UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

rosecat

Architektur und Implementierung eines Open-Source-eID-Clients

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von

Nico Jahn

Erstgutachter:	Prof. Dr. Rüdiger Grimm Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Dipl.-Inform. Andreas Kasten Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im Juli 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

Zu Beginn dieser Arbeit konnte ich aus einem großen Vorrat an Motivation schöpfen. Dies lag in der Tatsache etwas Neues zu schaffen begründet. Mit jedem Tag jedoch stieg der Umfang an Informationen derart an, dass der Aufwand der noch zu erbringenden Tätigkeiten immer schwerer abschätzbar wurde. Bei der Festlegung der Schwerpunkte habe ich allerdings seitens meiner Betreuer Prof. Dr. Rüdiger Grimm und Andreas Kasten die, für mich empfundene, größtmögliche Freiheit genossen. So war es mir und anderen Studenten, die an verwandten Themen arbeiteten, möglich kooperativ zu arbeiten und uns gegenseitig zu unterstützen. Ich möchte mich besonders bei Jonas Zitz, Andreas Wolf, Eugen Müller und Christian Merz für lange Diskussionen und gute Zusammenarbeit bedanken. Wir erhielten dabei stets Unterstützung von Sven Vowé vom Fraunhofer-Institut für Sichere Informationstechnologie in Darmstadt, der im „Kompetenzzentrum neuer Personalausweis“ tätig war. Durch ihn lernte ich außerdem Christian Kahlo kennen, der beruflich an einem eID-Client arbeitet und mit dem ich bis heute Informationen austausche. Der besondere Dank aber geht an meine Familie und vor allem an meine Eltern, die mich bei der freien Gestaltung meines bisherigen Weges in jeder Hinsicht unterstützt und mir mein Studium ermöglicht haben. Ohne sie wäre mein bisheriger Weg in seiner Form nicht möglich gewesen.

Kapitel 1

Chancen des neuen Personalausweises

Seit dem 01.11.2010 erhalten Bürger der Bundesrepublik Deutschland den neuen elektronischen Personalausweis in Größe einer Checkkarte. Dieser enthält einen Radio-Frequency Identification (RFID)-Chip, der die persönlichen Daten des Bürgers speichern und übertragen kann. Auf dem Chip befindet sich unter anderem die eID-Funktion, mit der sich ein Bürger online Ausweisen kann, um gegenüber Behörden oder Geschäftspartnern einen Identitätsnachweis zu erbringen. Dazu benötigt er einen mit dem Internet verbunden Computer, ein RFID-Lesegerät, seinen Personalausweis und die geheime PIN für die eID-Funktion. Der Chip entscheidet während dem Ausweisverfahren selbst, ob er angefragte Daten überträgt oder nicht. Der Zugriff durch Unbefugte wird durch kryptografische und organisatorische Maßnahmen verhindert. Ein Teil dieser Maßnahmen wird durch eine Software realisiert, die auf dem Computer des Bürgers ausgeführt wird und die unmittelbare Kommunikation mit dem Chip ermöglicht.

Die Bundesregierung stellt die Software „AusweissApp“ [1] zur Verfügung. Mit dieser Software können die Anwendungen des neuen Personalausweises genutzt werden. Hierzu zählen neben dem eID-Verfahren für die Online-Authentifikation auch die Möglichkeiten zur Erstellung einer elektronischen Signatur und der Konfiguration des Personalausweises, um beispielsweise seine geheime Persönliche

[1] http://www.ausweis-app.com

Identifikationnummer (PIN) zu ändern.

Abgesehen von einer Zertifizierung durch das Bundesministerium für Sicherheit in der Informationstechnik (BSI), findet die Entwicklung dieser Software unter Verschluss statt. Die Öffentlichkeit erhält keinen Einblick in den Quellcode und somit keine Gewissheit darüber was die Software tatsächlich tut. Bereits in der Vergangenheit wurden Pläne der Regierung, welche die Privatsphäre der Bürger gefährden können in der Öffentlichkeit kritisch diskutiert. Hierzu zählen Pläne zur Vorratsdatenspeicherung und dem Ausspähen von Informationen am heimischen Computer. Die Einführung einer Software, deren Inhalt nicht bekannt ist, kann die Akzeptanz der Bevölkerung mindern und eine erfolgreiche Einführung gefährden.

Zahlreichen Berichte im Fernsehen und Zeitschriften lässt sich entnehmen, dass diese Akzeptanz nicht gegeben ist. Als aktivster Gegner der AusweisApp hat sich der Chaos Computer Club präsentiert, der bereits vor der Einführung des neuen Personalausweises auf Risiken bei der Benutzung der eID-Funktion hinwies. Mitglieder zeigten, wie mit Hilfe von Schadsoftware die geheime PIN auf ungeschützten Computern in Erfahrung gebracht werden könnte.[2] Andere Kritiker sprechen direkt die Angst vor der totalen Überwachung aus.[3] Schließlich wurde in weniger als 24 Stunden nach der Einführung des neuen Personalausweises die erste Sicherheitslücke innerhalb der AusweisApp nachgewiesen [4]. Dies hatte zur Folge, dass sie nicht mehr als Download zur Verfügung gestellt wurde und und erst zwei Monate später als nachgebesserte Version zur Verfügung stand. [5]

Die Entwicklung der AusweisApp als Open-Source-Software hätte von Beginn an die Alternative zur gewählten Vorgehensweise sein können. Möglicherweise hätten Gründe für Zweifel an der Funktionsweise wesentlich früher von allen Interessierten untersucht und diskutiert und die gefundene Sicherheitslücke früher entdeckt werden können, um sie noch vor der Veröffentlichung zu beheben. Vor allem die gewonnene Transparenz durch Offenlegung von Code trägt in besonde-

[2] http://www.ccc.de/de/updates/2010/sicherheitsprobleme-bei-suisseid-und-epa
[3] http://www.tutsi.de/online-personalausweis-so-gefaehrlich-nutzlos-ist-der-neue-personalausweis/2011/02/06/tutsi-blog-aktuell
[4] http://janschejbal.wordpress.com/2010/11/09/ausweisapp-gehackt-malware-uber-autoupdate
[5] http://computer.t-online.de/personalausweis-nachgebesserte-ausweisapp-ist-da/id_43907924/index

rem Maße zur Akzeptanzsteigerung bei. Gerade als Entwickler kann man sich die Frage stellen, welchen Funktionsumfang AusweisApp mit über 150 MB Installationsvolumen enthält. Andere fertige Entwicklungen haben gezeigt, dass sich ein funktionsfähiger eID-Client mit weitaus Speicherbedarf realisieren lässt. Hierzu zählen die Java-Applet-basiert Version „Governirkus Autent“ von Bremen Online Service mit etwa 10 MB [6] und die sehr kleine Variante von Synchronity [7].

Diese Diplomarbeit zeigt die Entwicklung einer Softwarearchitektur zur Realisierung eines eID-Clients, die ein vom BSI entworfenes Framework zur Kommunikation mit Chipkarten benutzt. Dieses Framework ist in mehrere technische Richtlinien gegliedert, in denen Funktionen und Datenstrukturen beschrieben sind, die einen Standard zur Kommunikation zwischen Chipkarten-basierten Anwendungen darstellen. Vorbereitend dafür fanden bereits mehrere studentische Projektarbeiten zum Thema statt, deren Ergebnisse aufgearbeitet und weiter entwickelt werden sollen. Um dies zu ermöglichen, werden zunächst voneinander unabhängige Bereiche bezüglich der Entwicklung identifiziert. Hierzu zählen die Kommunikation mit dem Personalausweis, die Durchführung kryptografischer Protokolle, Realisierung von Webservicefunktionen, welche zur Kommunikation über ein Netz während eID benutzt werden, dem Entwurf der Benutzeroberfläche sowie deren interne Interaktion miteinander. Bei der Benutzung des eID-Clients soll die Transparenz der Vorgänge für den Anwender jederzeit gewährt sein. Der Anwender soll dabei selbst bestimmen können, wie detailliert die gebotenen Informationen sind. Für den „normalen“ Gebrauch soll zumindest jederzeit erkenntlich sein, mit welchen entfernten Systemen der eID-Client kommuniziert und welche Interaktionen mit dem Personalausweis durchgeführt werden. Ein erweiterter Überwachungsmechanismus soll außerdem spezielle Daten liefern, die beispielsweise für technisch versierte Anwender von Interesse sind. Eine gesonderte Anforderung an die zu entwickelnde Software ist ein Demonstrationsmodus. Die Software soll insbesondere für Lehr- und Demonstrationszwecke eingesetzt werden. Durch die Visualisierung kryptografischer Parameter können die Abläufe bei der Kommunikation mit dem Personalausweis transparent abgebildet werden. Hierbei sind verschiedene Anwendungsszenarien zu berücksichtigen, da der Personalausweis und auch ein entfernter Server als beteiligte Kom-

[6] http://www.bos-bremen.de/de/governikus_autent/1854605

[7] http://www.synchronity.de/de/news/news/aktuelles/90-erfolgreicher-start-fuer-die-ausweisapp-auf-der-cebit

ponenten, die Transparenz implizit einschränken. Mithilfe eines speziellen Simulationsmodus, der den kompletten eID-Ablauf abbildet, kann ein vollständiger Einblick ermöglicht werden.

Kapitel 2

Grundlagen

2.1 Vereinbarungen zur Terminologie

Der offizielle Name des neuen elektronischen Personalausweises lautet „neuer Personalausweis“. In dieser Arbeit wird ausschließlich die Bezeichnung „elektronischer Personalausweis“ verwendet.

Einige Begriffe werden je nach Abstraktionsebene sysnonym verwendet. Bei konzeptionellen Betrachtungen stehen sich meist der „Personalausweis“ bzw. der „Bürger“ und der „Dienstanbieter“ gegenüber. Bei technischen Betrachtungen stehen sich meist der „Chip“ und das „Terminal“ gegenüber. Bei der Verwendung des Wortes „Chip“ ist der Chip im elektronischen Personalausweis gemeint, ein „Terminal“ ist der Kommunikationspartner des Chips. Das Terminal kann sich lokal befinden, zB. in Form eines eID-Clients, dann wird es als „Local-Terminal“ bezeichnet oder es handel sich um ein entferntes terminal, z.B. in Form eines eID-Servers, dann wird es als „Remote-Terminal“ bezeichnet.

2.2 Vorangegangene Projekte und Arbeiten

In der Vergangenheit haben mehrere studentische Projekte zum Thema im Rahmen von Projektpraktika statt gefunden.

Zunächst wurden in einem ersten Projektpraktikum [Bol08] die grundlegenden

Funktionen und Protokolle der Anwendungen des elektronischen Personalausweises untersucht. Hierbei waren insbesondere die Abläufe der kryptografischen Protokolle von Interesse. Diese wurden durch eine Software in der Programmiersprache C-Sharp visualisiert, welche die drei beteiligten Komponenten elektronischer Personalausweis, Local Terminal und Remote Terminal simulierte.

Ein weiteres Projektpraktikum baute auf diese Erkenntnisse auf und portierte die Funktionalität in die Programmiersprache Java. Hierbei wurde die Kommunikation zwischen den beiden Terminals erstmals durch Webservices umgesetzt.

Im dritten Projektpraktikum [Eck10] wurden erstmals Bestandteile der Technischen Richtlinien des BSI zum eCard-API-Framework [BSI09a] in Java umgesetzt. Auf diese Richtlinien wird in Abschnitt 3.2 genauer eingegangen. Nach Abschluss des Projekts existierte eine Grundlage an Quellcode, der die Kommunikation mit dem elektronischen Personalausweis ermöglicht und einen gesicherten Kommunikationskanal zu ihm aufbaut. Die kryptografischen Abläufe wurden in einem speziell für den Anwendungszeck entwickelten CryptoProvider gekapselt.

2.3 Grundlagen des neuen Personalausweises

2.3.1 Kartenapplikationen

Der Chip im elektronischen Personalausweis beherrscht Funktionen, die durch das Übertragen von Befehlen ausführbar sind. Der Funktionsumfang kann durch Kartenapplikationen, die auf dem Chip installiert werden, erweitert werden. Das BSI beschreibt in der technischen Richtlinie TR-03110 [BSI10b] drei Kartenapplikationen für maschinenlesbare Reisedokumente, wie dem elektronischen Reisepass und dem elektronischen Personalausweis. Dies sind die ePassPort-Applikation, eID-Applikation und die eSign-Applikation.

ePassport-Applikation Die ePassport-Applikation ermöglicht das Auslesen der weniger sensiblen Daten, die bereits auf dem elektronischen Personalausweis oder Reisepass aufgedruckt sind. Diese Daten können mit einer entsprechenden Software und ohne spezielle Berechtigung ausgelesen werden. Hoheitliche Inspektionssysteme, wie sie von Behörden, Polizei und Bundesgrenzschutz

verwendet werden, ermöglichen außerdem Zugriff auf weitere Daten, um beispielsweise einen Abgleich der Fingerabdrücke durchzuführen.

eID-Applikation Der elektronische Personalausweis ermöglicht dem Ausweisinhaber sich elektronisch, also auch online auszuweisen. Um die eID-Applikation durchzuführen, muss der Ausweisinhaber eine geheime PIN eingeben, die sogenannte eID-PIN eingeben. Die Kombination von Besitz (Personalausweis) und Wissen (eID-PIN) soll sicherstellen, dass ausschließlich der tatsächliche Ausweisinhaber eID durchführen kann. Der eletronische Identitätsnachweis ist seit dem 01.10.2010 im „Art. 18, Abs. Elektronischer Identitäsnachweis, Personalausweisgesetz“[1] anerkannt.

eSign-Applikation Die eSign-Funktion ermöglicht das Signieren von digitalen Dokumenten mit einer qualifizierten elektronischen Signatur, die den Anforderungen von „Art. 2, Abs. Begriffsbestimmung, Signaturgesetz“[2] entspricht. Der Ausweisinhaber benötigt dazu einen sogenannten Komfort-Kartenleser, der den Zugriff auf die sichere Signaturerstellungseinheit des elektronischen Personalausweises ermöglicht.

2.3.2 Kartenlesegeräte

Bezüglich der Verwendung mit dem elektronischen Personalausweis werden Kartenlesegeräte in drei Kategorien mit unterschiedlichen Funktionsumfang unterteilt. Der einfache Basisleser, der Standardleser mit Tastatur und der Komfortleser zur Erstellung von Signaturen.

Basisleser Ein Basislesegerät ist die einfachste Form eines Lesegerätes. Ein solches Lesegerät bietet lediglich die Funktion einen ungesicherten Kommunikationskanal zum elektronischen Personalausweis aufzubauen sowie Kommandos an den Chip zu senden und die Antworten zu empfangen. Die Absicherung des Kanals muss bei Verwendung dieser Lesegeräte durch die auf dem Computer laufende Software erfolgen, was eine Eingabe der geheimen PIN am Computer vor-

[1] http://www.gesetze-im-internet.de/pauswg/BJNR134610009.html
[2] http://bundesrecht.juris.de/sigg_2001/index.html

aussetzt. Die Verwendung eines Lesegerätes diesen Typs muss kritisch betrachtet werden, da durch Schadsoftware potentiell die Möglichkeit besteht, die PIN in Erfahrung zu bringen.

Standardleser Im Gegesatz zu einem Basisleser, etabliert ein Standardleser selbst einen gesicherten Kommunikationkanal zum elektronischen Personalausweis. Die geheime PIN wird über eine Tastatur eingegeben, die sich auf dem Lesegerät selbst befindet. So kann verhindert werden, dass durch Schadsoftware auf dem Computer diese in Erfahrung gebracht werden kann.

Komfortleser Zusätzlich zum Funktionsumfang eines Standardleser enthält ein Komfortleser eine sichere Signaturerstellungseinheit, wie sie im Signaturgesetz gefordert ist. Daher sind Komfortleser für die Verwendung der eSign-Applikation geeignet.

Das BSI hat Anforderungen an Kartenlesegeräte in der technischen Richtlinie TR-03119 [BSIa] verfasst, nach denen sie zertifiziert werden können. Eine Auflistung der bereits zertifizierten Lesegeräte ist vom BSI online verfügbar. [3]. Zum Zeitpunkt der Erstellung dieser Arbeit enthielt diese Liste lediglich einen zertifizierten Standardleser und einen zertifizierten Komfortleser.

2.3.3 Ablauf von eID

Bei dem Authentifizierungsverfahren mittels eID kommunizieren vier Parteien miteinander. Abbildung 2.1 verdeutlich die einzelnen Schritte des gesamten Verfahrens. Dem Ablauf liegt ein Szenario zugrunde, in dem ein Benutzer einen Dienst einer Webseite in Anspruch nehmen möchte. Bevor der Dienst erfüllt werden kann, muss der Benutzer sich jedoch mittels eID ausweisen.

1. Ein Benutzer möchte den Dienst einer Webseite in Anspruch und ruft dazu eine entsprechende Webseite auf.

[3] https://www.bsi.bund.de/cln_165/sid_0F9C58D2212B77E9DF944B2C45EBA29F/DE/Themen/ZertifizierungundAnerkennung/ZertifizierungnachTR/ Zertifizierte-Produkte/zertifizierteprodukte_node.html

2. Der Dienstanbieter verlangt als Voraussetzung zur Diensterfüllung ein Authentifzierung durch eID und fordert bei einem eID-Server eine Authentifizierungsabwicklung an.

3. Der eID-Server antwortet darauf mit den Verbindungsdaten für das eID-Verfahren. Diese Verbindungsdaten enthalten eine Session-ID, einen Pre-Shared-Key und die Adresse des eID-Servers, die der eID-Client für den Verbindungsaufbau benutzen kann.

4. Der Webserver des Dienstanbieters antwortet nun auf die Anfrage aus 1) und überträgt die Verbindungsdaten für das eID-Verfahren, die von einem Browser-Plugin ausgelesen werden.

5. Nun werden diese Daten auf der Nutzerseite an den eID-Client übermittelt werden.

6. Der eID-Client stellt jetzt selbst eine Verbindung zum eID-Server her, die basierend auf den Verbindungsdaten mit Transport Layer Security (TLS)

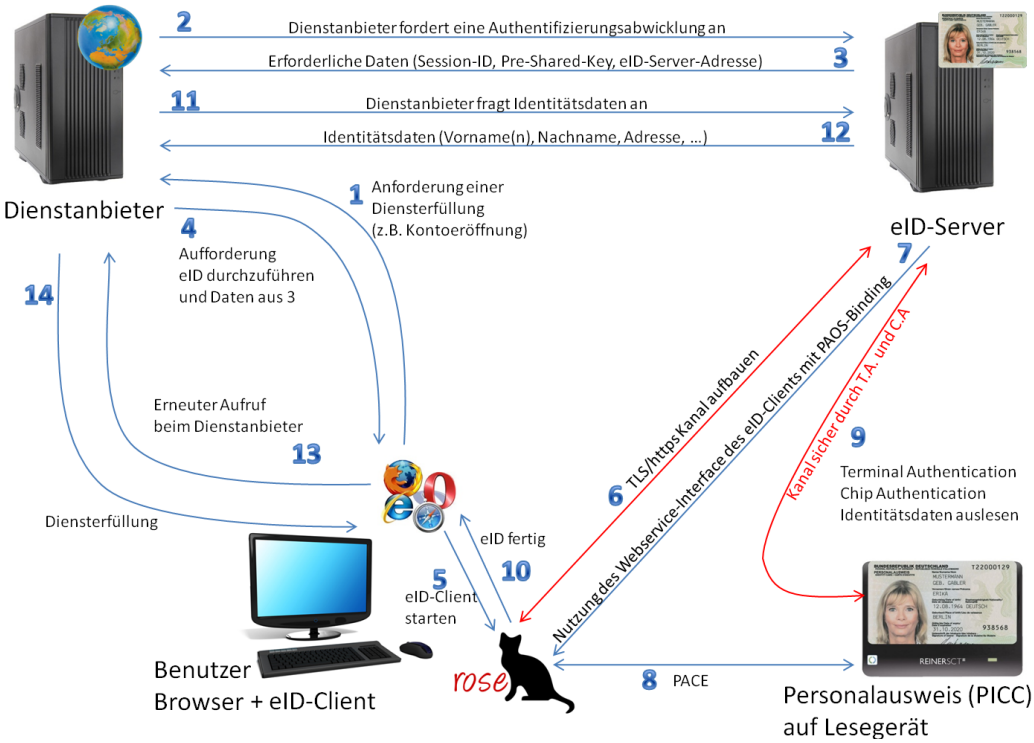

Abbildung 2.1: eID-Ablauf im Gesamtbild

abgesichert ist. Durch die Session-ID und den Pre-Shared-Key kann der eID-Server diese Verbindung der Authentifikationsanfrage zuordnen.

7. Der eID-Server stellt nun eine logische Verbindung zum eID-Client her. Dazu nutzt er die Webservice-Schnittstelle des eID-Clients, um die für eID notwendigen Befehle zu senden.

8. Um die Luftschnittstelle zwischen RFID-Lesegerät und Personalausweis abzusichern wird nun das Password Authenticated Connection Establishment (PACE)-Protokoll durchgeführt. Standard- und Komfortleser führen diesen Schritt selbstständig durch. Falls solch ein Lesegerät nicht verwendet wird, muss der eID-Client PACE durchführen.

9. Der eID-Server und der Personalausweis authentifizieren sich nun gegenseitig und etablieren einen gesicherten Kommunikationskanal. Danach sendet liest der eID-Server die gewünschten Informationen, z.B. Name oder Adresse, aus.

10. Nachdem der eID-Client die Verbindung mit dem eID-Server beendet hat, benachrichtigt er das Browser-Plugin.

11. Dienstanbieter fragt nun die Identitätsdaten des Benutzers vom eID-Server anfordern. Diese Anfrage wurde während des gesamten Ablaufs bereits öfter gestellt, allerdings antwortet der eID-Server erst mit einer positiven Nachricht, sobald er die Daten aus dem Personalausweis ausgelesen hat.

12. Der eID-Server sendet die Identitätsdaten an den Dienstanbieter

13. Das Browser-Plugin ruft erneut die Seite des Dienstanbieters auf, sobald es vom eID-Client in 10) die Durchführung von eID gemeldet bekommen hat.

14. Sind die Daten nun in gewünschten Umfang beim Dienstanbieter vorhanden, kann dieser den Dienst nun erfüllen.

2.4 Sicherheitsanforderungen

Beim Einsatz von elektronischen Ausweisdokumenten ist sicherzustellen, dass nur berechtigte Instanzen Kenntnis über die persönlichen Daten des Bürgers erhalten und der Empfänger diese Daten als authentisch ansehen kann. Es wird

gezeigt, dass durch die Sicherheitsanforderungen Vertraulichkeit und Authentizität diese Ziele erreicht werden.

Vertraulichkeit

> *Ein System gewährleistet Vetraulichkeit, wenn die in ihm enthaltenen Informationen nur berechtigten Subjekten zur Kenntnis gelangen. Eine Information ist dann vertraulich, wenn sie nur berechtigten Subjekten zur Kenntnis gelangt ist.*[Ama92]

In dieser Definition der Vertraulichkeit wird sowohl die Vertraulichkeit eines Systems, als auch die Vertraulichkeit einer Information explizit erwähnt. Bezogen auf die Nutzung des elektronischen Personalausweises besteht das System aus den drei Beteiligten Komponenten Personalausweis, eID-Client und eID-Server. Die Vertraulichkeit des Systems wäre verletzt, sobald der Personalausweis Daten frei gibt, ohne die Authentisierung, also die Berechtigung des Auslesenden, für diesen Vorgang festzustellen. Die Vertraulichkeit der Information wäre verletzt, falls die Kommunikation zwischen einem elektronischen Personalausweises und einem Auslesenden abgehört und der Inhalt interpretiert werden kann.

