Masterarbeit

Design und Implementierung einer Anwendungsanalyse für Smart Cards

Andreas Pöllabauer

Institut für Technische Informatik
Technische Universität Graz

Begutachter: Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger
Betreuer: Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger

Graz, im Jänner 2012

Kurzfassung

Spezifikationen von Smart Card Anwendungen sind aufgrund von ständig wachsenden Sicherheits-Anforderungen permanenten Änderungen ausgesetzt. Damit die Performance des gesamten Systems damit Schritt halten kann, muss die Hardware neue Funktionen unterstützen. Für die Entwicklung eines Systems ist die richtige Partitionierung der Aufgaben in Hardware und Software von großer Bedeutung. Zielsetzung dieser Arbeit ist es, aus der Sicht einer Anwendung die Anforderungen an die Hardware von neuen Plattformen zu definieren.

Da es viele unterschiedliche Smart Card Anwendungen gibt, wird im ersten Schritt versucht, die Anwendungen in Anwendungsdomains zu unterteilen. Aus den Anwendungen einer Domain werden Gemeinsamkeiten hinsichtlich der Anforderungen an die Hardware extrahiert. Zu diesem Zweck werden anhand der Anwendungen Kennwerte für die jeweilige Domain ermittelt.

Für diese Arbeit wurden verschiedene Anwendungen und Protokolle mit Hilfe eines SystemC Modells analysiert und ihre Anforderungen an die Hardware bestimmt. Es wurde überprüft, ob sich für alle Anwendungen, die sich in einer Domain befinden, die gleichen Kennwerte für die Hardwarenutzung ergeben. Abschließend wurden in weiterer Folge Ähnlichkeiten der unterschiedliche Domains betrachtet.

Abstract

There are often new specifications for applications and cryptography of smart cards. In order to support these new protocols the hardware support of the cards is required. For development of new applications the choice of the right hardware platform is important. The problem is how to determine the requirements of the platform for new applications.

Because of the big amount of different smart card applications, a set of applications will be divided into application domains. Common requirements to the hardware of the smart card for the different application domains were extracted according to the chosen applications.

The different applications were analyzed using a SystemC model of a smart card. Their requirements to the hardware were determined. It was verified if all applications within a domain have the same requirements.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 19. Jänner 2012 .....

(Unterschrift)

Danksagung

Diese Diplomarbeit wurde am Institut für Technische Informatik an der Technischen Universität Graz in Kooperation mit NXP Semiconductors Austria GmbH mit Sitz in Gratkorn durchgeführt.

Von Seiten des Instituts möchte ich mich bei meinem Betreuer Ass.Prof. Dipl.-Ing. Dr. techn. Christian Steger für seine Begleitung während dieser Arbeit bedanken.

Seitens NXP gilt mein Dank vor allem meinen beiden Betreuern Dipl.-Ing. Johannes Loinig und Dipl.-Ing. Dr. Wolfgang Eber die mich mit ihrem Wissen tatkräftig unterstützt haben. Auch allen anderen die mir mit ihrem Wissen zur Seite standen und die ich jetzt nicht alle namentlich erwähnen kann, möchte ich meinen Dank aussprechen.

Ein besonderer Dank gilt meiner Familie für ihre Unterstützung und Geduld.

Graz, im Jänner 2012 Andreas Pöllabauer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Chipkarten, oft auch als Smartcard oder Integrated Circuit Card (ICC) bezeichnet, sind spezielle Plastikkarten mit eingebautem integriertem Schaltkreis (Chip), der eine Hardware-Logik, Speicher oder auch einen Mikroprozessor enthält [Wik10].

1.1 Umfeld der Arbeit

In den folgenden Abschnitten wird erklärt, was Smart Cards sind und wie sie sich über die Zeit entwickelten. Es wird auf den Aufbau und die einzelnen Komponenten einer Smart Card eingegangen, die Kommunikation beschrieben und auch der Lebenszyklus einer Smart Card erklärt.

1.1.1 Grundlagen über Smart Cards

In den frühen 70er Jahren wurde das Konzept von Smart Cards eingeführt. Die ersten Anwendungsgebiete waren der Ersatz von Geld in Zahlungsanwendungen wie Telefonzellen oder Automatengeschäften. Das reduzierte die Kosten für die Geldmanipulation und auch das Risiko von Diebstahl. Den ersten großen Einsatz fanden Smart Cards in den französischen öffentlichen Telefonzellen. Von der einfachen Speicherkarte entwickelte sich die Plastikkarte in eine komplexe Mikrocontroller Karte mit der Möglichkeit, Daten zu verarbeiten und diese zu lesen und zu speichern. Mit der Einführung von Computer und Netzwerktechnologien wurde auch die Smart Card als Sicherheitstoken zur Authentifizierung eingeführt und ist ein nützliches Werkzeug in vielen Bereichen. Auch die Angebote von TV Sendern mit kostenpflichtigen Programmen gaben einen Aufschwung in Smart Card Bereich da diese zur Überprüfung der Berechtigung der Benutzer eingesetzt werden. Einen weiterer großer Markt für Smart Cards ist der Subscriber Identity Module (SIM) im Bereich für Mobiltelefone [Shi08].

Durch die verschiedenen Anforderungen der Anwendungsgebiete kam es zu zwei verschiedenen Arten, um mit einer Smart Card zu interagieren. Zum einen kann die Kommunikation und Versorgung der Karte kontaktbehaftet geschehen indem die Karte in ein Kartenlesegerät gesteckt wird und dadurch die Kontakte der Karte mechanisch mit denen des Kartenlesegeräts verbunden sind. Die andere Möglichkeit ist eine kontaktlose Kommunikation und Versorgung der Karte. Es gibt auch die Möglichkeit beide Methoden miteinander

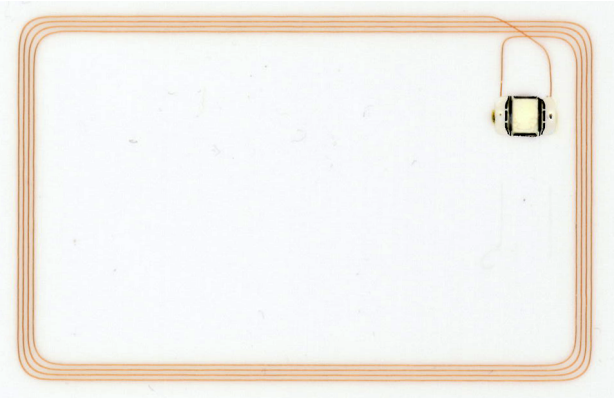

Abbildung 1.1: Aufbau einer kontaktlosen Smart Card.

zu kombinieren, sodass die Karte sowohl kontaktlos als auch kontaktbehaftet verwendet werden kann. In dieser Arbeit wird nur auf kontaktlose Smart Cards eingegangen.

1.1.2 Aufbau von Smart Cards

Eine typische kontaktlose Smart Card besteht aus einer Plastikkarte in die eine Antenne und ein Mikroprozessor eingearbeitet sind. In Abbildung 1.1 ist der Aufbau eine typische kontaktlosen Smart Card dargestellt. Am äußeren Rand kann man die Windungen der Antenne erkennen. Meist sind 4-5 Windungen aufgebracht. Auf der rechten Seite ist der eingegossene Mikrokontroller der Karte zu erkennen. Die Antenne dient zur Kommunikation und um den Mikrokontroller mit der nötigen Energie zu versorgen.

In Abbildung 1.2 ist das Blockschaltbild eines Smart Card Prozessors mit kontaktloser und kontaktbehafteter Schnittstelle dargestellt. Im Zentrum ist die Central Processing Unit (CPU). Diese ist meist ein Derivat der 8051 [www11] oder ARM [Ltd12] Plattform. Die CPU ist mit den einzelnen Hardwarekomponenten verbunden. Zu diesen Hardwarekomponenten zählen unter anderem die Kommunikationsschnittstellen, die Speicher, die Memory Management Unit (MMU) und die Koprozessoren für die Kryptographie. Diese Koprozessoren können wiederum eine eigene Schnittstelle zum Speicher haben. Im Folgenden wird kurz auf die wichtigsten Komponenten eingegangen.

