONA nach Common Criteria und dem Protection Profile BSI-PP-0066 [26] möglich erscheint, bleiben die Anforderungen an ein NFC-fähiges Gerät als Kartenleser für den Personalausweis ein schwieriger Sachverhalt. Ein Verzicht auf die Zertifizierung des Zufallszahlengenerators könnte noch damit verargumentiert werden, dass PACE nicht auf dem Kartenleser, sondern innerhalb der eID-Anwendung implementiert ist. Dann greifen nämlich die Anforderungen der TR-02102 [30] für die Seedgenerierung und nicht die der TR-03119. Aus Entwicklersicht sind die weiteren Anforderungen jedoch nicht zu realisieren. Daher müssen die Hersteller der mobilen Geräte in den Prozess involviert werden. Das Interesse internationaler Konzerne an Anforderungen für eine mobile Nutzung des deutschen Personalausweises wird aber eher verhalten sein. Die Anforderungen müssen daher für das mobile Szenario neu analysiert und definiert werden. Ein Augenmerk muss insbesondere auf die technische Realisierbarkeit gelegt werden. Offen bleibt auch, wie man der Vielfalt an unterschiedlichen Geräten und Plattformen gerecht wird.

Leider ist die Landschaft an Betriebssystemen im mobilen Umfeld nicht so homogen wie im stationären Fall. Neben den etablierten Betriebssystemen wie Android, iOS, Windows Mobile und BlackBerry OS existieren noch weitere wie bada, MeeGo und Nokia OS, die oft auf den Mittelklasse- und Einsteigergeräten installiert sind. Neben dem Betriebssystem gibt es darüber hinaus noch eine Vielzahl von mobilen Browsern, die mit unterschiedlichen Funktionen ausgestattet sind. Um das Starten einer mobilen eID-Anwendung, wie bei MONA gezeigt, durch einen Link auf der Website, für eine Vielzahl der Geräte und Plattformen, zu realisieren, muss das Gerät von der Web-Applikation mit Hilfe eines *Device Resolvers* identifiziert werden. Der Device Resolver ist eine Software, die unter anderem in der Lage ist, die Platt-

form und den Browser eines mobilen Endgerätes zu bestimmen und damit die Entscheidung zum Laden der passenden Authentisierungsmethode zu liefern, um die eID-Applikation per Link starten zu können. Sollte es dem Device Resolver gelingen auch das exakte Modell des Gerätes zu identifizieren, kann direkt überprüft werden, ob es sich überhaupt um ein NFC-fähiges Gerät handelt und damit eine Authentisierung mittels Personalausweis überhaupt möglich ist. Einen Device Resolver stellt beispielsweise WURFL (Wireless Universal Resource File) [107] bereit, das auch die Basis für die in Kapitel 17 genannte *Spring Mobile* Erweiterung ist. Wie viele unterschiedliche Konfigurationen nachher bereitgestellt werden müssen, um ein Starten der eID-Anwendung zu ermöglichen, kann zum jetzigen Zeitpunkt nicht abgeschätzt werden. Anzunehmen ist, dass für jedes Betriebssystem eine Konfiguration definiert werden muss. Die Funktionen für die mobile und stationäre Authentisierung, bei der in MONA entwickelten und in Kapitel 17 vorgestellten Web-Applikation, unterscheiden sich jedoch nur in 100 – 200 Zeilen Quellcode. Zusätzlich muss natürlich die eID-Anwendung auch für die jeweilige Plattform zur Verfügung gestellt werden.

Zusammenfassend lässt sich feststellen, dass mit der Entwicklung von MONA ein großer Schritt in Richtung einer mobilen Nutzung des neuen Personalausweises realisiert werden konnte. Spannend sind jetzt die Möglichkeiten und Erkenntnisse durch die neuen Endgeräte und die Realisierung der verbleibenden Anforderungen für den Wirkbetrieb.