Authentizität

> *Unter der Authentizität eines Objekts bzw. Subjekts (engl. authenticity) verstehen wir die Echtheit und Glaubwürdigkeit des Objekt bzw. Subjekt, die anhand einer eindeutigen Identität und charakteristischen Eigenschaften überprüfbar ist.*[Eck06]

Bevor der elektronische Personalausweis persönliche Daten übermittelt, muss er die Authentizität des Auslesenden feststellen. Um dies zu erreichen werden, je nach Anwendungsfall eine Reihe von kryptografischen Protokollen eingesetzt. Wer als authentisch anerkannt wird und wie umfangreich diese Prüfung ist, hängt von jeweiligen Anwendungsfall ab. Ebenfalls muss der Auslesende die Authentizität des Personalausweises und der ausgelesenen Daten prüfen können.

Sollen mit der ePassport-Applikation maximal die auf dem Personalausweis aufgedruckten Daten ausgelesen werden, ist die unmittelbare Nähe zwischen dem

Personalausweis und dem Auslesenden hinreichend. Durch die erfordernliche Eingabe von Daten, die sich ebenfalls auf der Oberfläche des elektronischen Personalausweises befinden und so nur jemanden aus der unmittelbaren Nähe bekannt sind, wird diese Nähe sicher gestellt. Nutzbare Daten dafür sind die Machine Readable Zone (MRZ), eine zeichenfolge die gescannt werden kann und die Card Access Number (CAN), eine Nummer die von Hand eingegeben werden kann. Die eID-Funktion hingegen verlangt eine Authentifikation, die auf starken kryptografischen Verfahren und einer Public Key Infrastruktur (PKI) basiert.

Nach [Gri07] wird Authentizität durch die Kombination von Integrität und Originalität erreicht.

Integrität

> *Integrität ist die Eigenschaft eines Systems, die besagt, dass es nur erlaubte und intendierte Veränderungen der in ihm enthaltenden Informationen zulässt. Eine Information ist integer, wenn an ihr nur zulässige Veränderungen vorgenommen wurden.*[Ama92]

Die Daten von hoheitlichen Ausweisdokumenten dürfen lediglich von hoheitlicher Instanz erstellt und manipuliert werden. Ein elektronischer Personalausweis lässt Änderungen an seinen gespeicherten Daten nur zu, falls ihm ein entsprechendes Berechtigungszertifikat mit diesen Berechtigungen vorliegt. Mit solch einem Berechtigungszertifikat kann sich ein Terminal als sogenanntes hoheitliches Terminal authentifizieren. Dadurch wird die Integrität des Personalausweises sicher gestellt. Die Informationen im elektronischen Personalausweis sind mit dem privaten Schlüssel einer Country Signer Certification Authority (CSCA) signiert und die Signatur ist ebenfalls im Personalausweis gespeichert. Solch eine CSCA wird vom Staat betrieben, die elektronische Ausweisdokumente herstellt und deren Inhalte signiert. Diese Signatur kann ausgelesen und mit Hilfe der CSCA-PKI die Integrität der bereits ausgelesenen Daten geprüft werden. Um die Daten während des Auslesevorgangs vor Manipulation zu schützen, werden die Kommunikationskanäle zwischen Chip und Terminal durch kryptografische Verfahren abgesichert.

Originalität

> *Originalität ist die Glaubwürdigkeit eines Objekts bzw. Subjekts. Ist ein Objekt oder Subjekt glaubwürdig, so können wir davon ausgehen, dass es sich um genau dieses handelt, für das wir es halten.*[4]

Der elektronische Personalausweis bestätigt seine Originalität, indem er ein bestimmtes Protokoll mit einem statischen geheimen Schlüssel durchführt. Dieser Private Key ist so in dem Chip gespeichert, dass er nicht ausgelesen werden kann. Würde jemand einen Personalausweis simulieren oder ihn klonen, müsste er ein neues Schlüsselpaar verwenden. So wie anderen Daten, ist auch der öffentliche Schlüssel des Personalausweises mit einer Signatur der CSCA versehen. Ein gefälschter öffentlicher Schlüssel würde eine Prüfung dieser Signatur nicht bestehen.

Die Originalität eines auslesenden Terminals prüft der Chip in ähnlicher Weise. Das Terminal authentifiziert sich mit einem Berechtigungszertifikat, das von einer Country Verifier Certification Authority (CVCA) augestellt wurde. Der Chip kennt das Wurzel-Zertifikat der PKI und kann die Authentizität des Zertifikates prüfen. Das Terminal verwendet bei der Authentifizierung seinen dazugehörigen privaten Schlüssel. Ohne Kenntnis dieses Schlüssel würde das Verfahren für diese Authentifizierung nicht erfolgreich abgeschlossen werden.

Bei Verwendung der eID-Funktion beispielsweise muss die durchführende Person ihre Originalität nachweisen. Dieser Nachweis wird durch die Kombination von Besitz und Wissen geführt. Um die eID-Funktion des elektronischen Personalausweises zu starten muss die eID-PIN verwendet werden, die nur der Ausweisinhaber kennen sollte.

[4] Vorlesung IT-Risk-Management SS2007, R. Grimm, Universität Koblenz-Landau

2.5 Sicherheitsmechanismen des neuen Personalausweises

2.5.1 BAC – Basic Access Control

Um den Zugriff auf die ePassport-Funktion zu ermöglichen wird unter anderem die Basic Access Control (BAC) eingesetzt. Mit Hilfe dieses Protokolls etabliert der Chip mit einem auslesenden Terminal einen abgesicherten Kanal. Im Wesentlichen findet während diesem Protokoll ein Diffie-Hellman-Schlüsselaustausch statt, wodurch ein gemeinsamer Schlüssel berechnet werden kann, ohne diesen übertragen zu müssen (vgl. [Buc08, S. 153]). Um dieses Verfahren zu starten benötigt das Terminal einen Schlüssel der aus der MRZ , zu deutsch maschinenlesbare Zone, eines Reisepasses oder Personalausweises ermittelt werden kann. Die MRZ wird in der Regel durch einen Scanner erfasst. Der so generierte Schlüssel wird genutzt um die Nachrichten während des Schlüsselaustauschs zu verschlüsseln. Diese Vorgehensweise stellt sicher, dass nur jemand mit unmittelbarem Kontakt zum Reisepass oder Personalausweis Zugriff auf die Daten erhält. Um sensible Daten auslesen zu können, wie in Abschnitt 2.3.1 beschrieben, sind allerdings weitere Authentifizierungsprotokolle notwendig, die das auslesnde Terminal als hoheitliches Inspektionssysteme authentifizieren (vgl. [BSI10b, S. 21]).

2.5.2 PACE – Password Authenticated Connection Establishment

Ebenso wie BAC sorgt auch das PACE dafür, dass ein geheimer Schlüssel für die Kommunikation zwischen einem Chip und einem Terminal erzeugt wird. PACE kann jedoch auch mit anderen Schlüsseln als der MRZ durchgeführt werden. Die Wahl des Schlüssels hängt i.d.R. vom Anwendungsfall ab. Weitere mögliche Schlüssel sind die geheime PIN (eID), die auf dem elektronischen Personalausweis aufgedruckte CAN (ePassport, eSign) oder die Personal Unblocking Key (PUK) (gesperrte PIN entsperren). Mit Hilfe einer sogenannten Key-Derivation-Function wird aus diesen Werten jeweils ein Schlüssel mit fester Länge hergeleitet.

Ziel des PACE-Protokolls ist die passwortbasierte Absicherung der Luftschnittstelle, also dem Kommunikationsweg zwischen Lesegerät und Chip. Die übertragenen Informationen müssen vertraulich bleiben und eine Zugriffskontrolle entscheiden, ob eine gesicherte Verbindung etabliert wird. Das Wort „Authenticated“ sollte in diesem Kontext nicht überbewertet werden. Zwar kann durch die Kombination aus Besitz des Personalausweises und Wissen über die PIN geprüft werden, ob der rechtmäßige Besitzer des Personalausweises den Verbindungsaufbau wünscht. Unter der Prämisse, dass die PIN tatsächlich nur dem Besitzer bekannt ist, könnte man hier sogar von einer Authentifikation sprechen. Dies gilt allerdings nur bei der Verwendung der eID-PIN für das eID-Verfahren. Bei der Verwendung der MRZ oder CAN erfolgt keine Authentifizierung während PACE, da kein Nachweis über das Wissen einer geheimen PIN erbracht werden muss. Es wird nur festgestellt, ob der Auslesende unmittelbaren Kontakt zum Personalausweis hat, andernfalls könnte er die aufgedruckten Werte nicht erfahren. Die reicht für die spätere Zugangskontrolle allerdings aus, da der Chip den Erfolg späterer Authentifizierungsversuche von dem für PACE verwendeten Schlüssel abhängig macht. Beispielsweise benutzen hoheitliche Inspektionssystem spezielle Berechtigungszertifikate, welche eine Authentifizierung nach PACE unter Verwendung der CAN ermöglichen. Nur so kann gewährleistet werden, dass auch ohne Zutun des Ausweisbesitzers ein Zugriff ermöglicht werden kann. Das Berechtigungszertifikat eines Dienstanbieters für eID würde jedoch nach der Durchführung von PACE mit der CAN nicht akzeptiert werden.

Abbildung 2.2 zeigt die einzelnen Schritte des PACE-Protokolls in ausführlicher Form. Die Terminologie wurde der Spezifikation des BSI [BSI10b] entnommen und weitestgehend beibehalten, um spätere Vergleiche zu erleichtern. Der Chip wird dort als Proximity Integrated Circuit Card (PICC) bezeichnet, ein Terminal, das mit dem Chip kommunizieren kann, als Public Coupling Device (PCD).

1. Der Chip erzeugt eine Zufallszahl `s` und verschlüsselt sie mit dem zuvor berechneten Wert `Kpi`. `Kpi` ist das Ergebnis einer Key-Derivation-Function, die aus dem verwendeten Schlüssel (MRZ, CAN, PIN) einen Schlüssel fester Länge erzeugt. Welcher Schlüssel das ist, wurde dem Chip bei der Aufforderung zur Durchführung von PACE explizit mitgeteilt und spielt für den konzeptuellen Ablauf von PACE keine Rolle. Die verschlüsselte Zufallszahl wird gemeinsam mit Domain-Parametern an das Terminal gesen-

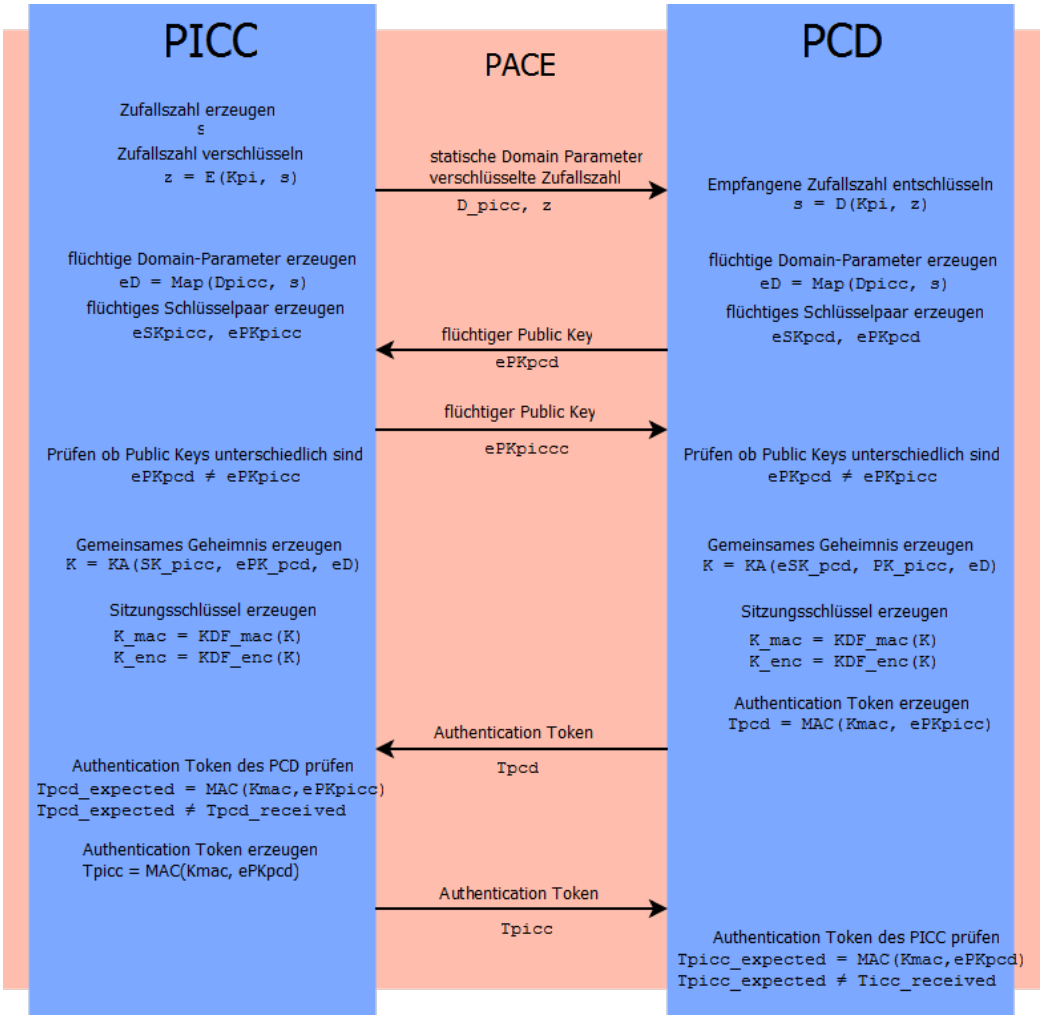

Abbildung 2.2: PACE - Password Authenticated Conenction Establishment

det. Die Domain-Parameter enthalten Informationen zu den Protokollen, die der Chip ausführen kann. Diese Informationen umfassen die Idenfikation der Protokolle selbst und Werte, die für die kryptografischen Berechnungen notwendig sind. In diesem Fall sind das unter anderem die Parameter für das Diffie-Hellman-Verfahren.

2. Das Terminal kann nach Eingabe des Schlüssels (MRZ, CAN, PIN) durch den Benutzer nun ebenfalls `Kpi` berechnen und die vom Chip erzeugte Zufallszahl entschlüsseln.

3. Für den folgenden Diffie-Hellman-Schlüsselaustausch erzeugen sich nun Chip und Terminal auf Basis der Domain-Parameter und der Zufallszahl neue Domain-Parameter `eD` und jeweils ein Schlüsselpaar `eSKpicc` und `ePKpicc`. Diese Werte werden nur für diesen einen Durchlauf des Verfahren verwendet und sind daher als flüchtig bezeichnet.

4. Die öffentlichen Schlüssel der Schlüsselpaare werden nun ausgetauscht.

5. Basierend auf den flüchtigen Werten erfolgt nun das Erzeugen des gemeinsamen Geheimnisses `K`, aus dem wiederum zwei Schlüssel abgeleitet werden. Der Schlüssel `K_enc` wird für die Verschlüsselung der künftig übertragenen Nachrichten genutzt, `K_mac` für deren Signierung.

6. Chip und Terminal erzeugen unter Verwendung vom `K_mac` beide ein Authentifizierungs-Token, dass sie austauschen und vom anderen prüfen lassen können. So kann sicher gestellt werden, dass das Verfahren erfolgreich durchgeführt wurde.

Nachdem PACE besitzen der Chip und das Terminal gemeinsame Sitzungsschlüssel `Kenc` und `Kmac`. Der Chip befindet sich von nun an im Modus SecureMessaging, was bedeutet, dass er sämtliche empfangenen Nachrichten mit den Sitzungsschlüsseln versucht zu entschlüsseln und deren Signatur prüft. Nachrichten die er sendet, verschlüsselt und signiert er mit den Sitzungsschlüsseln. Ab diesem Zeitpunkt ist der Kommunikationsweg zwischen dem lokalen Computer, bzw. dem Lesegerät und dem Chip abgesichert.

2.5.3 EAC – Extended Access Control

Im Gegensatz zur BAC, wird bei der Extended Access Control (EAC) geprüft, ob das Auslesende Terminal die Berechtigung hat, sensible Daten auszulesen. Die Grundlage für diesen Mechanismus bildet eine PKI, mit deren Zertifikaten diese Berechtigung eines auslesenden Terminals geprüft werden kann. Gleichzeitig authentifiziert sich der Chip gegenüber dem Terminal. Die EAC (Erweiterte Zugangskontrolle) besteht darin, dass durch die Terminal Authentication (TA) die Zugriffsberechtigung des Terminals geprüft wird und nach der Chip Authentication (CA) ein gesicherter Kanal besteht. Dabei wird abhängig vom Anwendungs-

fall entschieden in welcher Reihenfolge die beiden Protokolle durchgeführt werden. Soll eID ausgeführt werden, muss die TA vor der CA erfolgen. Hoheitliche Inspektionssysteme hingegen führen zuerst die CA durch.

Die TA dient der Authentifikation des Terminals und nach ihrer erfolgreichen Durchführung erhält das Terminal Zugriff auf die Daten. Um Daten vom Chip zu erhalten benötigt das Terminal ein Berechtigungszertifikat und einen dazugehörigen geheimen Schlüssel. Solche Berechtigungszertifikate werden von einer CVCA (vgl. [BSI10b, S. 14]) ausgestellt und enthalten eine Beschreibung darüber, auf welche Daten zugegriffen werden darf. Zur Beschreibung des Umfangs enthält das Berechtigungszertifikat eine Bitmaske, bei der jedes Bit eine Datengruppe repräsentiert. Zu jeden persönlichen Datum (Name, Vorname, Anschrift ...) existiert solch eine Datengruppe im Chip. Der Chip gibt genau die Datengruppen frei, für die die jeweiligen Bits gesetzt sind. Der Verwendungszweck wird in natürlicher Sprache gespeichert und wird dem Benutzer zur Kontrolle dargestellt. Diese Daten sind mit dem privaten Schlüssel eines CVCA-Zertifikats signiert, so dass der Chip die Signatur und somit die Authentizität der Daten prüfen kann.

Bevor der Chip den Zugriff auf die Daten endgültig erlaubt, entscheidet er, ob er die Daten ausschließlich über einen Ende-zu-Ende-gesicherten Kommunikationskanal überträgt, z.B. bei eID. Für die Etablierung des gesicherten Kanals dient die CA. Während ihr wird das gemeinsame Geheimnis berechnet. Die tatsächliche Verifizierung der Authentizität des Chips wird jedoch erst später durchgeführt, in dem während der Passive Authentication die ausgelesenen Datengruppen verifiziert werden.

2.5.4 Terminal Authentication

Bei der TA muss sich das auslesende Terminal gegenüber dem Chip authentifizieren. Während dieser Authentifizierung stellt der Chip fest, welche Berechtigungen das Terminal hat. Im Falle von eID wird insbesondere festgestellt auf welche persönlichen Daten das Terminal zugreifen darf. Der genaue Ablauf der TA ist von der Reihenfolge, in der die TA und die CA ausgeführt werden, abhängig. In der technischen Richtlinie [BSI10b] werden beide Protokolle jeweils in der Version 1 (siehe Abbildung 2.3, CA vor TA) und Version 2 (siehe Abbildung 2.4, TA vor CA) unterschieden. Die Unterscheidung in die beiden Versionen ist notwendig,

da einige Parameter in beiden Protokollen verwendet werden und je nach Reihenfolge in der TA oder der CA berechnet werden müssen. Zur Durchführung von eID werden diese Protokolle in der Version 2 durchgeführt.

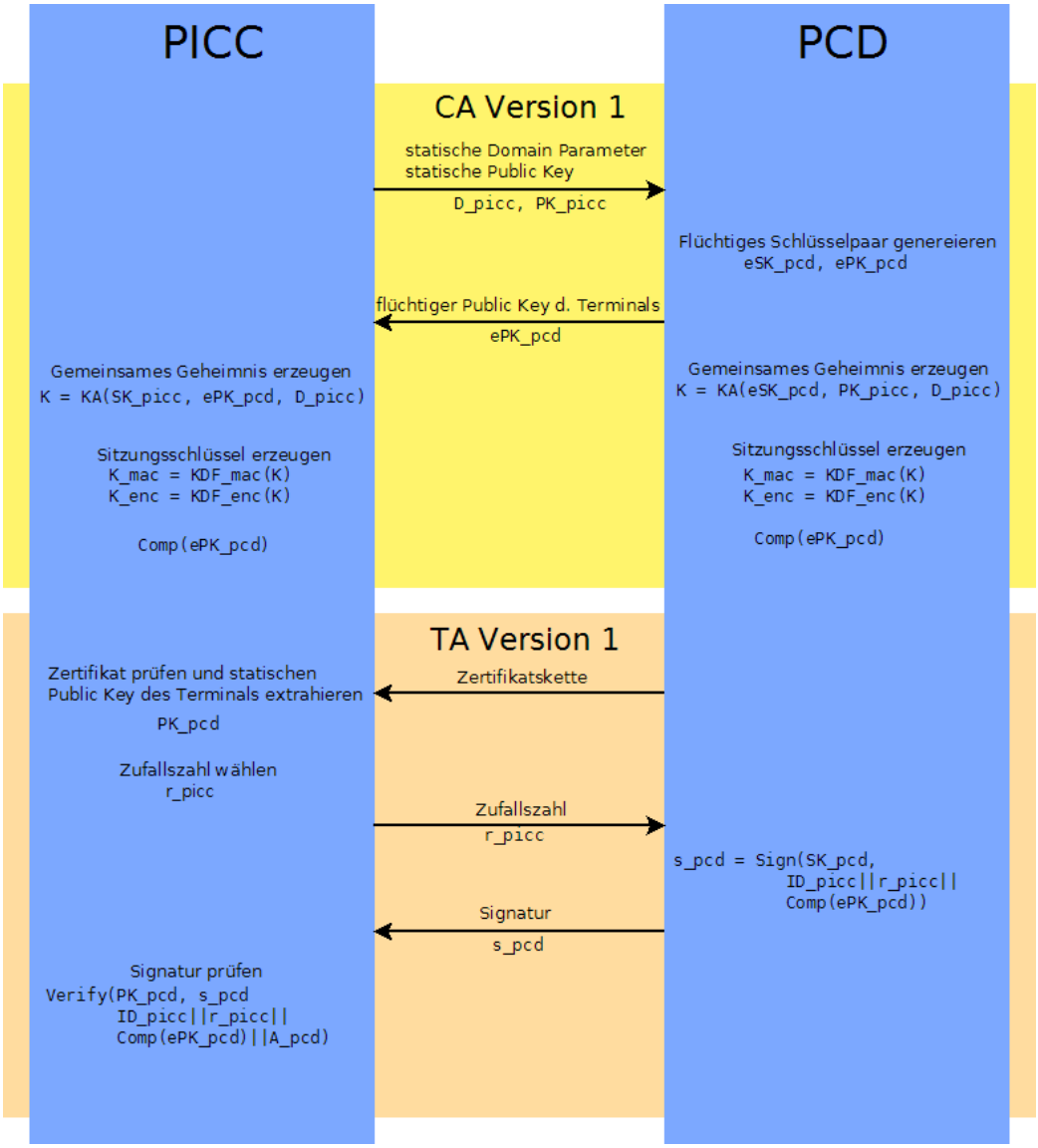

Abbildung 2.3: Chip Authentication vor Terminal Authentication

1. Zu Beginn sendet das Terminal seine Zertifikatskette an den Chip, welches der Chip prüfen kann und den öffentlichen Schlüssel des Berechtigungszertifikats `PKpcd` extrahiert. Die Zertifikatskette besteht aus dem Berech-

tigungszertifikat des Terminals und all seinen in der PKI übergeordneten Zertifikaten.

2. In der Version 2 generiert das Terminal nun ein flüchtiges Schlüsselpaar `eSKpcd` und `ePKpcd`. Dieses benutzt es nicht direkt zu seiner eigenen Authentifizierung, sondern für die spätere Aushandlung des Geheimnisses zur Absicherung des Kommunikationskanals. Der flüchtige öffentliche Schlüssel wird nun komprimiert und zusammen mit zusätzlichen Daten an den Chip gesendet. Zusätzliche Daten können beispielsweise angeben, dass eine Altersverifikation durchgeführt werden soll (vgl. [BSI10b, S. 65]).