CPU: Als CPU werden häufig von der 8051 oder ARM Plattform abgeleitete CPUs eingesetzt. Diese sind auf Sicherheit und komplexes Power-Management optimiert. Einerseits müssen sie den sehr strengen Anforderungen einer unabhängigen Sicherheitszertifizierung nach Common Criteria (CC) [Cri10] genügen, und andererseits müssen sie auch bei den sehr niedrigen und schwankenden Betriebsbedingungen bei der kontaktlosen Kommunikation nach ISO/IEC 14443 operieren können. Sie sind mit zusätzlichen Komponenten (z.B. Kryptographie-Koprozessoren) ergänzt, und ihr Befehlssatz kann mit umfangreichen Spezialbefehlen über den klassischen 8051-Befehlssatz hinaus erweitert sein. Abhängig von der gewählten Halbleitertechnologie werden diese bei Taktfrequenzen von mehreren zehn MHz betrieben. Die Taktfrequenz kann teilweise dem zur Verfügung stehenden Angebot an Versorgungsspannung angepasst werden.

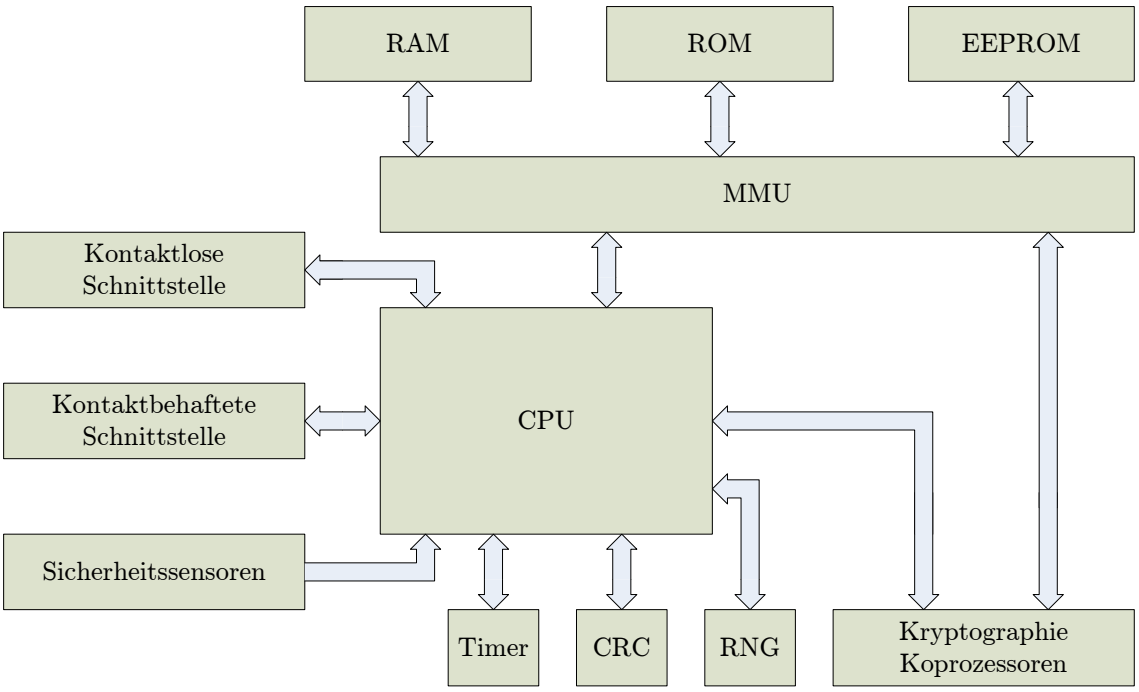

Abbildung 1.2: Blockdiagramm eines Smart Card Prozessors.

Kommunikation: Bei Smart Cards gibt es kontaktlose und kontaktbehaftete Kommunikation. Für die kontaktbehaftete Kommunikation gibt es mehrere verschiedene Protokolle unter anderem auch Universal Serial Bus (USB). Kontaktlose Kommunikation erfolgt bei den in dieser Arbeit betrachteten Smart Cards mit 13.56 MHz mit dem in der ISO/IEC 14443 definiertem Protokoll. Bei diesem Protokoll sind Datenraten bis 848 Kilobit (kbit) pro Sekunde in beide Richtungen möglich.

Kryptographie: Um eine annehmbare Übertragungsgeschwindigkeit bei der gesicherten Kommunikation zu erreichen sind die meisten Prozessoren mit Koprozessoren für Kryptographie ausgestattet. Meist sind es Advanced Encryption Standard (AES), Data Encryption Standard (DES) und eine Recheneinheit um die asymmetrische Kryptographie zu beschleunigen. Die Rechenleistung dieser Koprozessoren hängt von ihrer Registerbreite und Taktrate ab. Die Registerbreite kann von derjenigen der CPU deutlich abweichen (z.B. 32-Bit Register bei einer 8-Bit CPU). Die Taktrate der Kryptographie-Koprozessoren liegt in derselben Größenordnung wie diejenige der CPU, also im Bereich einiger zehn MHz.

RAM: Der Arbeitsspeicher kann zwischen den einzelnen Prozessoren stark variieren. Aktuelle Prozessoren greifen auf einen Arbeitsspeicher zwischen 2 - 8 Kilobyte (kB) zu.

ROM: Für den Programmspeicher des Prozessors wird meist ein Read Only Memory (ROM) verwendet, da eine Speicherzelle sehr flächeneffizient integriert werden kann und daher die Produktion billig ist. Ein weiterer Vorteil ist die geringe Zugriffszeit und dass daher auch die Ausführungszeit von Programmen aus dem ROM schnell ist. Die Größe

dieses Speichers kann zwischen einigen Kilobyte bis einigen hundert Kilobyte variieren.

EEPROM/Flash: Als nichtflüchtiger änderbarer Speicher kommt am häufigsten ein Electrically Erasable Programmable Read-Only Memory (EEPROM) vor, aber auch Flashspeicher sind heute schon in Verwendung. Die Speichergrößen variieren auch hier zwischen nur einigen hundert Bit bis zu einigen Hundert Kilobyte (und größeren Speichern bei dedizierten SIM-Prozessoren).

Mittlerweile wird auch teilweise für den Programmspeicher statt ROM ein Flashspeicher eingesetzt. Bei ROM als Programmspeicher werden meist mehrere verschiedene Anwendungen installiert, um nicht für jede Anwendung ein eigenes Produkt produzieren zu müssen. Flash hat den Vorteil, dass nach der Produktion nur die benötigte Anwendung geladen werden muss und somit der Speicherbedarf für den Programmspeicher geringer ist. Im Gegenzug dazu ist aber eine Flash-Speicherzelle um vieles größer als eine ROM-Speicherzelle. Daher sind auch die Kosten bei der Produktion höher. Ein weiterer Vorteil von Flash als Programmspeicher ist, dass bei neuen Programmversionen nicht die Produktion geändert werden muss.

Nichtflüchtige Speicher weisen aber auch Nachteile auf. Sie haben einen relativ hohen Energie- und Zeitbedarf beim Programmieren. Außerdem ist der Datenerhalt nicht so lange gewährleistet wie in ROM Speichern und sie sind auch nicht beliebig oft programmierbar.

Random Number Generator: Um eine sinnvolle kryptographische Verschlüsselung zu ermöglichen sind Zufallszahlen nötig. Diese Zufallszahlen werden mit einem Random Number Generator (RNG), also einem Zufallszahlengenerator erzeugt. Diese sind meist als Erweiterung zur CPU implementiert. Man unterscheidet hier zwischen Pseudozufallszahlengeneratoren und echten Zufallszahlengeneratoren.

CRC: Um zum Beispiel die Integrität einer Datenübertragung zu überprüfen, können die Daten mit einem Cyclic Redundancy Check (CRC) geschützt sein. Da die Berechnung eines CRCs eine komplexe mathematische Aufgabe ist, werden solche Berechnungen meist in ein eigenes Modul ausgelagert, welches diese Operationen schnell durchführen kann.

Timer: Manche Operationen von Smart Cards erfordern genaue Zeiten. Um dies zu erreichen haben die CPUs meist eine Erweiterung mit der sie auf exakte Zeiten zugreifen können.

Sicherheitssensoren: Zum Schutz eines Smart Card Prozessors gegen die verschiedensten Angriffe enthält der Chip verschiedene Sicherheitssensoren die Alarm schlagen falls eine Attacke detektiert wird.