Literaturverzeichnis

[1] R. Aarts and J. Kemp. Liberty Reverse HTTP Binding for SOAP Specification. Liberty Alliance Specification, Version 2.0, 2006. http://www.projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf. (31, 59, 60, 61, 98)

[2] Apache Software Foundation. Apache CXF: An Open-Source Services Framework. http://cxf.apache.org. (75, 79)

[3] Apache Software Foundation. Apache log4j. http://logging.apache.org/log4j/. (72)

[4] Apache Software Foundation. Apache MINA. http://mina.apache.org. (46)

[5] J. Bender, M. Fischlin, and D. Kügler. Security Analysis of the PACE Key-Agreement Protocol. In *Information Security Conference (ISC) 2009*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48. Springer, Sep 2009. (73)

[6] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. Request For Comments – RFC 3986, 2005. http://tools.ietf.org/html/rfc3986. (61, 94)

[7] T. Berners-Lee and L. M. M. McCahill. Uniform Resource Locators (URL). Request For Comments – RFC 1738, 1994. http://tools.ietf.org/html/rfc1738. (76, 94)

[8] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508. (30, 98)

[9] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003. ISBN: 3540431071. (13)

[10] U. Breymann and H. Mosemann. *Java ME. Anwendungsentwicklung für Handys, PDA und Co.* Hanser Fachbuch, 2008. ISBN: 3446413766. (27)

[11] Bundesregierung Deutschland. Gesetz über Rahmenbedingungen für elektronische Signaturen - SigG, 2001. http://bundesrecht.juris.de/sigg_2001/. (7)

[12] Codehaus Foundation. Groovy - An agile dynamic language for the Java Platform. http://groovy.codehaus.org. (75, 78)

[13] Common Criteria Maintenance Board (CCMB). Common Criteria for Information Technology Security Evaluation. http://www.commoncriteriaportal.org. (38)

[14] Ö. Dagdelen and M. Fischlin. Security Analysis of the Extended Access Control Protocol for Machine Readable Travel Documents. In *13th Information Security Conference*, Lecture Notes in Computer Science. Springer, Okt 2010. (73)

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. Request For Comments – RFC 4346, April 2006. http://www.ietf.org/rfc/rfc4346.txt. (36, 62)

[16] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. (13, 15)

[17] D. Eastlake and J. Reagle. XML Encryption Syntax and Processing, 2002. http://www.w3.org/TR/xmlenc-core/. (29)

[18] D. Eastlake, J. Reagle, D. Solo, F. David, and T. Roessler. XML Signature Syntax and Processing (Second Edition), 2008. http://www.w3.org/TR/xmldsig-core/. (29, 79)

[19] ECMA International. Standard ECMA-352 – Near Field Communication Interface and Protocol -2 (NFCIP-2), 2003. (24)

[20] ECMA International. Standard ECMA-340 – Near Field Communication Interface and Protocol (NFCIP-1), 2004. (24)

[21] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). Request For Comments – RFC 4279, December 2005. http://www.ietf.org/rfc/rfc4279.txt. (36, 62, 97)

[22] O. Falke, E. Rukzio, U. Dietz, P. Holleis, and A. Schmidt. Mobile Services for Near Field Communication. Technical Report LMU-MI-2007-1, Ludwig-Maximilians-Universität München, March 2007. (25)

[23] Federal Ministry of the Interior (Bundesministerium des Innern). Pressestelle des Bundesministeriums des Innern - Bildmaterial. http://www.personalausweisportal.de/DE/Presse/Bildmaterial/bildmaterial_node.html. (6)

[24] Federal Ministry of the Interior (Bundesministerium des Innern). Einführung des elektronischen Personalausweises in Deutschland Grobkonzept, Version 2.0, 2008. http://www.bmi.bund.de/cae/servlet/contentblob/122648/publicationFile/9169/Grobkonzept_Personalausweis.pdf. (6, 10)

[25] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). AusweisApp. https://www.ausweisapp.bund.de. (90)

[26] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Common Criteria Protection Profile: eID-Client based on eCard-API. BSI-PP-0066. (38, 99)

[27] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). AIS 20 - Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren. Anwendungshinweise und Interpretationen zum Schema (AIS), Version 1, 1999. https://www.bsi.bund.de/cae/servlet/contentblob/478150/publicationFile/30276/ais20pdf.pdf. (39)