3. Der Chip wählt eine Zufallszahl `r_picc`, die er dem Terminal sendet.

4. Das Terminal sendet nun eine Signatur `s_pcd` zurück, die es mit seinem statischen geheimen Schlüssel erzeugt hat. Signiert werden ein Hash des flüchtigen öffentlichen Schlüssel des Chips aus dem vorangeganenen PACE-Ablauf `IDpicc`, die Zufallszahl `r_picc`, der komprimierte flüchtige öffentliche Schlüssels des Terminals `Comp(ePKpcd)` und die zusätzliche Daten des Terminals `Apcd`. Falls Version 1 der Protokolle benutzt wird, wurde der flüchtige öffentliche Schlüssel während der, in diesem Fall vorangegangenen CA, bereits übertragen und zusätzliche Daten werden nicht übermittelt.

5. Der Chip prüft die Signatur und bricht die Kommunikation ab, falls die Prüfung negativ ausfällt.

Nachdem die Terminal Authentication erfolgreich durchgeführt wurde, lässt der Chip den Zugriff auf eine Datei zu, in denen Informationen für die folgende Chip Authentication enthalten sind. Dies sind der statische öffentliche Schlüssel und die statischen Parameter des Chips für einen neuen Diffie-Hellman-Schlüsselaustausch während der CA. Außerdem ist der Chip im Besitz des Berechtigungszertifikats, welches er verifiziert hat und anhand der Berechtigungen die in ihm gesetzt sind, den Zugriff auf die persönlichen Daten zulässt.

2.5.5 Chip Authentication

Ziel der CA ist die Etablierung einer gesicherten Ende-zu-Ende-Verbindung zwischen Terminal und Chip. Zwar wurde PACE durchgeführt und somit eine gesicherte Verbindung zwischen dem lokalen Terminal und dem Chip bereits etabliert, jedoch kann das auslesende Terminal auch entfernt und über ein Netz erreichbar sein. Also muss zwischen dem auslesenden Terminal und dem Chip erneut eine gesicherte Verbindung hergestellt werden. Das in der CA erzeugte gemeinsame Geheimnis wird zur Verschlüsselung und Signierung der übertragenen Nachrichten benutzt. Wie die TA wird auch die CA, je nach Ausführungreihenfolge, in der Version 1 (siehe Abbildung 2.3, CA vor TA) und der Version 2 (siehe Abbildung 2.4, TA vor CA) unterschieden.

1. Der Chip sendet zu Beginn der CA seine statischen Domain Parameter `Dpicc` und seinen statischen öffentlichen Schlüssel an das Terminal.

2. In der Version 1 hat noch keine TA stattgefunden. Also generiert das Terminal zunächst ein flüchtiges Schlüsselpaar `eSK_pcd` und `ePK_pcd`. In Version 2 wird dieser Schritt übersprungen und das Schlüsselpaar der TA benutzt.

3. Das Terminal sendet seinen flüchtigen öffentlichen Schlüssel`ePK_pcd` an den Chip.

4. In der Version 2 hat bereits eine TA stattgefunden. Der Chip versucht nun sicherzustellen, dass er immer noch mit dem selben Terminal kommuniziert. Er prüft, ob der während der TA empfangene Hashwert des flüchtigen öffentlichen Schlüssel des Terminals dem gerade empfangenen Schlüssel entspricht.

5.) Chip und Terminal erzeugen ein gemeinsames Geheimnis `K`.

6. v1) In der Version 1 leiten Chip und Terminal aus dem Geheimnis die Sitzungsschlüssel `K_enc` und `K_mac` ab. Außerdem erzeugen sie jeweils den Hashwert des öffentlichen Schlüssels des Terminals `Comp(ePK_pcd)`, der für die folgende TA in der Version 1 benötigt wird. In der Version 2 fließt in die Berechnung der Sitzungsschlüssel noch eine vom Chip gewählte Zu-

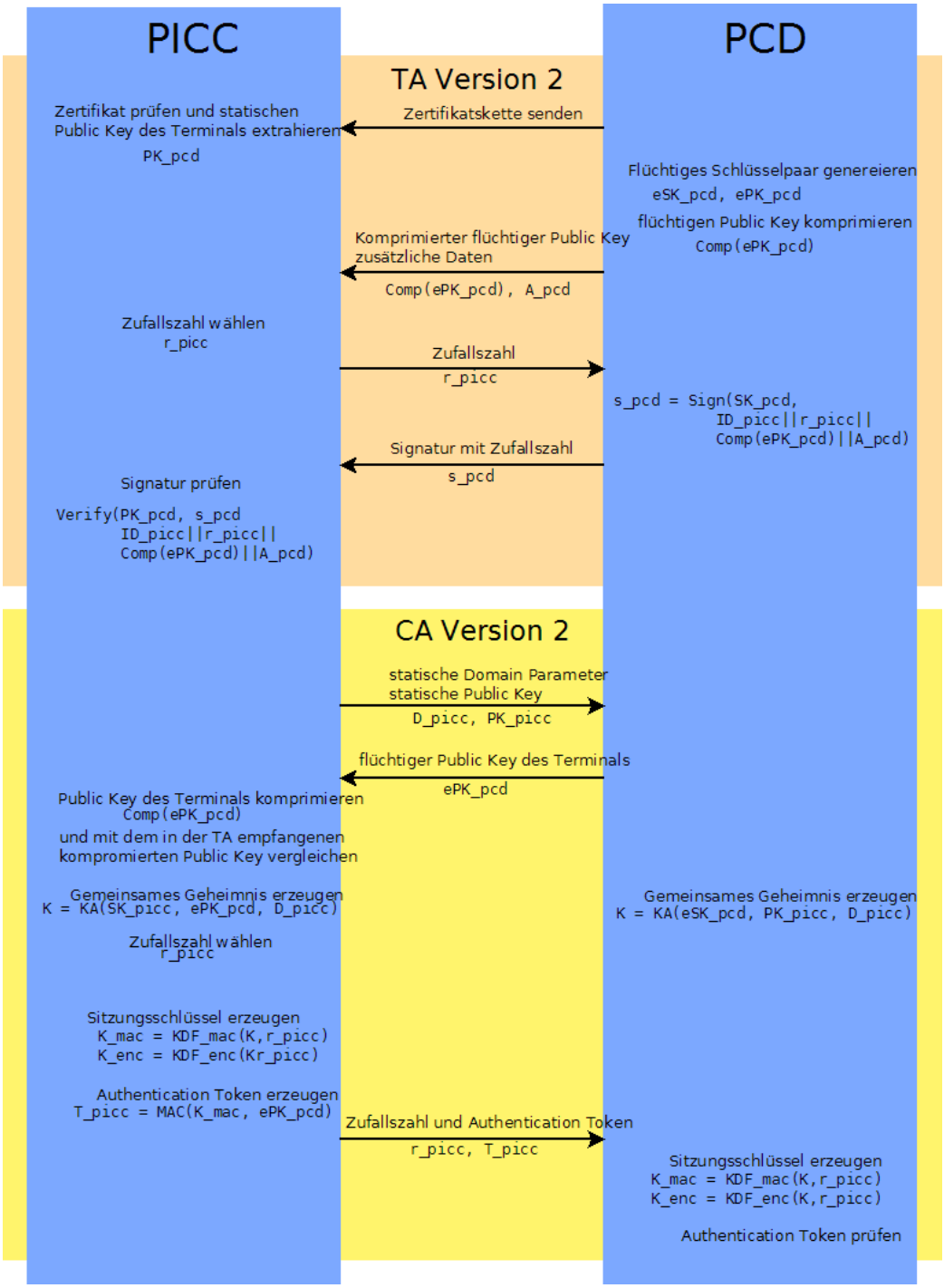

Abbildung 2.4: Terminal Authentication vor Chip Authentication

fallszahl `r_picc` mit ein, die der Chip zusammen mit einem Authentication-Token `T_picc` überträgt, das vom Terminal geprüft werden kann.

Nachdem diese Schritte erfolgreich durchgeführt wurden, sind Chip und Terminal nun im Besitz der Sitzungsschlüssel `Kenc` und `Kmac`. Der Chip befindet sich von nun an erneut im Modus SecureMessaging, allerdings verwirft er die Sitzungsschlüssel aus PACE und nutzt die neuen Sitzungsschlüssel der CA. Ab diesem Zeitpunkt ist eine gesicherte Ende-zu-Ende-verbindung zwischen dem Chip und dem Terminal etabliert.

2.5.6 Public-Key-Infrastruktur

Um die Authentizität zu gewährleisten werden PKIs eingesetzt. Die Ausweishersteller signieren die auf dem Chip gespeicherten Daten mit einem Document-Signer-Zertifikat. Dieses ist mit dem Zertifikat der CSCA signiert. Durch Kenntnis des CSCA-Zertifikats kann die Authentizität ausgelesener Daten verifiziert werden. Um Daten auslesen zu dürfen, benötigen Diensteanbieter ein Berechtigungszertifikat, das sie von einer Document-Verifier-Instanz erhalten. Um die Berechtigungszertifikate zu signieren, erhalten die Document-Verifier selbst ihre DV-Zertifikate von einer CVCA. Zu jeder Kartenapplikation existiert solch eine CVCA, die eine eigene PKI berteibt.

Der Chip besitzt keine eigene Uhr um die Gültigkeit eines Berechtigungszertifikats zu prüfen. Jedes mal, wenn er ein gültiges Zertifikat empfängt, speichert er dessen Herstellungsdatum, falls er noch kein jüngeres empfangen hat und akzeptiert keine Zertifikate deren Ablaufdatum älter ist, als das zuletzt gespeicherte Datum. Auf diese Weise wird ein Näherungswert des tatsächlichen Datums im Chip gespeichert. Falls ein Ausweisinhaber in außreichend kurzen und regelmäßigen Abständen die Verbindung mit einem Terminal mit gültigem Berechtigungszertifikat aufnimmt, kann er relativ sicher sein, dass nur gültige Zertfikate vom Chip akzeptiert werden.

2.5.7 Passive Authentication

Bei allen bisher genannten Protokolle war der Chip selbst aktiv. Die Passive Authentication basiert lediglich auf den signierten Daten des Chips und der CVCA-PKI. Sobald ein Terminal Daten aus dem Chip ausgelesen hat, kann es die Signatur dieser Daten, durch Kenntnis des CSCA-Zertfikates, welches die CSCA selbst verifiziert, prüfen. Bei erfolgreicher Prüfung kann das Terminal davon ausgehen, dass die Daten authentisch sind, also von einem berechtigten Document-Signer erstellt wurden und unverändert sind.

Die Passive Authentication wird der Extended Access Control durchgeführt. Abhängig davon welche Versionen der Protokolle TA und CA verwendetet werden, geschieht dies an einer anderen Stellen. Wird Version 1 genutzt, soll sie nach der CA durchgeführt werden. Falls Version genutzt wird, muss sie vor der CA durchgeführt werden. In diesem Fall wird ein Wert aus dem Chip ausgelesen, mithilfe dessen die Passive Authentication durchgeführt werden kann.

2.5.8 Restricted Identification

Für einige Anwendungsfälle genügen pseudonymisierte Authentifizierungsverfahren. Bei einer pseudonymisierten Authentifizierung spielt die Identität einer Person für den eigentlichen Authentifizierungsvorgang keine Rolle, vielmehr geht es darum jemanden wieder zu erkennen. Pseudonymisierte Anmeldeverfahren sind zum Beispiel aus Internet-Communities bekannt. Hier dient meist eine Email-Adresse oder ein Nickname (Fantasiename) als Pseudonym. In Verdindung mit einem Passwort können registrierte Nutzer Zugang zum System erhalten, ohne ihre tatsächliche Identität preisgeben zu müssen.

Mit der Restricted Identifaction (RI) bietet der elektronische Personalausweis ein Pseudonymisierungsverfahren. Für jedes Tupel aus Ausweisinhaber und Dienstanbieter wird ein einzigartiges Pseydonym gebildet. Für die Bildung dieses Pseudonyms existiert für jeden Personalausweis sowie für jeden Dienstanbieter ein Schlüsselpaar. Abbildung 2.5 zeigt den Ablauf des Verfahren. Das Termianl übermittelt dem Chip den Public Key des Dienstanbieters. Danach führt der Chip mit diesem Public Key und seinem eigenen Secret Key ein Key Agreement durch. Aus dem Ergenis dieses Key Agreements erzeugt der Chip den Sector-Specific Iden-

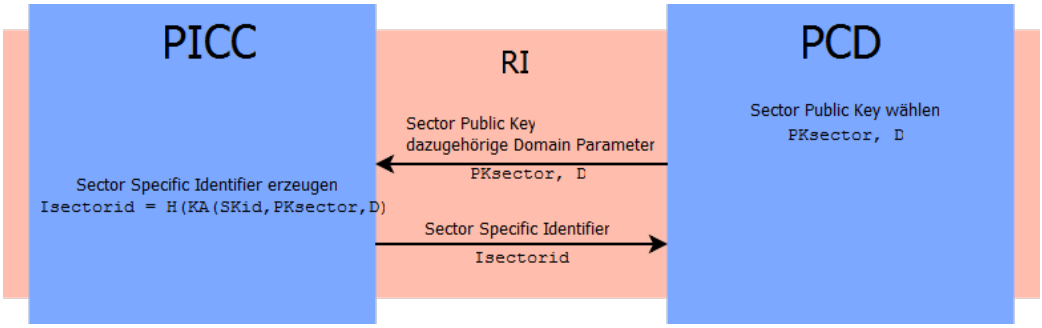

Abbildung 2.5: Restricted Identification

tifier, den er dem Terminal übermittelt. Im Gegensatz zu den bereits erwähnten Protokollen kommen keine Zufallswerte zum Einsatz, so dass bei jedem Durchlauf das selbe Ergebnis erzeugt wird.

2.5.9 Age Verification

Die Funktion Age Verification ermöglicht einen Altersnachweis seitens des Ausweisinhabers ohne sein genaue Alter preiszugeben. Ein auslesendes Terminal übermittelt dazu ein Datum. Der Chip antwortet mit einem boolschen Wert, also `ja` oder `nein`, je nachdem ob der Ausweisinhaber bis spätestens zu diesem Datum oder später geboren wurde.

2.5.10 TLS - Transport Layer Security

Der Kommunikationskanal zwischen eiD-Client und eID-Server wird mit TLS abgesichert (siehe Abbildung 2.1). TLS ist die neue Bezeichnung für Secure Sockets Layer (SSL) ab der Version 3.1 und unterstützt die Verwendung eines Pre Shared Key (PSK) und eines Session-Identifiers. Der Session-Identifier wird auch als PSK-Identity bezeichnet. So kann eine beim Server eingehende Verbindung einem zuvor bekannten Kommunikationspartner zugeordnet und mit dem PSK abgesichert werden. Beim Einsatz von eID werden zur Etablierung des sicheren Kanals die Ciphersuites des RFC4279 [Gro] benutzt (vgl. [BSI09g, S. 13]).

2.6 PAOS

Zur Formulierung von Webservice-Anfragen und deren Antworten wird üblicherweise das ursprünglich für Simple Object Access Protocol (SOAP) genutzt. Dabei werden sie als XML-Dokumente formuliert und können mittels HTTP über ein Computernetzwerk übertragen werden. Um eID durchzuführen, ist es jedoch notwendig, dass der eID-Server Webservice-Anfragen an den eID-Client stellt. Ein Client wird jedoch erwartungsgemäß in den seltensten Fällen HTTP-Anfragen entgegennehmen oder empfangen, da diese aus Sicherheitsgründen meist nicht angenommen oder von anderen Netzwerkkomponenten gar nicht erst weitergeleitet werden.

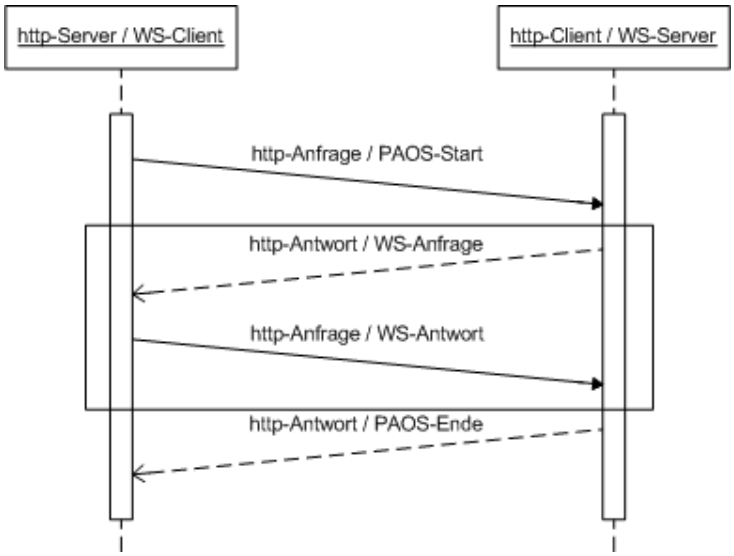

Abbildung 2.6: PAOS

Abbildung 2.6 zeigt, wie der HTTP-Server und der HTTP-Client bei der Verwendung von Liberty Reverse HTTP Binding for SOAP Specification (PAOS) [PAO] die Rollen als Webservice-Anbieter und Webservice-Konsument tauschen. Die Webservice-Anfragen werden in diesem Fall in HTTP-Antworten gekapselt. Dadurch wird es notwendig, dass der Anbieter des Webservice, der in diesem Fall der HTTP-Client ist, den Nachrichtenaustausch durch eine http-Anfrage initiiert. Der HTTP-Server, der den Webservice des HTTP-Client nutzen möchte, kann seine Webservice-Anfrag nun in der http-Antwort übermitteln. Die Webservice-Antwort wird in einer neuen HTTP-Anfrage übertragen, was es dem http-Server

ermöglicht, eine weitere Webservice-Anfrage zu stellen.

Kapitel 3

Entwurf des eID-Clients

Die zu entwerfende Software wird in zwei Bereiche gegliedert, der Realisierung eines vorgegebenen Frameworks und der eigentlichen Applikation. Um eine generelle Interaktion mit eCards zu ermöglichen, wurde vom BSI das eCard-API-Framework in einer technischen Richtlinie [BSI09a] entworfen. Dieses Framework berücksichtigt insbesondere Kommunikationtionswege zwischen drei Parteien: (1) der eCard selbst, (2) einer lokalen Software und (3) einem entfernten System. Letztere beiden besitzen jeweils eine Implementierung des eCard-API-Frameworks, mit dessen Hilfe sie kommunizieren. Da zum Zeitpunkt dieser Arbeit keine frei verfügbare Implementierung dieses Frameworks existierte, mussten alle notwendigen Teile der technischen Richtlinie entsprechend implementiert werden. Alle speziellen Anwendungen für eCards, wie eID oder eSign, finden in dieses Richtlinien keine weitere Berücksichtigung, außer dass sie einer Applikationsschicht zugeordnet werden. Sämtliche Funktionalität, die für die Nutzung der Kartenapplikationen, insbesondere der eID-Applikation notwendig ist, muss entworfen und implementiert werden. Hierbei werden wiederum Funktionalitäten unterschieden, die entweder speziell nur für das betrachtete Anwendungsszenario oder auch für andere Anwendungen relevant sind. Beispielsweise muss eine Kommunikation mit einer Karte hergestellt und verwaltet werden um eID durchzuführen. Dies betrifft jedoch auch andere Anwendungen mit eCards, wie beispielsweise der ePassport-Funktion oder einer dem Thema noch ferneren Anwendung, z.B. einer Software zur Nutzung der elektronischen Gesundheitskarte.

3.1 Abstrakte Anforderungen an eine Software-Architektur

Ian Sommerville nennt fünf nichtfunktionale Anforderungen bezüglich einer Software-Architektur [Som07]. Diese beziehen sich auf Software-Systeme unbestimmter Größe. Im Folgenden werden diese Anforderungen bezüglich der eID-Anwendung diskutiert und deren Relevanz beim Entwurf eines eID-Clients gezeigt.

Leistungsfähigkeit: Eine hohe Leistunsgfähigkeit wird erreicht, indem aufwändige Abläufe in einer möglichst kleinen Anzahl von Subsystemen konzentriert werden um die Anzahl der Kommunikationswege zwischen diesen zu minimieren. Er zielt hierbei vor allem auf verteilte Systeme ab, bei denen ein Nachrichtenaustausch zeitlich gesehen wesentlich aufwändiger ist, als eine lokale Berechnung. Eine größere Anzahl an ausgetauschten Nachrichten würde die Leistungsfähigkeit bezüglich der Geschwindigkeit verringern. Das System für die eID-Anwendung ist ebenfalls als verteiltes System anzusehen, da eine lokale Software, eine dort angeschlossener elektronischer Personalausweis und ein entfernter Server jeweils miteinander kommunizieren. Es wird sich im Laufe der Arbeit zeigen, dass sich die zeitkritischen Abläufe, durch die Gestaltung des eCard-API-Frameworks bereits, auf die lokale Seite konzentriert sind und eine minimale Anzahl an Nachrichten über das Netz ausgetauscht werden muss. Da die lokalen Abläufe innerhalb einer Softwareinstanz stattfinden, ist diese Anforderung bei der Entwicklung eines eID-Clients nicht zu berücksichtigen.

Zugriffsschutz: Um den Schutz vor Zugriffen von außen zu gewährleisten soll eine Architektur mit Schichtenstruktur verwendet werden und die kritischsten Inhalte in den untersten Schichten mit hohem Schutzniveau angeordnet sein. Er möchte damit offensichtlich erklären, dass die unteren Schichten über den erfolgreichen Zugriff entscheiden und nur über entsprechende Schnittstellen erreichbar sind, welche diese Entscheidungen unumgänglich machen. So lange ein Entwickler oder Angreifer keinen Zugriff auf die programmiertechnischen Inhalte unterhalb der Schnittstelle dieser Schichten hat, wird er also nur einen kontrollierten Zugriff auf die kritischen Inhalte erhalten. Die kritischsten Inhalte bezüglich der eID-Anwendung befinden sich auf dem Chip des Personalausweises und

sind durch dessen Sicherheitsmechanismen geschützt, die in Abschnitt 2.5 erklärt sind. Somit ist diese Anforderung bereits erfüllt.

Sicherheit: Der Aufwand für die Wartung und Validierung steigt mit dem Umfang einer Software. Der Grund liegt meist darin, dass immer mehr Programmteile an Prozessen beteiligt sind und deren Verknüpfungen schwerer nachvollziehbar werden. Eine gut gewählte und umgesetzte Architektur führt dazu, dass dieser Aufwand langsamer mit der Größe des Systems steigt. Sommerville schlägt vor, dass alle sicherheitsrelevanten Aktivitäten in einer möglichst kleinen Anzahl von Subsystemen gekapselt sein sollen, um Kosten und Probleme der Sicherheitsvalidierung gering zu halten. Ein einzelnes System kann einfacher getestet werden, als mehrere an einem Prozess beteiligte. Im eID-Client sollten die Durchführung der Sicherheits-Protokolle und kryptografischen Berechnungen also auch in eigenen Programmteilen stattfinden, die unabhängig von den anderen getestet und validiert werden können.

Verfügbarkeit: Es existieren Systeme, die stets verfügbar sein müssen. Beispielsweise muss während eines Space-Shuttle-Einsatzes der Board-Computer immer funktionieren. Die Konsequenz eines Ausfalls wäre der Verlust des Schiffes und der Besatzung und wäre nicht tragbar. Sommerville sagt, dass das Vorhandensein von redundanten Komponenten, von denen einzelne Komponenten im laufenden Betrieb abgeschaltet und ausgetauscht werden können, diese Verfügbarkeit sicherstellen. Im Falle des Space-Shuttles existieren vier vollwertige Board-Computer die stets parallel arbeiten. Falls ein eID-Client nicht funkioniert, wäre die Konsequenz die Notwendigkeit einer Neuinstallation. Ein eID-Client ist im wesentlichen eine eigenständig laufende Software, die ohne unverhältnismäßig großen Aufwand ausgetauscht oder neu installiert werden kann. Somit ist diese Anforderung nicht relevant.

