Humboldt-Universität zu Berlin

Mathematisch-Naturwissenschaftliche Fakultät II

Institut für Informatik

Display-Javakarte mit dynamischer eID-PIN für den neuen Personalausweis

Paul Bastian

Bachelorarbeit

Betreuer: Dr. Wolf Müller

Frank Dietrich

Gutachter: Prof. Jens-Peter Redlich

Dr. Manfred Paeschke

Berlin, den 7. November 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
AID	Application Identifier
API	Application Programming Interface
APDU	Application Process Data Unit
ATR	Answer to Reset
BSI	Bundesamt für Sicherheit in der Informationstechnik
CAN	Card Acces Number
CCC	Chaos Computer Club
CHAT	Certificate Holder Authorization Template
CLA	Class Byte
CMAC	ciper-based MAC
EAC	Extended Access Control
ECC	Elliptic Curve Cryptography
ECKA	Elliptic Curve Key Agreement
EEPROM	Electrical Erasable Programmable ROM
EF	Elementary File
eID	eletronic Identifiaction
INS	Instruction Byte
JCOP	Java Card Open Platform
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Maschine
KA	Key Agreement
MAC	Message Authentication Code
MF	Master File
MRTD	Maschine Readable Travel Document
NIST	National Institute of Standards and Technology
P1/P2	Parameter 1/2
PACE	Password Authenticated Connection Establishment
PCD	Proximity Coupling Device
PICC	Proximity Integrated Circuit Chip
PK	Public Key
RAM	Random Access Memory
RFID	Radio Frequency Identification
ROM	Read Only Memory
RSA	asymmetrisches Kryptoverfahren (Rivest, Shamir, Adleman)
SK	Secret Key
SSC	Send Sequence Counter
TR	Technische Richtlinie
USB	Universal Serial Bus
SW1/SW2	Statuswort 1/2

Zusammenfassung

Smart Cards sind heute eine weit verbreitete Möglichkeit zur Authentisierung und Identifikation, zum Beispiel in Ausweisdokumenten oder EC-Karten. Wird die Karte über ein Lesegerät ohne eigene Tastatur an einem kompromittierten Computer benutzt, so kann ein Angreifer mittels eines Keyloggers in Kenntnis der statischen PIN kommen und diese im folgenden mehrfach für Transaktionen im Namen des Nutzers verwenden. In dieser Arbeit wurde das Konzept einer teildynamischen PIN entwickelt, die aus der geheimen, statischen PIN und einem zufälligen, dynamisch generierten Anteil gebildet wird. Dieser dynamische Teil wird vor jeder PIN-Eingabe zufällig gewählt und anschließend auf einem Display der Smart Card angezeigt. Die abgehörte PIN ist für folgende Transaktionen wertlos, da der dynamische Teil nach der Authentisierung nicht mehr gültig ist. Möchte der Angreifer den Ausweis erneut benutzen, wird dies durch eine frische Auswahl von Zufallszahlen verhindert, die er nicht ablesen kann, weil er keinen Sichtkontakt zur Karte hat. Das Konzept wurde für das PIN-Management des neuen Personalausweises praktisch umgesetzt und analysiert. Da eine vollständige Kompatibilität mit dem neuen Personalausweis hinsichtlich des PIN-Managements erreicht werden kann, könnte eine neue Ausweisgeneration auch lokal begrenzt oder zeitlich überlappend mit der alten Generation eingeführt werden, ohne andere Komponenten ändern zu müssen.

1. Einleitung

Die Eingabe einer PIN bildet eine gängige Methode zur Authentisierung mit einer Smart Card. Dabei bildet die Eingabe über eine Kartenlesegerät ohne eigene Tastatur ein Sicherheitsrisiko. Über einen kompromittierten Computer kann der Angreifer die statische PIN mittels eines Keyloggers mitlesen. Er braucht die Eingabe dafür nur ein einziges Mal abhören und kann nachfolgend die Karte für seine Zwecke beliebig oft benutzen. Dieses Szenario eröffnet sich prinzipiell für jede Smart Card Anwendung, im Rahmen dieser Arbeit wird speziell ein Angriff auf den neuen Personalausweis betrachtet.

1.1. Der neue Personalausweis

Mit der Einführung des neuen Personalausweises am 1. November 2010 steht eine neue Möglichkeit zur Authentisierung im Internet zur Verfügung. Der im Ausweisdokument integrierte RFID-Chip ermöglicht neben hoheitlichen Zwecken auch eine eID-Funktionalität. Diese soll als Identitätsnachweis äquivalent sein zu einer Authentisierung mit dem Personalausweis als Sichtdokument außerhalb des Internets. Zu diesem Zweck wurde in einer Kooperation von Siemens, OpenLimit und der Bundesdruckerei die AusweisApp geschaffen, die eine Nutzung der eID-Funktion ermöglicht. Diensteanbieter können von einer hoheitlichen Behörde ein elektronisches Berechtigungszertifikat erwerben und sich damit gegenüber dem Kunden authentisieren. Der neue Personalausweis überprüft die Berechtigung des Diensteanbieters und schickt nach der PIN-Eingabe die angeforderten Daten zurück.

1.2. Angriffsszenario

Im Zuge der Einführung des neuen Personalausweises verteilte das Bundesamt für Sicherheit in der Informationstechnologie (BSI) viele Basiskartenleser gratis oder subventioniert diese, um die Akzeptanz des neuen Ausweisdokumentes zu fördern. Im Gegensatz zu den Komfort- und Standardkartenlesegeräten verfügt die Basisklasse nicht über eine eigene Tastatur. Die eID-PIN muss somit über die Tastatur des Computers eingegeben werden, alternativ über eine mit der Maus zu bedienenden Bildschirmtastatur. Diese Schwachstelle der Basiskartenleser eröffnet einen Angriffsvektor, über den der Chaos Computer Club (CCC) folgendes bereits im September 2010 berichtete [14]. Der Ausweisinhaber möchte sich im Internet gegenüber einem Diensteanbieter authentisieren und benutzt die eID-Funktionalität. Da er nur über

ein Basiskartenlesegerät verfügt wird mittels eines Keyloggers die vom Nutzer über die Tastatur eingegebene eID-PIN mitgeschnitten. Der Nutzer hat sich erfolgreich gegenüber dem Diensteanbieter authentisiert und vergisst im Anschluss seinen Personalausweis auf dem Kartenlesegerät. Der Angreifer hat auf einem hier nicht näher betrachteten Weg die Kontrolle über den Computer des Nutzers erlangt. Mittels eines Relay-Angriffs erlangt er nun indirekten Zugriff auf die Karte. Dies geschieht entweder durch Forwarding der USB-Pakete des Kartenlesers, wie es Max Moser und Thorsten Schröder bei der Suisse ID demonstrierten [24] [16] oder auf PC/SC-Ebene mit der Virtual Smart Card Architecture von Frank Morgner und Dominik Oepen [15]. Der Angreifer kann nun selbst die eID-Funktion mit den Personalien des Ausweisinhabers nutzen, da er im Besitz der Karte sowie in Kenntnis der PIN ist. Darüber hinaus hat der Angreifer die Möglichkeit die eID-PIN beliebig zu ändern. Der Computer bildet in diesem Szenario die Schwachstelle für den Angriff, der neue Personalausweis an sich hat keine derzeit bekannte Sicherheitslücke. Als Antwort auf die Veröffentlichungen des CCC verwies das BSI auf die Richtlinien um den eigenen Computer auf den jeweiligen Stand der Sicherheit zu bringen. Nach § 27 des Personalausweisgesetztes muss der Personalausweisinhaber

> „[...]durch technische und organisatorische Maßnahmen gewährleisten, dass der elektronische Identitätsnachweis gemäß § 18 nur in einer Umgebung eingesetzt wird, die nach dem jeweiligen Stand der Technik als sicher anzusehen ist. Dabei soll er insbesondere solche technischen Systeme und Bestandteile einsetzen, die vom Bundesamt für Sicherheit in der Informationstechnik als für diesen Einsatzzweck sicher bewertet werden.“[20]

Diese Vorstellung ist in der Welt des heutigen Internets surreal, denn die wenigsten Bürger haben die Kenntnisse und Möglichkeiten diese Empfehlungen umzusetzen.

Ziel dieser Arbeit ist es eine Chipkarte mit der eID-Funktionalität im Sinne eines elektronischen Personalausweises zu konstruieren und durch Modifizierung des PIN-Verfahrens die Sicherheit zu erhöhen.

2. Konzept

Das grundlegende Sicherheitsproblem einer statischen PIN ist die Tatsache, dass ein Angreifer die PIN-Eingabe nur ein einziges Mal abhören braucht. Anschließend kann er mit dem erlangten Wissen die Karte später beliebig oft benutzen. Um trotz eines möglichen Angriffs den Schutz vor einer unberechtigten Nutzung zu steigern, darf der Nutzer bei der Eingabe der PIN nicht sein komplettes Wissen preisgeben. Die vorgeschlagene Lösung ist ein dynamischer Anteil in der PIN, der bei jeder Authentisierung neu generiert wird. Der Angreifer muss somit den statischen und dynamischen Anteil der PIN wissen, um die Karte benutzen zu können. Ein Display auf der Karte bildet hierbei einen zweiten Kanal zwischen Ausweis und Nutzer, um den dynamischen Teil der PIN anzuzeigen. Auf diese Anzeige hat der Angreifer jedoch ohne weitere Maßnahmen keinen Sichtkontakt, sodass ein Missbrauch der Karte erschwert wird.

2.1. Teildynamische PIN

Die teildynamische PIN wird sich aus dem geheimen, statischen Teil der PIN und einem zufälligen, dynamisch generierten Anteil gebildet. Dabei teilt sich die n-stellige PIN in $0 < d < n$ dynamische und $n - d$ statische Stellen auf. Die dynamischen Stellen werden vor jeder PIN-Eingabe frisch mit Zufallszahlen gefüllt und anschließend auf dem Display angezeigt. Hierbei können die Positionen und die Anzahl der Zufallsziffern[1] sowohl statisch als auch zufällig gewählt werden. Die teildynamische PIN entsteht, wenn man bestimmte Stellen der statischen PIN durchdie den dynamisch generierten Zufallsziffern ersetzt. An allen Positionen, an denen keine dynamischen Ziffern generiert wurden, bleibt die statische PIN effektiv. Bevor eine neue PIN-Eingabe erfolgt, wird der dynamische Teil der PIN mit neuen Zufallsziffern gefüllt.

Im Falle eines Angriffs wird jetzt mittels des Keyloggers nur die teildynamische PIN mitgeschnitten. Möchte der Angreifer nun eine Authentisierung im Namen des Ausweisinhabers durchführen, wird die Zufallsziffernfolge erneut generiert. Da er jedoch keinen Sichtkontakt zur Karte hat, kann er den dynamischen Teil der PIN nicht auslesen. Die Informationen der zuvor mitgeschnittenen PIN sind nicht ausreichend um einen Angriff auszuführen.

[1] Zufallsziffern sind nicht auf die Zahlen 0 bis 9 beschränkt und können Buchstaben oder andere Symbole beinhalten.