[28] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Grundlagen der elektronischen Signatur, 2006. https://www.bsi.bund.de/cae/servlet/contentblob/487196/publicationFile/31102/esig_pdf.pdf. (7)

[29] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Advanced Security Mechanism for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI). Technical Guideline BSI-TR-03110, Version 2.0, 2008. (15)

[30] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Technical Guideline BSI-TR-02102, Version 1.0, 2008. https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr02102/index_htm.html. (97, 99)

[31] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Advanced Security Mechanism for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI). Technical Guideline BSI-TR-03110, Version 2.01, 2009. (97)

[32] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework. Technical Guideline BSI-TR-03112, Version 1.1, Part 1-7, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (32, 38, 97, 98)

[33] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – eCard-Interface. Technical Guideline BSI-TR-03112-2, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29)

[34] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – IFD-Interface. Technical Guideline BSI-TR-03112-6, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29)

[35] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – ISO24727-3-Interface. Technical Guideline BSI-TR-03112-4, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29)

[36] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – Management-Interface. Technical Guideline BSI-TR-03112-3, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29)

[37] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – Overview. Technical Guideline BSI-TR-03112-1, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29, 55)

[38] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – Protocols. Technical Guideline BSI-TR-03112-7, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (51, 74, 77)

[39] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework – Support-Interface. Technical Guideline BSI-TR-03112-5, Version 1.1, 2009. https://www.bsi.bund.de/cln_156/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html. (29)

[40] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Test Plan for Official Electronic ID Documents with Secure Contactless Integrated Circuit. Technical Guideline BSI-TR-03105, Version 2.2, Part 2, 2009. https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr03105/index_htm.html. (86)

[41] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Advanced Security Mechanism for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI). Technical Guideline BSI-TR-03110, Version 2.05, 2010. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/TR-03110_v205_pdf.pdf. (11, 13, 14, 17, 22, 28, 32, 38, 43, 63, 65, 84, 97)

[42] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Advanced Security Mechanism for Machine Readable Travel Documents - Extended Access Control (EAC), Password Authenticated Connection Establishment (PACE), and Restricted Identification (RI). Technical Guideline BSI-TR-03110, Version 2.03, 2010. (97)

[43] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Certificate Policy für die ePass-Anwendung der hoheitlichen Dokumente. Version 1.27, 2010. (7)

[44] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Certificate Policy für die eSign-Anwendung des ePA - Elektronische Signaturen mit dem elektronischen Personalausweis. Version 1.01, 2010. (7)

[45] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Conformity Tests for Official Electronic ID Documents. Technical Guideline BSI-TR-03105, Version 2.5, Part 1-5, 2010. https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr03105/index_htm.html. (38)

[46] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). EAC-Box Architecture and Interfaces. Technical Guideline BSI-TR-03131, Version 1.1, 2010. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03131/BSI_TR03131.pdf. (54)

[47] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Technische Richtlinie eID-Server. Technical Guideline BSI-TR-03130, Version 1.4.1, 2010. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03130/TR-03130_TR-eID-Server_V1_4_pdf.pdf. (32, 39, 74, 77, 98)

[48] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Anforderungen an Chipkartenleser mit nPA Unterstützung. Technical Guideline BSI-TR-03119, Version 1.2, 2011. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03119/BSI-TR-03119_V1_pdf.pdf. (39, 40, 97)

[49] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Architektur elektronischer Personalausweis und elektronischer Aufenthaltstitel. Technical Guideline BSI-TR-03127, Version 1.14, 2011. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03127/BSI-TR-03127_pdf.pdf. (39)

[50] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). Certificate Policy für die eID-Anwendung des ePA - Elektronischer Identitätsnachweis mit dem elektronischen Personalausweis. Version 1.27, 2011. (6, 38, 98)

[51] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request For Comments – RFC 2616, 1999. http://www.ietf.org/rfc/rfc2616.txt. (30, 61, 94)

[52] E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN: 0201633612. (71)

[53] Gartner. Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012. http://www.gartner.com/it/page.jsp?id=1622614. (92)