Wartbarkeit: Eine hohe Wartbarkeit besteht dann, wenn die Behebung von Fehlern oder die Erweiterung des Funktionsumfangs einen geringem Aufwand in Form von Zeit und Kosten darstellen. Systemarchitekturen mit kleinen, unabhängigen Komponenten, die schnell geändert werden können, verringern den Aufwand bei Wartungen. Diese Komponenten sollten ein möglichst hohes Maß

an Entkopplung besitzen und über Schnittstellen mit wenigen Interaktionspunkten verfügen. Dies gilt sowohl für die vertikale Verteilung innerhalb einer Schichtenarchitektur, also untergeordnete Komponenten die übergeordneten Komponenten Dienste zur Verfügung stellen, als auch für die horizontale Verteilung, also Komponenten die gemeinsam an einem Prozess beteiligt sind.

Bei der Entwicklung des eID-Clients werden also vorrangig die Anforderungen Sicherheit und Wartbarkeit verfolgt. Zusätzlich ist zu bedenken, dass ein eID-Client in zwei Rollen auftreten kann. Er kann selbst der Kommunikationspartner eines elektronischen Personalausweises sein, aber auch als Nachrichtenvermittler zwischen einem entfernten eID-Server und dem Personalausweis dienen. Insbesondere kann dies auch für andere Anwendungen zutreffen, die das eCard-API-Framework benutzen. Es ist absehbar, dass Teile der zu entwickelnden Software in einer potientiellen eID-Server-Implementierung oder in anderen Softwareprojekten Verwendung finden können. Daher sollten einzelne Komponenten so aufgebaut sein, dass sie in anderen Projekten wiederverwendet werden können.

3.2 Das eCard-API-Framework

Das eCard-API-Framework aus Abbildung 3.1 beschreibt eine Reihe von plattformunabhängigen Schnittstellen zur Kommunkation zwischen Anwendungen und Chipkarten. Motivation zur Entwicklung dieses Frameworks waren die vom Bundeskabinett am 09.03.2005 beschlossenen Eckpunkte der eCard-Strategie. Die wichtigsten Punkte dieser Strategie sind die elektronische Authentisierung und die qualifizierte elektronische Signatur. Das eCard-API-Framework unterstützt hierbei nicht speziell die Verwendung des neuen Personalausweises, sondern zielt auch auf andere Projekt wie der elektronischen Gesundheitskarte und dem elektronsichen Reisepass ab. Diese Chipkarten unterscheiden sich grundlegend nicht vom neuen Personalausweis und können mit Hilfe dieses Frameworks angesprochen werden. Im Folgenden werden die einzelnen Bestandteile des Frameworks beschrieben.

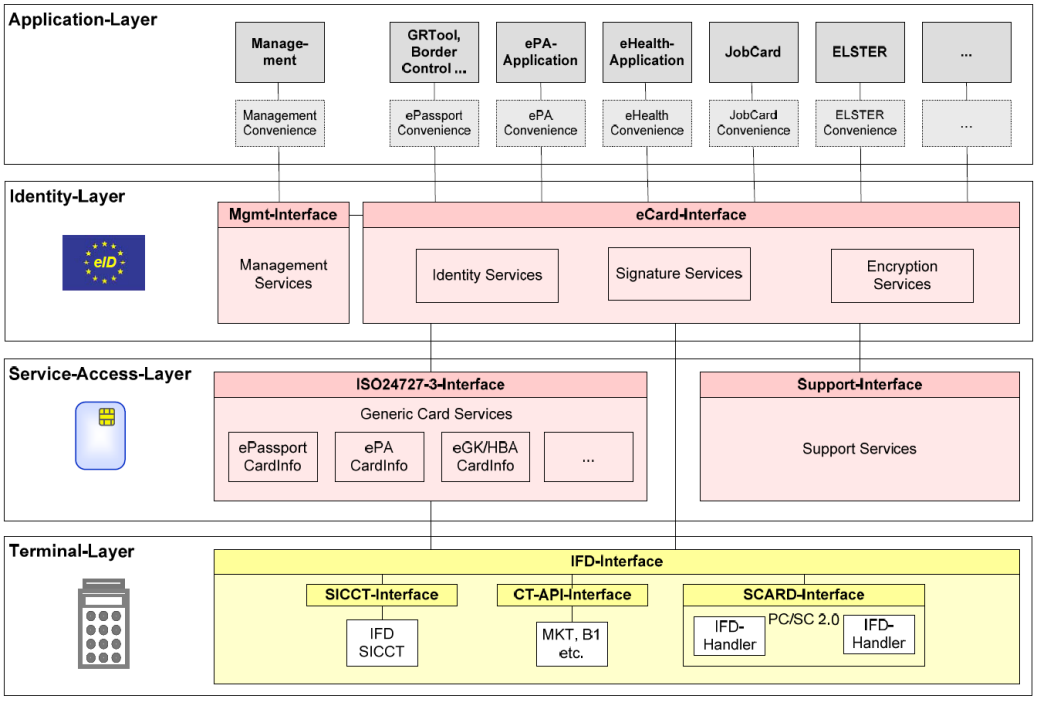

Abbildung 3.1: eCard-API-Framework [BSI09a]

Application-Layer

Der Application Layer beinhaltet die jeweiligen Anwendungen, die das eCard-API-Framework nutzen wollen. Ein eID-Client ist solch eine Anwendung. Außerdem sind anwendungsspezifische Schnittstellen vorgesehen, die wiederkehrende Aufruffolgen kapseln. Eine „ePA-Convenience-Schnittstelle“ könnte beispielsweise die Durchführung der Sicherheitsprotokolle, wie PACE, TA und CA, und deren kryptografische Berechnungen anbieten. Diese Schnittstellen werden als „Convenience-Schnittstellen“ bezeichnet und sind im eCard-API-Framework derzeit nicht weiter definiert.

Identity-Layer

Der Identity-Layer beinhaltet das eCard-Interface und das Management-Interface. Das eCard-Interface [BSI09b] bietet Funktionen für den Bezug von Zertifikaten und Verschlüsselungs- und Signaturfunktionen. Das ebenfalls dort angesiedelte

Management-Interface [BSI09c] dient dem Update des Frameworks und der Verwaltung von vertrauenswürdigen Identitäten, Chipkarten und Kartenterminals. Abgesehen von einer Funktion des Management-Interfaces, die zu Beginn einer Kommunikation zwischen eID-Server und eID-Client verwendet wird, findet der Identity-Layer zunächst keine Verwendung.

Service-Access-Layer

Im Service-Access-Layer ist ein Webservice-Interface und das Support-Interface gekapselt.

Mit dem ISO24727-3-Interface [BSI09d] wird eine Webservice-Schnittstelle realisiert. Sie bietet Funktionen um Verbindungen von entfernten Systemen, wie eID-Servern, zu Chipkarten herzustellen und kryptografische Funktionen der Anwendungen auszuführen.

Das Support Interface [BSI09e] bietet unterstützende Funktionen, die eine eCard nicht ausführt.

Terminal-Layer

Der Terminal-Layer enthält das IFD-interface [BSI09f]. Mit Hilfe dieses Interfaces werden verschiedene Typen von Lesegeräten und Schnittstellen zu Chipkarten generalisiert. So entfällt für den Anwender die Relevanz, welche Hardware er nutzt.

Protocols

Die verwendeten Protokolle, wie dem Aufbau eines Kommunikationskanals oder die Verwendung kryptografischer Funktionen, werden in [BSI09g] und [BSIb] beschrieben.

3.3 Die Komponenten von Rosecat

Das eCard-API-Framework kapselt durch seine Schnittstellen wesentliche Teile die ein eID-Client benötigt. Hierzu zählen unter anderem das IFD-Interface für die Kommunikation mit dem Chip und das ISO24727-3-Interface für die Kommunikation mit entfernten Applikationen, wie beispielsweise ein eID-Server, die ihrerseite eine Implementierung des eCard-API-Frameworks besitzen. Neben der Realisierung notwendiger Komponenten des eCard-API-Framework werden für die vollständige Applikation noch weitere Teile benötigt. Einerseits muss speziell für eID eine Applikation entworfen werden. Außerdem werden unterstützende Funktionen zum Umgang mit eCards allgemein entworfen, die potentiell in anderen eCard-Anwendungen wieder verwendet werden können. Der entstehende Code soll, basiernd auf der eCard-API-Framework-Implementation, als Convenience-Schnittstelle genutzt werden können. Im eID-Client soll diese dann mit möglich wenig Aufwand und entkoppelt von der eCard-API-Framework-Implementation integriert werden.

Die einzelnen Komponenten aus denen der eID-Client besteht sollen klar voneinander getrennt sein und durch wohldefinierte Schnittstellen miteinander interagieren. Diese Vorgehensweise ermöglicht die Umsetzung der in Abschnitt 3.1 genannten wichtigsten Anforderungen an die Architektur. Die folgende Grafik zeit einen Überblick der betrachteten Komponenten.

3.3.1 Core

Die Kernkomponente ist für den Start und das Beenden von Rosecat verantwortlich. Außerdem besteht ihre Hauptaufgabe darin die Komponenten der obersten Granularitätsebene zusammenzuhalten und zu Beginn zu initialisieren. Sie stellt den zentralen Punkt dar, an dem auf andere Komponenten zugegriffen werden kann.

3.3.2 GUI (Graphical User Interface)

Die Benutzungsschnittstelle oder auch Präsentationsschicht ist von der Kernkomponente und dem Rest klar getrennt und greift über Schnittstellen, auch Fassa-

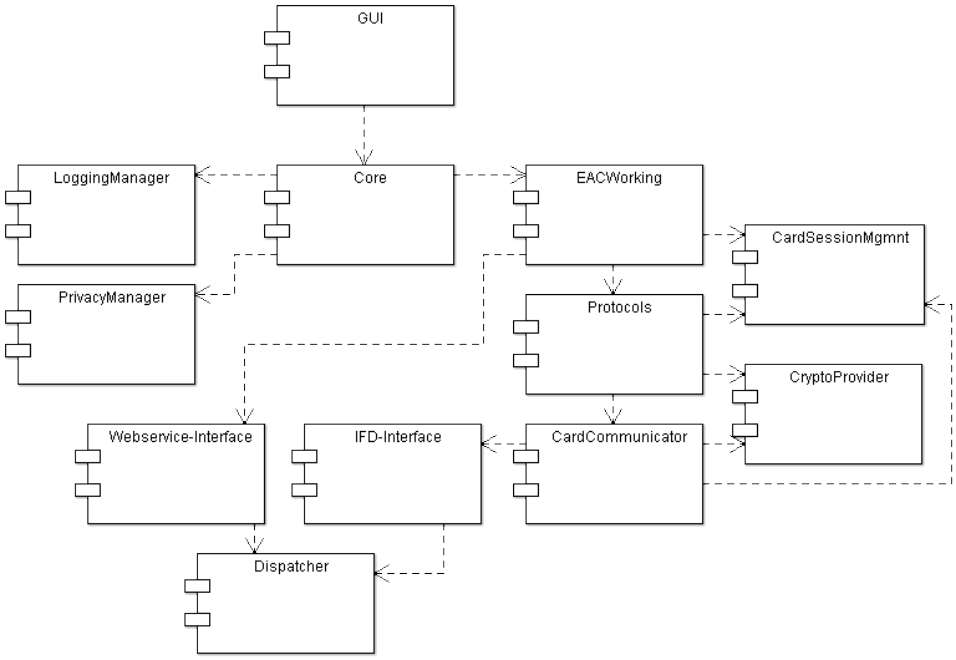

Abbildung 3.2: Gesamtansicht der Komponenten

den genannt, auf die Kernaktivitäten zu. Die GUI-Komponent umfasst auf dieser Granularitätsebene (1) die Präsentation in Form von geeigneten Darstellungsmitteln wie beispielsweise Fenstern und deren Fensterkomponenten, (2) die Dialogkontrolle, die Ereignisse des Benutzers auswertet und darauf reagiert und (3) die Anwendungsschnittstelle, die Prozesse und fachliche Klassen der Anwendungsschicht verwendet und über einen Benachrichtigungs-Mechanismus von der Anwendungsschicht über Änderungen und Ergebnisse informiert werden kann. Dieses Konzept folgt dem Seeheim-Modell [Vos98] und wird in [Dun03] entsprechend der gewählten Umsetzung beschrieben.

3.3.3 Protocols

Ein eID-Client muss die EAC-Protokolle durchführen können, also PACE, die TA und die CA (vgl. Abschnitt 2.5). Auch wenn die TA und CA Protokollabläufe zwischen dem eID-Server und dem elektronsichen Personalausweis beschreiben, agiert der eID-Client dabei unterstützend und führt die Kommunikation mit dem

Ausweis durch (vgl. [BSI09g, S. 46-53]). Die genauen Protokollabläufe müssen also im eID-Client selbst implementiert sein. Die Protocol-Komponente enthält für die einzelnen Authentifizierungsporotokolle jeweils spezialisierte Einzelkomponenten.

3.3.4 CardCommunicator

Kommandos an den Chip und dessen Antworten sind in Application Protocol Data Unit (APDU)s in Form von Byte-Arrays formuliert. Sämtliche Funktionalität zur Erstellung dieser Kommandos und Interpretation der Antworten wird in dieser Komponente implementiert. Neben der Möglichkeit beliebige APDUs an eine verbundene eCard zu senden, beherrscht diese Komponente bereits grundlegende Kartenkommandos, die jede eCard versteht und die notwendig sind, um eine weitere Kommunikation mit ihr durchzuführen. Dazu gehören die „Standard"-APDUs, mit denen in Erfahrung gebracht werden kann, welchen Funktionsumfang die Karte besitzt. Dies umfasst unter anderem die Menge der Kartenapplikationen und die von der Karte verwendeten kryptografischen Algorithmen. Außerdem stellt diese Komponente anwendungsspezifische CardCommunicator zur Verfügung, die von den Protocol-Komponenten verwendet werden und genau die Menge der Kartenkommandos enthalten, die jeweils benötigt werden.

Die CardCommunicator müssen außerdem auch die Kommunikation mit einem gesicherten Kanal mit dem Chip beherrschen. Sie müssen erkennen ob der Chip im Modus Secure Messaging kommuniziert und einen Teil der CryptoProvider-Komponente mit den kryptografischen Berechnungen beauftragen.

3.3.5 CryptoProvider

Zur Durchführung der Protokolle müssen kryptografische Berechnungen durchgeführt werden. Diese werden, je nach Protokoll, von einem spezialisierten CryptoProvider zur Verfügung gestellt. Dieser CryptoProvider bietet genau die kryptografischen Funktionen die benötigt werden und bietet Funktionen an, die entsprechend zu den Definitionen in der technischen Richtlinie „BSI TR-03110 Advanced Security Mechanisms for Machine Readable Travel Documents" [BSI10b] aufgebaut sind. Die tatsächlichen Berechnungen sollen dabei nicht selbst imple-

mentiert werden, sondern durch die Nutzung bestehender Implementationen realisiert werden. Der komplexe Funktionsumfang für diese Berechnungen wird also hinter einer Fassade von der aufrufenden Schicht entkoppelt. Bei der Integration des CryptoProviders im Zielsystem reicht es dann aus, mit der Terminologie der technischen Richtlinie vertraut zu sein. Ein weiterer positiver Effekt ist die leichte Validierbarkeit des kryptografischen Funktionsumfangs, da die in der Richtlinie spezifizierten Funktionen gekapselt aufgerufen werden können.

Eine weitere Aufgabe dieser Komponente ist die Ver- und Entschlüsselung, sowie das Signieren von APDUs und validieren von signierten APDUs, sobald ein gesicherter Kommunikationskanal mit einem Chip etabliert ist. Die ist jeweils nach PACE und der CA der Fall. Die CardCommunicator erhalten dazu einen APDU-Crypter, der diese Aufgabe übernimmt.

3.3.6 CardSessionManagement

Es reicht aus, dass ein eID-Client genau eine Verbindung zu einer Karte aufrecht halten kann. Diese Verbindung wird mit Hilfe des IFD-Interface erzeugt und durch einen sogenannten SlotHandle identifiziert. Jede Interaktion mit einer Karte, sei sie vom eID-Client selbst durchgeführt oder von einem eID-Server weitergeleitet, basiert auf diesem SlotHandle. Ein eID-Client benötigt im Verlauf aller Abläufe noch weitere Parameter, die der Karte zugeordnet werden können. Diese Parameter können z.B. Inhalte von Dateien, die aus dem Chip ausgelesen wurden oder Schlüssel für Secure Messaging sein. Diese Informationen werden in einer CardSession gekapselt.

Ein CardSessionManager kann mehrere solcher CardSessions verwalten, damit sie einzelnen Prozessen zugeordnet werden können. Auch wenn der Aufwand, mehrere Verbindungen zu verwalten obwohl nur eine benötigt wird, zunächst unnötig erscheint, sind Vorteile dieser Strategie zu erkennen. Die eine für den Anwendungsfall notwendige CardSession müsste ohnehin an einem bestimmten Punkt gespeichert werden. Der CardSessionManager bietet eine verweisbare Stelle hierfür und kann Funktionalitäten, wie das Starten und Beenden einer CardSession, zusätzlich kapseln. Außerdem entsteht so ein Modul, das für andere Anwendungszwecke verwendet werden kann. Es könnte potentiell in einem eID-Server implementiert werden, der die gleichzeitige Ausführung mehrerer eID-

Verfahren beherrschen muss (vgl. [BSI10a]).

Die Verwendung eines CardSessionManagers erscheint vor allem dann sinnvoll, sobald man die Bemerkung der Webservice-Funktion CardApllicationPath aus [BSI09d, S. 21] berücksichtigt. In ihr wird ein Hintergrundprozess gefordert, der eine Liste aller verfügbaren Lesegeräte und eCards vorrätig hält. Diese Aufgabe kann vom CardSessionManager übernommen werden. Sobald eine Karte erkannt wurde, wird eine Verbindung zu ihr hergestellt und über einen SlotHandle referenziert.

3.3.7 LoggingManager

Rosecat soll auch der Lehre dienen und die Abläufe der verwendeten Protokolle darstellen. Um dies zu erreichen wird ein Logging-Mechanismus entwickelt, mithilfe dessen die Vorgänge erfasst und visualisiert werden können. Dabei finden vor allem die verschiedenen Aspekte dieser Vorgänge Berücksichtigung. Relevante Aspekte sind (1) die Parameter für die kryptografischen Abläufe, (2) die Kommandos an den Chip und deren Antworten und (3) die Kommunikation zwischen den Webservices. Um diesen Gedanken zu verdeutlichen werden folgend einige Beispiele erklärt.

(1) Bei der Durchführung des PACE-Protokolls sendet der Chip zunächst eine verschlüsselte Zufallszahl an das Terminal. Diese kann das Terminal mit Hilfe der eingegeben PIN entschlüsseln. Die drei Parameter sollen nun sichtbar werden, um zu zeigen wie das PACE-Protokoll genau funktioniert.

(2) Jede Interaktion mit dem Chip wird durch ein Kommando ausgelöst, das mit in einer APDU übertragen wird. Der Chip sendet die Antwort ebenfalls mit einer APDU. Um die Funktionsweise der Chipkartenkommunikation zu verdeutlichen, ermöglicht der Logging-Mechanismus das Mithören von Nachrichten an und von einem Chip Card Interface Device (IFD).

(3) Die TA und CA können zwischen einem entfernten eID-Server und dem Chip durchgeführt werden. Dies ist bei eID der Fall. Bei eID beispielsweise kommuniziert der lokale eID-Client mit dem eID-Server und auch der Chip kom-

muniziert mit dem eID-Server. In beiden Fällen findet die Kommunikation mit Webservices statt. Der Logging-Mechanismus soll die Webservice-Aufrufe und deren Antworten darstellen können.

Es stellt sich nun die Frage ob es zulässig ist alle Informationen mitzuhören und darzustellen. Eine geheime PIN sollte niemals dargestellt werden. Man könnte sogar so weit gehen zu sagen, das eine geheime PIN innerhalb des Programms niemals an mehr stellen verarbeitet werden sollte, als tatsächlich notwendig. Daher ist es notwendig, dass verschiedene Anwendungsfälle betrachtet werden.

Vor Einführung des elektronischen Personalausweises fand ein offener Anwendungstest statt. Als Teilnehmer an diesem Test erhielt man Testausweise und Berechtigungszertifikate, die im Rahmen einer Test-PKI ausgestellt wurden. Zwar sind die Berechtigungszertifikate und das Zertifikat des Testausweises nicht mehr gültig. Aufgrund des Zeitmechanismus im elektronischen Personalausweis, der in Abschnitt 2.5.6 erklärt wird, sind die Testausweise in Verbindung mit den Berechtigungszertifikaten auch weiterhin benutzbar. So ist zu Demonstrationszwecken eine völlige Einsicht in die Werte der tatsächlich durchgeführten Protokolle möglich, ohne dass, bei der Verwendung eines Personalausweises des Wirkbetriebs, die Vertraulichkeit von sensiblen Daten, wie einer geheimen PIN, verletzt wird. Solch ein volltransparenter Modus ist für Demonstrationen, die Lehre und die Nachvollziehbarkeit zu gebrauchen.

Im alltäglichen Gebrauch wird es den Benutzer sehr wahrscheinlich nicht interessieren, welche Parameter errechnet werden. Daher ist ein nontransparenter Modus für den Wirkbetrieb von Rosecat am wahrscheinlichsten. Ein weiterer Fall tritt allerdings ein, falls jemand trotzdem Einblick in die kryptografischen Protokolle haben möchte, während er seinen eigenen elektronischen Personalausweis benutzt. Da ohnehin nur PACE lokal ausgeführt wird und das auch nur so lange kein Komfortleser verwendet wird, wird es auch nicht viel zu sehen geben. Daten wie geheime Schlüssel und gemeinsame Geheimnisse der anderen Protokolle bleiben nämlich im Server und dem Chip verborgen. In jedem Fall gilt aber, dass die Vertraulichkeit von sensiblen Daten zu keinem Zeitpunkt gefährdet sein darf. Dies betrifft sowohl die Darstellung einer PIN, als auch das gemeinsame Darstellen eines Chiffrats und eines Klartextes. Ein semitransparenter Modus soll daher möglichst viele Daten zeigen, die Sicherheitsanforderungen dabei aber nicht gefährden.

3.3.8 PrivacyManager

Ein PrivacyManager wird im Rahmen dieser Diplomarbeit lediglich in Ausblick gestellt. Er kann Aufschluss darüber geben, welchen Dritten man welche persönlichen Daten bereits preisgegeben hat. In einer möglichen Kooperation mit dem Unabhängigen Landeszentrum für Datenschutz Schleswig-Holstein könnten Anforderungen an solch einen PrivacyManager erarbeitet und umgesetzt werden. Der LoggingManager könnte die grundlegende Komponente sein um die entsprechenden Informationen aus den einzelnen Prozessen eines eID-Clients zu generieren.

3.3.9 ConfigurationManager

Der ConfigurationManager bietet die Möglichkeit Daten für die Zeiträume zwischen Beendigung und Neustart des eID-Clients vorrätig zu halten. Diese können unter anderem Speicherorte für Logfiles des LoggingManagers und des PrivacyManagers sowie sämtliche Einstellungen des Benutzers sein.

3.3.10 eID-Initiator

Zur Durchführung von eID werden im eID-Client Daten benötigt, die der Benutzer zuvor, mit einem Webbrowser in einem html-Dokument, empfangen hat (vgl. Abbildung 2.1 Schritt 5). Diese Daten müssen an den eID-Client übermittelt werden. In den technischen Richtlinien des BSI ist die Verwendung von Browser-Plugiins vorgesehen, die die notwendigen Daten auslesen und an einen lokalen Webservice des eID-Client weitergeben (vgl. [BSI09g, S. 12]).

Eine andere Möglichkeit wird in [Kah11] vorgeschlagen. Bei diesem Ansatz wird auf die Verwendung der lokalen Webservice-Schnittstelle verzichtet und statt dessen lediglich ein http-Request mit zusätzlichen Parametern an den eID-Client benutzt. Dieser http-Request kann durch die Verwendung html-Formularen und Links erzeugt werden. Somit entfällt die Verwendung von Browser-Plugins.