1.1.3 Kommunikation

Die Kommunikations- und Spannungsversorgungschnittstelle einer kontaktlosen Karte ist in [ISO08a] definiert. Als Träger für die Daten wird ein 13.56 MHz Signal verwendet welches auch zur Übertragung der benötigten Energie verwendet wird. Das von der Spule des Lesegeräts erzeugte Magnetische Feld ist induktiv mit der Karte gekoppelt und überträgt dadurch die Energie (Abbildung 1.3). Es sind zwei Standards definiert dies sind ISO/IEC

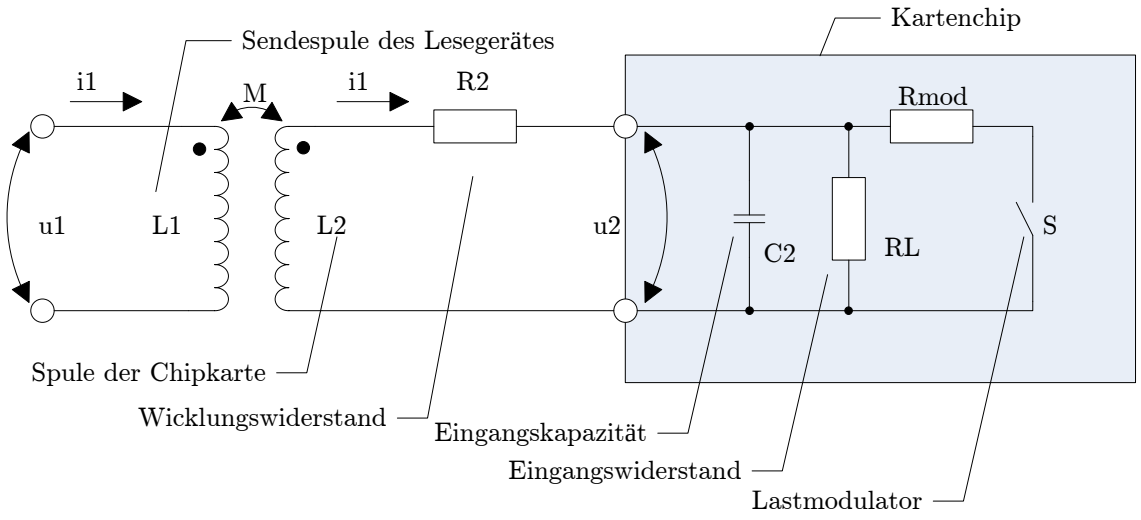

Abbildung 1.3: Energieversorgung und Datenübertragung einer Smart Card [Fin08].

14443-A und ISO/IEC 14443-B. Sie unterscheiden sich hauptsächlich durch die Codierung der Daten und den Modulationsindex.

Die Kommunikation erfolgt vom Lesegerät zur Karte über Modulation des Felds welches die Karte detektieren und auswerten kann (In Abbildung 1.4 ist eine beispielhafte Übertragung nach ISO/IEC 14443-A dargestellt). In umgekehrter Richtung wird das Signal durch eine zusätzliche Last, welche von der Karte beeinflusst wird, moduliert was wiederum das Lesegerät detektiert (Abbildung 1.5 stellt eine Antwort nach ISO/IEC 14443-A dar).

Für die Kommunikation sind 4 verschiedene Datenraten definiert die sich alle vom 13.56 MHz Trägersignal ableiten, dies sind 106kbit/sec (ein Bit benötigt 128 Sinusperioden des 13.56 MHz Signals), 212kbit/sec (64 Perioden), 424kbit/sec (32 Perioden) und als höchste Datenrate 848kbit/sec (16 Perioden).

Um auch kommunizieren zu können, wenn sich mehrere Karten im Feld des Readers befinden ist es nötig einen Mechanismus zu haben mit dem man einzelne Karten auswählen kann. Ein solcher Mechanismus ist in [ISO08b] definiert. Auch hier wird zwischen Type-A und Type-B Karten unterschieden.

Bei Type-A müssen die Karten bei ihren Antworten definierte Zeiten einhalten und die Codierung der Daten ist so gewählt, dass eine Kollision der Daten (also wenn eine Karte eine Null und die andere eine Eins schickt) detektiert werden kann. Es wird ein Anfrage an die Karten geschickt (Request-A (REQA))und nach einer definierten Zeit antworten alle Karten mit Daten über die Identifikationsnummer der Karte. Nach diesem Befehl wird ein weiterer Befehl zur Karte geschickt auf den alle Karten mit ihrer einmaligen Identifikationsnummer antworten. Sind nun mehrere Karten im Feld, ist sicher eine Stelle der Antwort verschieden. Daher kommt es bei der Antwort der Karten zwangsläufig zu einer Kollision. Diese Kollision wird vom Reader detektiert. Die Rückantworten der Karten bis zur ersten Position einer Kollision sind bei allen Karten gleich. Daher sind die Identifikationsnummern der Karten bis zur ersten Kollisionsposition bekannt. Der Reader wählt nun eine Karte bzw. auch mehrere Karten aus indem er der bereits bekannten Identifikations-

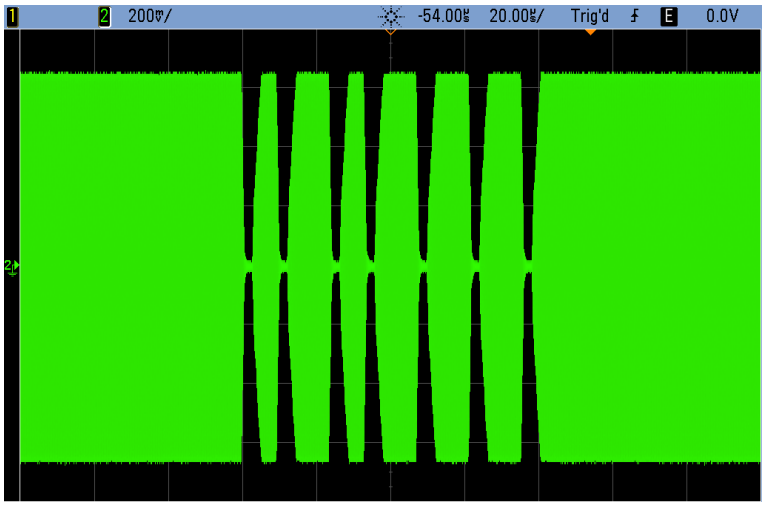

Abbildung 1.4: Beispiel für eine Datenübertragung zwischen Reader und Smart Card.

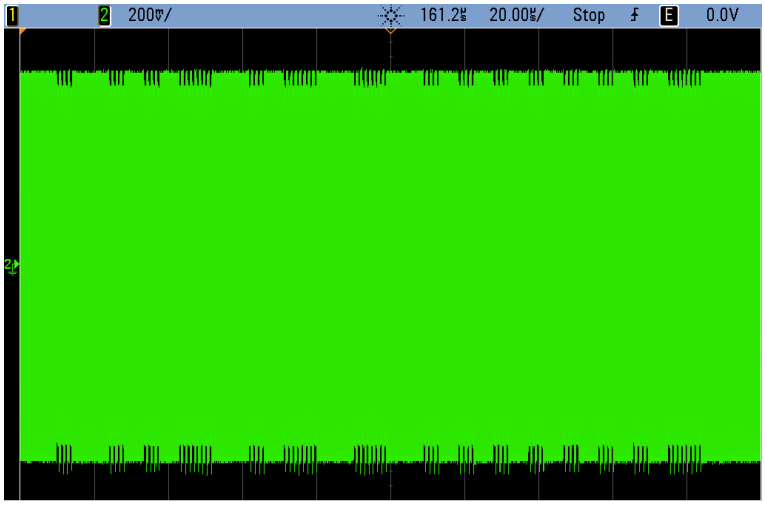

Abbildung 1.5: Beispiel für die Rückmodulation einer Smart Card.

nummernteil eine Null oder Eins hinzufügt. Die nun gewählte ID wird wiederum zu den Karten gesendet. Alle Karten bei denen die ID übereinstimmt antworten wieder mit ihrer restlichen ID. Dies wird solange wiederholt bis nur mehr eine Karte selektiert ist und es daher zu keiner Kollision mehr kommt.

Eine Ausnahme stellen Karten dar die eine zufällige Identifikationsnummer generieren. Durch diese zufällige Nummer sind die Karten nicht verfolgbar da bei jedem Kontakt eine neue Identifikationsnummer generiert wird. Diese zufällige Identifikationsnummer ist vier Byte lang. Sind nun zwei solche Karten in Reichweite des Readers, kann es zu dem sehr unwahrscheinlichen Fall kommen das die Karten nicht eindeutig selektiert werden können, falls beide Karten die gleiche Identifikationsnummer gewählt haben.