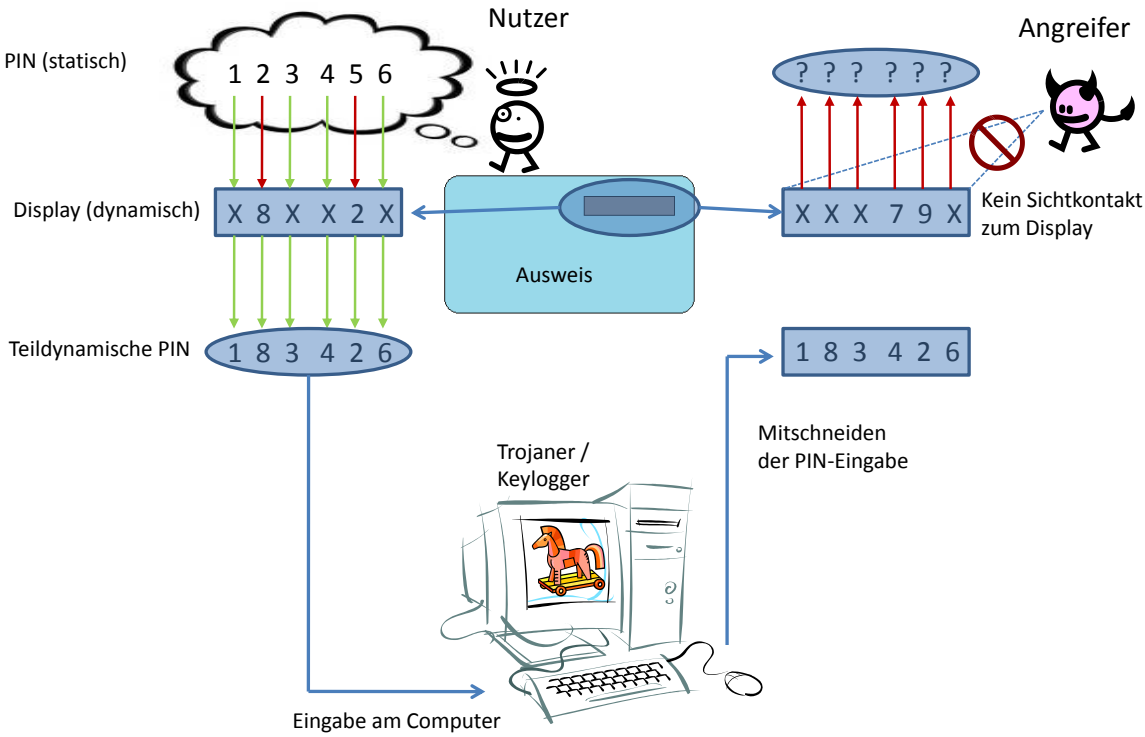

Abbildung 2.1.: Funktionsweise der teildynamischen PIN und des Keyloggerangriffs

2.2. Varianten

Die Varianten der teildynamischen PIN unterscheiden sich hinsichtlich der Position und Anzahl der zufälligen Ziffern. Zunächst werden die beiden im Rahmen der Bachelorarbeit implementierten Verfahren vorgestellt, danach folgt ein Ausblick, welche weiteren Varianten denkbar wären.

2.2.1. Statische Positionierung

Beim ersten Verfahren sind die Zufallziffern stets an der gleichen, statischen Position. Beispielsweise können die ersten $n - d$ Positionen auf dem Display mit „X“ markiert sein und für die letzten d Positionen werden Zufallsziffern generiert und angezeigt. Die statische PIN des Nutzers hat die Länge n, ebenso wie die Ziffernfolge auf dem Display der Smart Card. Die erste Stelle der teildynamischen PIN ergibt sich aus der ersten Position der statischen PIN, wenn diese auf dem Display mit „X“ markiert ist, ansonsten benutzt man die auf dem Display angezeigten Zufallsziffern. Nach diesem Verfahren errechnet man alle n Stellen der teildynamischen PIN, als effektive PIN zur

Authentisierung gibt der Nutzer nur diese teildynamische PIN ein.

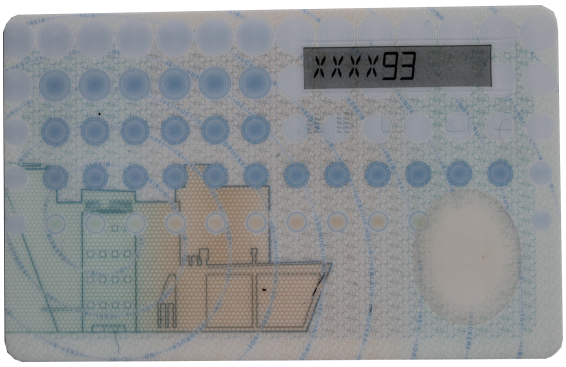

Abbildung 2.2.: Displaykarte mit statischer Positionierung der Zufallszahlen

2.2.2. Randomisierte Positionierung

Beim zweiten Verfahren sind sowohl die Zufallsziffern dynamisch generiert, als auch die jeweiligen Posistionen der d Zufallsziffern. Dabei wird die erste Position mit einer Zufallszahl zwischen 1 und n berechnet. Die Berechnung der weiteren Positionen wird danach nach dem gleichen Verfahren errechnet bis d unterschiedliche Positionen bestimmt sind. Danach werden für die soeben bestimmten Positionen Zufallsziffern generiert und auf dem Display angezeigt, alle anderen Positionen werden mit „X“ markiert. Analog zum ersten Verfahren bildet der Nutzer für die Authentisierung nun die teildynamische PIN, indem er an allen mit „X“ markierten Stellen die jeweilige Position seiner statischen PIN einsetzt.

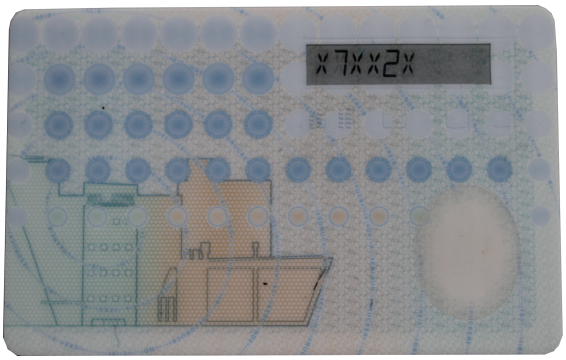

Abbildung 2.3.: Displaykarte mit randomisierter Positionierung der Zufallszahlen

2.2.3. Alternative Varianten

Desweiteren kann man die Anzahl der Zufallsziffern dynamisch gestalten und diese wiederum in einem bestimmten Rahmen zufallsgeneriert bestimmen. Je nach Durch-

lauf würden dann zum Beispiel eine, zwei oder drei Zufallsziffern auf dem Display ausgegeben werden.

3. Wahrscheinlichkeitsbetrachtung

In diesem Abschnitt wird die Sicherheit der teildynamischen PIN gegen folgende zwei Angriffsarten betrachtet:

- Schutz vor Missbrauch nach einem Diebstahl der Smart Card
- Schutz vor einem Relay-Angriff mit Keylogger

Dabei werden die in Kapitel 2.2 vorgestellten Varianten untersucht und verglichen. Die beschriebenen Wahrscheinlichkeiten beziehen sich auf die Sichtweise des Angreifers. Zur besseren Vergleichbarkeit ist die Gesamtlänge der PIN für alle Varianten gleich n. Die Anzahl der zur Eingabe möglichen Ziffern beträgt s. Ein gängiges Verfahren zum Schutz vor Brute-Force-Angriffen ist eine Sperrung durch einen Fehlbedienungszähler. Dieser Mechanimus wird im folgenden durch die Anzahl der möglichen Versuche k abgebildet.

3.1. Statische PIN

Eine rein statische PIN ohne dynamisch generierten Anteil ist am besten gegen einen Missbrauch nach einem Diebstahl geschützt. Ein Angreifer müsste mittels Brute-Force n Stellen erraten um die Karte benutzen zu können. Je nach Länge der PIN sinkt die Wahrscheinlichkeit für einen Treffer mit jedem neuen Versuch an.

$$\underbrace{\frac{1}{s^n} + \frac{1}{s^n - 1} + \cdots + \frac{1}{s^n - k - 1}}_{k-mal} \approx \frac{k}{s^n} \qquad , s \gg 1$$

Im Falle eines Keylogger-Angriffs ist beispielsweise die sechsstellige PIN des Personalausweises nutzlos, da der Angreifer sie im Klartext mitlesen kann und die selbe PIN danach für seine eigenen Zwecke wiederverwenden kann. Die Wahrscheinlichkeit für einen erfolgreichen Angriff beträgt unter diesen Voraussetzungen **100%**.

3.2. Dynamische PIN mit statischer Positionierung

In dieser ersten Variante der teildynamischen PIN ist die Schutzfunktion nach einem Diebstahl gegenüber der rein statischen PIN verringert, weil der Angreifer die

zufallsgenerierten Ziffern vom Display ablesen kann. Der Erfolg eines Brute-Force-Angriffs steigt somit, weil nur die verbleibenden, nichtdynamischen Ziffern erraten werden müssen:

$$\underbrace{\frac{1}{s^{n-d}} + \frac{1}{s^{n-d}-1} + \cdots + \frac{1}{s^{n-d}-k-1}}_{k-mal} \approx \frac{k}{s^{n-d}}$$

Versucht der Angreifer jedoch den Personalausweis über einen Relay-Angriff zu benutzen, so verringert die teildynamische PIN seine Erfolgsaussichten. Dem Angreifer sind zwar nach einem abgehörten PIN-Durchlauf die ersten $n-d$ Stellen der PIN bekannt, die letzten d Ziffern der mitgeschnittenen Ziffernfolge sind jedoch wertlos, weil diese nun neu generiert werden. Er muss auch hier wieder einen Rateversuch machen. Die Wahrscheinlichkeiten für die k Versuche sind identisch, da die Ereignisse unabhängig sind.

$$\underbrace{\frac{1}{s^{d}} + \frac{1}{s^{d}} + \cdots + \frac{1}{s^{d}}}_{k-mal} = \frac{k}{s^{d}}$$

3.3. Dynamische PIN mit randomisierter Positionierung

Bei der zweiten vorgestellten Variante der teildynamischen PIN finden sich keine Veränderungen bei Betrachtung des Diebstahlszenarios. Lediglich die Position der Zufallsziffern ist bei jedem Authentisierungsdurchlauf unterschiedlich, was die Chancen für den Angreifer jedoch nicht ändert.

Der wesentliche Unterschied findet sich beim Keyloggerszenario: Beim erstmaligen Abhören erlangt der Angreifer keine Erkenntnisse darüber, bei welchen Ziffern die PIN mit Zufallszahlen überschrieben wurde. Er weiß jedoch, dass mindestens $n-d$ Ziffern zur korrekten, statischen PIN gehören. Der Angreifer hat durch die wechselden Positionen der Zufallsgenerierung über mehrere Abhörvorgänge die Möglichkeit die statische PIN probabilistisch herauszufinden. Für einen Brute-Force-Angriff bringt ihm dieses Wissen jedoch keinen Vorteil, denn zunächst muss er die Positionierung der dynamischen Stellen erraten und zusätzlich noch die Werte der Zufallsziffern. Bei d Zufallsstellen auf n PIN-Ziffern kommt eine Anzahl von l verschiedenen

Anordnungsmöglichkeiten zustande. Für $n = 6$ und $d = 2$ ergeben sich daraus zum Beispiel $k = 15$ mögliche Zufallspositionierungen:

$$l_{(6;2)} = \binom{n}{d} = \frac{n!}{d!(n-d)!} = \frac{6!}{2! \cdot 4!} = 15$$

Da der dynamische Anteil nach jedem Fehlversuch neu generiert wird, sind die Ereignisse unabhängig und die Chancen somit identisch. Die Wahrscheinlichkeit für einen erfolgreichen Angriff beträgt somit:

$$\underbrace{\frac{1}{s^d \cdot l} + \frac{1}{s^d \cdot l} + \cdots + \frac{1}{s^d \cdot l}}_{k-mal} \approx \frac{k}{s^d \cdot l}$$

3.4. Alternative Varianten

Eine andere Variante würde die in 2.2.2 vorgestellte randomisierte Positionierung erweitern: Neben den zufälligen Werten und der zufälligen Positionierung könnte man ebenso die Anzahl der Ziffern des dynamisch generierten Teils zufällig gestalten. Im Vergleich schneidet diese Variante jedoch entweder beim Diebstahlszenario oder beim Relay-Angriff schlechter ab als eine vergleichbare randomisierte Positionierung. Im Folgenden betrachten wir die Variante „Randomisierte Anzahl" mit $d_{\min}$ bis $d_{\max}$ zufälligen, dynamischen Stellen. Ein möglicher Angreifer, der die Smart Card durch Diebstahl erlangt hat, legt die Karte so lange neu auf den Kartenleser bis die dynamische Zufallsgenerierung ihm $d_{\max}$ dynamische Stellen vorgibt. Der zufällige Teil der teildynamischen PIN wird bei jedem Auflegen neu generiert, weil der Inhaber diesen sonst bei einem Vorzeigen der Karte Preis geben müsste. Weil er anstatt $n - d$ Ziffern jetzt nur noch $n - d_{\max}$ erraten muss, steigen somit die Chancen für eine erfolgreiche Brute-Force-Attacke:

$$\underbrace{\frac{1}{s^{n-d_{\max}}} + \frac{1}{s^{n-d_{\max}} - 1} + \frac{1}{s^{n-d_{\max}} - k - 1}}_{k-mal} \approx \frac{k}{s^{n-d_{\max}}}$$

Der Vorteil scheint dafür zunächst bei einem Relay-Angriff zu liegen. Zusätzlich muss der Angreifer nun noch erraten, wieviele dynamische Stellen generiert wurden:

$$k \cdot \sum_{i=d_{\min}}^{d_{\max}} \left(\frac{1}{\frac{\binom{n}{i}}{\sum_{j=d_{\min}}^{d_{\max}} \binom{n}{j}} \cdot \frac{1}{\binom{n}{i}} \cdot s^i} \right) = k \cdot \sum_{i=d_{\min}}^{d_{\max}} \left(\frac{1}{\frac{1}{\sum_{j=d_{\min}}^{d_{\max}} \binom{n}{j}} \cdot s^i} \right)$$

$$= \frac{k}{\underbrace{\sum_{j=d_{\min}}^{d_{\max}} \binom{n}{j}}_{\widehat{=}l} \cdot \underbrace{\sum_{i=d_{\min}}^{d_{\max}} \left(s^i\right)}_{\widehat{=}s^d}}$$

Bei einem direkten Vergleich mit der randomisierten Positionierung ist die randomisierte Anzahl bei gleichem $d_{\max}$ stets schlechter.