Um den Plugin-Ansatz konsequent zu verfolgen, müssten für verschiedene Browser und Betriebssysteme mehrere Plugins realisiert werden. Der Aufwand für Entwicklung, Wartung und Weiterentwicklung wäre dabei so hoch, dass zumin-

dest zum jetzigen Zeitpunkt, der alternative Ansatz verfolgt wird.

3.3.11 Webservice

Im eCard-API-Framework sind Verbindungen zu anderen eCard-API-Instanzen vorgesehen. Die Kommunikation findet über einen Webservices statt, dessen Funktionumfang im eCard-API-Framework selbst in der technischen Richtlinie TR-03112-4 [BSI09d] definiert ist. Für einen eID-Client sind vor allem Funktionen zum Herstellen von Verbindungen zu einer Chipkarte, dem etablieren von gesicherten Kommunikationskanälen zur Chipkarte, und dem Übertragen von Kommandos an den Chip von Bedeutung.

3.3.12 Dispatcher

In jeder eCard-API-Instanz soll ein Dispatcher vorhanden sein (vgl. [BSI09d, S. 16]. Zwei Instanzen nutzen ihre Dispatcher um Kommunikationskanäle über ein Netz aufzubauen und mit Sicherheitsmechanismen abzusichern. Sobald zwei Dispatcher einen Kommunikationskanal miteinander aufgebaut haben, können ihre eCard-API-Instanzen Nachrichten untereinander austauschen. Die Dispatcher leiten diese Nachrichten dann zwischen den lokalen Schichten und dem entfernten eCard-API-Framework weiter.

Ein Kommunikationskanal wird in seiner Eigenschaft durch das Binding und den Sicherheitsmechanismus definiert. Zur Absicherung des Kanals kommen übliche Sicherheitsmechanismen zum Einsatz. Deren Verwendung unterscheided sich jedoch je nach verwendetem Binding.

Als Binding kommen SOAP und PAOS in Frage. SOAP kann zur einfachen Formulierung von Webservice-Anfragen und deren Antworten genutzt werden. Laut [BSI09g] muss bei der Verwendung von SOAP als Binding mindestens TLS in der Version 1.1 als Sicherheitsmechanismus verwendet werden. Wie schon bei SSL üblich, agiert in diesem Fall der TLS/http-Client als Konsument eines Webservice und der TLS/http-Server als Anbieter des Webservice.

Wird für eine Verbindung PAOS als Binding benutzt, wie es für eID gefordert und notwendig ist, tauschen Server und Client ihre Rollen. Der TLS/http-Client

ist nun Anbieter des Webservice und der TLS/http-Server ist Konsument des Webservice. PAOS funktioniert ähnlich wie SOAP, allerdings ermöglicht es Webservice-Anfragen in html-Anworten zu kapseln, die bezüglich vorangeganen html-Anfragen übertagen werden. Auf diese Weise können Webservice-Anfragen an Systeme übertragen werden, die sonst durch Sicherheitsmechanismen, wie Firewalls oder sonstige Maßnahmen, nicht erreichbar wären (vgl. Abschnitt 2.6).

3.4 Interaktion der Komponenten

Das eCard-API-Framework ermöglicht die Ausführung mehrerer Anwendungsfälle, bei denen die Art und Weise der Interaktion mit dem Chip unterschiedlich sein kann. Die im Folgenden beschrieben Vorgehensweisen gelten nicht ausschließlich für einen eID-Server und einen eID-Client, sondern können bei anderen Anwendungsfällen ebenso Verwendung finden. Um diese Allgemeingültigkeit hervorzuheben und gleichzeitig kürzere Begriffe zu verwenden, wird die lokale eCard-API-Instanz (eID-Client) als Local-Terminal und die entfernte eCard-API-Instanz (eID-Server) als Remote-Terminal bezeichnet.

Das Local-Terminal kann durch das IFD-Interface direkt mit dem Chip kommunizieren. Die APDUs werden von einem CardCommunicator erzeugt der auch die Antworten des Chips aufbereitet. Die Interaktion zwischen einem Remote-Terminal und dem Chip kann jedoch auf unterschiedliche Art erfolgen. Das Remote-Terminal kann die Function ExecuteAction des ISO24727-Interface [BSI09d, S. 47] nutzen, um APDUs an einen Chip zu senden, der mit dem Local-Terminal verbunden ist. Diese Vorgehensweise wird derzeit in der Praxis nicht verwendet. Stattdessen wird clientseitig die Transmit-Funktion [BSI09f, S. 35] des IFD-Interface als Teil des Webservice verfügbar gemacht. Das Local-Terminal dient in diesem Fall als weiterleitende Instanz zwischen Remote-Terminal und Chip. Eine derartige Durchführung der, in Abschnitt 2.5 beschriebenen, kryptografischen Protokolle würde zu einer hohen Anzahl an Nachrichten führen, die über das Netz ausgetauscht werden müssten. Das ISO24727-Interface bietet, mit der DIDAuthenticate-Funktionen [BSI09d, S. 86], eine Möglichkeit diese Protokolle mit einer minimalen Anzahl an Nachrichten durchzuführen. In diesem Fall übernimmt das Local-Terminal die Kommunikation mit dem Chip. Das Remote-Terminal berechnet lediglich die kryptografischen Parameter, die das Local-Termi-

nal aufgrund von fehlendem Wissen über private Schlüssel nicht selbst berechnen kann.

Eine Betrachtung der beschriebenen Abläufe soll verdeutlichen, wie die beteiligten Komponenten mitereinander interagieren.

3.4.1 Lokale Kommunikation mit dem Chip

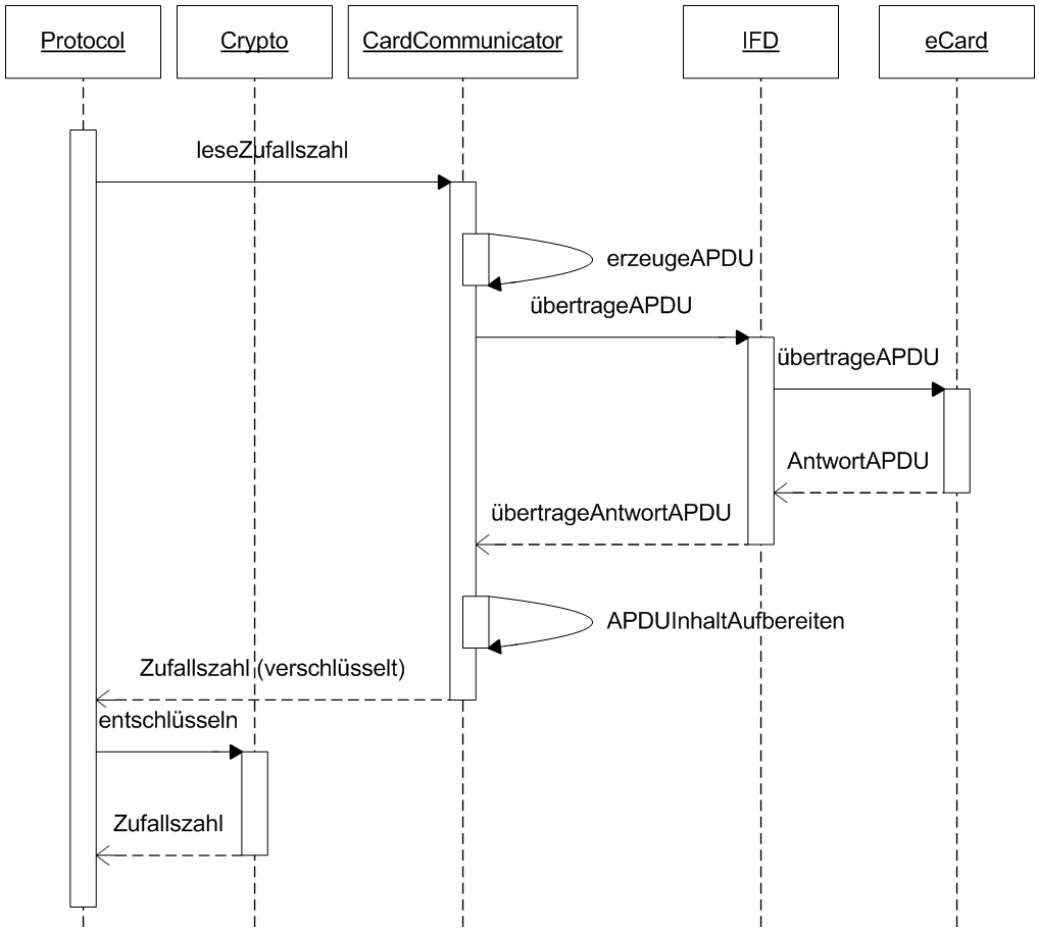

Abbildung 3.3: Protokollfunktion bis Kartenkommando

Der einfachste Fall ist die Kommunikation mit einem Chip, der mit dem lokalen Framework verbundenem ist. Abbildung 3.3 zeigt die Beteiligten Komponenten während des ersten Schritts des PACE-Protokolls. Der Chip übermittelt dem Terminal als erstes eine verschlüsselte Zufallszahl (vgl. Abschnitt 2.5.2). Da der

Chip jedoch nicht aus eigener Entscheidung heraus agiert, sondern auf Kartenkommandos reagiert, muss ihm das Kommando zur Anforderung dieser Zufallszahl übertragen werden. Die Komponente Protocols nutzt dazu einen CardCommunicator, der die entsprechende APDU erzeugt und an die lokale IFD-Instanz übergibt. Nachdem die Antwort des Chip bis an den CardCommunicator zurückgegeben wurde, wird sie dort aufbereitet und in entsprechender Form an die Protocol-Komponente zurückgegeben. Neben einem CardCommunicator besitzt jede Protocol-Komponente einen CryptoProvider, der die kryptografischen Berechnung durchführen kann. In diesem Fall übernimmt er die Entschlüsselung der Zufallszahl.

3.4.2 Protokollinitiierung durch eID-Server

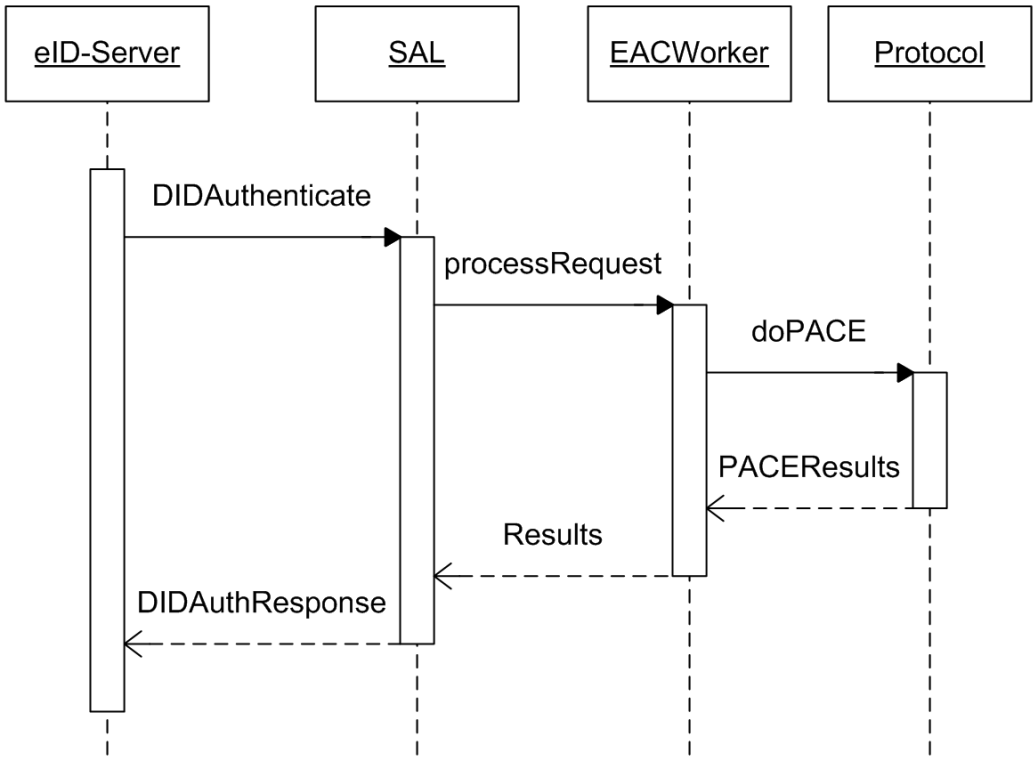

Abbildung 3.4: Start eines Authentifizierungsprotokoll

Als nächstes wird am Beispiel von PACE betrachtet, wie eines der EAC-Protokolle gestartet wird. Während PACE wird keine Authentifizierung im eigentlichen Sin-

ne vorgenommen (vgl. Abschnitt 2.4). Daher kann der Name der Funktion DIDAuthenticate aus dem eCard-API-Framework, welche die Durchführung der EAC-Protokolle ermöglicht, missverstanden werden. Abbildung 3.4 zeigt wie durch den Aufruf der Webservice-Funktion DIDAuthenticate (vgl. [BSI09d, S. 86] und [BSI09g, S. 45]) ein Local-Terminal veranlasst eines der EAC-Protokolle durchzuführen. Die Funktion DIDAuthenticate wird von einem EACWorker realisiert, der anhand der übergebenen Parameter erkennt, welches Protokoll mit dem Chip durchgeführt werden soll. Soll beispielsweise PACE durchgeführt werden, delegiert er die Durchführung von PACE an die Protocol-Komponente. Die Ergebniswerte von PACE (u.a. Inhalt der Datei EF.CardAccess, ChipIdentifier der Karte aus PACE) werden in der Antwort der Webservice-Funktion an das Remote-Terminal übermittelt, das diese für die Durchführung weiterer EAC-Protokolle benötigt.

3.4.3 Verteilte Protokolldurchführung

Bei der Verwendung von DIDAuthenticate delegiert das Remote-Terminal die unmittelbare Protokolldurchführung an das Local-Terminal. Dies ist möglich, da das eCard-API-Framework die Durchführung von Authentifizierungsprotokollen und insbesondere den EAC-Protokollen grundlegend unterstützt. Komplizierter als bei PACE gestalten sich jedoch die darauf folgenden Protokolle Terminal-Authentication und Chip-Authentication, da sie zwischen dem Chip und dem Remote-Terminal durchgeführt werden. Die Kommunikation mit dem Chip, also das Verfassen und Interpretieren von APDUs, übernimmt nach wie vor das Local-Terminal. Die kryptografischen Berechnungen müssen jedoch durch das Remote-Terminal erfolgen. Den Beschreibungen beider Protokolle aus Abschnitt 2.5 ist zu entnehmen, dass die Berechnungen auf Basis von geheimen und öffentlichen Schlüsseln erfolgen. Da der geheime Schlüssel nur dem Remote-Terminal bekannt sein darf, müssen sie also auf dem Remote-Terminal erfolgen. Schließlich soll ja eine gesicherte Verbindung zwischen dem Remote-Terminal und dem Chip aufgebaut werden. Die tatsächliche Kommunikation mit dem Chip, also das Formulieren und Senden der APDUs, wird jedoch vom Local-Terminal übernommen.

Wie PACE, werden auch die anderen Protokolle durch die Verwendung von DI-

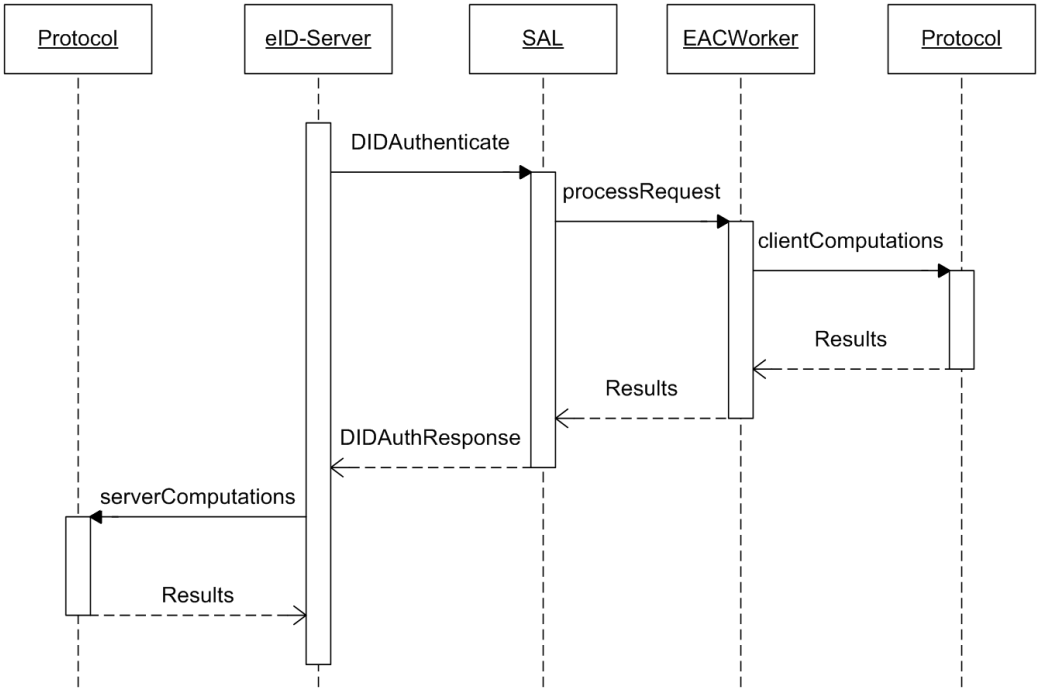

Abbildung 3.5: Verteilte Protokolldurchführung

DAuthenticate durchgeführt. Abbildung 3.5 zeigt, wie das Remote-Terminal diese Funktion an der Webservice-Schnittstelle des Local-Terminals aufruft. Da diese im Service-Access-layer des eCard-API-Framworks enthalten ist, wird sie als SAL bezeichnet. Der EACWorker beauftrag jedoch diesmal die Protocol-Komponente, nur die clientseitigen Schritte durchzuführen, also das Versenden und Empfangen von APDUs. Dabei werden die Ergebnisse der kryptografischen Berechnungen des eID-Servers an den Chip und die des Chips an den eID-Server weitergeleitet. Die Protocol-Komponente soll so gestaltet sein, dass sie auf beiden Seiten gleichermaßen eingesetzt werden kann. Zum besseren Verständnis sei an dieser Stelle anzumerken, dass die Authentifizierungsprotokolle mit Hilfe der Funktion DIDAuthenticate zwar durchgeführt werden können, die Reihenfolge des ausgetauschten Informationen den idealisierten Protokolldefinitionen aus [BSI10b] jedoch nicht mehr entspricht. Die Zuordenbarkeit zwischen den einzelnen Protokollschritten und der Bündelung ihres Informationsgehalt in den DIDAuthenticate-Nachrichten kann in [BSI09g, S. 46-53] nachgeschlagen werden.

3.4.4 Kartenkommandos über das Netz

Kartenkommandos können in Form von APDUs auch direkt auf dem Remote-Terminal erzeugt und vom Local-Terminal lediglich an den Chip weitergeleitet werden. Diese Vorgehensweise findet spätestens nach der Durchführung der Authentifizierungsprotokolle Verwendung. Diese können mit Hilfe der Funktion DIDAuthenticate durchgeführt werden. Für das Auslesen von Datengruppen existiert innerhalb des eCard-API-Frameworks keine solche Funktion. Außerdem besteht ab diesem Zeitpunkt eine gesicherte Verbindung zwischen eID-Server und dem Chip. Beides führt dazu, dass der eID-Server die APDUs selbst formuliert und den eID-Client zur Weiterleitung nutzt.

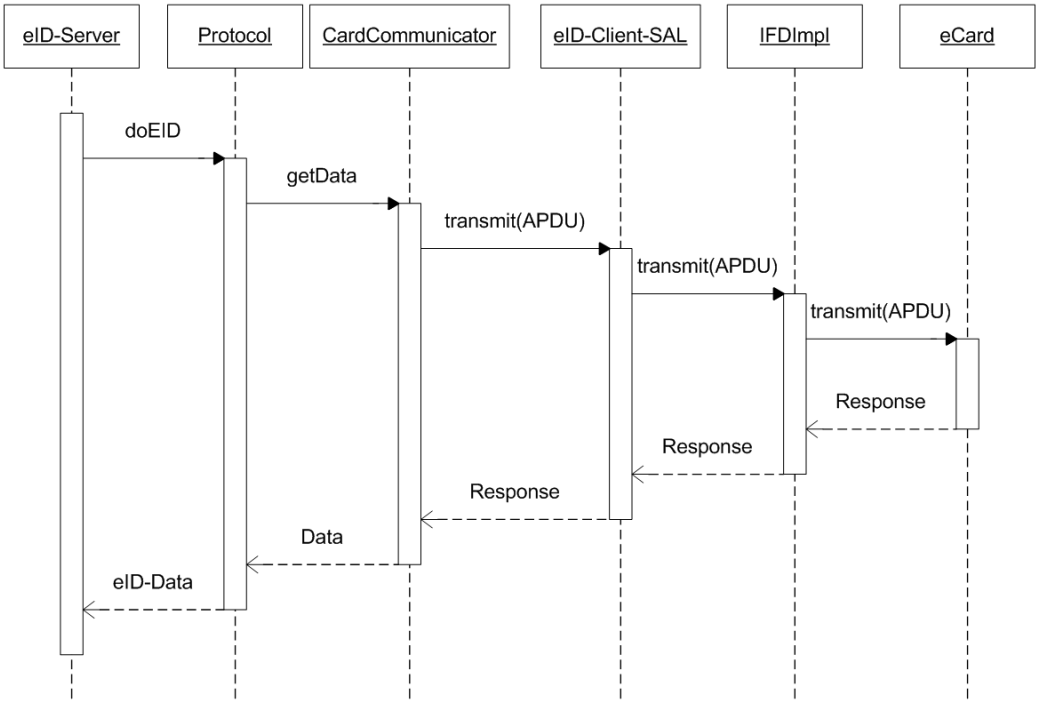

Abbildung 3.6: Übermittlung von APDUs über das Netz

Abbildung 3.6 zeigt wie der eID-Server, zum Auslesen der Datengruppen, selbst als Kommunikationspartner des Chips agiert. Zunächst beauftragt er seine Protocol-Komponente damit die Datengruppen auszulesen. Die Kartenkommandos werden von einen CardCommunicator formuliert, der die APDUs diesmal jedoch nicht an ein lokales IFD weitergibt, sondern die Transmit-Funktion des Webservice des eID-Clients benutzt. Die Antworten des Chips werden dann an den eID-

Server übermittelt. Ebenso wie die Protocol-Komponente, muss auch die Card-Communicator-Komponente so gestaltet sein, dass sie auf einem Local-Terminal und einem Remote-Terminal eingesetzt werden kann.

Kapitel 4

Implementierung

Zunächst muss entschieden werden, welche Programmiersprache und welche Entwicklungsumgebung sich für die Realisierung eines eID-Clients eignen. Dazu werden drei Anforderungen besrücksichtigt:

1. Die Zielgruppe der Anwender sind alle Bürger, was bedeutet, dass alle Systemumgebungen, wie Windows, Linux und Mac OS Berücksichtigung finden müssen.
2. Als mögliches Open-Source-Projekt, sind potentiell Entwickler beteiligt, die ebenfalls mit verschiedenen Systemumgebungen arbeiten.
3. Es finden keine besonders leistungssensitiven Berechnungen statt, also ist der Einsatz einer effizienten Programmiersprache nicht notwendgig.

Aus der Menge der Objekt-Orientierten Programmiersprachen wird Java und als Entwicklungsumgebung Eclipse gewählt. Durch die Plattformunabhängkeit von Java kann die Software mit wenig Aufwand auf verschiedenen Betriebssystemen eingesetzt werden. Die Entwicklungsumbegung Eclipse ist für die gängigen Betriebssysteme frei verfügbar und schränkt somit die Menge der potentiellen Entwickler nicht ein. Die Leistungseinbußen von Java spielen für die wenig leistungssenitiven Berechenungen eines eID-Clients nur eine untergeordnete Rolle.