Type-B Karten verwendet für diese Aktivierung ein anderes Verfahren. Wird eine Type-B Karte in das Feld gebracht, wartet sie auch auf ein Kommando vom Reader (Request-B (REQB)). Beim REQB werden als Parameter mitgegeben welche Anwendungsfamilie (Application Family Identifier (AFI)) man haben will (z.B. Transport) und wie viele Zeitslots für die Antwort zur Verfügung stehen. Dieser Wert kann zwischen eins und 16 eingestellt werden. Die Karten die nun zur richtigen Anwendungsfamilie gehören wählen zufällig einen Zeitslot aus den verfügbaren aus. Der Reader schickt nun sequentiell für jeden Zeitslot einen kurzen Befehl und alle Karten die diesen Zeitslot gewählt haben antworten darauf (Answer To Request-B (ATQB)). Diese Antwort enthält wichtige Parameter der Karte wie die unterstützten Übertragungsraten, die maximale Größe der Daten, die übertragen werden dürfen, Informationen über mehrere auf der Karte verfügbare Anwendungen und auch eine 4 Byte Identifikationsnummer. Wird nun eine Antwort richtig empfangen kann diese Karte über die Identifikationsnummer selektiert werden [Fin08].

Das geschieht alles mit der niedrigsten Datenrate. Ist nun eine einzelne Karte ausgewählt kann ein Befehl zur Datenratenänderung geschickt werden und mit der Karte laut in [ISO08c] definierten Protokoll kommuniziert werden. Dieses schreibt vor, wie der Beginn der Übertragung ausschauen muss, wie darin die Bytes codiert sind und dass die Übertragung mit einem CRC geschützt sein muss.

Aufbauend auf diesem Protokoll ist in [ISO04] definiert, welche verschiedenen Befehle es gibt, die zur Karte geschickt werden können, was deren Parameter sind und wie diese codiert werden müssen. Ein Kommando das zwischen dem Reader und der Karte ausgetauscht wird, nennt sich Application Protocol Data Unit (APDU). Alle APDUs sind gleich aufgebaut (Abbildung 1.6). Am Anfang steht das Klassen Byte (CLA) mit dem zwischen verschiedenen Klassen der Kommunikation unterschieden wird, zum Beispiel ob die Verbindung verschlüsselt ist oder nicht. Das zweite Byte enthält eine Identifikation für den gewählten Befehlt (INS). P1 und P2 sind etwaige Parameter, danach kommt ein Feld welches die Länge der nachfolgenden Daten angibt (Lc) (falls keine Daten folgen ist auch dieses Feld nicht vorhanden). Als nächstes folgen optional die Daten. Als letztes Byte steht die maximale Länge der Antwort (Le). Als Antwort können Daten und ein auch in der Norm definierter Statuscode (zusammengesetzt aus SW1 und SW2) von der Karte zurück zum Reader gesendet werden.

1.1.4 Java Card

Im Jahr 1996 wurde von der Firma Schlumberger eine Chipkarte entwickelte welche in der Lage war Programme die in Java programmiert waren auszuführen. 1997 kam es dann

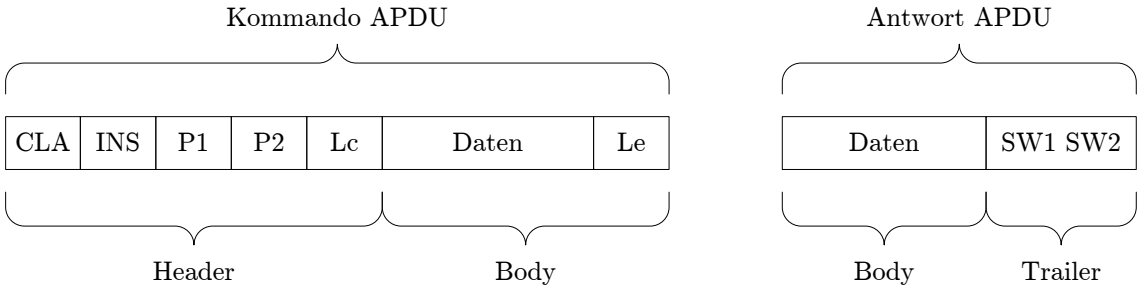

Abbildung 1.6: Aufbau eines in ISO/IEC 7816 definierten Kommandos und der Antwort.

zum Treffen vieler großen Chiphersteller und das Java Card Forum (JCF) wurde gegründet [RE08]. Das JCF ist das international tätige Standardisierungsgremium für Java auf Chipkarten.

Java für Chipkarten bietet nicht den ganzen Funktionsumfang von Java. So gibt es zum Beispiel Einschränkungen bei den Dateitypen, denn es dürfen nur einfache Typen wie zum Beispiel Byte oder Short verwendet werden. Typen wie String oder List sind nicht verfügbar. Weiters gibt es Einschränkungen beim dynamischen Speichermanagement. Im Gegenzug sind spezielle Anwendungsschnittstellen für Smart Cards vorhanden.

Der große Vorteil von Java Card gegenüber der nativen Implementierung auf einer Karte ist nicht nur, dass die Programmierung einfach über eine Hochsprache erfolgt, sondern dass durch das standardisierte Interface eine Unabhängigkeit von Kartenherstellern erzeugt wird. Das bedeutet ein Programm das für eine Java Card geschrieben ist, sollte auch auf Karten einen anderen Herstellers problemlos lauffähig sein.

Die Java Card Spezifikation [Mic06] definiert das eine Chipkarte mit Java eine Java Virtuelle Maschine (VM) besitzt welche bei der Kartenfertigung aktiviert und am Ende das Lebenszyklus der Karte deaktiviert wird.

Das Programm und die zugehörigen Daten sind ein sogenanntes Applet welches auf die Karte geladen wird. Beim Laden werden die Applets in den nichtflüchtigen Speicher der Karte abgelegt. Über eine eindeutige Anwendungs-ID kann das Applet selektiert werden und danach werden automatisch alle Kommandos an das Applet weitergereicht. Das Applet kann nun die Kommandos auswerten und die entsprechenden Operationen ausführen. Dadurch erreicht man die maximale Flexibilität da bei verschiedenen Applets die Abarbeitung der Kommandos verschieden definiert werden kann. Die einzigen Befehle die unabhängig implementiert sind, sind die Befehle zur Verwaltung der Applets.

In Abbildung 1.7 ist der Aufbau einer Java Card in den verschiedenen Schichten dargestellt. Die unterste Schicht auf die alles aufbaut, ist die Hardware. Auf diese setzt eine Softwareschicht auf, die den Zugriff auf die Hardware ermöglicht. Auf diese Abstraktionsschicht bauen die Java VM und auch die verschiedenen Schnittstellen, die von den Applets verwendet werden können, auf. Diese Schnittstellen verwenden auch die VM. Über die verschiedenen Interfaces können dann die Applets auf der Karte ausgeführt werden.

Um eine Software für eine Java Karte zu entwickeln muss das Java Programm (mit dem eingeschränkten Java Umfang) wie für ein normales Java Programm in unabhängigen Bytecode kompiliert werden. Dieser Bytecode wird danach vom Offline Teil der Java VM überprüft um sicherzustellen, dass er auch auf der Karte korrekt lauffähig ist. Aus

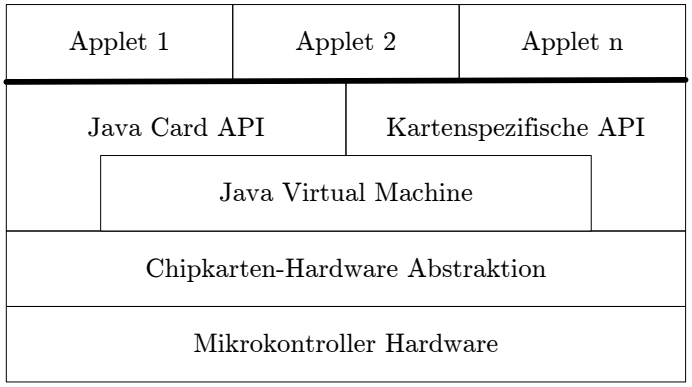

Abbildung 1.7: Abstrakter Aufbau einer Java Card.

dem Bytecode wird eine Card Application (CAP) Datei erzeugt und diese wird auf die Chipkarte geladen. Das Applet wird an die Java VM übergeben welche den Code Zeile für Zeile prüft und aus den Bytecodes Maschinenbefehle für die Hardware erzeugt.

1.1.5 Lebenszyklus einer Smart Card

Der Lebenszyklus einer Smart Card kann vereinfacht in 4 Phasen geteilt werden. In [ISO00] wird auf den Lebenszyklus der Karte und der einzelnen Dateien die sich auf der Karte befinden eingegangen.