Dies liegt vor allem daran, dass der Faktor $\left(\frac{1}{s}\right)^i$ mit $i = d_{\min}$ das Ergebnis stärker beeinflusst als die zusätzlichen Kombinationsmöglichkeiten.

4. Der neue Personalausweis

Die Authentisierung mit dem neuen Personalausweis wird durch zwei Faktoren sichergestellt: Er muss zum einen im Besitz des Personalausweises sein und zum anderen das Wissen über die geheime eID-PIN haben. Der Authentisierungsfaktor Besitz bedeutet strenggenommen den Zugriff auf den neuen Personalausweis. Der Zugriff ist beispielsweise bei einem kompromittierten Computer in Verbindung mit einem Basisleser gegeben. Die Durchführung eines Relay-Angriffs wurde in [15] und [24] vorgestellt und soll hier nicht näher betrachtet werden. Neben einem Basisleser kann auch ein kompromittiertes, öffentliches Terminal ein Einfallstor in das System bilden. Der zweite Authentisierungsfaktor Wissen wird über die eID-PIN realisiert, deren Sicherheit in diesem Kapitel untersucht wird.

4.1. eID-PIN

Bei der eID-PIN handelt es sich in der derzeitigen Ausführung des neuen Personalausweises um eine sechsstellige, alphanumerische Ziffernfolge, jedoch können an Standard-, Komfortleser und über die AusweisApp nur die Ziffern $0 - 9$ verwendet werden. Der Inhaber bekommt nach Ausstellung des Ausweises zunächst eine Transport-PIN per Brief zugeschickt, mit der er sich zum Setzen einer operationellen eID-PIN authentisiert. Durch diesen Ablauf wird sichergestellt, dass die eID-PIN allein dem Ausweisinhaber bekannt ist. Die eID-PIN ist ein „blocking“ Passwort, das bedeutet, dass sie mit einem Fehlbedienungszähler gegen Brute-Force-Attacken gesichert ist. Nach drei falschen Eingaben wird die eID-PIN gesperrt.

4.2. CAN

Mit der Sperrung durch einen Fehlbedienungszähler eröffnet sich gleichzeitig auch die Möglichkeit einer Denial-of-Service-Attacke. Ein Angreifer könnte unbemerkt in der Öffentlichkeit mit einem starken Lesegerät drei falsche PIN-Eingaben tätigen und so die Karte bis zur Eingabe der PUK vorübergehend unbenutzbar machen. Dagegen ist der Ausweis zusätzlich mit einer Card Access Number (CAN) gesichert, deren Eingabe nach dem zweiten Fehlversuch gefordert wird, bevor man den letzten Versuch zur Eingabe der eID-PIN starten kann[7]. Dazu wird die eID-PIN nach zwei Falscheingaben auf den Status „suspend“ gesetzt. Um sie wieder freizuschalten, muss der Nutzer seinen Besitz der Karte mit der CAN nachweisen. Nach erfolgreicher Eingabe wird der Status der eID-PIN auf „resumed“ geändert und der Nutzer ist für den

dritten Versuch freigeschaltet.

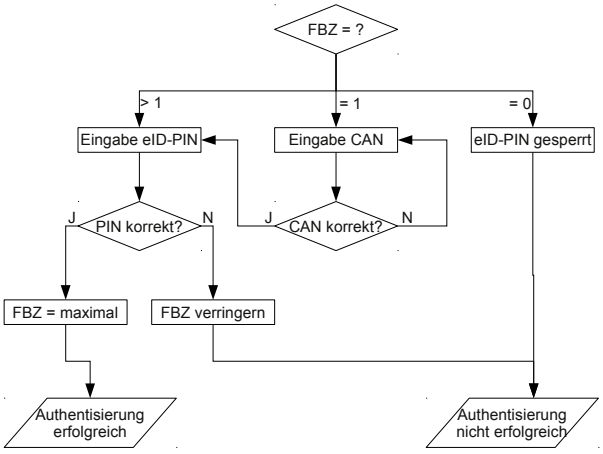

Abbildung 4.1.: Eingabe der eID-PIN nach TR-03127 [12]

Die CAN ist ein „non-blocking“ password, das heißt sie besitzt keinen Fehlbedienungszähler. Die Technische Richtlinie TR-03110 schreibt dazu:

> „While this Technical Guideline does not recommend any specific size for passwords, each non-blocking password MUST contain sufficient entropy or the MRTD chip MUST employ additional countermeasures to protect against brute-force attacks. Countermeasures MAY include delays but MUST NOT block the password after incorrect trials.“[5]

Obwohl ein zusätzlicher Schutz der CAN gegen Brute-Force-Angriffe gefordert ist, wurde dieser nicht im aktuellen Personalausweis implementiert. Wie Frank Morgner und Dominik Oepen in [15] berichteten, lässt sich die CAN in 4,5 Tagen knacken. Zudem könnte ein Angreifer zunächst zwei Fehlversuche durchführen und die CAN-Eingabe des Nutzers bei einer nächsten Aktion belauschen. Im Folgenden stellt die Card Access Number keinen eindeutigen Sicherheitsgewinn dar und wird deswegen außen vor gelassen.

4.2.1. Dynamische CAN

Durch die Verwendung einer Displaykarte für den neuen Personalausweis würde sich zudem die Möglichkeit einer dynamischen CAN anbieten, wie sie bereits in der technischen Richtlinie [4] vorgeschlagen wird. Gegenüber einem statischen Aufdruck ist eine dynamische CAN gegen ein vorangehendes Auslesen, zum Beispiel beim Vorzeigen als Sichtdokument, geschützt, da die CAN erst direkt vor der Eingabe zufällig generiert wird. Zudem steigt der Aufwand für eine Brute Force Attacke: Bei einer statischen CAN mit sechs Stellen muss der Angreifer nur alle 1000000 verschiedenen Möglichkeiten durchprobieren. Mit jeder weiteren Eingabe steigt somit die Wahrscheinlichkeit, dass er einen Treffer landet, sodass er im Mittel

$$
\begin{aligned}
n &= \frac{1000000}{2} \\
&= 500000
\end{aligned}
$$

Versuche benötigt. Bei einer dynamischen CAN sind die bisherigen Fehlversuche jedoch wertlos, da nach jeder Eingabe die CAN neu generiert wird und die Ereignisse somit unabhängig sind. Die Wahrscheinlichkeit für einen Treffer nach dem n-ten Versuch beträgt:

$$
\begin{aligned}
p(n) &= \underbrace{(1-p)\cdots(1-p)}_{(n-1)-mal} \cdot p \\
&= (1-p)^{n-1} \cdot p
\end{aligned}
$$

$$
\begin{aligned}
\langle n \rangle &= \sum_{n=1}^{\infty} n \cdot p(n) \\
&= \sum_{n=1}^{\infty} n \cdot (1-p)^{n-1} \cdot p \\
&= \frac{p}{1-p} \cdot \sum_{n=1}^{\infty} n \cdot (1-p)^{n} \\
&= \frac{p}{1-p} \cdot \frac{1-p}{p^2} \\
&= \frac{1}{p}
\end{aligned}
$$

Mit $p = 10^{-6}$ beträgt der Erwartungswert für den ersten Treffer $n = 1000000$ Versuche. Der Aufwand für einen Brute Force Angriff ist somit doppelt so groß wie bei der statischen CAN.

4.3. Anwendung der teildynamischen PIN auf den nPA

Im Rahmen der Bachelorarbeit war die Implementierung auf eine volle Kompatibilität zur AusweisApp fokussiert. Die Anzahl der PIN-Ziffern ist deswegen auf $n = 6$ gesetzt. Die Anzahl der dynamisch generierten Zufallsziffern ist auf $d = 2$ festgelegt.

Damit ergibt sich der statische Anteil von $n - d = 4$ statischen Ziffern. Eine weitere Limitierung der AusweisApp eröffnet sich durch die Menge der Ziffern s für die PIN. Obwohl PACE und der neue Personalausweis für eine alphanumerische PIN ausgelegt sind, ermöglichen die AusweisApp als auch Standard- und Komfortleser nur die Eingabe der Zahlen von null bis neun, sodass $s = 10$ gilt. In Tabelle 4.1 sind die errechneten Wahrscheinlichkeiten für einen Erfolg des Angreifers mit dem neuen Personalausweis aufgelistet. Neben der zunächst gewählten Verteilung des statischen und dynamischen Verhältnisses von vier zu zwei wurden noch die Werte für eine Aufteilung drei zu drei zu Vergleichszwecken hinzugezogen.

PIN-Modus	Diebstahlszenario	Trojanerszenario
neuer Personalausweis	0,0003%	100,0%
statische Positionierung[a]	0,03%	3,0%
statische Positionierung[b]	0,3%	0,3%
randomisierte Positionierung[a]	0,03%	0,2%
randomisierte Positionierung[b]	0,3%	0,015%
randomisierte Anzahl($d_{\min} = 1$, $d_{\max} = 2$)	0,03%	1,57%
randomisierte Anzahl($d_{\min} = 2$, $d_{\max} = 3$)	0,3%	0,094%
randomisierte Anzahl($d_{\min} = 1$, $d_{\max} = 3$)	0,3%	0,81%

[a] 4 Ziffern statisch; 2 Ziffern dynamisch
[b] 3 Ziffern statisch; 3 Ziffern dynamisch

Tabelle 4.1.: Zusammenfassung der Erfolgswahrscheinlichkeiten für eine Angreifer

4.4. Kompromiss zwischen Komfort und Sicherheit

Als bestes Ergebnis schneidet die randomisierte Positionierung mit einem Verhältnis von vier statischen und zwei dynamischen Ziffern ab. Betrachtet man auswertend die Wahrscheinlichkeiten für einen erfolgreichen Angriff, so erscheint die zufallspositionierte Lösung der Statischpositionierten in Hinsicht auf das Trojanerszenario überlegen. Der Grund dieser erhöhten Sicherheit liegt in der zusätzlichen Varianz, die durch die Kombinationsmöglichkeiten der Anordnungen der Zufallsziffern ensteht. Diese zusätzliche Sicherheit bringt allerdings auch einen Rückschritt an Komfort mit sich. Das menschliche Gehirn merkt sich Ziffernfolgen am besten sequenziell. Dafür ist die erste Variante am besten geeignet, welche nur die vier ersten PIN-Stellen erfordert. Demgegenüber ist die zufällige Positionierung entgegen der Gedächtnisstruktur des Menschen. Dieser muss unter Umständen nach jeder vom Display abgetippten Ziffer die PIN erneut von vorne durchdenken, da er die Abfolge der Ziffern kennt, aber nicht sofort weiß, welche Ziffer an einer bestimmten Stelle steht. Allerdings variiert

dieser Umstand je nach Konzentration und Gedächtnis des Nutzers. Im Durchschnitt dürfte die Eingabezeit und die Fehlerquote für die zufallspositionierte PIN leicht ansteigen.

Weitere Modifikationen könnten über die Beschränkung von sechs PIN-Ziffern hinausgehen. So wäre eine unveränderte eID-PIN äquivalent zum Auslieferungsstand des derzeitigen neuen Personalausweises mit einer zusätzlichen, optionalen Zufallsziffterngenerierung denkbar. Alternativen zu den hier vorgestellten Möglichkeiten bieten sich, indem man zur derzeit statischen, sechsstelligen PIN weitere dynamische Ziffern hinzufügt. Somit würde die Länge des Geheimnisses steigen und damit die Wahrscheinlichkeit für einen Brute-Force-Angriff weiter sinken. Ebenso bleibt die bestehende PIN unverändert. Damit würde die hohe Sicherheit des Personalausweises gegen einen Diebstahl erhalten bleiben und mit der erhöhten Sicherheit der teildynamsichen PIN gegen Relay-Angriffe kombiniert.