Die Realisierung der einzelnen Komponenten erfolgte und erfolgt im Rahmen von Projektpraktika, Qualifikationsarbeiten und Tätigkeiten als Hilfswissenschaftlern von mehreren Personen. Den daraus entstandenen Implementationen sind

die in Abschnitt 3.1 genannten Anforderungen an die Architektur zugrunde gelegt. Insbesondere wurden die Datenstrukturen und die prozeduralen Schnittstellen (vgl. [Som07, S. 167]), welche die API für die Implementierung im Zielsystem darstellen, vorab definiert und in den einzelnen Komponenten entsprechend implementiert. So konnten einzelne Teile des eID-Clients in Form von entkoppelten Komponenten entwickelt und deren aufwandsarme Wartung ermöglicht werden.

Die folgende Tabelle enthält die Komponenten, an deren Realisierung weitere Personen beteiligt waren und den Quellen zu den entsprechenden Ausarbeitungen.

IFD	[Eck10] und [Wol11]
CryptoProvider	[Eck10] und [Wol11]
Protocols	[Eck10] und [Wol11]
CardCommunicator	[Eck10] und [Wol11]
PAOS	[Mül11]

Wie in Abschnitt 3.3 beschrieben, sollen Teile der Software (z.B. die Komponenten Protocol, CardCommunicator unn CryptoProvider) als Convenience-Schnittstellen nutzbar sein. Das heißt, sie sollen anwendungsspezifische Prozesse kapseln und eigenständig durchführen können. Bei der Implementierung dieser Komponenten im Zielsystem, soll der Entwickler von diesen Prozessen kein besonderes Wissen haben müssen und nur geringen Implementierungsaufwand haben. Am Beispiel der Durchführung von PACE wird dies deutlich: Um PACE auszuführen benutzt der Entwickler die Protocol-Komponente, die ihm die Eingabe von benötigten Parametern (z.B. eID-PIN) und das starten von PACE ermöglicht. Dass dabei tatsächlich die Komponenten IFD, CardCommunicator und CryptoProvider ebenfalls beteiligt sind, bleibt verborgen. Somit reicht die Kenntnis über die Schnittstellen der Protocol-Komponente aus um eine Anwendung zu entwickeln, die PACE durchführt. Um dieses Ziel zu erreichen, müssen vor allem zwei softwaretechnische Ziele erfolgt werden. Die entwickelten Komponenten müssen jeweils über eine Application Programmable Interface (API) verfügen, also über klare Programmierschnittstellen implementierbar sein und müssen selbst eine Implementierung des eCard-API-Frameworks benutzen um mit anderen Entwicklungen interoperabel zu werden. Bei der Entwicklung spezieller Programme, hier der gesamte eID-Client, können die Komponenten so unabhängig von einer vorliegenden eCard-API-Framework-Implementierung integriert werden. Zwar wer-

den die nötigen Teile des Frameworks auch als Komponenten realisiert, jedoch ist es denkbar dass in einigen Zielimplementationen diese Teile ausgetauscht werden müssen. Z.B. wäre für eine mobile Version eines eID-Clients, die vorliegende Realisierung des IFD-Interfaces nicht brauchbar, da sie Teile von Java benutzt, die auf mobilen Endgeräten nicht unbedingt funktionieren.

4.1 Erzeugung der Klassen der eCard-API

Das eCard-API-Framework beschreibt webservicebasierte Schnittstellen und die Datentypen, welche mit Funktionen dieser Schnittstellen ausgetauscht werden können. Zur Beschreibung der Schittstellen wird die Webservice Description Language (WSDL) benutzt, welche die Funktionen und deren Datentypen XML-basiert darstellt. Die Datentypen selbst werden in einem XML Schema (XSD) definiert. Das BSI stellt eine Dateisammlung zur Verfügung, welche die Definitionen des eCard-API-Frameworks als WSDL- und XSD-Darstellung enthält. Mit einem Generierungswerkzeug, wie „wsimport“ aus einem Java Developement Kit, können automatisch Java-Klassen generiert werden, die den Funktionsumfang der Schnittstellen als Methoden besitzen. Die verwendeten fachlichen Klassen, welche die Datentypen darstellen und von den Funktionen benutzt werden, erzeugt wsimport ebenfalls. Dadurch kann die aufwändige Programmierung der eCard-API-Framework-Klassen von Hand vermieden und eine spezifikationskonforme Implementierung ermöglich werden.

Da künftig neuere Versionen des eCard-API-Frameworks zu erwarten sind und somit die Klassen den eCard-API-Frameworks erneut erzeugt werden müssen, werden dem interessierten Leser Probleme bei der Verwendung von wsimport genannt. In einer Schemadefinition mit XSD können zwei Arten der objekt-orientierten Spezialisierung erfolgen, die „extension base“ und die „restriction base“. Durch Verwendung der „extension base“ werden Klassen wie in Java spezialisiert, wobei die spezialisierende Klasse alle Attribute der generalisierenden Klasse erbt. Bei der Verwendung der „restriction base“ müssen alle zu vererbenden Attribute in der spezialisierenden Klasse erneut definiert werden und können eingeschränkt werden. Solch eine Einschränkung kann sich auf die Anzahl der enthaltenden Intsanzen eines Typs in einer Klasse beziehen, falls die Superklasse zum Beispiel eine beliebige Anzahl an Instanzen eines Objekts ent-

halten kann, die Subklasse jedoch mindestens eine enthält. Einige XSD-Dateien des eCard-API-Frameworks enthalten Vererbungen durch restriction-base, wobei an vielen Stellen durch wsimport tatsächlich keine Attribute vererbt werden. Es ist nicht ersichtlich, ob das ein Definitionsfehler innerhalb der XSD-Dateien ist oder wsimport fehlerhaft arbeitet. Durch Substituierung der entsprechenden restriction-base-Angaben mit extension-base-Angaben konnte dieser Mangel umgangen werden. Die aus den manipulierten Dateien generierten Klassen entsprechen nun den Definitionen innerhalb der technischen Richtlinien des eCard-API-Frameworks.

Außerdem enthält die Sammlung der Dateien verschiedene Versionen einzelner Teile der eCard-API, von denen wiederum einige auf verschiedene Versionen von externen (z.B. Dateien für die Verwendung von TSL) und teilweise speziell für die eCard-API manipulierten Dateien verweisen. Sämtliche Bezüge zwischen diesen Dateien wurden so aufbereitet, dass für das Anwendungsszenario brauchbare Klassen generiert werden können. So ist eine Sammlung von Dateien entstanden, welche WSDL- und XSD-basierten Definitionen der eCard-API enthalten und die für Implementierungen des eCard-API-Frameworks benutzt werden können. Innerhalb dieses Kapitels wird der Begriff eCard-API als Menge aller aus diesen Definitionen generierten Klassen verstanden.

4.2 IFD-Interface

In Teil 6 des eCard-API-Frameworks [BSI09f] wird der interne Aufbau der IFD-Schicht beschrieben. Dort ist eine komplexe Struktur beschrieben, die aus (1) dem Interface selbst, (2) einem eCard-API-IFD-Ressource-Manager, (3) verschiedenen Handlern für verschiedene Arten von Lesegeräten und (4) den Implementationen der Software für die Lesegeräte besteht (vgl. [BSI09f, S. 51]). Die hier beschriebene Implementation folgt dieser Anforderung nicht und implementiert lediglich die Funktionen der Klasse IFD aus der eCard-API, wie in Abbildung 4.1 dargestellt. Es existiert lediglich eine Klasse, welche die Funktionen des IFD-Interface realisiert. Der Funktionsumfang, wie in der technischen Richtlinie beschrieben, wird von dieser Klasse erfüllt und sie kann direkt als Typ IFD, welcher der entsprechenden WSDL-Definition entspricht, instanziiert werden. Auf den Funktionsumfang eines angeschlossenen Lesegeräts greift die IFD-Implementation durch

Verwendung des Packages `javax.smartcardio` zu. Die vorliegende IFD-Implementierung ist also nicht ohne weiteres auf andere Systemumgebungen übertragbar. Für eine mobile Variante, die beispielsweise auf ein im Smartphone integriertes Lesegerät zugreift, müsste das IFD-Interface neu realisiert werden. Um in einer späteren Weiterentwicklung die geforderte interne Architektur der IFD-Schicht zu erreichen, kann der Inhalt Klasse dann teilweise zu einem Handler (siehe oben Punkt 3) und der Implementation für javax.smartcardio aufgeteilt werden.

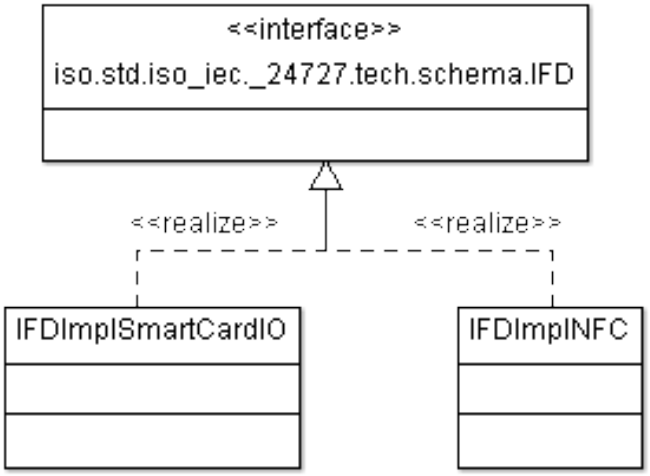

Abbildung 4.1: Klassendiagramm IFD

4.3 CardCommunicator

Um mit einem Chip zu kommunizieren werden CardCommunicator-Klassen eingesetzt. Jeder Prozess, der mit einem Chip kommunizieren soll, besitzt solch einen CardCommunicator und definiert seine Sitzung mit dem Chip über eine CardSession. In dieser CardSession wird unter anderen ein SlotHandle gespeichert, über den ein, mit einer Instanz der Klasse IFD verbundener Chip, assoziiert wird. Die CardSession selbst ist nicht Bestandteil der Komponente CardCommunicator, sondern dient als Datencontainer für den Austausch von Informationen zwischen den einzelnen Komponenten. Die CardCommunicator besitzen das Wis-

sen über die Erstellung und Interpretation der APDUs, mit denen mit dem Chip kommuniziert wird. In der eCard-API ist die Klasse `Transmit` definiert, in der APDUs gekapselt werden. Solche Transmit-Objekte werden dann an eine IFD-Instanz übergeben, das die enthaltenen APDUs an den Chip weiterleitet und anschließend die Antwort liefert. Der angesprochene Chip wird dabei über den SlotHandle der CardSession definiert wird, der ebenfalls im Transmit-Objekt gekapselt ist.

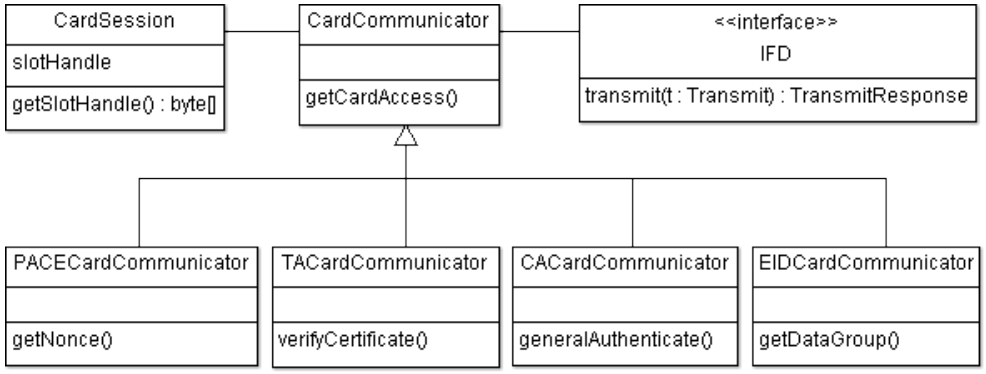

Abbildung 4.2: Klassendiagramm CardCommunicator

Anstatt einen CardCommunicator für den gesamten Funktionsumfang des Chips zu implementieren, werden spezialisierte CardCommunicator entwickelt, die genau die Funktionen des Chips benutzen können, welche entsprechend benötigt werden. Die Superklasse CardCommunicator enthält Methoden mit denen die grundlegenden Kartenkommandos gesendet werden können. Dazu gehört beispielsweise die Methode `getCardAccess()`, die vom Chip die Datei `EF.CardAccess` (vgl. [BSI10b, S. 48]) anfordert und deren Inhalt zurückgibt. Anwendungsspezifische CardCommunicator spezialisieren die Superklasse und bieten zusätzliche Methoden. Es existieren unter anderem für die Sicherheits-Protokolle PACE, TA und CA (siehe Abschnitt 2.5) und für die anschließende Durchführung von eID spezielle CardCommunicator-Klassen, wie Abbildung 4.2 zeigt. Die Namensgebung der Methoden orientiert sich hierbei an der Terminologie der technischen Richtlinie [BSI10b]. Somit können die dort beschriebenen Abläufe leicht durch Verwendung der entsprechenden CardCommunicator-Klasse abgebildet und vollzogen werden. Die Funktionalität zur Kommunikation mit einem Chip

bleibt hinter den Methodenaufrufen verborgen und ist von den oberen Schichten entkoppelt.

4.4 CryptoProvider

Sämtliche kryptografische Berechnungen werden von den Klassen der Komponente CryptoProvider durchgeführt. Zu diesen Berechnungen gehören (1) eine KeyDerivationFunction um Schlüssel herzuleiten, (2) CipherAlgorithmen, mit denen Klartexte verschlüsselt und Schiffrate entschlüsselt werden, (3) das Mapping von DomainParametern, (4) das Generieren von Schlüsselpaaren, (5) eine KeyAgreement-Funktion zur Berechnung eines gemeinsamen Geheimnisses, (6) das Berechnen von Authentication Token, (7) das Erstellen einer Signatur und (8) eine Komprimierungsfunktion. Diese Auflistung entspricht den in Abschnitt 2.5 dargestellten Tätigkeiten des Terminals während PACE, CA und TA. Abbildung 4.3 zeigt, dass zu jeder dieser Tätigkeiten ein Interface existiert. In diesen sind die notwendigen Methoden deklariert, deren Namensgebung sich an der Terminologie der technischen Richtlinie [BSI10b] orientiert. Die Klasse Cryptoprovider erhält den vollen benötigten Funktionsumfang, in dem sie die einzelnen Interfaces implementiert, allerdings ohne sie selbst zu realisieren. Die Klasse CryptoProvider stellt daher gleichzeitig eine Fassade für den gesamten Funktionsumfang dar, was einen zentralisierten Zugriff auf die verschiedenen kryptografischen Funktionen ermöglicht.

Die tatsächlichen Berechnungen werden von einzelnen Klassen durchgeführt, in denen der Funktionsumfang der Interfaces realisiert ist. In Abbildung 4.3 sind das die Klassen der unteren zwei Reihen. Zum Zeitpunkt dieser Arbeit nutzen diese Klassen die frei verfügbare Klassensammlung von BouncyCastle [BC]. Die in BouncyCastle selbst definierten Datentypen zur Kapselung der kryptografischen Parameter, dürfen außerhalb der realisierenden Klassen, also in den Interfaces, nicht verwendet werden. Andernfalls würden die der CryptoProvider stets von den Klassen von BouncyCastle abhängig sein. Daher sind alle notwendigen Parameter (z.B. `KeyPair`, `DomianParameters`) in eigenen Klassen der CryptoProvider-Komponente gekapselt. Um einen CryptoProvider zu benutzen, müssen maximal die Klasse CryptoProvider selbst und die Klassen der Datentypen bekannt sein, wodurch die Realisierung entkoppelt und leicht austauschbar

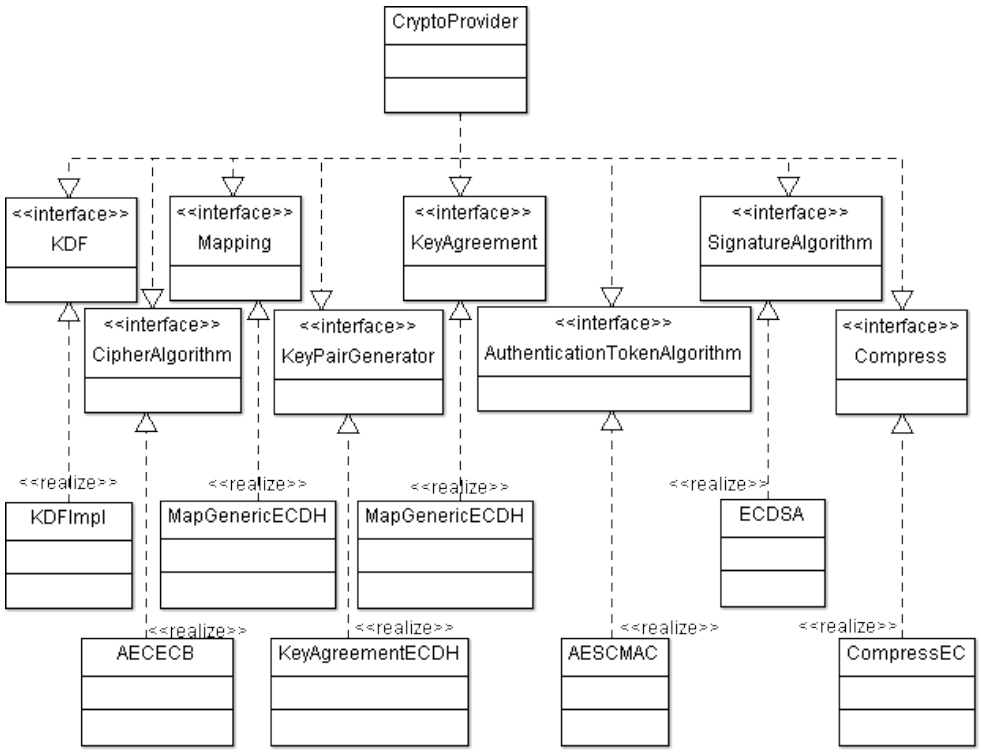

Abbildung 4.3: Klassendiagramm Crypto

wird. Für eine Implementierung auf bestimmten Zielsystemen ist eine alternative Realisierung des entwickelten CryptoProvider-Frameworks vorteilhaft oder gar notwendig. Gründe hierfür können Kompabilität oder Performance sein. Außerdem ist es möglich, dass künftige Versionen des elektronischen Personalausweises andere kryptografische Funktionen nutzen. Diese können in neuen Realisierungen der Interfaces implementiert werden.

Die Klasse CryptoProvider implementiert zum einen die Interfaces, welche die kryptografischen Funktionen bereits stellen, gleichzeitig enthält er selbst Referenzen auf Instanzen dieser Interfaces, was in Abbildung 4.3 nicht erkennbar ist. Diese Strategie vereinfacht den Erzeugungsprozess eines CryptoProviders. Zunächst werden die Objekte für die kryptografischen Funktionen erzeugt und Referenzen auf diese dem CryptoProvider übergeben. Dort werden später die Funktionsaufrufe an die realisierenden Klassen deligiert. Listing 4.1 verdeutlicht dies an einem minimalistischen Beispiel eines CryptoProviders, der aus einer PIN einen Schlüs-

sel ableiten kann. Im CryptoProvider ist das Interface KDF implementiert (Zeile 1). Bei seiner Erzeugung wird dem Konstruktor die Schlüssellänge und eine Referenz auf eine Instanz einer Klasse, die das Interface KDF realisiert, übergeben (Zeile 6). Wird von CryptoProvider nun die Methode deriveKey() aufgerufen, deligiert er diese Berechnung an die Instanz von KDF(Zeile 11-12).

```
1  public class CryptoProvider implements KDF{
2
3    private int protocolKeySize;
4    private KDF kdf;
5
6    public CryptoProvider(int protocolKeySize, KDF kdf){
7      this.protocolKeySize = protocolKeySize;
8      this.kdf = kdf;
9    }
10
11   public byte[] deriveKey(byte[] sharedSecret) {
12     return kdf.deriveKey(sharedSecret);
13   }
14 }
```

Listing 4.1: CryptoProvider

Um einen CryptoProvider zu erzeugen müssen also zunächst alle Klassen der kryptografischen Funktionen instanziiert werden. Welche realisiernden Klassen im Einzelfall benutzt werden, kann von der Systemumgebung und von der Version des elektronischen Personalausweises abhängen. Dieser komplexe Vorgang wird in der `CryptoProviderFactory` gekapselt. Um verschiedene Versionen des elektronischen Personalausweises, also verschiedene auf ihm installierte kryptografische Algorithmen zu unterstützen, werden die Dateien `EF.CardAccess` und `EF.CardSecurity` ausgelesen. Sie enthalten die Informationen, welche die Algorithmen mit ihren Schlüssellängen definieren. Diese Informationen sind als Object Identifier (OID) dargestellt. Ein OID besteht aus einer Folge von Zahlen, die hieraschisch geordnet ein beliebiges Objekt identifizieren. Um das zu verdeutlichen wird ein OID für eine in Listing 4.1 verwendete KDF gezeigt. Der OID `0.4.0.127.0.7.1.1.5.3` beispielsweise steht für eine spezielle Key-Derivation-Function. Das Präfix `0.4.0.127.0.7` steht für das BSI. Der Rest des OID, also `1.1.5.3`, wird als `Algorithmen.Elliptische Kurven.Schlüsselerzeugung.AES128` verstanden. Es handelt sich also um einen vom `BSI` vergebenen OID für einen `Algorithmus`, der auf `Elliptischen Kurven` arbei-

tet, zur `Schlüsselerzeugung` benutzt wird und `AES` mit `128` Bit verwendet. Auf diese Weise sind sowohl die Funktionen (z.B. PACE) als auch die kryptografischen Algorithmen definiert, welche der Chip beherrscht. Die OIDs sind in der Interface-Klasse `ObjectIdentifiers` statisch und öffentlich sichtbar deklariert. Zunächst werden also die Inhalte der Dateien `EF.CardAccess` und `EF.CardSecurity` interpretiert und die entsprechenden OIDs in der CardSession abgelegt. Die CryptoProviderFactory erzeugt dann anhand dieser OIDs zunächst die einzelnen Instanzen der kryptografischen Klassen und übergibt diese dann einem neu erzeugten CryptoProvider. Es gilt außerdem zu berücksichtigen, dass alternativ zu den bestehenden Implementationen, die BouncyCastle nutzen, andere Klassen entwickelt und benutzt werden können. Dafür kann eine neue Factory entwickelt oder die bestehende erweitert werden. Letzteres würde einen größeren Umfang der Software im Zielsystem bedeuten, eine einzige Version der CryptoProvider-Komponente allerdings flexibel einsetzbar machen.

4.5 Protocol

Die eigentliche Interaktion mit dem Chip führen die Klassen der Protocol-Komponente durch. Diese Klassen enthalten die anwendungsspezifischen Funktionen, um mit dem Chip zu interagieren. Insbesondere sind die einzelnen Schritte der Sicherheitsprotokolle (PACE, TA CA) dort über Methoden aufrufbar, in denen Nachrichten mit dem Chip ausgetauscht und die kryptografischen Parameter berechnet werden. Die tatsächliche Kommunikation benutzen sie Instanzen der Klasse CardCommunicator, für kryptografische Berechnungen Instanzen der Klasse CryptoProvider. Den höheren Schichten bleibt dies allerdings verborgen, so dass dort nur die Kenntnis über die Protokoll-Klassen notwendig ist.

Es existieren Protokoll-Klassen für PACE, TA, CA, eID und RI. Die eID-Anwendung ist genau genommen kein Protokoll, aufgrund der analogen Verwendung ist sie jedoch in einer Klasse der Protocol-Komponente realisiert. Die Terminologie innerhalb dieser Klassen entspricht den Definitionen in der technischen Richtlinie TR-03110 [BSI10b], in der die Sicherheitsmechanismen (siehe Abschnitt 2.5) und die Kartenapplikationen (siehe Abschnitt 2.3.1) beschrieben sind. Um den Inhalt dieser Klassen leicht verständlich zu halten, wurde die Namensgebung der Parameter und den Berechnungsprozessen in den Klassen übernommen.