In Abbildung 1.8 ist der Lebenszyklus einer Smart Card dargestellt. Die Einzelnen Zustände können im einfachsten Fall nur von der Erstellung der Karte zur Personalisierung in den betriebsfähigen Zustand übergehen. Von diesem Zustand aus kann die Karte dann terminiert werden.

1. Erstellung

   Die Fertigung der Karte erfolgt beim Kartenhersteller. Es werden die Wafer produziert und getestet. Aus diesen Wafern werden die Mikrocontroller in die Karte eingebracht, Personalisierungsschlüssel aufgespielt, verschiedene Parameter des Mikrokontrollers konfiguriert und die Karte getestet. Die einzelnen Fertigungsschritte können auch von verschiedenen Herstellern ausgeführt werden. Zum Beispiel kann der Smart Card Chip in einer Fabrik gefertigt werden und die Antenne von einem anderen Hersteller bezogen werden. Nach den abschließenden Tests wird die Karte an den Herausgeber ausgeliefert.

2. Personalisierung

   In dieser Phase werden vom Herausgeber der Karte (das ist zum Beispiel eine Behörde) die personenbezogenen Daten aufgespielt. Dies kann zum Beispiel die Persönliche Identifikationsnummer (PIN) bei einer Bankanwendung oder das Passbild bei einem Reisepass sein. Auch personenbezogene Schlüsseldaten werden entweder direkt auf der Karte generiert oder aufgespielt. Nachdem alle Daten aufgespielt sind, wird die Karte in den eigentlichen Betriebszustand versetzt. In diesem Zustand können dann zum Beispiel beim Reisepass die Daten nicht mehr geändert werden.

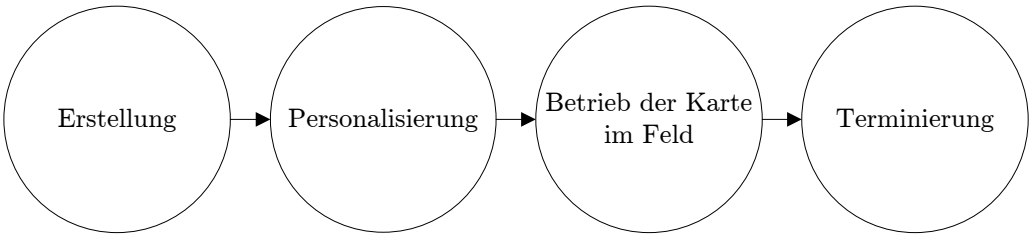

Abbildung 1.8: Lebenszyklus einer Smart Card.

3. Nutzung der Karte

   Nachdem alle Daten aufgespielt sind und die Karte gegen Manipulationen gesichert ist, kann sie an den Kunden ausgeliefert werden. In dieser Phase erfolgt die eigentliche Nutzung der Karte. Diese Phase kann mehrere Jahre dauern und daher ist es notwendig, dass die Karte genügend Schutzmechanismen beinhaltet um über den langen Zeitraum gegen Angriffe geschützt zu bleiben.

4. Terminierungsphase

   Ist die Lebensdauer der Karte abgelaufen wird sie terminiert und kann nicht mehr benutzt werden. Die Karte kann sich je nach Implementierung auch terminieren wenn Attacken auf die Karte detektiert werden. In diesem Fall ist es ein Schutzmechanismus gegen Attacken.

1.2 Motivation

Die ganze Entwicklung begann mit der Einführung einer einfachen Smart Card als Telefonwertkarte. Durch die Weiterentwicklung der Technologie konnten sich Smart Cards in immer mehr Bereichen etablieren. Mittlerweile gibt es eine große Menge an unterschiedlichsten Anwendungen in den verschiedensten Bereichen. Der Bereich der wohl am meisten mit Smart Cards verbunden wird, sind die Bankanwendungen. Allein die schier endlose Anzahl an Kreditkarten und Bankomatkarten geben ein großes Anwendungsfeld ab. Aber auch bei Anwendungen für das offline Bezahlen wie beim österreichischen System Quick [Gmb12] kommen Smart Cards zum Einsatz. Auch in anderen Bereichen hat sich die Smart Card etabliert. Als SIM Karte in jedem Handy, als Ticket bei öffentlichen Transportmittel oder auch als Eintrittskarte für Konzerte sind Smart Cards in Verwendung.

Ein weiterer großer Einsatzbereich von Smart Cards ist die elektronische Identität. Mögliche Einsatzszenarien dafür sind unter anderem [Cor08]:

- Krankenakte und Rezept
- Reisepass
- Nationale ID
- Fahrzeug Anmeldung
- Führerschein

- Visa
- Bürger ID

Die elektronische Identität bringt auch viele Vorteile für die Einwohner als auch für die Verwaltung mit sich [Cor08]:

- Die Smart Chip Technologie in einer e-ID Karte agiert als sicherer Speicher um Missbrauch von verlorenen oder gestohlenen e-ID zu verhindern und die persönlichen Informationen des Besitzers zu schützen.
- e-ID verbessert die Privatsphäre indem sie nur Zugriff auf speziell autorisierte Information erlaubt.
- Abläufe werden vereinfacht, zum Beispiel kann Verifikation und Identitäts-Authentifizierung offline gemacht werden.
- e-ID Karten sind extrem schwer zu fälschen und bieten daher eine starke Gegenmaßnahme gegen Identitätsdiebstahl.
- e-ID in Kombination mit Digitaler Signatur vereinfacht und beschleunigt den angebotenen Service.

Allein an diesen Beispielen ist ersichtlich, dass Smart Cards vielseitig einsetzbar sind. Um nun ein neues Produkt in einem der Bereiche zu platzieren ist es nötig auch die richtige Hardware Plattform zu wählen. Auch bei der Hardware gibt es ein großes Angebot von verschiedensten Herstellen und die Entscheidung für das richtige Produkt ist schwer zu fällen.

Auch beim Design einer neuen Plattform für Smart Cards sollen die Anforderungen der Anwendungen, für die die neue Plattform entworfen wird, möglichst gut erfüllt werden. Dazu ist es notwendig die Anforderungen der Anwendungen an die Hardware zu kennen. Bei der großen Menge an verschiedenen Anwendungen ist das eine schwierige Aufgabe.

Eine Möglichkeit die vielen verschiedenen Anwendungen zu minimieren ist, sie zu verschiedenen Anwendungsdomains zusammenzufassen. Es stellt sich dabei die Frage ob alle Anwendungen in einer Domain die annähert gleichen Anforderungen an die Hardware stellen. Wenn das der Fall ist, dann kann anhand der Domain der eine Anwendung zugeordnet werden kann, auch die Entscheidung für die richtige Hardwareplattform getroffen werden. Außerdem kann anhand der Domain, für die eine neue Plattform entwickelt wird, Rückschlüsse auf die Anforderungen, die die neue Hardwareplattform erfüllen muss, gezogen werden.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist zu zeigen, ob es möglich ist, anhand der Domain einer Anwendung auf die Anforderungen an die Hardware (durch die bekannten Anwendungen in der Domain und deren Anforderung) zu schließen.

Es ist eine Umgebung zur Analyse der Anforderungen von Anwendungen verschiedener Domains zu entwerfen. Mit Hilfe dieser Testumgebung ist zu verifizieren, ob neue Anwendungen die gleichen Anforderungen haben, wie die Anwendungen die bereits in der zugehörigen Domain analysiert wurden.

1.4 Gliederung

Die verschiedenen Anwendungsdomains und Anwendungen die es für Smart Cards gibt werden in **Kapitel 2** behandelt. Es werden die unterschiedlichen Anwendungen kurz vorgestellt.

Kapitel 3 beschäftigt sich mit den unterschiedlichen Protokollen, die es bei Smart Cards gibt. Es werden die unterschiedlichen Protokolle für die verschiedenen kryptographischen Algorithmen vorgestellt und auch auf die übliche Speicherstruktur bei Smart Cards eingegangen.

In **Kapitel 4** wird auf Arbeiten eingegangen die sich mit einem ähnlichen Thema wie diese Arbeit beschäftigen. Zum einen werden Arbeiten vorgestellt, die sich mit der Vorhersage von Leistungssteigerungen von Smart Cards beschäftigen. Weiters wird eine Arbeit vorgestellt in der Java Card Implementierungen mit nativen Implementierungen verglichen werden. Als letzte Punkte in Kapitel 4 wird Benchmarking von Java Cards und ein Framework zur Ermittlung der Performance von Java Cards vorgestellt.