4.5. Auswertung

Die in dieser Arbeit vorgestellten Verfahren zur teildynamischen PIN erfüllen die primäre Anforderung zur Erhöhung der Sicherheit gegen Keylogger-Angriffe. Dieser Zuwachs an Sicherheit geht allerdings zu Lasten der Sicherheit im Falle eines Diebstahls, wie aus Tabelle 4.1 ersichtlich wird. Die Variante der statischen Positionierung ermöglicht dem Nutzer einen größtmöglichen Komfort. Dagegen steigert das zufallspositionierte Verfahren die Sicherheit gegenüber dem Keylogger-Angriff deutlich, bringt aber gleichzeitig einen Verlust an Benutzerfreundlichkeit mit sich. Zudem bringt die randomisierte Variante den Vorteil mit sich, dass nach einem Mitschneiden der PIN und anschließendem Diebstahl des Ausweises der Angreifer den Ausweis nicht bedienen kann. Der Nutzer gibt bei jeder Eingabe nur vier von sechs Stellen der statischen PIN ein und der Angreifer kann nach einmaligem Abhören nicht wissen, welche Stellen dynamisch generiert wurden und welche zur statischen PIN gehören. Nur nach einer Vielzahl von Abhöraktionen kann der Angreifer die statische PIN probabilistisch herausfinden.

Unabhängig vom Verfahren ist die Reduzierung der Sicherheit gegen eine Brute-Force-Attacke bei einem Diebstahl in sofern verkraftbar, als dass ein Diebstahl des Personalausweises eher vom Ausweisinhaber bemerkt wird als ein Trojanerangriff über seinen kompromittierten Computer. Im Falle eines Verlusts des Ausweisdokuments besitzt der Nutzer ein Sperrkennwort über welches ein Missbrauch verhindert werden kann. Demgegenüber bleibt ein Keylogger-Angriff fast immer unbemerkt. Somit bietet die teildynamische PIN einen großen Zuwachs an Sicherheit für den Nutzer.

Die beste Lösung wäre eine Ergänzung von dynamischen Stellen zur derzeitigen Ausführung des neuen Personalausweises. So bleibt die Sicherheit gegenüber einem Diebstahl vollständig erhalten und gleichzeitig wird der Ausweis gegen einen Relay-Angriff geschützt.

Trotz der erhöhten Sicherheit mit einer teildynamischen PIN kann ein Angreifer,

der die PIN-Eingabe mithört trotzdem noch mit einer online Man-in-the-Middle-Attacke die eingegebene PIN in genau diesem Moment einmalig benutzen. In Analogie zum Online-Banking kann eine iTAN genau im Moment der Eingabe für eine Überweisung missbraucht werden, während dem Nutzer eine erfolgreiche Transaktion vorgegaukelt wird. Ebenso wäre es möglich, die eingegebe teildynamische PIN in genau diesem Moment für eine andere Authentisierung zu nutzen. Ein Lösungsansatz wäre hier die Möglichkeit der Überprüfung des Nutzers für welche Transaktion er seine PIN eingibt, beispielsweise über das Display als zweiten Kanal, ähnlich zum smsTAN-Verfahren. Denn der Nutzer kann bei Eingabe seiner PIN nicht wissen, ob zum Beispiel die eID, eSign oder ePass Anwendung genutzt wird. Hierbei könnte das Display anzeigen, welche Funktion genutzt wird und sogar die verschiedenen eID-Varianten, wie Altersverifikation oder RestictedID unterscheiden. Das Problem liegt hier jedoch derzeit auf Seiten der AusweisApp, welche den CHAT mit den entsprechenden Daten erst nach der PIN-Eingabe an den Ausweis schickt.

Zusätzlich ist eine Sicherheitsverschärfung für die CAN sehr empfehlenswert. Hierbei ist zunächst eine einfache Verzögerung hilfreich, um Brute-Force-Attacken wesentlich zu erschweren. Zudem ist eine dynamische CAN wünschenswert, wie sie auch in der technischen Richtlinie TR-03110 vorgeschlagen wird [5]. Diese könnte mit Hilfe der Displaykarte mit wenig Aufwand umgesetzt werden.

5. Chipkarten

5.1. Aufbau

Chipkarten, auch Smart Cards genannt, sind Plastikkarten mit einem integrierten Schaltkreis. Anders als Magnetstreifenkarten haben sie nicht nur einen Speicherbereich, sondern funktionieren als eigenständiger Computer. Das physikalische Format ist normiert nach ISO 7810 [10]. Kreditkarten, EC-Karten sowie auch der neue Personalausweis nutzen dabei das weit verbreitete ID-01 Format mit einer Größe von $85,60mm \times 53,98mm$ und einer Höhe von $0,76mm$. Diese unterteilen sich wiederum in kontaktbehaftete und kontaktlose Karten.

Im Folgenden werden nur die kontaktlosen RFID-basierten Smart Cards betrachtet. Das Lesegerät erzeugt ein hochfrequentes elektromagnetisches Wechselfeld, welches dem Transponder (Chipkarte) als Kommunikationskanal sowie auch als Stromversorgung dient. Die Chipkarte besitzt dafür eine Antennenspule, die in der Plastikkarte eingebaut und mit dem Mikrocontroller verbunden ist. Der Mikrocontroller bildet das Herzstück der Smart Card: Er enthält die Prozessoreinheit, mathematische oder kryptografische Coprozessoren und verschiedene Arten von Speicher. Die Coprozessoren sind Recheneinheiten, welche auf bestimmte komplexe Operationen spezialisiert sind, die für den Hauptprozessor sonst zu zeitaufwändig wären. Meist handelt es sich um Kryptoprozessoren, die zum Beispiel AES, RSA oder andere Algorithmen implementiert haben. Das Speichersystem von Smart Cards unterteilt sich in drei unterschiedliche Bereiche[1]: Der ROM (read only memory) ist ein festgeschriebener, unveränderbarer Speicherbereich, der während der Herstellung einmalig beschrieben wird. Er enthält das Betriebssystem der Chipkarte und andere permanente Daten, die auch ohne Stromzufuhr erhalten bleiben. Der EEPROM (electrical erasable programmable read-only memory) ist genauso wie der ROM persistent, jedoch kann er auch modifiziert werden. Er dient somit als Programm- und Datenspeicher für Applikationen, die auf die Karte geladen werden. Die Zellen des EEPROM reichen für mindestens 100000 Schreibzyklen und 10 Jahre aus. Der Lesezugriff auf die Daten ist genauso schnell wie beim flüchtigen RAM, das Beschreiben benötigt jedoch 1000 Mal mehr Zeit. Der RAM (random access memory) ist nicht persistent, dass heißt die Daten gehen nach einer Trennung vom Terminal verloren. Er dient als Zwischenspeicher für Daten und Berechnungen und kann unendlich oft beschrieben werden. Wie bereits erwähnt sind Schreibzyklen für den RAM sehr kurz, dafür ist er aber sehr groß und teuer, weshalb RAM eine stark limitierte Ressource auf Smart Cards darstellt.

[1] Die Speicherbereiche sind in modernen Karten nicht von einander getrennt, sondern überall auf der Karte durchmischt angeordnet.

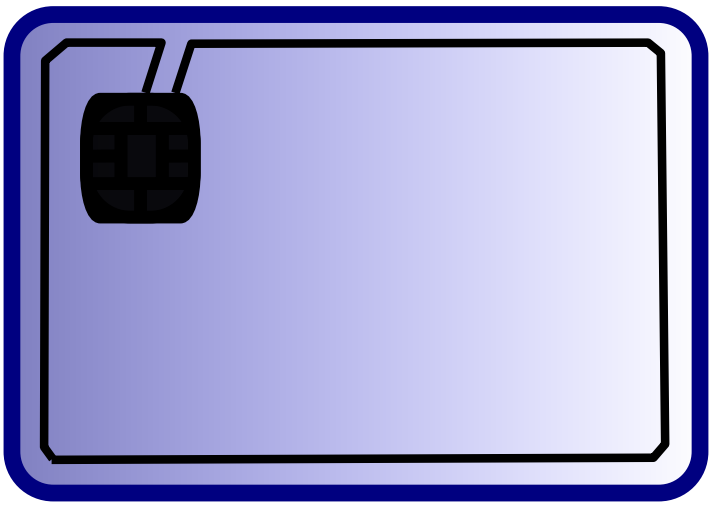

Abbildung 5.1.: Aufbau einer kontaktlosen Smart Card

5.2. Kommunikation

Die Datenübertragungsprotokolle und die elektrischen Parameter für die kontaktlose Kommunikation sind in ISO 14443-3 [9] beschrieben. Der Kommunikationskanal zwischen Smart Card und Terminal funktioniert über eine Halbduplex-Verbindung: Die Datenübermittlung ist dabei in beide Richtungen möglich, allerdings kann nur einer der beiden Teilnehmer Daten senden. Somit müssen die Kommunikationspartner im Wechselbetrieb Daten schicken. Terminal und Smart Card arbeiten deshalb nach dem Master-Slave-Prinzip: Die Initiative geht stets vom Terminal aus und die Smart Card ist nur darauf ausgerichtet auf Anfragen zu antworten, nicht aber selbstständig zu agieren. Die Kommunikation auf Anwendungsebene erfolgt dabei über APDUs (application protocol data units), deren Aufbau in ISO 7816-4 [11] spezifiziert ist. Diese werden unterschieden in „command APDUs“, die einen Befehl an die Chipkarte darstellen und den zugehörigen „response APDUs“, welche die Antwort der Karte bilden. Command APDUs sind aus einem obligatorischen Header und einem optionalen Body aufgebaut. Dieser Header besteht aus einem Class-Byte (CLA), einem Instruction-Byte (INS) und zwei Parameter Bytes (P1 und P2). Optional folgt dahinter ein L_c-Byte, das die Länge der Daten im Body angibt. Nach dem L_c-Byte und den zugehörigen Daten folgt optional dahinter noch ein L_e-Byte, das die Länge der zu erwartenden Antwort bestimmt.

Header				**optionaler Body**		
CLA	INS	P1	P2	L_c	Data	L_e

Abbildung 5.2.: Struktur einer Command APDU

Response APDUs bestehen aus einem voranstehenden optionalen Body und einem nachfolgenden, obligatorischen Trailer. Der Body enthält das Datenfeld und der

Trailer beinhaltet ein Statuswort, welches den Prozesszustand nach einem durchgeführten Befehl widergibt. Das Statuswort „0x9000“ bedeutet, dass das Kommando erfolgreich durchgeführt wurde.

optionaler Body	**Trailer**	
Data	SW1	SW2

Abbildung 5.3.: Struktur einer Response APDU

Beispielhaft wird im Folgenden eine zusammengehörige command und reponse APDU analysiert: Das Terminal schickt folgende APDU an die Chipkarte:

» 00 A4 04 00 08 42 65 69 73 70 69 65 6C .

Das Class-Byte indiziert bestimmte Eigenschaften, wie zum Beispiel Secure Messaging oder Command Chaining, die in dieser APDU allerdings nicht gegeben sind. Im Instruction-Byte findet sich die eigentliche Anweisung des Terminals wieder: „A4“ ist der Befehl [11] für das „Select“-Kommando, welches eine Anwendung auswählt, die geladen werden soll. Die Parameter P1 und P2 definieren weitere Eigenschaften zur jeweils gewählten Instruction. Im Datenfeld folgt nun der Bezeichner des zu ladenden Applets, genannt AID. Das L_c-Byte „08“ gibt die Länge des zu ladenden AID an. Die nachfolgenden acht Byte bilden eine ASCII-codierte AID des zu ladenden Applets. Das abschließende L_e-Byte bedeutet, dass keine Antwortdaten von der Karte erwartet werden. Anschließend antwortet die Karte nach der Auswahl des Applets mit der Response APDU:

« 90 00 .

Die Antwort enthält keine Daten, sondern nur die Statuswörter SW1 und SW2, wie das L_e-Feld der command APDU gefordert hat. Das Statuswort bedeutet eine erfolgreiche Durchführung des Befehls. Die Smart Card ist nun wieder bereit und erwartet einen Befehl vom Terminal.

5.3. Java Card

Die Java Card Technologie ist ein Mitglied der Java-Familie [13], die es ermöglicht Java Card Applets als Anwendungen auf Smart Cards auszuführen. Das Design dieser Programmiersprache ist auf Portabilität und Sicherheit ausgerichtet. Mittels der Java Card Virtual Machine (JCVM) wird in Ähnlichkeit zu Java eine Plattformunabhängkeit erreicht. Gleichzeitig bietet die Plattform durch eine strikte Trennung von der Java Card Runtime Environment (JCRE) und dem zugrundeliegenden Smart Card Betriebssystem ein hohes Maß an Sicherheit. Die Runtime Environment verwaltet die Ressourcen der Chipkarte und stellt diese über die Java Card API bereit. Gleichzeitig reguliert die Java Card Virtual Machine mit einer Applet Firewall die Separation und Interoperabilität mehrerer Applets.