Zu jeder Protocol-Klasse existiert ein CardCommunicator und ein CryptoProvider. Beispielsweise besitzt die Protokollklasse PACE einen PACECardCommunicator und einen speziell für PACE erzeugten CryptoProvider. In Abschnitt 3.4 ist beschrieben, wie diese drei Komponenten miteinander interagieren. Durch Verwendung einer sogenannten ProtocolFactory wird die Erzeugung einer Instanz von PACE mit nur einem Befehl ermöglicht. Notwendige Informationen für den weiteren Erzeugungsprozess werden mit Hilfe einer CardSession gewonnen, die man der Factory-Methode übergibt. Listing 4.2 zeigt die Methode zum Erzeugen einer PACE-Protokoll-Klasse. Innerhalb der Factory-Methode wird zunächst ein PACECardCommunicator erzeugt (Zeile 6-7), mit dem die Datei `EF.CardAccess` ausgelesen und deren Inhalt in der CardSession gespeichert wird (Zeile 9). Ein OIDInterpreter liest diese Datei und schreibt die interpretierten Paramater in die CardSession (Zeile 11). Falls der CryptoProvider für PACE erzeugt werden kann (Zeile 15), wird nun ein neues PACE-Objekt erzeugt und zurückgegeben (Zeile 21).

```
1  public class ProtocolFactory {
2
3    static public Pace createPaceProtocol(CardSession cs)
4                          throws CreateProtocolException {
5
6    PACECardCommunicator cc =
7      CardCommunicatorFactory.createPaceCardCommunicator(cardSession);
8
9    cs.setEfCardAccess(cardCommunicator.getCardAccess());
10
11   OIDInterpreter.interpret(cs);
12
13   CryptoProvider cp = null;
14   try {
15     cp = CryptoProviderFactory.getCryptoProvider(cs, CryptoProviderFactory.PACE);
16   }
17   catch (UnknownOidException e) {
18     e.printStackTrace();
19   }
20
21   return new PACE(cs, cardCommunicator, cp);
22
23   }
24   ...
25 }
```

Listing 4.2: Factory-Methode der ProtocolFactory

Durch den Einsatz der Protocol-Factory, die selbst zur Erzeugung eines Crypto-Providers und CardCommunicators deren Factorys benutzt wird und die Verwendung der CardSession als Datencontainer, bleibt der Implementierungsaufwand für die Durchführung des PACE-Protokolls gering. Um PACE in einem eID-Client durchzuführen, müssen die in Listing 4.3 dargestellten Befehle ausgeführt werden. Aus dem vom, eID-Server als Bytearray empfangenen Terminalzertifikat, wird ein Objekt vom Typ CVCertificate erzeugt (Zeile 1). Die bereits vorhandenen CardSession wird der ProtocolFactory übergeben (Zeile 3). Befor PACE gestartet werden kann, müssen das Terminalzertifikat und die PIN übergeben werden (Zeile 5-6 und 9), wobei noch definiert wird, ob nach PACE eine TerminalAuthentication folgt (Zeile 7) und welcher Schlüssel benutzt wird (Zeile 10) (siehe Abschnitt 2.5.2). Zur Durchführung von PACE muss nun die Methode `doPace()` aufgerufen werden (Zeile 11). Nach erfolgreicher Durchführung enthält die CardSession nun die Information, dass der Chip sich im Modus Secure Messaging befindet, sowie das entsprechende Schlüsselmaterial. Der für die folgende TerminalAuthentication komprimierte öffentliche Schlüssel kann danach aus der PACE-Instanz selbst ausgelesen werden (Zeile 13).

```
1  CVCertificate terminalCert = new CVCertificate(certContent);
2
3  PACE pace = ProtocolFactory.createPaceProtocol(cardSession);
4
5  pace.setCertificateHolderAuthorizationTemplate(
6          terminalCert.getHolderAuthorizationTemplate(true));
7  pace.setFollowingTerminalAuthentication(true);
8  pace.setKeyType(Pace.PIN);
9  pace.setPin(pin);
10
11 pace.doPace();
12
13 byte[] idPicc = pace.getIDpicc()
```

Listing 4.3: Benutzung der Klasse PACE

Im Gegensatz zu PACE werden die TA- und CA-Protokolle zwischen dem Remote-Terminal und dem Chip ausführt. Die tatsächliche Kommunikation mit dem Chip übernimmt i.d.R. jedoch das Local-Terminal. In Abschnitt 3.4.4 ist beschrieben,

wie diese Protokolle verteilt durchgeführt werden. Die Protokollklassen werden so gestaltet, dass sie auf einem Local-Terminal und einem Remote-Terminal eingesetzt werden können. Dadurch wird ihre Verwendung als Convenience-Schnittstelle ermöglicht, kapseln also die komplexen Prozesse während der Protokoll-Durchführung um über schmale Schnittstellen in veschiedenen verschiedenen Systemen verwendet werden können. In Listing 4.3 wird in Zeile 11 durch einen einzigen Befehl PACE durchgeführt. Die Klassen TerminalAuthentication und ChipAuthentication besitzen analog dazu die Methoden `doTA()` bzw. `doCA()`. So könnte ein lokales Terminal mit einem Chip die EAC-Protokolle durchführen, tatsächlich macht das allerdings ein Remote-Terminal und wird lediglich vom lokalen Terminal unterstützt. Für diesen verteilten Einsatz enthalten diese Klassen weitere Methoden, die jeweils die Schritte durchführen, die auf der entsprechenden Seite benötigt werden. Auf dem Local-Terminal wird dabei mit Hilfe eines CardCommunicators die tatsächliche Kommunikation mit dem Chip durchgeführt. Das Remote-Terminal führt hingegen lediglich die kryptografischen Berechnungen mit Hilfe des CryptoProviders durch. Die Ergebnisse der Kommunikation mit dem Chip bzw. der kryptografischen Berechnungen können dann von den Protokollklassen abgefragt werden.

4.6 EACWorker

Der EACWorker übernimmt bei der verteilten Protokolldurchführung auf lokaler Seite die Vermittlung zwischen dem Remote-Terminal und den lokalen Protocol-Klassen. Im eCard-API-Framework ist im ISO24727-3-Interface die Funktion DIDAuthenticate beschrieben [BSI09d, S. 86], welche das Local-Terminal dem Remote-Terminal zur Verfügung stellt. Mit dieser Funktion kann das Remote-Terminal das Local-Terminal zur Durchführung der EAC-Protokolle auffordern und Parameter die vom Chip erwartet werden (z.B. das Berechtigungszertifikat) übertragen. Parameter vom Chip an das Remote-Terminal (z.B. eine Zufallszahl) werden in der Antwort dieser Funktion vom lokalen Terminal übertragen. Die Verwendung von DIDAuthenticate ist in Teil 7 des eCard-API-Frameworks beschrieben [BSI09g, S. 45ff]. Der EACWorker realisiert die Funktion DIDAuthenticate des Local-Terminals und führt abhängig von den übergebenen Parametern entweder PACE oder die beiden Protokolle TA und CA durch. In Abbildung 4.4 ist erkenn-

bar, dass nur seine Methode `didAuthenticate()` öffentlich sichtbar ist. Das übergebene gleichnamige Objekt `DIDAuthenticate` der eCard-API enthält ein Objekt des Typs `AuthenticationProtocolData`, welches in einer von drei möglichen Spezialisierungen verarbeitet wird. Anhand des speziellen Typs werden die Methoden `processEAC1InputType()`, `processEAC2InputType()` oder `processEACAdditionalInputType()` augerufen, was der Überschaubarkeit des Codes dient. Erstere sorgt für die Durchführung von PACE, letztere beiden für die Durchführung der TA und der CA, wobei `EACAdditional-InputType()` nur notwendig ist, falls der Chip nicht fähig ist bereits vor der TA die Challenge für die Signatur zu übertragen und das in einer eigenen Nachricht nachgeholt werden muss (vgl. Abschnitt 2.5 und [BSI09g, S. 47-51]). In diesen Methoden werden durch Verwendung der ProtocolFactory die Instanzen der Protokollklassen erzeugt und die Protokolle durchgeführt. Für die Kommunikation mit dem Chip nutzen diese Klassen eine zuvor übergebene Instanz der IFD-Interface-Implementierung. Das Remote-Terminal übergibt mit dem `DID-Authenticate-Objekt` ein Objekt vom Typ `ConnectionHandle`, das u.a. den `SlotHandle` der anzusprechenden Karte kapselt (vgl.[BSI09d, S. 86 u 24]). Aus einem `CardSessionManager`, der die aktuell existierenden `CardSession`s verwaltet, wird diejenige referenziert, die diesem `SlotHandle` entspricht. Da die Protokolle während zwei bzw. drei Aufrufen durch das Remote-Terminal durchgeführt werden, müssen die Ergebnisse der Protokolle zwischen der Laufzeit der Methodenausführungen vorrätigt gehalten werden. Die Klasse `EACSession` kapselt diese Ergebnisse.

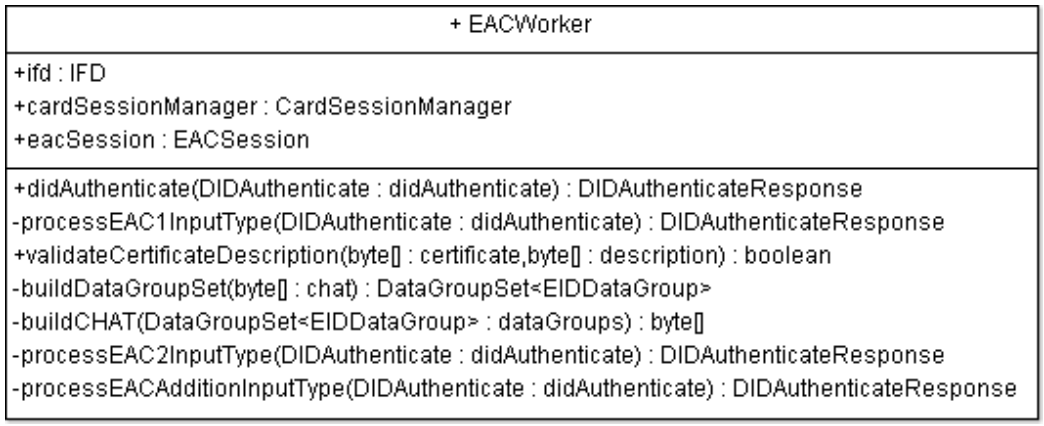

Abbildung 4.4: EACWorker

Bevor die EAC-Protokolle durchgeführt werden, muss der Nutzer der Zertifikatsbeschreibung des Dienstanbieters zustimmen und die auszulesenden Datengruppen angeben. Die Authentizität der Zertifikatsbeschreibung können Instanzen von `EACWorker` mit ihrer Methode `validateCertificateDescprition()` prüfen. Die Datengruppen werden vom Dienstanbieter in einem sogenannten Certificate Holder Authorization Template (CHAT) übermittelt. Aus wird dann ein Objekt vom Typ DataGroupSet erzeugt, der den höheren Schichten bekannt ist. Die durch den Benutzer festgelegten Datengruppen werden mit der Methode `buildCHAT()` wiederum in ein CHAT umgewandelt.

Mit der Klasse EACWorker ist der letzte Teil der Convenience-Schnittstelle realisiert. Im eID-Client kann sie nun genutzt werden, um die EAC-Protokolle mit dem Chip und dem eID-Server durchzuführen zu lassen.

4.7 Logging-Komponente

Die Logging-Komponente stellt einen Mechanismus bereit um Informationen über Aktivitäten, wie das Senden von Kartenkommandos oder ein Verbindungsaufbau zu einem eID-Server und deren Ergebnisse zu dokumentieren. Dies soll jedoch möglichst unabhängig von den internen Abläufen der Prozesse eines eID-Clients geschehen. Also musste ein Konzept entworfen werden, dass es den Prozessen ermöglicht, diese Informationen gesondert bereit zu stellen. Dabei sollten die notwendigen Änderungen an bestehendem Quellcode jedoch so gering wie möglich gehalten werden. Bei der Auswahl der Realisierung wurde zwischen drei Möglichkeiten abgewägt.

Als erste Möglichkeit wurde die Realisierung durch das, in Java bereits verfügbare, Observer-Pattern betrachtet, welches es erlaubt, dass Objekte, andere Objekte über Änderungen informieren. Dazu wird in einer beobachtenden Klasse das Observer-Interface implementiert und beobachtete Klassen spezialiseren die Superklasse Observable. Observer können bei Observables registriert und später über Änderungen benachrichtigt werden. Sämtlichen Klassen, die Prozesse ausführen oder relevante Daten der Prozesse kapseln, würden die Klasse Observable spezialisieren. So könnten Klassen, in denen das Obersever-Interface implementiert worden wäre, über Änderungen informiert und mit Daten versorgt werden

oder sich diese nach einer Änderung, welche die beobachtende Klasse gemeldet hat, selbst beschaffen. Der Verwaltungs- und Implementationsaufwand dafür wäre jedoch im Gesamten sehr hoch gewesen. Jeder Observer (beobachtende Klasse) hätte bei jedem Observable (beobachtete Klasse) registriert werden müssen. Außerdem hätten die Observer in einigen Fällen umfangreiche Kenntnis über die Observables haben müssen, um die Daten aus ihnen auszulesen und deren Inhalt im richtigen Kontext zu verstehen.

Eine weitere Möglichkeit wäre die Verwendung der Listener-Strategie. In diesem Fall implementieren Klassen, die auf Ereignisse reagieren sollen, ein Listener-Interface. Andere Objekte können Instanzen dieser Klassen verwalten und sie über die Interface-Methoden dann über die Ereignisse informieren. Da Listener an einigen Stellen ohnehin verwendet werden, beispielsweise um beispielsweise Elemente der GUI über Ereignisse zu informieren, hätten diese auch für das Logging verwendet werden können. Wie bei der Möglichkeit der Realisierung mit Observern, wäre auch hier der Verwaltungsaufwand sehr hoch gewesen, da die Listener sich ebenfalls bei allen relevanten Klassen hätten registieren müssen, um über Ereignisse informiert zu werden. Außerdem wäre so eine klare Trennung zwischen den prozesseigenen Datenflüssen und dem Logging-Mechanismus nicht gewährleistet gewesen.

Um Log-Nachrichten mit möglichst wenig Implmentierungsaufwand und flexibel senden und empfangen zu können, wurde also ein eigenes Konzept entwickelt. Es orientiert sich am Observer-Pattern, aber mit erweitertem Umfang und vereinfachtem Gebrauch. Es sollen Nachrichten der internen Abläufe übermittelt werden, falls beispielsweise ein Objekt der Protokoll-Komponente einen neuen Parameter vom Chip empfangen hat oder eine Klasse der IFD-Komponente Nachrichten mit den Chip austauscht. Die Vermittlungsstrategie ist in Abbildung 4.5 dargestellt. Zentrale Stelle ist der `LoggerManager`, der alle Nachrichten empfängt. Objekte, die Nachrichten an den `LoggerManager` senden sollen, erben von der abstrakten Klasse `Loggable`, die eine Referenz auf den `LoggerManager` und eine Methode `sendMessage(RosecatMessage message)` enthält, welche eine Nachricht an den `LoggerManager` übertragt. Um zu vermeiden, dass während der Laufzeit Referenzen auf einen `LoggerManager` übergeben werden müssen, ist er nach dem Singleton-Entwurfsmuster (vgl. [Dun03, S. 43]) implementiert. Seine Funktionsweise basiert auf einer statischen Referenz auf sich

selbst, innerhalb des LoggerManagers. Durch die statische Methode getInstance(), welche diese Referenz zurückgibt, kann er so direkt referenziert werden. Im Gegensatz zum Oberserver-Pattern kann eine Klasse, die von Loggable erbt, mit einem einzigen Befehl Nachrichten übertragen und muss außerdem keine Referenzen auf beobachtende Objekte verwalten. Klassen, die Nachrichten empfangen sollen, erben von der abstrakten Klasse Logger und müssen die abstrakte Methode processMessage(RosecatMessage message) implementieren. Die Klasse Logger enthält außerdem die Methoden addMessageType(RCMType type) und removeMessageType(RCMType type), mit denen definiert werden kann welche Arten von Nachrichten (z.B. die Protokollausführung betreffend) berücksichtigt werden und welche gefiltert werden sollen. Abbildung 4.5 liegt folgende Annahme zugrunde: LoggerA ist an Nachrichten vom Typ typeA interessiert, LoggerB sowohl an Nachrichten vom Typ typeA als auch vom type typeB. Beide wurden zuvor beim LoggerManager registriert. Die Klassen LoggableA und LoggableB senden beide Nachrichten, indem sie die Methode receiveMessage() des LoggerManagers aufrufen. Danach werden sie an alle registrierten Logger weitergeleitet, die diese nach dem Nachrichtentyp filtern und in der Methode processMessage() dann verarbeiten, falls sie an Nachrichten des entsprechenden Typs interessiert sind. Diese Methode ist in den Spezialisierungen der Klasse Logger zu implementieren.

Die Arten der Nachrichten sind in hierarchisch strukturierte Enumerations definiert. Das sind spezielle Klassen in Java, in denen Objekte gleichen Typs erzeugt und letzlich durch statische Referenzen global sichtbar bemacht werden können. Die Enumeration RCM enthält eine vollständige Auflistung der Nachrichtentypen, die gesendet werden können. Jedes Enumeration-Objekt enthält außerdem Referenzen auf weitere Enumeration-Objekte, welche definieren, von welchem Typ die Nachricht ist, optional auf welches Protokoll (PACE, TA, CA etc.) sie sich bezieht, von wem sie erwartungsgemäß gesendet wurde und welchen Identifier sie hat. Beispielsweise existiert das Objekt PACE_PCD_pi vom Typ RCM, welches durch den Nachrichtentyp RCMType.PROTOCOL, das die Nachricht betreffende Protokoll RCMProtocol.PACE, des Teilnehmers dieses Protokolls RCMParty.PCD und einem Identifizierer RCMID.pi definiert wird. Ein Loggable kann durch Aufruf seiner Methode sendMessage(PACE_PCD_pi, „123456“) mitteilen, dass es gerade PACE durchführt, das lokale Terminal ist und Kenntnis über die PIN mit dem Wert „123456“ erhalten hat. Durch get-Methoden in RCM,

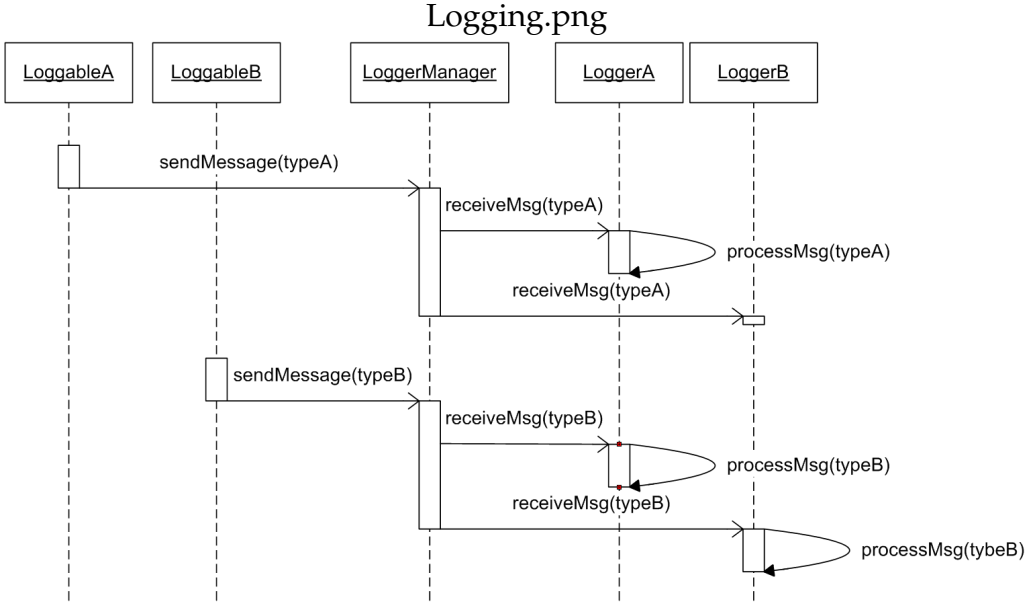

Abbildung 4.5: Nachrichtenvermittlung beim Logging

welche die gerade genannten Parameter zurückgeben, können diese Informationen dann von einem `Logger` gewonnen werden.

Die Integration des Logging-Mechanismus in den informationserzeugenden Klassen, also den `Loggables`, erfordert tatsächlich wenig Änderungen am Quelltext dieser Klassen. An jeder Stelle im Quelltext, an der eine Nachricht gesendet werden sollen, reicht die Verwendung eines befehls aus. Die Realisierung eines `Loggers` allerdings erfordert einen höheren Implementierungsaufwand, da er in der Lage sein muss, die empfangenen Nachrichten zu auswerten zu können. Die Verwendung von hierarchisch geordneten Enumerations erweist sich in diesem Zusammenhang als effizient. Eine Nachricht kann durch wenige Schritte identifiziert werden, wodurch der Quelltext in den Spezialisierungen von `Logger` übersichtlich bleibt.

Wie die Informationen der Nachrichten verarbeitet werden sollen, entscheidet sich anhand der Implementierung der Methode `processMessage()`. Falls die Informationen der Nachrichten textuell verarbeitet werden sollen, also als Abbildung der Nachricht auf einen String, bieten die Klasse `TextualLogger` und das Interface `TextualLogConsumer` die Möglichkeit einen automatisch generierten Log-Text verfassen zu lassen. Dies ist zum Beispiel für das fortwährende

Erstellen von Log-Dateien sinnvoll. Um die Semantik der Nachrichten auf menschenlesbare Strings abzubilden, existieren `MessageDictionary`s. Diese ordnen eine Instanz der Enumeration `RCM`, einem String zu. Der `TextualLogger` besitzt einen `MessageTranslator`, den er zur Übersetzung der Nachrichten benutzt und dem ein `MessageDictionary` zugewiesen werden kann. Um die `MessageDictionary`s ohne weiteren Aufwand erzeugen zu können, wird eine `DictionaryFactory` benutzt. Diese erzeugt aus Properties-Dateien, die Schlüssel-Wert-Paare enthalten, ein `MessageDictionary`. Die Properties-Dateien können unabhängig von einer Implementation oder einer bereits installierten Software erzeugt werden. Die Klasse, welche die menschenlesbaren Nachrichten schließlich verarbeiten soll, muss die beiden Methoden `setTextualLogger(TextualLogger logger)` und `consumeLogText(String log)` implementieren und einen `TextualLogger` zugewiesen bekommen. Im Falle einer fortwährenden Protokollierung könnte der übergebene String dann in eine Datei geschrieben werden.

4.8 GUI (Graphical User Interface)

Bei der Implementierung einer grafischen Oberfläche ist in erster Linie darauf zu achten, dass die Prozesse von ihrer Visualisierung getrennt werden. Die Prozesse sind hierbei als die Abläufe innerhalb der Anwendungsschicht und nicht als die Interaktion zwischen der der Software und dem Benutzer zu verstehen. Grundlegend kann man sagen, dass der Benutzer durch eine Eingabe einen Prozess der Anwendungsschicht auslöst und dessen Ergebnisse nach Prozessbeendigung visualisiert werden. Falls die Prozesse der Anwendungsschicht so komplex sind, dass sie während ihrer Ausführung weitere Benutzereingaben erfordern, gestaltet sich diese Koordination aufwändiger. Diese Koordination, auch Dialogkontrolle genannt, übernehmen daher Komponenten, die zwischen dem Benutzer und der Anwendungsschicht vermitteln. Diese Vorgehensweise entspricht dem Seeheim-Modell [Vos98]. Die Implementierung folgt dem „Konzept für den Entwurf der Präsentitaionsschicht“ aus [Dun03].