Das Design der Arbeit wird in **Kapitel 5** behandelt. In diesem Kapitel wird eine Übersicht der Arbeit vermittelt und die einzelnen Designentscheidungen beschrieben. **Kapitel 6** beschreibt die Implementierung der gesamten Testumgebung. Zuerst wird auf das ganze Testsystem eingegangen und in weiterer Folge auch auf den eigentlichen Kern der Simulation und die Werkzeuge zur Auswertung der Ergebnisse.

In **Kapitel 7** werden die Resultate der Arbeit präsentiert und diese interpretiert und anschaulich dargelegt. **Kapitel 8** geht auf die weitere Verwendung der Arbeit und die Möglichkeiten die diese Arbeit noch hat ein.

Kapitel 2

Anwendungsdomains

Es gibt für Smart Cards die verschiedensten Einsatzbereiche. Angefangen von einer einfachen Speicherkarte bis hin zu komplexen Anwendungen mit Einsatz von Kryptographie. Für jeden Bereich gibt es verschiedene Anwendungen. In diesem Kapitel wird auf verschiedene Anwendungen für Smart Cards für die verschiedenen Bereiche eingegangen. Es wird versucht die Bereiche nach den zusammengehörigen Anwendungen zu teilen. In der weiteren Arbeit wird näher auf die Bereiche Identifikations- und Bankanwendungen eingegangen

2.1 Identifikationsanwendungen

Als erste Anwendungsdomain werden Identifikationsanwendungen vorgestellt. Diese Anwendungen dienen dazu die Identität des Inhabers festzustellen. Beispiel für so eine Anwendung ist ein Reisepass, ein Personalausweis oder ein Führerschein. Bei diesen Anwendungen kann man anhand eines Zugriffsschlüssels personenbezogene Daten aus dem gesicherten Datenspeicher auslesen und anhand dieser Daten wird die Identität des Inhabers verifiziert. Diese Daten können zum Beispiel ein Bild des Inhabers oder die Fingerabdrücke sein.

In dieser Arbeit wird auf zwei Anwendungen in diesem Bereich eingegangen. Die erste ist ein Reisepass auf dem ein Passbild und die Daten des Inhabers gespeichert sind. Um auf die Daten des Passes zugreifen zu können muss man sich zuerst mit der Chipkarte authentifizieren. Die folgende Datenübertragung erfolgt dann verschlüsselt. Für diese Operationen muss die Chipkarte symmetrische Kryptographie unterstützen und die große Menge an Daten auch in angemessener Zeit über die kontaktlose Schnittstelle übertragen können.

Bei der zweiten Anwendung handelt es sich ebenfalls um einen Reisepass aber auf diesen sind auch die Fingerabdrücke des Inhabers gespeichert. Um auf diese Daten zuzugreifen, muss sich das Lesegerät speziell beim Pass authentifizieren. Dafür benötigt das Lesegerät Zertifikate mit denen die Chipkarte am Pass feststellen kann ob das Lesegerät überhaupt berechtigt ist, auf die sensiblen Daten zuzugreifen. Für die Überprüfung der Zertifikate wird asymmetrische Kryptographie benötigt, die die Chipkarte rechnen muss. Zusätzlich dazu muss auch auf die gleichen Daten wie beim ersten Reisepass zugegriffen werden und die gleichen Sicherheitsmechanismen verwendet werden.

Beide Anwendungen müssen vor der Ausgabe des Dokuments personalisiert werden, wobei die Daten des Inhabers im nichtflüchtigen Speicher gespeichert werden müssen. Anhand dieser Anwendungsbeschreibung lassen sich Anforderungen an die Hardware der Chipkarte ableiten. Diese sind im Folgenden beschrieben:

- Beschreibbarer nichtflüchtiger Speicher um die Daten des Inhabers und das Passbild sowie etwaige Fingerabdrücke zu speichern.
- Schnelle kontaktlose Kommunikation um die Daten zu übertragen.
- Unterstützung von symmetrischer Kryptographie zur Sicherung der Datenübertragung.
- Hardwarebeschleunigung oder Koprozessor für asymmetrische Kryptographie die für das Verifizieren der Zertifikate benötigt wird.

2.2 Bank Anwendungen

Im Bereich des Bezahlens mit einer Chipkarte gibt es auch verschiedene Anwendungen. Zum einem gibt es Karten auf welche ein Guthaben geladen werden kann und mit dem Guthaben bezahlt werden kann. Ein Beispiel dafür ist die Österreichische Quick System [Gmb12]. Diese Systeme haben meist nur nationale Gültigkeit.

Zum anderen sind Chipkarten im internationalen Zahlungsverkehr im Einsatz um Zahlungen abzusichern. Der Ablauf einer solchen Zahlung mit einer Chipkarte ist zum Beispiel in [LLC11] beschrieben. Bei diesem Verfahren wird zuerst von der Karte ausgelesen welche Protokolle unterstützt werden. Danach wird verifiziert, ob die Karte nicht gefälscht ist oder Daten manipuliert wurden. Anhand eines PIN Codes kann sich nun der Inhaber authentifizieren. Das Terminal und die Karte handeln nun anhand der Transaktionsdaten (wie zum Beispiel dem Betrag der Zahlung) aus, in welcher Art die Zahlung durchgeführt wird. Also ob das Terminal sich mit dem System des Kartenherausgebers verbinden muss um die Transaktion zu überprüfen oder ob die Transaktion auch offline erfolgen kann. Generell setzt sich eine Bankanwendung vereinfacht aus folgenden Schritten zusammen:

- Authentifizierung des Inhabers (zum Beispiel anhand eines PIN Codes).
- Überprüfung ob die Transaktion gültig ist (genug Guthaben vorhanden, Karte nicht gesperrt, ...).
- Speicherung der Transaktionsdaten oder des neuen Guthabens.

Auch bei dieser Anwendungsdomain können anhand der groben Beschreibung der Funktionalitäten Anforderungen an die Hardware abgeleitet werden.

- Beschreibbarer nichtflüchtiger Speicher um die Daten des Inhabers zu speichern und bei den Transaktionen Daten mitzuspeichern.
- Asymmetrische Kryptographie zur Erstellung einer Signatur über die Transaktionen.
- Kontaktlose Kommunikation um die Daten der Transaktion zwischen Terminal und Karte auszutauschen.

2.3 Weitere Anwendungen

Zu erwähnen sind noch Anwendungen für Tickets. Diese können zum Beispiel bei elektronischen Fahrscheinen, wie zum Beispiel bei der Oyster-Card die für einen Großteil des öffentlichen Verkehrs in London zum Einsatz kommt, in Verwendung sein.

Eines der größten Einsatzbereiche von Smart Cards ist der Einsatz als SIM Karte in Mobiltelefonen. Der Zugriff auf die Daten, die auf der Karte gespeichert werden können ist meist mit einem PIN geschützt. Auf diesen großen Anwendungsbereich wurde in dieser Arbeit aber nicht näher eingegangen, da es sich bei SIM Karten um kontaktbehaftete Karten handelt und in der Arbeit der Schwerpunkt auf kontaktlosen Karten liegt.

Um die privaten Schlüssel einer Public Key Infrastruktur (PKI) sicher zu speichern, sind Smart Cards ein guter Weg. Wenn die Signaturen auch auf der Karte berechnet werden, ist die Übertragung des privaten Schlüssels auf ein nichtsicheres Medium nicht nötig.

Ein weiteres Einsatzgebiet von Smart Cards ist das Speichern von Daten, wobei es je nach Anwendung die verschiedensten Zugriffskontrollen geben kann.

Auch bei Gesundheitskarten wie bei der österreichischen e-Card [uEm11] kommen Smart Cards zum Einsatz. Auf dieser Karte sind digital die Daten des Inhabers gespeichert, anhand welcher er eindeutig identifiziert werden kann. Anhand dieser Daten und eines Schlüssels den die Karte erzeugt, kann danach online zum Beispiel der Versicherungsstatus abgefragt werden.

Einer der Ersten Einsatzgebiete vom Smart Card waren auch der Ersatz von Geld bei öffentlichen Telefonzellen. Dabei kann über kryptographische Zugriffsprotokolle auf den Inhalt der Karte wie zum Beispiel das Guthaben zugegriffen werden.

Bei Pay-TV Anwendungen kommt eine Smart Card meist zur Überprüfung der Berechtigungen für den Empfang geschützter Programme zum Einsatz.

Bei manchen Karten können auch mehrere verschiedene Anwendungen installiert sein. Zum Beispiel kann auf der österreichischen e-Card auch die Bürgerkartenfunktionalität aktiviert werden [fsIA12].