Aufgrund der beschränkten Ressourcen von Smart Cards unterstützt Java Card nur eine Untermenge von Java. Keine Unterstützung finden zum Beispiel große primitive Datentypen wie long, double, float sowie Strings und Characters. Außerdem fehlen das dynamische Laden von Klassen, mehrdimensionale Felder, der Security Manager, Threads, Garbage collection, Object serialization und Object cloning. Speziell die fehlende Unterstützung des Garbage Collectors ist bei den knappen Speicherressourcen bedeutend für die Programmierung von Java Card Applets.

Java Card Programme werden von `javacard.framework.Applet` abgeleitet. Von dieser Klasse erben sie die wichtigsten Methoden für den Lebenszyklus eines Applets: `install()`, `register()`, `select()`, `process()` und `deselect()`. Ein Java Card Applet wird zu Beginn installiert und dann bei der JCRE regisitiert. Es kann dann mittels eines eindeutigen application identifiers (AID) selektiert werden und nach Verwendung wieder deselektiert werden, zum Beispiel um ein anderes Applet zu laden. Die Kommunikation mit dem Terminal über APDUs erfolgt über die Methode `process()`.

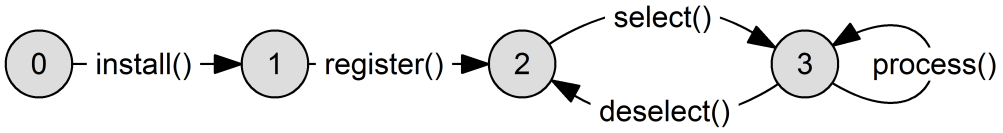

Abbildung 5.4.: Lebenszyklus eines Java Card Applets

6. Implementation

6.1. Aufbau, Hardware und verwendete Software

Die Hardwarebasis der Implementierung bildet die Displaykarte der Bundesdruckerei. Der von NXP Semiconductors gefertigte Chip auf der Karte unterstützt die Java Card API 2.2.2 [13], die Entwicklungsinfrastuktur GlobalPlatform sowie die Extended JCOP API (DoC NX823A-p5 only), die eine Erweiterung zur Java Card API bildet. Auf der Smart Card läuft das Java Card OpenPlatform (JCOP) Betriebssystem.

Bei der Anzeige auf der Smart Card handelt es sich um ein elektrophoretisches Display, wie es auch in heutigen eBooks verwendet wird. Das Display besteht aus einer transparenten Elektrode auf der Vorderseite und segmentierten Elektroden auf der Rückseite. Das Prinzip basiert darauf, dass geladene Teilchen durch Anlegen von Spannung im Feld in Bewegung geraten. Streben die Teilchen an die Oberfläche des Displays erzeugen sie einen visuellen Effekt mit dem sich Informationen, wie zum Beispiel alphanumerische Zeichen darstellen lassen. Der so erzeugt Displayinhalt bleibt bistabil, dass heißt die Partikel verharren auch nach Abbau des elektrischen Feldes in ihrer Position. Eine Änderung der Daten ist nur über die Ansteuerungselektronik möglich, der Inhalt kann von außen weder manipuliert oder gelöscht werden. Auf der verwendeten Displaykarte der Bundesdruckerei lassen sich sowohl Zahlen als auch Buchstaben darstellen. Die benötigte Zeit zur Änderung des Displays beträgt circa 0,9 bis 1,1 Sekunden. Zur softwareseitigen Ansteuerung des Displays wird eine weitere Bibliothek eingebunden.

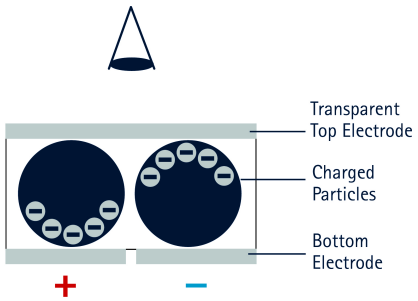

Abbildung 6.1.: Funktionsweise eines elektrophoretischen Displays

Für den Versuchsaufbau wurde das Basiskartenlesegerät (Kategorie Cat-B nach TR-03119 [6]) SDI 010 von SCM Microsystems verwendet. Um die APDU Kommunikati-

on mitlesen zu können, wurde der Treiber des Kartenlesers unter Linux installiert und der PC/SC-Daemon anschließend im Foreground Modus mit APDU Tracing neugestartet. Dies stellte sich als eines der hilfreichsten Tools zur Problemdiagnose der AusweisApp heraus, die sonst eher die Rolle einer Blackbox übernahm.

Als Entwicklungsumgebung diente Eclipse in der Version 3.2.2 mit einem JCOP Tools Plugin. Dieses Plugin enthielt einen Simulator zum Offcard Debugging von JCOP-basierten Java Cards. Allerdings waren die Funktionen der Extended JCOP API nicht für den Debugger implementiert, sodass ein Debugging nur ohne Verwendung dieser Methoden möglich war. Da jedoch diese Funktionen für die Umsetzung des Projektes grundlegend notwendig waren, musste ein Großteil der Programmierarbeit ohne Debugmöglichkeiten gestaltet werden. Umso wichtiger war deshalb das APDU Tracing über den Linux Treiber des Kartenlesegeräts. Außerdem enthält das JCOP Plugin die JCOP Shell, ein Kommandozeilenwerkzeug zum Management von Java Card Applets und zur APDU Kommunikation mit der Smard Card oder dem Simulator. Zur Automatisierung der Installation eines neuen Applets wurde ein JCOP Shell Script geschrieben, das folgende Aktionen ausführte:

- Verbindung zum Kartenlesegerät
- Auswahl und Authentisierung des Card Managers
- Löschen von Applets
- Löschen von Packages
- Upload des Packages
- Installation des Applets
- APDU-Kommandos

Als Hauptreferenz zur Programmierung des Applets diente die Java Card API 2.2.2 von Orcale [13]. Zusätzlich gab es eine Dokumentation zu den Erweiterungen der Extended JCOP API und eine weitere Dokumentation zur Ansteuerung des Displays. Die Erweiterungen der Extended JCOP API sind notwendig, weil die Methoden der Java Card API für eine Implementierung von PACE nicht ausreichend sind, wie bereits in [22] untersucht wurde. Folgende Operationen werden von der Standard Java Card API nicht unterstützt:

- EC Punktaddition
- EC Diffie-Hellman Schlüsselaustausch (ohne Key Derivation)
- CMAC-AES

Die ersten beiden Operationen werden von der Extended JCOP API abgedeckt, CMAC-AES wurde von Hand umgesetzt.

6.2. Entwicklung

Das Ziel der Implementierung war die Umsetzung des PACE-Protokolls mit einer teildynamischen PIN. Zur Vorführung der Ergebnisse war eine volle Kompatibilität zur PIN-Verwaltung der AusweisApp geplant. Als erster Schritt wurde das PACE-Protokoll gemäß der TR-03110 [5] implementiert. Die Funktionsfähigkeit wurde im Fortlauf der Programmierarbeit stets mit den Werten des EAC Worked Sample [23] abgeglichen. Zu diesem Zweck wurden die zufallsgenerierten Werte zunächst statisch festgeschrieben und die APDUs des Terminals mittels eines Shell Scripts an die Karte geschickt. Probleme bereitete hier der Schritt „Mutual Authentication“, weil das benutzte CMAC-AES nicht in der Standard API implementiert war. Nach der erfolgreichen Implementation des PACE-Protokolls wurden dann die statischen Werte entfernt und durch Zufallsgeneratoren ersetzt. Das EAC Worked Sample hatte zu diesem Zeitpunkt damit seinen Zweck erfüllt.

6.2.1. AusweisApp

Neben den APDUs für PACE mussten natürlich auch noch andere Anfragen der AusweisApp korrekt beantwortet werden. Um diese Abfragen zu identifizieren wurde die AusweisApp unter Linux installiert und gleichzeitig der Datenverkehr zum Kartenleser über den Treiber mitgelesen. Mittels dieser Untersuchungen konnten die Abfragen der AusweisApp detailliert analysiert werden. Die Ergebnisse sind in Tabelle 6.1 aufgelistet.

Die Aufgabe ist somit, alle Abfragen der AusweisApp mit validen APDUs zu beantworten. Während auf dem echten Personalausweis ein Dateisystem die Dateiabfragen verwaltet, wurde bei dieser Implementierung aufgrund des geringen Umfangs darauf verzichtet. Alle Dateisystemabfragen der AusweisApp sind also statisch eincodiert. Im Folgenden werden die einzelnen Abfragen kurz erläutert und in Tabelle 6.2 spezifisch aufgeführt:

Select MF

Das Selektieren des Masterfiles ist Teil der Dateisystemverwaltung und wird statisch mit „0x9000“ beantwortet.

Select EF_CARDACCESS

Das Selektieren des Files EF_CARDACCESS dient dem nachfolgenden Auslesen und wird ebenfalls statisch mit „0x9000“ beantwortet.

Read Binary:

Der Befehl dient zum Auslesen der vorher selektierten Datei und wird verwendet um EF_CARDACCESS zu lesen. Der Inhalt der EF_CARDACCESS ist als statisches

Tabelle 6.1.: APDU-Abfragen der AusweisApp

Aktion	Abfrage der AusweisApp
Karte auf Lesegerät legen	Read Binary(short identifier) Kartenlesermerkmale
Aufruf des Konfigurationsmenüs	Select MF MSE:AT
Aufruf des PIN-Änderungsdialogs	Select MF MSE:AT
Abbruch des PIN-Änderungsdialogs	Select MF (2x) MSE:AT
PIN-Änderung	Select MF Select EF_CARDACCESS Read Binary MSE:AT General Authenticate: Get Nonce General Authenticate: Map Nonce General Authenticate: Perform Key Agreement General Authenticate: Mutual Authentication Reset Retry Counter[a] Select MF Card Reset
Bestätigungsdialog nach PIN-Änderung	Select MF MSE:AT

[a] nur bei erfolgreicher Authentisierung; Secure Messaging Mode

Byte Array hinterlegt und wird mittels dieses Kommandos in 80-byte Blöcken ans Terminal geschickt.

Read Binary(short identifier)

Der Befehl benutzt den Parameter P1="9C". Durch das Setzen von bit8 werden bit5 bis bit1 als short identifier ausgewertet. Dadurch wird ein vorheriges Selektieren überflüssig. Der Befehl ist im Extended Length APDU Format, sodass das gesamte EF_CARDACCESS in einer APDU übermittelt werden kann.

MSE: Set AT

Das Kommando dient zum Auswählen und Initializieren der kryptografischen Protokolle. In diesem Fall wird mittels P1/P2="C1A4" das PACE-Protokoll spezifiziert. Im Tag 0x80 werden die kryptografische Mechanismen festgelegt und Tag 0x83 gibt das verwendete Passwort an. Die Antwort der Karte richtet sich nach dem Status der PIN und dem Wert des Fehlbedienungszählers. Steht der Fehlbedienungszähler auf 3, so ist die Antwort der Karte „0x9000". Nach mindestens einem Fehlversuch errechnet sich die Antwort der Karte aus „0x63C0"+ dem Wert des Fehlbedienungszählers. Im Falle von „0x63C1" befindet sich die eID-PIN im Status „suspend" und die Ausweis-App führt automatisch eine Authentisierung mittels CAN durch. Die Antwort der Karte ist darauf stets „0x9000", da die CAN ein „non-blocking" Passwort ist. Nach erfolgreichem Durchlauf des PACE-Protokolls mit der CAN befindet sich die Karte im Status „resumed". Ist auch der letzte Eingabeversuch der eID-PIN nicht erfolgreich so gibt die Karte „0x63C0" zurück und verweigert weitere Abläufe.

General Authenticate

Diese Kommandos dienen dem Ablauf des PACE-Protokolls und werden in Abschnitt 6.3 näher erläutert. Sie enthalten im Datenfeld das Statustag sowie die kryptografischen Kurvenpunkte nach ANSI X9.62 [1] im Format „04 || X || Y".

Reset Retry Counter

Mit diesem Befehl wird eine neue PIN festgelegt, die sich im Datenfeld der APDU befindet. Die Karte antwortet nach erfolgreicher Änderung mit „0x9000". Das erste Kommando wird zudem immer im Secure Messaging Modus übertragen.

6.2.2. Konzept

Für die Implementierung erfolgte eine Aufteilung in sechs Klassen, die strikt nach ihren Aufgabenbereichen getrennt sind.