Die Komponente „GUI“ gliedert sich in die Bestandteile Präsentation, Dialogkontrolle und Anwendungsschnittstelle. Diese Aufteilung wird in Abbildung 4.6, die einige Teile der GUI zeigt, verdeutlicht. Der Benutzer interagiert über die Be-

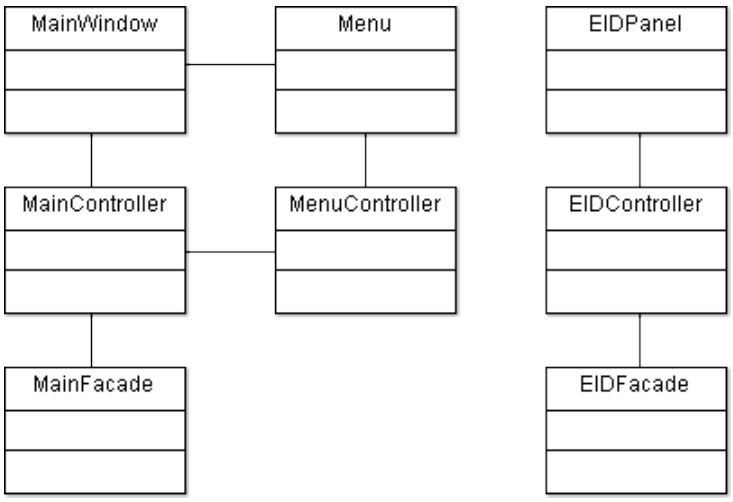

Abbildung 4.6: GUI

standteile der Präsentation mit der Anwendung. Im Beispiel sind dies das Hauptfenster (`MainWindow`), ein Menü (`Menu`) und eine Oberfläche für die Durchführung von eID (`EIDPanel`). Jede dieser Präsentations-Komponenten besitzen einen eigenen Controller, denen eigene Fassadenklassen zugewiesen werden. Der `MenuController` hingegen greift auf Funktionen des `MainController`s zu, da seine Aufgabe lediglich das Öffnen von neuen Darstellungs-Komponenten ist. Diese Darstellungs-Komponenten können allerdings auch ohne Benutzereingabe erscheinen, wodurch dies die Aufgabe des `MainController`s wird. Die einzelnen Controller-Komponenten können so übersichtlich gehalten werden und sind für künftige Änderungen wartbar. Löst ein Benutzer nun ein Ereignis, z.B. in Form eines Mausklicks auf eine Schaltfläche, aus, erkennen die Präsentations-Komponenten dieses Ereignis selbst und rufen eine entsprechende Methode ihres Controllers auf. Der Controller entscheidet dann, welchen Prozess er mit Hilfe seiner Fassaden-Klasse in der Anwendungs auslöst. Durch die Rückgabe der Methoden der Fassaden-Klassen erhalten die Controller die Ergebnisse der Prozesse, welche sie in den Präsentations-Komponenten in aufbereiteter Form darstellen können.

Die Fassadenklassen bieten genau den Funktionsumfang, welcher benötigt wird um Benutzereingaben zu verarbeiten und die internen Prozesse zu starten oder laufende Prozesse mit weiteren Informationen zu versorgen. Diese Fassadenklassen sind die Schnittstellen zur Anwendung und sind jeweils für die entsprechen-

den Anwendungsfälle entworfen. Dadurch kann die gesamte Schnittstelle zur Anwendung übersichtlich gehalten werden und die einzelnen Teile der Dialogkontrolle erhalten genau den Umfang an Funktionszugriff, den sie benötigen.

Eine weitere Möglichkeit Benutzereingaben zu verarbeiten ist die Verwendung des Befehlsentwurfsmusters, das ebenfalls in [Dun03, S. 273] beschrieben ist. In Java ist dieses Entwurfsmuster mit dem Action-Mechanismus realisiert. Schaltflächen können Instanzen der Klasse `AbstractAction`, welche die abstrakte Methode `actionPerformed()` enthalten, zugewiesen werden. Sobald die Schaltfläche gedrückt wird, wird diese Methode aufgerufen. Außerdem können den Action-Klassen beliebig viele Parameter in Form von Schlüssel-Wert-Paaren übergeben werden, die sie später verarbeiten. Vor allem komplexere Abläufe, die innerhalb der Controller zu umfangreichem Quelltext führen würden, können so in einer eigenen Klasse gekapselt werden und verschiedenen Stellen verwendet werden, ohne den Funktionsumfang in verschiedenen Controllern implementieren zu müssen.

```
1  public class MonitorTab{
2
3    ...
4
5    storeButton = new JButton();
6    storeButton.setText("Log␣in␣Datei␣speichern");
7    storeButton.setAction(controller.getActionFactory().getStoreLogAction());
8
9    storeButton.addActionListener(new ActionListener() {
10     public void actionPerformed(ActionEvent e) {
11       doStoreLogActionPerformed(e);
12     }
13   });
14
15   ...
16
17   public void doStoreLogActionPerformed(ActionEvent e) {
18     Object src = e.getSource();
19     Action action = ((JButton) src).getAction();
20     action.putValue(LogActionKey.LOG_CONTENT.getName(), logger.getLog());
21     action.putValue(LogActionKey.RCM_TYPES.getName(), logger.getMessageTypes());
22   }
23 }
24
25 public class StoreLogAction extends AbstractAction {
26
```

```
27    ...
28
29    public void actionPerformed(ActionEvent ae) {
30      String content = (String) getValue(LogActionKey.LOG_CONTENT.getName());
31      Set<RCMType> types = (Set<RCMType>) getValue(LogActionKey.RCM_TYPES.getName());
32      facade.storeLogFile(content, types);
33    }
34  }
```

Listing 4.4: Verwendung des Befehlsentwurfsmusters

Listing 4.4 zeigt die Verwendung einer Action-Klasse zur Speicherung eines Textes, der durch den Logging-Mechanismus erzeugt wurde. In Zeile 7 wird eine neue `StoreLogAction` einem `Button` zugeordnet. Eine `AbstractAction` implementiert das Interface `ActionListener`. Nach dem Aufruf in Zeile 7 fügt der `Button` die `StoreLogAction` sich selbst hinzu. Danach wird in Zeile 9-13 ebenfalls ein neuer `ActionListener` dem Button übergeben, der nur die Aufgabe hat die Methode `doStoreLogActionPerformed` aufzurufen. Diese Methode setzt in den zeilen 20-21 zwei Parameter im `Action`-Objekt des `Button`s. Sobald nun der `Button` gedrückt wird, werden die `ActionListener` in umgekehrter Reihenfolge aufgerufen, in denen sie dem `Button` zugeordnet wurden. Also wird zuerst die Methode in Zeile 17 aufgerufen und dann die Methode der `StoreLogAction` in Zeile 29, welche die zuvor gesetzten Parameter verarbeitet und die entsprechende Fassaden-Methode in Zeile 32 aufruft.

Kapitel 5

Beitrag zur weiteren Entwicklung eines Open-Source-eID-Clients

Durch die Einführung des neuen Personalausweises können in Zukunft viele Vorgänge des täglichen Lebens vereinfacht werden. Beispielsweise kann ein Identitätsnachweis gegenüber einer Bank oder einer Versicherung direkt online erfolgen. Bislang war dies nur durch einen augenscheinlichen Vergleich zwischen dem Passbild auf einem Personalausweis und dem Gesicht der Person möglich. Die Herausforderung für eine erfolgreiche Etablierung setzt allerdings die technischen Möglichkeiten und die Akzeptanz der Bürger voraus. Ist mindestens eines dieser beiden nicht gegeben und somit der zu erwartende Benutzerkreis eingeschränkt, sinkt zugleich das Interesse seitens der Dienstanbieter. Die technischen Voraussetzungen sind die Verfügbarkeit eines eID-Clients und der Besitz eines elektronischen Personalausweises sowie eines Lesegeräts. Aufgrund der Gültigkeit eines Personalausweises von maximal zehn Jahren, sind voraussichtlich erst im November 2020, zehn Jahre nach der Einführung, alle Bundesbürger mit einem elektronischen Personalausweis ausgestattet. Die Verfügbarkeit von Lesegeräten ist nur bedingt gewährleistet, da vor allem die Standardleser, welche als sicherer anerkannt werden, durch ihre Anschaffungskosten eine Kaufentscheidung hemmen können. Die Software „AusweisApp“ stand bereits mehrfach in der Kritik. Ein, von der Mehrheit anerkannter eID-Client, kann die Akzeptanz steigern und zu einer höheren Anzahl an Bürger die eID nutzen und somit zu einer schnelleren Verbreitung von eID-Anwendungen führen.

Diese Arbeit zeigt die programmiertechnischen Aufgaben, die bei der Entwicklung eines eID-Clients zu erfüllen sind. Teile des eCard-API-Frameworks wurden erfolgreich implementiert und mit den fachlichen Klassen des Frameworks, welche dem Austausch von Informationen dienen, die Prozesse während eID durchgeführt. Es war vor allem möglich, die einzelnen Aufgaben der Software derart zu identifizieren und möglichen Anwendungen zuzuordnen, dass wiederverwandbare Module enstanden sind, die auch in anderen Umgebungen, als in einem eID-Client, verwendet werden können. Ein ähnliches Projekte könnte z.B. eine Software für die Gesundheitskarte sein. In diesem Fall müsste ebenfalls mit einer Karte kommuniziert und Authentifzierungsprotokolle durchgeführt werden, was die bestehende Implementierung, ggf. durch Erweiterung, realisieren kann. Für die speziellen Anwendungen der Gesundheitskarte können die Kartenkommandos in einem neuen CardCommunicator definiert werden. Die bestehenden Komponenten müssen also vielmehr um Wissen erweitert werden, als dass tatsächlich neue Entwicklungen notwendig sind. Außerdem wurden vielen Komponenten so entworfen, dass sie auf der Clientseite und der Serverseite verwendet werden können.

Leider war es im Rahmen dieser Arbeit nicht möglich die Kommunikation über ein Netz zu durchzuführen. Die voll funktionsfähige Implementierung eines über PAOS kommunizierenden Webservice, der TLS-Verbindungen zu einem Server aufbaut, konnte aus Zeitgründen nicht beendet werden. Der Nachrichtenaustausch konnte allerdings vom sogenannten EACWorker abgebildet werden, der die entsprechenden Webservicefunktionen bereits realisiert. Da der EACWorker alle anderen, an der Durchführung der Authentifikationsprotokolle beteiligten, Komponenten selbst erzeugt und verwaltet, bleibt der Implementationsaufwand möglichst gering. Die Implementierung eines EACWorkers für einen eID-Server kann sich später an der vorliegenden Implementierung orientieren.

Die in Abschnitt 3.1 erarbeiteten relevanten Architekturanforderungen sind erfüllt. Die unteren Schichten Protocol, CryptoProvider und CardCommunicator sind vollständig entkoppelt, was die Wartbarkeit vereinfacht. Gleichzeitig finden die sicherheitsrelevanten Berechnungen in einem einzigen Subsystem, nämlich dem CryptoProvider statt, wodurch sie austauschbar und validierbar sind. Berücksichtigt man den potentiellen Einsatz solch einer Software auf mobilen Endgeräten, gewinnen die erfüllten Anforderungen an Gewicht, da gerade dort teil-

weise auf andere Techniken als bei einem Desktop-PC zurückgegriffen werden muss. Dies betrifft die Komponente CryptoProvider, welche durch die hier verwendeten Bibliotheken für kryptografische Berechnungen, in der jetzigen Form auf einem mobilen Endgerät nicht effizient genug arbeiten würde. Durch die aufwändige Interface-Struktur kann ihre Realisierung alternativ erfolgen, die Verwendung bleibt jedoch die selbe.

Der geforderte Demonstrationsmodus ist durch den Entwurf eines Logging-Konzepts ermöglicht. Die realisierten Komponenten nutzen diesen bereits zum Senden von Nachrichten über Ereignisse und Komponenten der grafischen Oberfläche benutzen diese Nachrichten um die Ereignisse visuell darzustellen. In Zukunft könnte dieser Logging-Mechanismus für einen Observationsmodus relevant sein, der den Nutzer über seinen Privacystatus informiert. Unter Privacystatus wird die Menge an persönlichen Informationen verstanden, die ein Bürger von sich preisgegeben hat und wem er sie preisgegeben hat. Dieses Bewusstsein hilft dem Bürger bei der Wahrung seines Rechts auf informationelle Selbstbestimmung.

Sollte sich ein Open-Source-Projekt zur Entwicklung eines Open-Source-eID-Clients durchsetzen, sind mit den Ergebnissen dieser Arbeit viele Grundlagen dafür geschaffen. Die modular einsetzbaren Komponenten, wurden im Software-Projekt Rosecat integriert und durch Schnittstellen einer ebenfalls implementierten grafischen Benutzeroberfläche verfügbar gemacht. Auch wenn Rosecat in seiner Gesamtheit nicht als Referenzimplementierung dienen sollte, sind trotzdem einzelne Komponenten hochgradig genug entkoppelt um unabhängig voneinander verwendet zu werden. Erste Schritte für ein Open-Source-Projekt sind durch interessierte Entwickler [1] bereits zu Stande gekommen. Ähnliche Softwareprojekte finden aus diesem Personenkreis beispielsweise an der Technischen Universität Darmstadt und der Fachhochschule Coburg statt. In diesem Rahmen sind bereits Kooperationen und der Austausch von Programmteilen in Planung. Vor allem für das Verständnis des eCard-API-Frameworks zeigte sich der Informationsaustausch als sehr nützlich, da einige Inhalte offenbar auf unterschiedliche Weise interpretiert wurden und Fehlinterpretationen so behoben werden konnten. Auch die Kommunikation mit Unternehmen zeigte, dass dort ebenfalls Interesse an einem Open-Source-eID-Client besteht. In einigen Fällen profitierten

[1] http://openecard.de

bereits beide Parteien von einem Informations- und Ressourcenaustausch. Diese alles motiviert dazu, einen eID-Client auf Basis eines Open-Source-Projektes zu entwickeln. Die hier vorgestellte Arbeit zeigt die ersten Schritte in diese Richtung.

Glossar

Age Verification

Ein Protokoll, mit dem der Personalausweis einen Test auf Gleichheit oder Ungleichheit bezüglich des Geburtsdstum durchführen kann. Das tatsächliche Alter und das Geburtsdatum werden bei Prüfung auf Ungleichheit („größer als“ oder „kleiner als“) nicht preisgegeben..

APDU

Eine Application Protocol Data Unit ist eine Folge von Bytes. Diese Bytefolge stellt eine Kommunikationseinheit bei der Chipkartenkommunikation dar. Es existieren command APDUs, die Kommandos an die Chipkarte enthalten und response APDUs, die Antworten der Chipkarte enthalten..

Berechtigungszertifikat

Dieses Zertifikat erhalten Dienstanbieter, die den eID-Service nutzen möchten. Es enthält eine Beschreibung des Dienstes in textueller Form und eine Bitmaske, die definiert welche Datengruppen ausgelesen werden dürfen..

Certificate Holder Authorization Template

Ein Bytearray in Datengruppen definiert sind, die der Dienstanbieter auslesen möchte oder der Benutzer zum Auslesen freigegeben hat..

Country Signer Certification Authority

Eine solche Zertifikats-Autorität stellt die Document-Signer-Zertifikate her, die von den Ausweisherstellern zum Signieren der Daten im Ausweis benutzt werden..

Country Verifier Certification Authority

Eine solche Zertifikatsautorität stellt Berechtigungszertifikate her. Für jede

Kartenapplikation existiert eine eigene CVCA.

EF.CardAccess
In dieser, auf auf einer eCard gespeicherten Datei, sind die Informationen zu Kartenapplikationen und verwendbaren Sicherheitsmechanismen enthalten.

EF.CardSecurity
In dieser, auf auf einer eCard gepsiecherten Datei, sind die Informationen zum statischen Schlüsselmaterial enthalten. Sie kann erst nach erfolgreich durchgeführter Terminal Authentication ausgelesen werden..

eID
Mit diesem Verfahren kann sich ein Bürger gegenüber einem Dienstanbieter online authentifizieren..

Extended Access Control
Erweiterte Zugriffskontrolle der Daten in einem Ausweis, unter Verwendung der Protokolle PACE, TA und CA.

Kartenapplikation
Eine Anwendung auf einer Chipkarte. Die eID-Applikation auf dem elektronischen Personalausweis beispielsweise enthält die Logik, um die Protokolle für eID und die notwendigen Berechnungen dafür durchzuführen..

Komfortleser
Ein Lesegerät mit eigenem Tastaturfeld, das selbst in der Lage ist PACE durchzuführen.

Object Identifier
Eindeutige Bezeichnungen der Fähigkeiten einer eCard. Kryptografische Algorithmen und Kartenapplikationen werden in den Dateien EF.CardAccess und EF.CardSecurity definiert.

PAOS
PAOS ist ein Protocol zur Nutzung eines cleintseitigen Webservice und ist in der „Liberty Reverse HTTP Bining for SOAP Specification definiert“. Ein

Client ist in der Regel durch Firewall oder andere blockierende Mechanismes nicht ohne weiteres erreichbar. Durch das „Verpacken“ von Webservice-Anfrage in http-Antworten erreichen die Anfragen die Instanz auf der der Webservice läuft. Dazu muss jedoch zunächst eine http-Anfrage gesendet werden, die dem Aufrufenden mitteilt, dass die nächste http-Antwort als eine solche Webservice-Anfrage interpretiert werden kann..

Passive Authentication

Der Chip enthält ein Document Security Object. Dieses enthält eine Signatur, um die Authentizität der Inhalte der Datengruppen prüfen zu können..

Pre Shared Key

Ein zuvor ausgehandelter Schlüssel, den zwei oder mehrere Parteien nutzen können um kryptografische Protokolle durchzuführen.

Public Coupling Device

Bezeichnung der Leseeinheit, also des RFID-Lesegeräts, nach der Norm ISO/IEC 14443.

Restricted Identifcation

Die Restricted Identification ist ein Teil der eID-Kartenapplikation und dient dem Altersnachweis eines Bürgers. Dabei wird jedoch eine Altergrenze geprüft ohne das Geburtsdatum selbst preiszugeben..

Secure Messaging

Sobald zwischen Terminal und Chip ein gesicherter Kanal etabliert ist, wird die Kommunikation als Secure Messaging bezeichnet. Die APDUs werden dabei verschlüsselt und signiert zwischen Chip und Terminal übertragen..

Secure Sockerts Layer

Protokoll zur Absicherung von Kommunikationskanälen im Internet, oberhalb der TCP-Schicht.

SOAP

Netzwerkprotokoll zum Austausch von Daten im XML-Format. Insbesondere Webserviceanfragen und -antworten können mit SOAP verfasst werden..

Terminal

Der Begriff Terminal wird synonym zu Public Coupling Device verwendet. Er beschreibt ein lokales Terminal, z.B. in Form einer Client-Software oder ein entferntes Terminal, z.B. in Form eines eID-Servers..

Terminal Authentication

Die "Terminal Authentication" dient der Authentifizierung eines Terminals, das mit dem Personalausweis kommunizieren möchte. Zusammen mit der „Chip Authentication" bilden beide Protokolle ein „Key Agreement".

Transport Layer Security

Nachfolger von SSL. TLS bietet die Möglichkeit vorher ausgehandelte Schlüssel und Sitzungsdaten im Aushandlungsprozess zu berücksichtigen. So können beim Server eingehende Verbindungen bereits existierenden Sitzungen zugeordnet werden..

Abkürzungsverzeichnis

APDU	Application Protocol Data Unit.
API	Application Programmable Interface.
BAC	Basic Access Control.
BSI	Bundesministerium für Sicherheit in der Informationstechnik.
CA	Chip Authentication.
CAN	Card Access Number.
CHAT	Certificate Holder Authorization Template.
CSCA	Country Signer Certification Authority.
CVCA	Country Verifier Certification Authority.
EAC	Extended Access Control.
IFD	Chip Card Interface Device.
MRZ	Machine Readable Zone.
OID	Object Identifier.
PACE	Password Authenticated Connection Establishment.
PAOS	Liberty Reverse HTTP Binding for SOAP Specification.

PCD	Public Coupling Device.
PICC	Proximity Integrated Circuit Card.
PIN	Persönliche Identifikationnummer.
PKI	Public Key Infrastruktur.
PSK	Pre Shared Key.
PUK	Personal Unblocking Key.
RFID	Radio-Frequency Identification.
RI	Restricted Identifaction.
SOAP	ursprünglich für Simple Object Access Protocol.
SSL	Secure Sockets Layer.
TA	Terminal Authentication.
TLS	Transport Layer Security.
WSDL	Webservice Description Language.
XSD	XML Schema.

Literaturverzeichnis

[Ama92] Esther Amann, Hugo Atzmüller (1992): *IT-Sicherheit - was ist das? Datenschutz und Datensicherheit*, 6:286 – 292.

[BC] The Legion of the Bouncy Castle. *Bouncy Castle Crypto API*. URL http://www.bouncycastle.org

[Bol08] Matthias Boll, Michael T. Boos, Florian Dietz, Nico Jahn, Stefan Schröder, Christopher Walg (2008). *Projektpraktikum Elektronischer Personalausweis*.

[BSIa] BSI - Bundesamt für Sicherheit in der Informationstechnik. *BSI TR-03119 - Anforderungen an Chipkartenleser mit ePA Unterstützung v1.1*.

[BSIb] BSI - Bundesamt für Sicherheit in der Informationstechnik. *eCard-API-Framework - Protocols - Amendment 1 (eID-SSCD ActicationProtocol)*. URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil7_amd_pdf.pdf?__blob=publicationFile

[BSI09a] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-1 - eCard-API-Framework*. URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil1_pdf.pdf?__blob=publicationFile

[BSI09b] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-2 - eCard-API-Framework - eCard-Interface*. URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/

BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil2_pdf.pdf?__blob=publicationFile

[BSI09c] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-3 - eCard-API-Framework - Management-Interface*.
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil3_pdf.pdf?__blob=publicationFile

[BSI09d] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-4 - eCard-API-Framework - ISO24727-Interface*.
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil4_pdf.pdf?__blob=publicationFile

[BSI09e] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-5 - eCard-API-Framework - Support-Interface*.
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil5_pdf.pdf?__blob=publicationFile

[BSI09f] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-6 - eCard-API-Framework - IFD-Interface*.
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil6_pdf.pdf?__blob=publicationFile

[BSI09g] BSI - Bundesamt für Sicherheit in der Informationstechnik (2009). *BSI TR-03112-7 - eCard-API-Framework - Protokolle*.
URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03112/API/api1_teil7_pdf.pdf?__blob=publicationFile

[BSI10a] BSI - Bundesamt für Sicherheit in der Informationstechnik (2010). *BSI TR-03130 - Technische Richtlinie eID-Server, Version 1.4.1*.

[BSI10b] BSI - Bundesamt für Sicherheit in der Informationstechnik (2010). *BSI TR-3110 Advanced Security Mechanisms for Machine Readable Travel Documents*.

URL https://www.bsi.bund.de/cae/servlet/contentblob/532066/publicationFile/27971/TR-03110_v201_pdf.pdf

[Buc08] Johannes Buchmann (2008): *Einführung in die Kryptographie*, volume 4. Springer-Verlag.

[Dun03] Jürgen Dunkel (2003): *Softwarearchitektur für die Praxis*. Springer-Verlag Berlin Heidelberg New York.

[Eck06] Claudia Eckert (2006): *IT-Sicherheit*, volume 4.

[Eck10] Sven Ecker, Malte Knauf, Johannes Normann, Olga Roht, Florian Schlünß, Ansgar Taflinski, AndreasWolf, Jonas Zitz (2010). *Proektpraktikum ePa3*.

[Gri07] Rüdiger Grimm (2007). *Vorlesung - IT-Risk-Management - V3: Sicherheitsbegriffe und -konzepte*.

[Gro] Network Working Group. *Request for Comments: 4279*.

[Kah11] Christian Kahlo (2011). *Alternativer Aktivierungsmechanismu für eID Clients (ÄusweisApp")*.

[Mül11] Eugen Müller (2011): *Implementierung des PAOS-Protokolls*. Master's thesis.

[PAO] *Liberty Reverse HTTP Binding for SOAP Specification*.
URL http://projectliberty.org/liberty/content/download/909/6303/file/liberty-paos-v2.0.pdf

[Som07] Ian Sommerville (2007): *Software Engineering*. Pearson Studium.

[Vos98] Josef Voss, Dietmar Nentwig (1998): *Graphische Benutzungsschnittstellen*. Hanser Verlag.

[Wol11] Andreas Wolf (2011): *Implementierung der Extended Access Control des neuen Personalausweises*. Master's thesis, Universität Koblenz-Landau.