Kapitel 3

Protokolle

Bei denn im vorigen Kapitel vorgestellten Anwendungen für Smart Cards gibt es natürlich auch die verschiedensten Protokolle die die Kommunikation und die Kryptographie definieren. Auf einige der Wichtigsten wird in diesem Kapitel näher eingegangen.

3.1 Smart Card Dateisystem

In [ISO04] wird ein Dateisystem für die Speicherung von Daten auf Smart Cards definiert. Da meist verschiedene Daten gespeichert werden müssen, ist es sinnvoll die Daten auch strukturiert abzulegen. Daher ist es üblich auch bei limitiertem Speicher auf ein Dateisystem zu setzen und auch Zugriffsmechanismen dazu zu definieren. Ähnlich wie auch bei einem Personal Computer baut sich das Dateisystem auf einer Smart Card hierarchisch auf. Quasi das Root-Verzeichnis ist auf einer Smart Card das Master File (MF). Ein Dedicated File (DF) ist vergleichbar mit einem Ordner und ein Elemantary File (EF) entspricht einer Datei. Unter dem MF können sich mehrere EFs und mehrere DFs befinden. Auch innerhalb eines DFs können sich wieder weitere EFs bzw. DFs befinden.

Diese EFs und DFs haben eine eindeutige Identifikationsnummer (bei EFs ist dies eine 2 Byte ID und bei DFs eine bis zu 16 Byte lange ID als symbolischer Name) mit der sie angesprochen werden können. Weiters können auch noch sogenannte Short EF Identifier vorhanden sein, mit denen die Dateien angesprochen werden können.

In Abbildung 3.1 ist ein beispielhafter Aufbau einer solchen Dateistruktur dargestellt. Es gibt ein MF unter dem sich einzelnen EFs befinden. Weiters sind DFs dargestellt die selbst wiederum EFs und DFs enthalten.

Da eine Karte auch verschiedene Anwendungen enthalten kann, werden DFs meist dazu eingesetzt um die Daten der einzelnen Anwendungen zu separieren.

Bei einer Smart Card gibt es verschiedene Speicherarten (siehe auch Abschnitt 1.1.2). Für die Speicherung von Daten kommt nur ein nichtflüchtiger Speicher in Frage, da die Karte ja nicht immer mit Spannung versorgt wird. Da der Inhalt des ROMs bei der Produktion nur einheitlich für eine ganze Charge gesetzt werden kann, müssen die personenbezogenen Daten im EEPROM oder in einem anderen nichtflüchtigen programmierbaren Speicher liegen. Eventuell können Daten die für die ganze Lebensdauer der Karte fix sind bei der Produktion schon in der ROM Maske berücksichtigt werden.

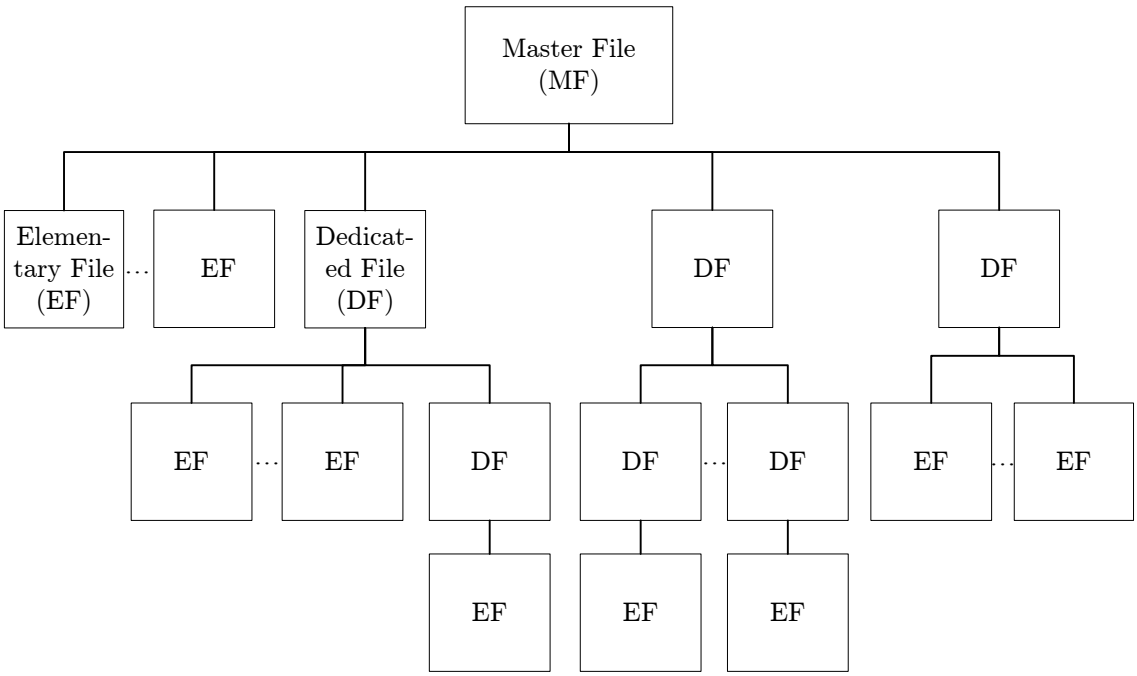

Abbildung 3.1: Beispielhafter Aufbau einer Speicherstruktur auf einer Smart Card.

Um auf die Dateien zuzugreifen sind auch etliche Grundfunktionen definiert, dazu gehören unter anderem:

- Select File:

  Mit dieser Funktion kann ein EF oder eine DF anhand der File ID (2 Byte) oder anhand des Namens ausgewählt werden.

- Read Binary:

  Erlaubt das Lesen von Daten aus dem aktuell selektierten EF, oder auch das Lesen eines EFs das mithilfe der Short EF ID beim Lesebefehl mit angegeben ist.

- Write Binary:

  Um Daten in das aktuell ausgewählte EF zu schreiben oder anhand der Short EF ID zu schreiben.

- Update Binary:

  Um Daten im ausgewählten EF zu ändern (auch mit Short EF ID möglich).

Bei den Befehlen wird meist der Offset innerhalb der Datei und die Länge der Daten auf die zugegriffen wird mit angegeben.

Da es nicht erwünscht ist das immer alle Daten auch geändert werden können, gibt es die Möglichkeit Zugriffsbedingungen zu definieren damit die EFs auch geschützt werden können. Diese Zugriffsbeschränkungen können auch vom aktuellen Life Cycle (Siehe 1.1.5) abhängen. So ist meist nur im Personalisierungszustand das Anlegen von Dateien erlaubt. Je nach Anwendungsfall kann es zum Beispiel auch sein, dass im Operation State nur das Selektieren und Auslesen von EFs erlaubt ist.