Abfrage der AusweisApp	CLA	INS	P1	P2	L_c	Data Field	L_e
Select MF	00	A4	00	0C	02	3F 00	
Select EF_CARDACCESS	00	A4	02	0C	02	01 1C	
Read Binary	00	B0	00	00			80
Read Binary(short identifier)	00	B0	9C	00			00
MSE:AT	00	22	C1	A4	0F	80 0A ...83 01 *password*	
GA: Get Nonce	10	86	00	00	02	7C 00	00
GA: Map Nonce	10	86	00	00	45	7C 43 81 41 *EC Point*	00
GA: Perform Key Agreement	10	86	00	00	45	7C 43 83 41 *EC Point*	00
GA: Mutual Authentication	00	86	00	00	0C	7C 0A 85 08 *MAC*	00
Reset Retry Counter	00	2C	02	03	06	*new PIN*	00

Tabelle 6.2.: Aufbau der Kommando APDUs

Die Hauptklasse leitet sich von der Applet-Klasse ab und bildet somit das Grundgerüst des Java Card Applets. Zusätzlich wird das Extended Length Interace implementiert. Zum Installationsaufruf werden Instanzen der PACE-, PIN-, CAN-, CMAC-AES- und der Secure Messaging-Klasse erzeugt. Somit wird bereits zu diesem Zeitpunkt der Speicherbedarf alloziiert, den die Karte in ihrem Lebenszyklus benötigt. Sämtliche eingehenden APDUs werden von der Klasse verwaltet und die enthaltenen Daten an die anderen Klassen weitergereicht. Die errechneten Antworten werden dann wieder an die Hauptklasse zurück übergeben und diese schickt sie als APDU zurück an das Terminal. Desweiteren steuert diese Klasse mit den Daten der PIN-Klasse das Display an, um die teildynamische PIN anzuzeigen.

Die kryptografischen Abläufe des PACE-Protokolls sind in der gleichnamigen Klasse untergebracht. Sie erhält zur Initialisierung die Referenz der Passwörter-Klassen PIN und CAN sowie zur Hilfsklasse CMAC-AES. Die Kryptomechanismen des PACE-Protokolls sind in der AusweisApp statisch festgelegt, sodass auch nur eine Umsetzung mit genau diesen Parametern erfolgte:

- Elliptische Kurven Kryptografie mit Bitlänge 256
- statische Domainparameter BrainpoolP256r1
- symmetrische Verschlüsselung AES mit Bitlänge 128
- Message Authentication Code mit CMAC-AES

Die Umsetzung erfolgte in Kombination von der Java Card API und der Extended JCOP API.

Die PIN-Klasse verwaltet den Zugriff auf die PIN sowie den Fehlbedienungszähler. Sie beherrscht dabei drei Modi:

- statische PIN ohne dynamische Stellen
- teildynamische PIN mit statischer Positionierung

- teildynamische PIN mit randomisierter Positionierung

Dieser Modus wird zur Initialisierung festgelegt. Die Klasse ermöglicht eine dynamische Generierung der PIN und schickt die Zufallszahlen über die Hauptklasse an das Display. Die CAN-Klasse ist analog zur PIN aufgebaut und enthält die Möglichkeit einer dynamischen CAN, deren Inhalt auf dem Display ausgegeben werden kann.

Ein Secure Messaging Kanal entsteht aus den berechneten Schlüsseln des PACE-Protokolls. Diese werden an die Secure-Messaging-Klasse weitergegeben, welche dann das Ver- und Entschlüsseln von APDUs übernimmt. Auch hierbei wird wieder der CMAC-AES-Algorithmus benötigt.

Da in der Standard Java Card API kein CMAC-AES enthalten ist, wurde dieser in einer eigenen Klasse selbst implementiert. Es handelt sich um eine Weiterentwicklung bestehender MACs und wurde vom NIST unter dem Titel „Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication“ [19] veröffentlicht.

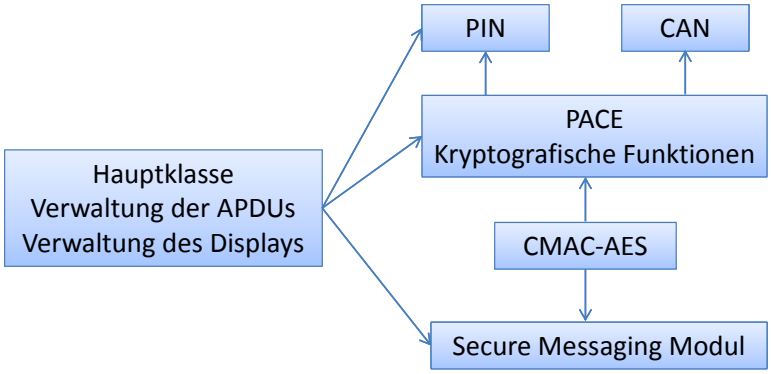

Abbildung 6.2.: Abhängigkeiten der Klassen

Als weiteres Problem stellte sich heraus, dass Java Card Applets stets mit einem Selektierungskommando ausgewählt werden müssen, bevor sie ihre Arbeit beginnen können. Die AusweisApp erwartet aber keine Java Card und sendet somit auch kein Selektierungskommando. Die Selektierung musste somit implizit geschehen mit Hilfe einer Flag im Installationsaufruf der Karte. Mittels der Option `--default` wird das Applet immer beim ATR (Answer to Reset) automatisch selektiert und kann so die Kommandos der AusweisApp empfangen.

6.3. PACE

Der Übergang von kontaktbehafteten zu kontaktlosen Karten bringt neben einem Komfortgewinn auch neue Angriffsvektoren mit sich. Anders als bei kontaktbehafteten Karten muss die Funkschnittstelle abgesichert werden. Die Aufgabe des PACE-Protokolls ist hierbei die kryptografische Sicherheit der Bindung zwischen Smart

Card und Kartenleser. PACE ist als Teil des Extended Access Control (EAC) der Nachfolger des Basic Access Control (BAC) und wurde vom BSI für den Einsatz im neuen Personalausweis entwickelt. Das Protokoll verwendet als Eingabe ein Passwort geringer Entropie, mit dem sich der Nutzer authentisiert und etabliert nachfolgend Sitzungsschlüssel mit starker Entropie, mit der die fortfolgende Kommunikation verschlüsselt und authentisiert wird. PACE ist in der technischen Richtlinie [5] zunächst generisch beschrieben und kann sowohl mit elliptischen Kurven (ECC) als auch mit RSA in verschiedenen Schlüssellängen durchgeführt werden. Da ECC bei vergleichbarer Sicherheit gegenüber RSA eine geringere Schlüssellänge benötigt und somit die nötige Effizienz für Kryptografie auf Smart Cards erbringt, wird sich diese Kurzdarstellung des Protokolls auf erstere beschränken. Zudem wird im neuen Personalausweis nur die Kryptografie mit elliptischen Kurven verwendet.

Die Kryptografie mit elliptischen Kurven basiert auf dem endlichen Körper $\mathbb{F}_p$, p prim. Die elliptische Kurve E über $\mathbb{F}_p$ wird nach der Weierstraß Gleichung beschrieben mit

$$y^2 = x^3 + ax + b \qquad a, b \in \mathbb{F}_p \qquad 4a^3 + 27b^2 \neq 0$$

Die Domain Parameter der elliptischen Kurve ergeben sich aus dem endlichen Körper $\mathbb{F}_p$, den Koeffizienten a und b, einem Basispunkt $G \in E(\mathbb{F}_p)$ sowie seiner Ordnung n und dem Kofaktor h. Der Basispunkt G generiert dabei eine zyklische Gruppe der Ordnung n in $E(\mathbb{F}_p)$:

$$\langle G \rangle = \{G, [2]G, \ldots, [n-1]G, [n]G\}$$

Beim neuen Personalausweis sind die möglichen Suiten von Domain Parametern vorgegeben und im EF_Cardaccess angegeben.

Das Problem des Diskreten Logarithmus auf elliptische Kurven lautet: Seien die Domainparamter D und ein Punkt $P \in \langle G \rangle$ gegeben, so wird ein integer k gesucht mit $0 \leq k \leq n-1$, sodass $[k]G = P$. Für den Diffie Hellman Schlüsselaustausch für elliptische Kurven folgt daraus: Seite A wählt einen integer $\widetilde{d_A}$ als zufälligen privaten Schlüssel und berechnet den öffentlichen Schlüssel $\widetilde{P_A} = [\widetilde{d_A}]G$. Seite B wählt analog einen integer $\widetilde{d_B}$ als zufälligen privaten Schlüssel und berechnet den öffentlichen Schlüssel $\widetilde{P_B} = [\widetilde{d_B}]G$. A und B tauschen nun ihre öffentlichen Schlüssel $\widetilde{P_A}$ und $\widetilde{P_B}$ aus und berechnen ein gemeinsames Geheimnis $S_{AB} = ECKA(\widetilde{P_A}, \widetilde{d_B}, D) = ECKA(\widetilde{P_B}, \widetilde{d_A}, D)$. Auf weitere Details zum Elliptic Curve Key Agreement ECKA sowie zu elliptischen Kurven soll an dieser Stelle verzichtet werden.

Im Folgenden werden die einzelnen Schritte des PACE-Protokolls erläutert, die auch noch einmal in Abbildung 6.3 dargestellt sind. Im Vorfeld werden die statischen Domain Parameter aus dem EF_Cardaccess ausgelesen. Die folgenden Operationen von PACE können in vier Schritte untergliedert werden:

(1) Get Nonce

Im ersten Schritt wird von der Smart Card zunächst eine zufällige Zahl s erzeugt. Smart Card und Terminal leiten nun mit einer Hashfunktion aus dem Passwort π einen symmetrischen Schlüssel K_π ab. Die Zufallszahl s wird nun mit K_π verschlüsselt und das Ergebnis z zum Terminal geschickt. Nach Eingabe des Passworts π am Terminal, wird die Nachricht z enschlüsselt und man erhält die Zufallszahl s.

(2) Map Nonce

Im zweiten Schritt wird der statische Basispunkt G mit der Zufallszahl s zu einem dynamischen Basispunkt $\widetilde{G}$ abgebildet. Dafür wird zunächst ein Diffie Hellman Schlüsselaustausch auf Basis von G durchgeführt um einen zufälligen Punkt H zu erstellen. Danach wird der neue Basispunkt $\widetilde{G} = s \cdot G + H$ errechnet.

(3) Perform Key Agreement

In Schritt drei wird ein weiterer Diffie Hellman Schlüsselaustausch auf Basis des neuen Punkts $\widetilde{G}$ durchgeführt. Das Ergebnis ist der geheime Punkt K, den sowohl Terminal als auch Smart Card kennen.

(4) Mutual Authentication

Aus dem Punkt K werden nun die Schlüssel K_{Enc} zur Verschlüsselung und K_{MAC} zur Errechnung von Message Authentication Codes (MAC) abgeleitet. Der Schlüssel K_{MAC} wird daraufhin zur Mutual Authentication verwendet, bei der die Authentizität des Kommunikationspartners überprüft wird. Nach erfolgreichem Ablauf des PACE-Protokolls wird mit den K_{Enc} und K_{MAC} ein Secure Messaging Kanal aufgebaut, der die weitere Kommunikation sichert.

Die Vorteile von PACE liegen in den starken, zufälligen Session Keys, die aus einem gemeinsamen Passwort mit niedriger Entropie abgeleitet werden. Somit ist Brute Force der einzig mögliche Angriff auf das Passwort. Dieses wird jedoch durch einen Fehlbedienungszähler verhindert. Auf eine detaillierte Analyse zur Sicherheit von PACE wird verzichtet.