[54] E. Haselsteiner and K. Breitfuß. Security in Near Field Communication (NFC), 2006. http://events.iaik.tugraz.at/RFIDSec06/Program/papers/002%20-%20Security%20in%20NFC.pdf. (25)

[55] S. Haustein. kXML 2, 2005. http://kxml.sourceforge.net/kxml2/. (57, 94)

[56] S. Haustein and A. Slominski. XML Pull API, 2005. http://www.xmlpull.org. (94)

[57] C. Hornig. A Standard for the Transmission of IP Datagrams over Ethernet Networks. Request For Comments – RFC 894, April 1984. http://tools.ietf.org/rfc/rfc894.txt. (62, 88)

[58] M. Horsch. MobilePACE - Password Authenticated Connection Establishment implementation on mobile devices. Bachelor Thesis, TU Darmstadt, September 2009. http://www.cdc.informatik.tu-darmstadt.de/reports/reports/Moritz_Horsch.bachelor.pdf. (51, 63, 65)

[59] D. Hühnlein, M. Bach, and R. Oberweis. Das eCard-API-Framework. *Tagungsband zum 10. Deutschen IT-Sicherheitskongress des BSI*, 2007. http://www.ecsec.de/pub/2007_BSI.pdf. (28)

[60] T. Icart. How to Hash into Elliptic Curves. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*, pages 303–316. Springer, 2009. (16)

[61] Information Sciences Institute, University of Southern California. Transmission Control Protocol. Request For Comments – RFC 793, September 1981. http://www.ietf.org/rfc/rfc793.txt. (36)

[62] INSIDE Secure. Open NFC. http://www.opennfc.org. (49)

[63] International Civil Aviation Organization (ICAO). Machine Readable Travel Documents. ICAO Doc 9303, Part 1-3, 2006 - 2008. http://www2.icao.int/en/MRTD/Pages/Doc9393.aspx. (7, 11, 17, 19)

[64] International Civil Aviation Organization (ICAO). Machine Readable Travel Documents - Part 3: Machine Readable Official Travel Documents, Specifications for electronically enabled official travel documents with biometric identification capabilities. ICAO Doc 9303-3, 2008. (6)

[65] International Civil Aviation Organization (ICAO). Supplemental Access Control for Machine Readable Travel Documents. Technical Report, Version 1.01, 2010. (13)

[66] Internet Assigned Numbers Authority (IANA). http://www.iana.org. (94)

[67] ISO/IEC. Information processing systems – Open Systems Interconnection – Basic Reference Model, Part 1-4. International Standard, ISO/IEC 7498, 1989 - 1997. (30)

[68] ISO/IEC. Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1). International Standard, ISO/IEC 18092, 2004. (24)

[69] ISO/IEC. Identification cards – Integrated circuit cards, Part 4: Interindustry commands for interchange. International Standard, ISO/IEC 7816-4, 2005. (22, 43)

[70] ISO/IEC. Information technology – Telecommunications and information exchange between systems – Near Field Communication Interface and Protocol -2 (NFCIP-2). International Standard, ISO/IEC 21481, 2005. (24)

[71] ISO/IEC. Identification cards – Contactless integrated circuit(s) cards – Vicinity cards. International Standard, ISO/IEC 15693, Part 1-3, 2006 - 2010. (24)

[72] ISO/IEC. Identification cards – Integrated circuit card programming interfaces – Part 3: Application interface. International Standard, ISO/IEC 24727-3, 2008. (29)

[73] ISO/IEC. Identification cards – Contactless integrated circuit cards – Proximity cards. International Standard, ISO/IEC 14443, Part 1-4, 2008 - 2011. (8, 24)

[74] ISO/IEC. Identification cards – Contactless integrated circuit cards – Proximity cards – Part 2: Radio frequency power and signal interface. International Standard, ISO/IEC 14443-2, 2010. (86)

[75] J. Karlsson. Microlog, 2010. http://sourceforge.net/projects/microlog/. (72)

[76] F. Kiefer. Effiziente Implementierung des PACE- und EAC-Protokolls für mobile Geräte. Bachelor Thesis, TU Darmstadt, July 2010. (63)

[77] J. B. Knudsen. *Wireless Java: Developing with J2ME*. Aspress, 2003. ISBN: 1590590775. (27)

[78] J. Krhovjak, V. Matyas, and J. Zizkovsky. Generating Random and Pseudorandom Sequences in Mobile Devices. In *Security and Privacy in Mobile Information and Communication Systems*, volume 17, pages 122–133. Springer Berlin Heidelberg, 2009. (97)

[79] E. Lafortune. ProGuard. http://proguard.sourceforge.net. (73)

[80] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. Request For Comments – RFC 4122, July 2005. http://tools.ietf.org/rfc/rfc4122.txt. (59, 60)

[81] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger. NFC Devices: Security and Privacy. In *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 642–647, Washington, DC, USA, 2008. IEEE Computer Society. (25)

[82] J. H. Mahmoud, M. Naghshineh, J. Inouye, O. J. Joeressen, and W. Allen. Bluetooth: Vision, goals, and architecture. *ACM Mobile Computing and Communications Review*, 2:38–45, 1998. (24)

[83] media transfer AG. mtG-eID. http://www.mtg.de. (98)

[84] J. Mogul and S. Deering. Path MTU Discovery. Request For Comments – RFC 1191, November 1990. http://www.ietf.org/rfc/rfc1191.txt. (62, 88)

[85] C. Mulliner. Vulnerability Analysis and Attacks on NFC-enabled Mobile Phones, 2009. http://mulliner.org/collin/academic/publications/vulnanalysisattacksnfcmobilephones_mulliner_2009.pdf. (25)

[86] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Services Security: SOAP Message Security 1.1, 2006. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. (79)

[87] Nokia Corporation. Nokia 6212 classic. http://europe.nokia.com/support/product-support/nokia-6212-classic. (41)

[88] Nokia Corporation. DRM Developer's Guide for Nokia Devices, Version 3.0, 2006. http://www.forum.nokia.com/info/sw.nokia.com/id/f269e8db-bb9b-4de8-9009-bb9c2ec810f2/DRM_Developers_Guide_for_Nokia_Devices_v3_0_en.pdf.html. (73)

[89] Open Mobile Alliance. OMA Digital Rights Management V2.0, 2008. http://www.openmobilealliance.org/Technical/release_program/drm_v2_0.aspx. (73)

[90] Oracle Corporation. Java Applets. http://www.oracle.com/technetwork/java/applets-137637.html. (26)

[91] Oracle Corporation. Java Web Start Technology. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html. (26)

[92] Oracle Corporation. VirtualBox. http://www.virtualbox.org. (41)

[93] Oracle Corporation. JSR 30: J2ME Connected, Limited Device Configuration. http://jcp.org/en/jsr/detail?id=30, 2000. (26)

[94] Oracle Corporation. JSR 37: Mobile Information Device Profile for the J2ME Platform. http://jcp.org/en/jsr/detail?id=37, 2000. (26)

[95] Oracle Corporation. JSR 63: Java API for XML Processing. http://jcp.org/en/jsr/detail?id=63, 2002. (58)

[96] Oracle Corporation. JSR 172: J2ME Web Services Specification. http://jcp.org/en/jsr/detail?id=172, 2004. (58)

[97] Oracle Corporation. JSR 36: Connected Device Configuration. http://jcp.org/en/jsr/detail?id=36, 2005. (26)

[98] Oracle Corporation. JSR 118: Mobile Information Device Profile 2.0. http://jcp.org/en/jsr/detail?id=118, 2006. (26, 68, 74, 93)

[99] Oracle Corporation. JSR 218: Connected Device Configuration (CDC) 1.1. http://jcp.org/en/jsr/detail?id=218, 2006. (26)

[100] Oracle Corporation. JSR 268: Java Smart Card I/O API. http://jcp.org/en/jsr/detail?id=268, 2006. (44, 49, 50)

[101] Oracle Corporation. JSR 139: Connected Limited Device Configuration 1.1. http://jcp.org/en/jsr/detail?id=139, 2007. (26)