Terminal		**Smart Card**	
		wähle zufällig nonce s	(1)
$K_\pi = H(\pi\|\|1)$		$K_\pi = H(\pi\|\|1)$ $z = ENC(K_\pi, s)$	
	$\xleftarrow{z}$		
$s = DEC(K_\pi, z)$			
erzeuge zufälliges KeyPair (PK_{PCD}, SK_{PCD}, D)		erzeuge zufälliges KeyPair $(PK_{PICC}, SK_{PICC}, D)$	(2)
	$\xrightarrow{PK_{PCD}}$		
	$\xleftarrow{PK_{PICC}}$		
$H = KA(PK_{PICC}, SK_{PCD}, D)$ $\widetilde{G} = s \cdot G + H$		$H = KA(PK_{PCD}, SK_{PICC}, D)$ $\widetilde{G} = s \cdot G + H$	
erzeuge zufälliges KeyPair $(PK_{PCD}, SK_{PCD}, \widetilde{D})$		erzeuge zufälliges KeyPair $(PK_{PICC}, SK_{PICC}, \widetilde{D})$	(3)
	$\xrightarrow{PK_{PCD}}$		
	$\xleftarrow{PK_{PICC}}$		
$K = KA(PK_{PICC}, SK_{PCD}, \widetilde{D})$		$K = KA(PK_{PCD}, SK_{PICC}, \widetilde{D})$	
$K_{ENC} = H(K\|\|2)$ $K_{MAC} = H(K\|\|3)$		$K_{ENC} = H(K\|\|2)$ $K_{MAC} = H(K\|\|3)$	(4)
$T_{PCD} = MAC(K_{MAC}, PK_{PCD})$		$T_{PICC} = MAC(K_{MAC}, PK_{PICC})$	
	$\xrightarrow{T_{PCD}}$		
	$\xleftarrow{T_{PICC}}$		
$T'_{PICC} = MAC(K_{MAC}, SK_{PCD})$ überprüfe $T'_{PICC} = T'_{PICC}$		$T'_{PCD} = MAC(K_{MAC}, SK_{PICC})$ überprüfe $T'_{PCD} = T'_{PCD}$	

Tabelle 6.3.: PACE

6.4. Secure Messaging

Die von PACE generierten Schlüssel werden nun benutzt, um einen sicheren Kommunikationskanal zwischen Karte und Terminal aufzubauen und zu authentisieren. Der Mechanismus für die Verschlüsselung und Entschlüsselung der APDUs wird in der technischen Richtlinie TR-03110 [3] erläutert. Die zu verschlüsselnde Kommando APDU wird zunächst in Commandheader (CLA, INS, P1, P2), L_c, Data und L_e aufgeteilt. Der Commandheader bleibt unverschlüsselt, allerdings wird das Bit8 des CLA-Byte auf 1 gesetzt, um den Secure-Messaging-Modus anzukündigen. Das Datenfeld wird zunächst in Blöcke zu je 16 Byte aufgeteilt und optional mit Padding aufgefüllt. Die Datenverschlüsselung erfolgt dann mit AES im CBC-Mode mit dem Schlüssel K_{enc} und dem Initialisierungsvektor $IV = E(K_{enc}, SSC)$. Der Send Sequence Counter (SSC) ist ein unsigned integer, der vor jeder Command und Response APDU um eins erhöht wird. Den verschlüsselten Daten geht das Tag '87' voran, gefolgt vom optionalen Tag '97', welches das L_e Byte der originalen APDU speichert. Über diese Daten wird nun eine kryptografische Prüfsumme mittels CMAC-AES [19] erzeugt und mit dem Tag '8E' abgespeichert. Die verschlüsselte APDU setzt sich zusammen aus dem unverschlüsselten Command Header, einem neu berechneten $L_{c'}$, den verschlüsselten Daten, dem optionalen L_e Byte sowie dem Message Authentification Code (MAC). Um eine Secure Messaging APDU zu entschlüsseln wird in der umgekehrten Reihenfolge verfahren, dabei beweist speziell der MAC die Integrität der empfangenen Nachricht.

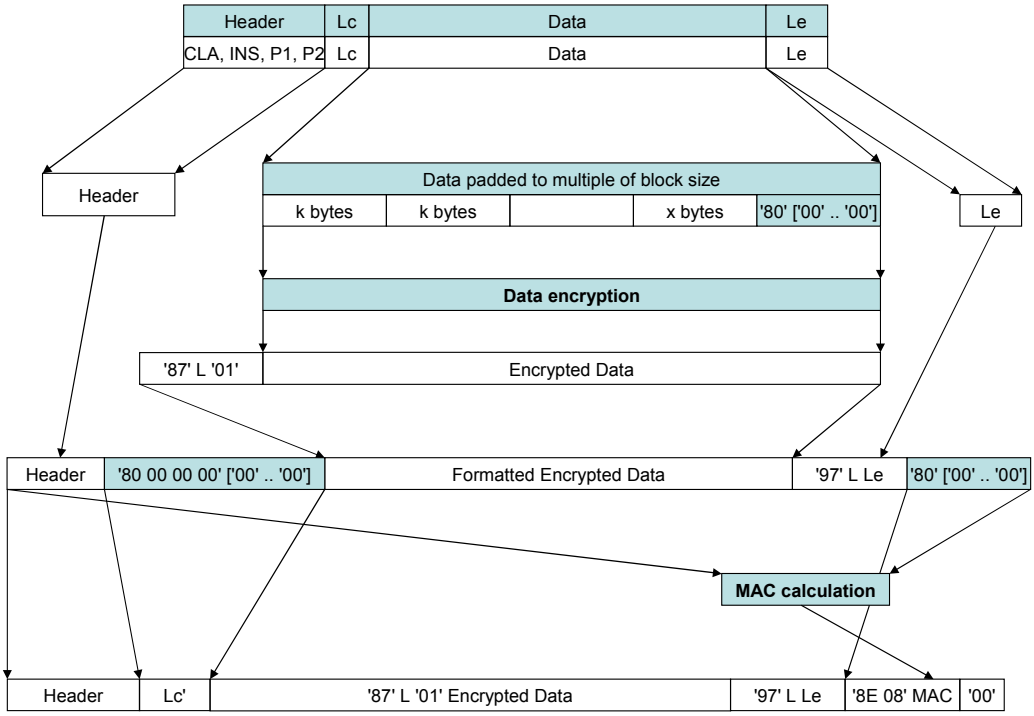

Abbildung 6.3.: Secure Messaging für Command APDUs [5]

Die Verschlüsselung von Response APDUs verläuft nach einem ähnlichen Verfahren. Hierbei werden wiederum die Daten in 16 Byte Blöcke aufgeteilt und mit dem oben beschriebenen Verfahren symmetrisch verschlüsselt. Das Statuswort der Antwort wird unverschlüsselt nach den Daten mit dem Tag '99' übertragen. Über den SSC, die verschlüsselten Daten und das Statuswort wird nun wieder ein MAC gebildet und hinter dem Tag '8E' gespeichert. Die verschlüsselte APDU bildet sich aus den verschlüsselten Daten, dem Statuswort der originalen APDU, dem MAC sowie dem Statuswort der Secure Messaging APDU. Das Terminal entschlüsselt die Daten auf umgekehrtem Weg und überprüft die Echtheit der Daten mittels dem MAC.

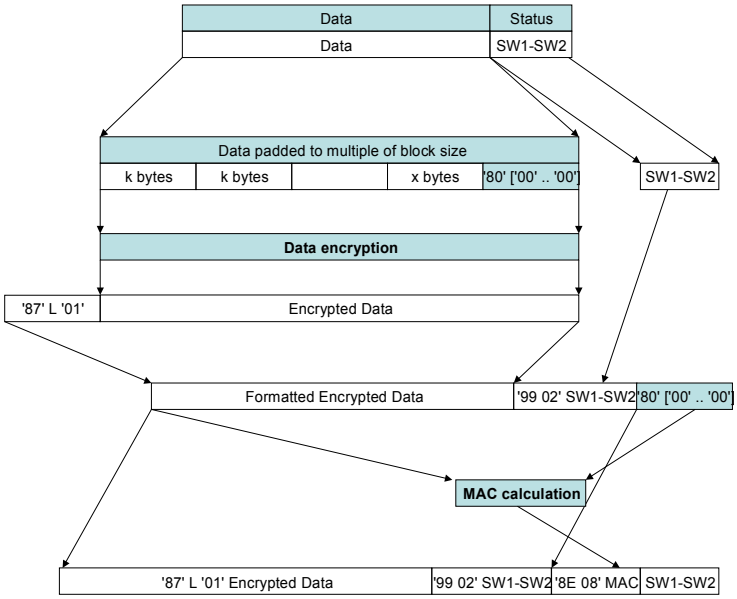

Abbildung 6.4.: Secure Messaging für Response APDUs [5]

6.5. Auswertung

Die Implementation aller für das PIN-Management erfoderlichen Protokolle konnte erfolgreich durchgeführt werden und die Karte verhält sich diesbezüglich konform zu einem echten Personalausweis. Lediglich in der Laufzeit macht sich ein Unterschied bemerkbar: Während der neue Personalausweis für eine PIN-Änderung 0,7 Sekunden benötigt, beträgt die durhschnittliche Laufzeit mit einer Java-Displaykarte 3,4 Sekunden. Befindet sich die eID-PIN nach zweimaliger Fehleingabe im Status „resumed“, so wird ein Durchlauf des PACE-Protokolls mit der CAN durchgeführt (wiederum 3,4s) und innerhalb des entstandenen Secure Messaging Kanals ein PACE-Durchlauf mit der PIN durchgeführt. Dieser Protokollablauf muss jedoch durchgehend verschlüsselt werden, sodass die Zeit für den Ablauf von PACE in diesem Fall auf 5,3 Sekunden ansteigt (vergleiche Tabelle 6.5).

Die erhöhte Rechenzeit im Vergleich zum echten Personalausweis ist zu etwa 75% den Elliptic-Curve-Operationen zuzuschreiben, sodass das Potential für Optimierungen nicht sehr groß ist. Da beim Secure Messaging sowohl die Nutzdaten verschlüsselt werden als auch ein MAC gebildet wird steigt hierbei die Ausführungszeit pro APDU um 300-400 ms an.

Schritt	Zeit in ms
Select EF_MF	5
Select EF_CARDACCESS	5
Read Binary	30
MSE:AT	9
General Authenticate: Get Nonce	97
General Authenticate: Map Nonce	1581
General Authenticate: Perform Key Agreement	941
General Authenticate: Mutual Authentication	385
Reset Retry Counter	367
Select EF_MF	5
Card Reset	-
Select EF_MF	19
MSE:AT	1505

Tabelle 6.4.: Zeitablauf für einen Durchlauf der PIN-Änderung

Schritt	Zeit in ms	Differenz in ms
MSE:AT	335	326
General Authenticate: Get Nonce	480	383
General Authenticate: Map Nonce	2030	449
General Authenticate: Perform Key Agreement	1399	458
General Authenticate: Mutual Authentication	743	358

Tabelle 6.5.: Zeitablauf für PACE mit Secure Messaging

7. Ausblick

Nach der vollständigen Implementierung von PACE wäre es möglich zusätzlich die Protokolle der Terminal Authentication (TA), der Chip Authentication (CA) und weitere benötigte Kompontenen auf die Displaykarte zu programmieren um so die Funktionalität eines neuen Personalausweises mit integriertem Display zu verwirklichen. Die Laufzeit von PACE wäre bei einer Umsetzung der dynamischen PIN in einer neuen Ausweisgeneration kein Problem, da sich an den kryptografischen Protokollen außer der verwendeten PIN nichts ändert. Gleichzeitig könnte man weitere Vorteile des Displays miteinbauen. So wäre die dynamische CAN eine leicht umzusetzende Möglichkeit. Je nach Größe und Umfang des Displays wäre eine Ausgabe auf dem Display möglich, die den Nutzer kontrollieren lässt, welche eID-Funktion vom Terminal angefordert wird, wie in Abschnitt 4.5 erläutert wurde.