[102] Oracle Corporation. JSR 257: Contactless Communication API. http://jcp.org/en/jsr/detail?id=257, 2009. (24, 41, 49, 84)

[103] Oracle Corporation. JSR 56: Java Network Launching Protocol and API. http://jcp.org/en/jsr/detail?id=56, 2009. (27)

[104] Oracle Corporation. Light Weight User Interface Toolkit (LWUIT), 2010. http://www.oracle.com/technetwork/java/javame/tech/lwuit-141954.html. (68)

[105] Oracle Corporation. Light Weight User Interface Toolkit (LWUIT) - Developer's Guide, 2010. http://download.oracle.com/javame/dev-tools/lwuit-1.4/LWUIT_Developer_Guide.pdf. (69, 70)

[106] Oracle Corporation. Java API for XML Web Services (JAX-WS). http://jax-ws.java.net, 2011. (79)

[107] L. Passani. WURFL - Wireless Universal Resource File. http://wurfl.sourceforge.net. (100)

[108] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. Request For Comments – RFC 2988, 2000. http://tools.ietf.org/html/rfc2988. (87)

[109] C. Petit, F.-X. Standaert, O. Pereira, T. Malkin, and M. Yung. A Block Cipher based PRNG Secure Against Side-Channel Key Recovery. In *Proceedings of ASIACCS*. ACM, 2008. (16)

[110] J. Postel and J. Reynolds. A Standard for the Transmission of IP Datagrams over IEEE 802 Networks. Request For Comments – RFC 1042, February 1988. http://tools.ietf.org/rfc/rfc1042.txt. (62, 88)

[111] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, February 1978. (36)

[112] G. Rocher and J. Brown. *The Definitive Guide to Grails - Second Edition*. Apress, 2009. ISBN: 9781590599952. (75)

[113] J. Rosenberg and D. Remy. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Sams Publishing, 2004. ISBN: 978-0672326516. (79)

[114] Sagem. MorphoMapping. Patent FR09-50189, 2009. (16)

[115] K.-D. Schmatz. *Java 2 Micro Edition. Entwicklung mobiler Anwendungen mit CLDC und MIDP.* Dpunkt Verlag, 2004. ISBN: 3898642712. (27)

[116] SCM Microsystems. Dual interface reader SDI 010. http://www.scmmicro.com/security/view_product_en.php?PID=19. (41)

[117] SpringSource. Grails Web Application Framework. http://www.grails.org. (75)

[118] SpringSource. Spring Mobile. http://www.springsource.org/spring-mobile. (79)

[119] W. Stallings. *Network Security Essentials*. Prentice Hall International, 2010. ISBN: 9780137067923. (36)

[120] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994. ISBN: 0-201-63346-9. (88)

[121] Technische Universität Darmstadt. CoDec ASN.1 En-/Decoder Library. http://sourceforge.net/projects/codec/. (65)

[122] Technische Universität Darmstadt. FlexiProvider. http://www.flexiprovider.de. (63, 93)

[123] E. Tews. Entwicklung von MicroTLS als sichere Ende-zu-Ende-Verbindung über Bluetooth und TCP/IP, 2006. (62)

[124] The Legion Of The Bouncy Castle. Bouncy Castle Crypto API. http://www.bouncycastle.org. (62)

[125] J. Vinet and A. Griffin. Arch Linux. https://www.archlinux.org. (41)

[126] A. Wiesmaier, M. Horsch, J. Braun, F. Kiefer, D. Hühnlein, F. Strenzke, and J. Buchmann. An efficient PACE Implementation for mobile Devices. In *ASIA CCS '11: 6th ACM Symposium on Information, Computer and Communications Security*, March 2011. (82, 95)

[127] World Wide Web Consortium (W3C). HTML 4.01 Specification, 1999. http://www.w3.org/TR/html401/. (74)

[128] World Wide Web Consortium (W3C). Web Services Addressing. W3C Member Submission, 2004. http://www.w3.org/Submission/ws-addressing/. (60)

[129] World Wide Web Consortium (W3C). XHTML 1.1 - Module-based XHTML - Second Edition, 2010. http://www.w3.org/TR/xhtml11/. (74)