A. APDU-Trace PACE

```
00000000 APDU: 00 A4 00 0C 02 3F 00
00005262 SW: 90 00

00000455 APDU: 00 A4 00 0C 02 3F 00
00005424 SW: 90 00

00000195 APDU: 00 A4 02 0C 02 01 1C
00005436 SW: 90 00

00000179 APDU: 00 B0 00 00 80
00018773 SW: 31 81 82 30 0D 06 08 04 00 7F 00 07 02 02 02 02 01 02
   30 12 06 0A 04 00 7F 00 07 02 02 03 02 02 02 01 02 02 01 41 30
   12 06 0A 04 00 7F 00 07 02 02 04 02 02 02 01 02 02 01 0D 30 1C
   06 09 04 00 7F 00 07 02 02 03 02 30 0C 06 07 04 00 7F 00 07 01
   02 02 01 0D 02 01 41 30 2B 06 08 04 00 7F 00 07 02 02 06 16 1F
   65 50 41 20 2D 20 42 44 72 20 47 6D 62 48 20 2D 20 54 65 73 74
   6B 61 72 74 65 90 00

00000159 APDU: 00 B0 00 80 80
00010909 SW: 20 76 32 2E 30 04 49 02 1D 41 19 28 80 0A 01 B4 21 FA
   07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 10
   10 29 12 33 62 82

00045975 APDU: 00 22 C1 A4 0F 80 0A 04 00 7F 00 07 02 02 04 02 02
   83 01 03
00008555 SW: 90 00

00000252 APDU: 10 86 00 00 02 7C 00 00
00097373 SW: 7C 12 80 10 49 B9 AE C2 E0 CA 6F F1 B2 30 96 C1 A3 FA
   79 6E 90 00

00008408 APDU: 10 86 00 00 45 7C 43 81 41 04 42 32 E8 6D 12 A9 66
   55 26 5B 23 77 E1 3A 0A D5 46 FB B2 08 AB 12 69 1A 8D 6F 39 3A
   78 C7 63 41 80 B7 57 3F 28 5D 33 FE D6 77 4E FC E1 DF 32 38 C3
   5B EC AB 13 3F 57 DD 0C 4E EF BE AD 8C 1F CB 00
01581035 SW: 7C 43 82 41 04 03 5F CA 88 D4 2E FD 1E F4 29 FA A0 C0
   CE 5C 0F 54 28 0D 96 54 28 77 B7 67 46 FA A1 85 74 F9 DF 2F 93
   28 5A B5 6A 3B 2F 63 A0 23 62 C1 87 FD 2F B4 26 63 6E 6F B3 20
   B1 09 86 AD F5 6C 02 D6 B2 90 00
```

```
00020461 APDU: 10 86 00 00 45 7C 43 83 41 04 47 56 D4 C5 58 3A EA
   19 B6 68 04 37 78 96 EE 7C 18 37 95 59 58 EE 6F 88 15 CF 34 46
   17 BD 94 1C 6E F4 EA 95 1D F9 DF 55 A7 CB 10 5B 0F 4F 98 A0 85
   71 91 E7 41 CB 12 D1 13 E3 E2 3A AE BB 15 0B 00
00941389 SW: 7C 43 84 41 04 2B 79 34 86 B1 A8 19 75 64 5A E1 B3 6E
   5C 55 DE 40 76 DA 38 40 08 69 CD D3 93 85 75 A8 E4 56 0D 5D 3D
   07 B2 24 6E 95 69 7B B9 9C B3 2F 1C 4F 2C 2B F4 D8 98 B1 71 C1
   4B D0 DB 06 B7 03 18 9F 04 90 00

00007253 APDU: 00 86 00 00 0C 7C 0A 85 08 DE 8B CE 00 5A 03 F5 A9 00
00385207 SW: 7C 0A 86 08 BC 43 5B 21 35 E3 C5 B1 90 00

00000487 APDU: 0C 2C 02 03 1D 87 11 01 AD BB 8F AA 84 E7 33 9A 0F
   4E 91 52 77 49 45 01 8E 08 58 4E A0 96 3D B1 E2 69 00
00366698 SW: 99 02 90 00 8E 08 BC C5 DC D3 75 C3 DD 8D 90 00

00000981 APDU: 00 A4 00 0C 02 3F 00
00005534 SW: 90 00

00000021 winscard.c:498:SCardReconnect() Attempting reconnect to
   token.
00062038 winscard.c:583:SCardReconnect() Reset complete.

00000153 Card ATR: 3B FE 91 00 FF 91 81 71 FE 40 00 41 28 00 11 80
   B1 4A 43 4F 50 76 32 34 32 52

00000000 APDU: 00 A4 00 0C 02 3F 00
00019161 SW: 90 00

00000187 APDU: 00 22 C1 A4 0F 80 0A 04 00 7F 00 07 02 02 04 02 02
   83 01 03
01505045 SW: 90 00
```

B. APDU-Trace PACE mit Secure Messaging

```
00000000 APDU: 0C 22 C1 A4 1D 87 11 01 53 34 F9 9E AB A0 1E 0C 6C
   94 C7 50 6B 23 1D 7A 8E 08 5A 96 50 25 E8 C7 8B DA 00
00334922 SW: 99 02 63 C1 8E 08 1D A5 BD C4 6F 7E F2 04 63 C1

00000402 APDU: 1C 86 00 00 20 87 11 01 1E B3 F1 35 0E 79 A5 2A 7E
   20 7E 26 54 53 F3 A6 97 01 00 8E 08 2F E0 DE 5E 79 67 3D F1 00
00479502 SW: 87 21 01 2E 71 C3 FF 2C 10 25 82 78 44 AD 14 FB C2 4D
   94 F6 ED 34 00 20 B8 7D 0D 7F 46 F5 9C B6 50 B0 5F 99 02 90 00
   8E 08 BB 93 5C DE 31 51 F5 04 90 00

00009330 APDU: 1C 86 00 00 60 87 51 01 16 1B 5B 6C 75 DF 0A 3B CF
   08 F9 48 87 BC 5B D8 2C 42 02 6C 67 49 36 4E 01 EE 91 F6 3E 83
   9F 9A E6 88 65 26 34 B9 EE 9E 37 4D 7E 5B F3 81 BB 93 3F ED 25
   AB DA CB C7 61 B8 74 71 3C 01 B5 04 AE 55 A1 AD BA 86 73 99 D8
   B0 B5 F0 C7 B0 CF E3 34 97 01 00 8E 08 2E 16 55 74 5B 0C E5 EB 00
02029805 SW: 87 51 01 FA F6 C5 2C 5F 0E A6 EA 31 89 CE 47 95 7F C0
   D0 0A CA 7C 81 34 2D 83 E2 7D 06 96 44 F8 E7 DF BC 41 37 79 B0
   BA B9 E1 78 38 EC 66 21 93 D8 65 CF B3 C5 AD 7A 1A 3C 3D E5 59
   57 A9 43 40 94 36 DC CB 38 9B E1 11 E7 A3 B1 47 98 31 DD 42 A3
   12 38 99 02 90 00 8E 08 4C D2 48 86 AA 37 33 05 90 00

00017172 APDU: 1C 86 00 00 60 87 51 01 3F 7A BE 28 D6 D6 04 EF 03
   CA E2 44 B2 8D 04 FA FF 3A 8D 6F 8B 2A 39 BD 40 B7 41 41 86 E1
   88 85 1F 6F 12 50 E8 6A 9B 7B E8 D5 D8 9D 24 77 DD 39 7E 22 98
   85 DF 0A 26 D5 C4 CE 24 F0 E9 2F 59 22 47 46 93 1B 39 BA 48 A2
   61 E8 DA 27 77 9C 16 1C 97 01 00 8E 08 A7 6C 6C C0 5D 83 96 7C 00
01398661 SW: 87 51 01 6A AC CE 5C 45 FE 31 0C 21 0D A0 31 4A 05 93
   8C A6 5E 07 99 1E DA 2C A7 94 1B 84 12 C7 FF 51 6F CF 38 81 74
   48 B0 C0 58 8A A6 81 57 09 D3 BA 94 D9 CD C9 23 58 C3 47 9D AC
   FD 80 3C 0F AA A8 1D 7E 93 12 47 44 F4 2A C7 48 94 94 E2 7A EA
   71 4F 99 02 90 00 8E 08 FD 78 8C 59 A9 DD 8F 70 90 00

00005880 APDU: 0C 86 00 00 20 87 11 01 B1 A1 0B 98 C0 5B 26 AB 9E
   61 9C 02 7E 3A D1 83 97 01 00 8E 08 2C 96 C2 5A CA 0A A7 16 00
00743001 SW: 87 11 01 B4 05 2E 6B CF EA A5 DA 67 F1 F8 AD DE FD C1
   CC 99 02 90 00 8E 08 61 E2 CF DF 20 A1 22 89 90 00
```

```
00000478 APDU: 0C 2C 02 03 1D 87 11 01 67 8B EF 8E 7D EA A1 D2 83
   4D 2A 7D 6B AF E7 83 8E 08 17 EC 67 F9 48 AE 66 01 00
00324223 SW: 99 02 90 00 8E 08 32 51 70 DF A8 15 45 E0 90 00

00000899 APDU: 00 A4 00 0C 02 3F 00
00005294 SW: 90 00

00000021 winscard.c:498:SCardReconnect() Attempting reconnect to
   token.
00061730 winscard.c:583:SCardReconnect() Reset complete.

00000019 Card ATR: 3B FE 91 00 FF 91 81 71 FE 40 00 41 28 00 11 80
   B1 4A 43 4F 50 76 32 34 32 52

00000012 APDU: 00 A4 00 0C 02 3F 00
00019273 SW: 90 00

00000170 APDU: 00 22 C1 A4 0F 80 0A 04 00 7F 00 07 02 02 04 02 02
   83 01 03
01493326 SW: 90 00
```

Literatur

[1] *ANSI X9.62 - The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute.

[2] Prof Dr. Johannes Buchmann. *Einführung in die Kryptographie*. Springer, 2004.

[3] Bundesamt für Sicherheit in der Informationstechnik. *Technical Guideline TR-03110, Anhang F, Seiten 99-103. Advanced Security Mechanisms for Machine Readable Travel Documents*. 2.05. 2010. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/TR-03110_v205_pdf.pdf.

[4] Bundesamt für Sicherheit in der Informationstechnik. *Technical Guideline TR-03110, Seite 24, 3.3.1. Advanced Security Mechanisms for Machine Readable Travel Documents*. 2.05. 2010. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/TR-03110_v205_pdf.pdf.

[5] Bundesamt für Sicherheit in der Informationstechnik. *Technical Guideline TR-03110. Advanced Security Mechanisms for Machine Readable Travel Documents*. 2.05. 2010. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/TR-03110_v205_pdf.pdf.

[6] Bundesamt für Sicherheit in der Informationstechnik. *Technische Richtlinie TR-03119. Anforderungen an Chipkartenleser mit ePA Unterstützung*. 1.1. 2009. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03119/BSI-TR-03119_V1_pdf.

[7] Bundesamt für Sicherheit in der Informationstechnik. *Technische Richtlinie TR-03127. Architektur Elektronischer Personalausweis*. 1.13. Okt. 2010. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03127/BSI-TR-03127_pdf.pdf.

[8] Zhiqun Chen. *Java Card Technology for Smart Cards. Architecture and Programmer's Guide*. Sun Microsystems, 2004.

[9] *ISO 14443 - Identification cards - Contactless integrated circuit cards*. International Organization for Standardization.

[10] *ISO 7810 - Identification cards*. International Organization for Standardization.

[11] *ISO 7816 - Identification cards - Integrated circuit cards*. International Organization for Standardization.

[12] Bundesamt für Sicherheit in der Informationstechnik. *Technische Richtlinie TR-03127, Seite 18, Eingabe der eID-PIN. Architektur Elektronischer Personalausweis*. 1.13. Okt. 2010. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03127/BSI-TR-03127_pdf.pdf.

[13] *Java Card Platform Specification 2.2.2*. Oracle. URL: http://java.sun.com/javacard/specs.html.

[14] Axel Kossel. *CCC zeigt Sicherheitsprobleme beim elektronischen Personalausweis auf [Update]*. URL: http://heise.de/-1083649.

[15] Frank Morgner und Dominik Oepen. „"Die gesamte Technik ist sicher". Besitz und Wissen: Relay-Angriffe auf den neuen Personalausweis“. In: *27th Chaos Communication Congress*. Dez. 2010, S. 26–31. URL: http://events.ccc.de/congress/2010/Fahrplan/attachments/1720_27C3_Morgner_Oepen.pdf.

[16] Max Moser und Thorsten Schröder. *SuisseID / Smartcard USB Takeover*. 2010. URL: http://www.vimeo.com/15155073.

[17] *PKI for Machine Readable Travel Documents offering ICC Read-Only Access*. Techn. Ber. Version 1.1. INTERNATIONAL CIVIL AVIATION ORGANIZATION, 2004.

[18] Wolfgang Rankl und Wolfgang Effing. *Handbuch der Chipkarten*. Hanser, 2002.

[19] *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. National Institute of Standards und Technology. URL: http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.

[20] *Sicherheitsmassnahmen für den Umgang mit dem neuen Personalausweis*. BSI. URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/SicherheitImNetz/Personalausweis/Sicherheitstipps/Sicherheitstipps_node.html.

[21] Stefan Trölenberg. „Integration flexibler Elektronikkomponenten in Smart Cards Aus Polykarbonat“. Magisterarb. Technische Hochschule Wildau, 2010.

[22] Markus Ullmann u. a. *Password Authenticated Key Agreement for Contactless Smart Cards*.

[23] *Worked Example for Extended Access Control (EAC)*. Version 1.01. BSI. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/BSI_EAC-Worked-Example.zip?__blob=publicationFile.

[24] erdgeist. *Praktische Demonstration erheblicher Sicherheitsprobleme bei Schweizer SuisseID und deutschem elektronischem Personalausweis*. Chaos Computer Club, Okt. 2010. URL: http://ccc.de/de/updates/2010/sicherheitsprobleme-bei-suisseid-und-epa.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfmittel angefertigt habe.

Berlin, den 7. November 2011

Paul Bastian