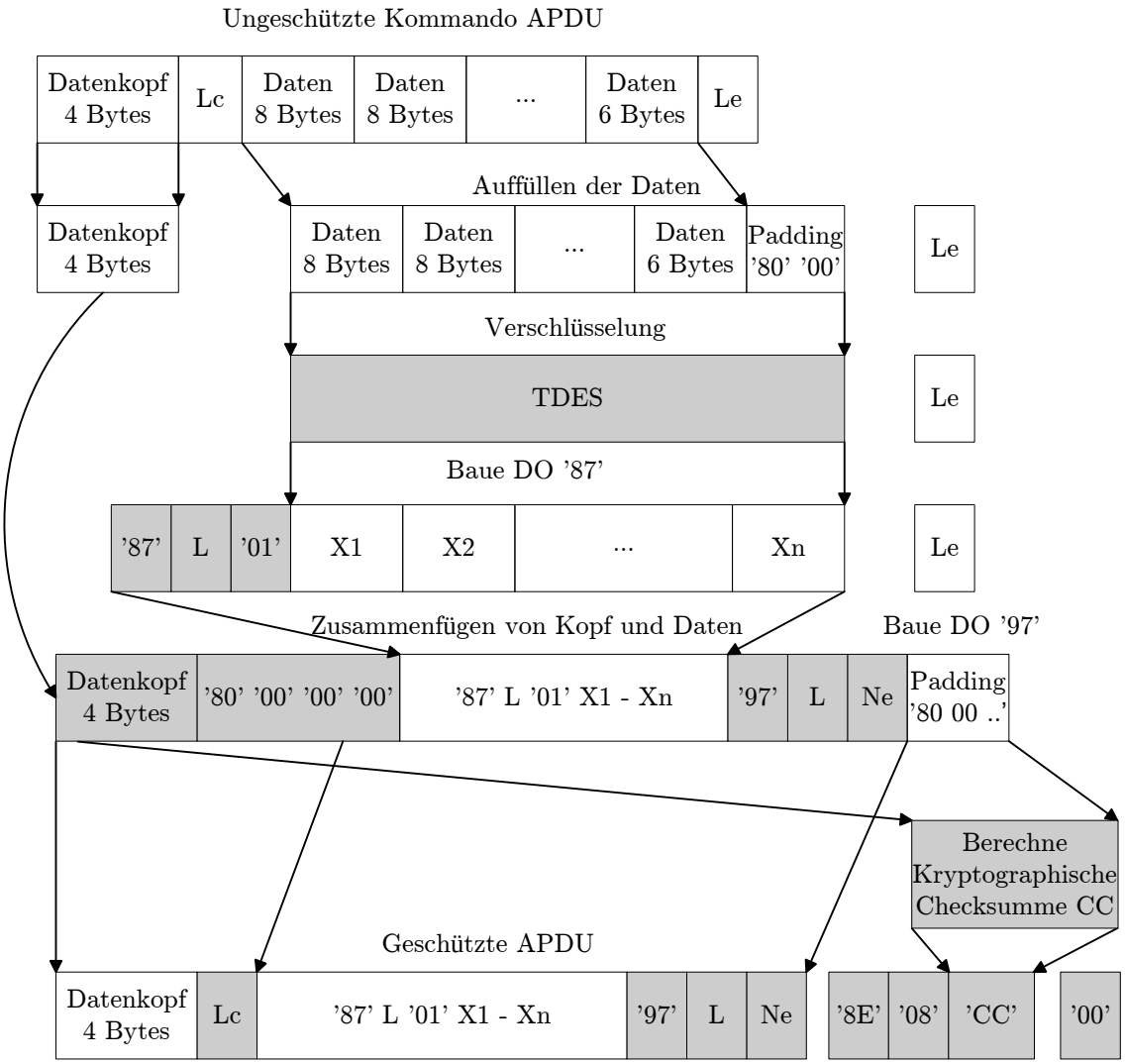

Abbildung 3.2: Berechnung einer Secure Messaging Kommando-APDU [Int06].

3.2 Secure Messaging

Unter Secure Messaging (SM) versteht man im Zusammenhang mit Smart Cards die verschlüsselte und gesicherte Übertragung von Daten zwischen dem Lesegerät und der Karte über das Radio Frequency (RF)-Interface. Diese gesicherte Übertragung ist bei sensiblen Daten nötig, da ein RF-Interface leicht abgehört werden kann.

In [ISO04] ist das SM-Protokoll für Smart Cards definiert. Durch SM werden alle Teile eines Kommando- und Antwort Paares geschützt. Es werden zwei Basisfunktionalitäten erfüllt und zwar die Geheimhaltung der Daten und die Authentifizierung der Daten.

In Abbildung 3.2 ist ein beispielhafter Ablauf für eine Kommando APDU dargestellt. Die Daten werden in diesem Beispiel zuerst auf die erforderliche Länge für die Verschlüsselung gebracht und danach mit Triple DES (TDES) verschlüsselt. Diese Verschlüsselten Daten werden danach in das in [ISO04] definierten Format gebracht und zusammen mit

Abbildung 3.3: Maschinell lesbaren Zone auf einem Reisepass [Wik11].

dem Datenkopf und dem Byte für die erwartete Länge wird eine kryptographische Checksumme über diese Daten berechnet. Als neues durch SM geschütztes Kommando werden nun die verschlüsselten Daten und die Checksumme geschickt. Damit ist sichergestellt, dass die Daten nicht durch mithören der Kommunikation ausgespäht werden können und auch eine Manipulation der Daten kann festgestellt werden. Für die Verschlüsselung und die Berechnung der Checksumme werden zwei verschiedene Schlüssel verwendet. Wie diese Schlüssel (KS_{ENC} und KS_{MAC}) erzeugt werden ist in Abschnitt 3.3.1 beschrieben.

Bei der Antwort von der Karte erfolgt die Verschlüsselung ähnlich.

3.3 Protokolle für maschinell lesbare Reisedokumente

3.3.1 Basic Access Control

Basic Access Control (BAC) ist ein Protokoll für maschinell lesbare Reisedokumente (zum Beispiel ein Reisepass). Die Daten die auf der Karte gespeichert sind können nur ausgelesen werden wenn vorher mit dem richtigen Schlüssel authentifiziert wurde. Dieser Schlüssel leitet sich aus dem Aufdruck des Reisedokuments ab. Dieser Aufdruck ist maschinenlesbar und wird Machine Readable Zone (MRZ) genannt. In Abbildung 3.3 ist ein Reisepass dargestellt die MRZ ist zur besseren Sichtbarkeit rot umrandet. Dadurch soll gewährleistet werden, dass man nur auf die gespeicherten Daten Zugriff hat wenn man im Besitz des Dokuments ist.

Chip (PICC)		**Terminal (PCD)**
		Optisches Einlesen der MRZ und Ableitung des Schlüssels K
Wähle ein Zufallszahl r_{PICC}		Wähle ein Zufallszahl r_{PCD} und Schlüsselmaterial K_{PCD}
	$\xrightarrow{r_{PICC}}$	
		$e_{PCD} = \mathbf{EM}(K, r_{PCD}\|\|r_{PICC}\|\|K_{PCD})$
	$\xleftarrow{e_{PCD}}$	
$r'_{PCD}\|\|r'_{PICC}\|\|K'_{PCD} = \mathbf{DM}(K, e_{PCD})$		
Überprüfe ob $r'_{PICC} = r_{PICC}$		
Wähle Schlüsselmaterial K_{PICC}		
$e_{PICC} = \mathbf{EM}(K, r_{PICC}\|\|r'_{PCD}\|\|K_{PICC})$		
	$\xrightarrow{e_{PICC}}$	
		$r'_{PICC}\|\|r''_{PCD}\|\|K'_{PICC} = \mathbf{DM}(K, e_{PICC})$
		Überprüfe ob $r''_{PCD} = r_{PCD}$

Tabelle 3.1: Ablauf des BAC Protokolls.

Im Folgenden werden die einzelnen Schritte des BAC Protokolls erklärt (Tabelle 3.1).

- Der Inhaber des Dokuments legt das Dokument zur Kontrolle vor und die MRZ wird eingelesen oder händisch eingegeben.
- Die gewonnenen Daten werden mit Hilfe einer Prüfsumme überprüft und aus den Daten werden zwei Schlüssel (K_{ENC} und K_{MAC}) berechnet mit welchen die nachfolgenden Kommandos zur Authentifizierung gesichert werden. In Tabelle 3.1 ist aus Gründen der besseren Lesbarkeit ein Schlüssel K dargestellt.
- Das Terminal fordert von der Karte eine Zufallszahl an und diese erzeugt die Zufallszahl r_{PICC} und schickt diese als Antwort zurück.
- Nun generiert auch das Terminal eine Zufallszahl r_{PCD} und auch ein Schlüsselmaterial K_{PCD}. Diese beiden Daten werden zusammen mit der Zufallszahl r_{PICC} mit Hilfe des Schlüssels K_{ENC} verschlüsselt und zusätzlich mit einer Kryptographischen Checksumme die von K_{MAC} abhängt versehen. Diese Daten werden nun wieder zur Karte geschickt (e_{PCD}).
- Die Karte wiederum verifiziert die Checksumme, entschlüsselt die Daten und überprüft ob die von ihr gesendete Zufallszahl (r_{PICC}) mit der wieder empfangenen übereinstimmt. Danach wird auch von der Karte ein Schlüsselmaterial K_{PICC} generiert und dieses zusammen mit den beiden Zufallszahlen wieder verschlüsselt und mit einer Prüfsumme versehen zurück zum Terminal geschickt (e_{PICC}).
- Das Terminal verifiziert nun auch die Prüfsumme und ob die Zufallszahl korrekt ist.
- Sind alle Schritte erfolgreich absolviert werden aus den Schlüsseldaten K_{PICC} und K_{PCD} die Session Schlüssel KS_{ENC} und KS_{MAC} abgeleitet mit denen die weitere Kommunikation mit Hilfe von SM verschlüsselt wird.

Das BAC Protokoll weist aber einige Schwächen auf, auf die auch in [Jen09] eingegangen wird. Da für die Verschlüsselung der Übertragung symmetrische Kryptographie verwendet wird kann die kryptographische Stärke der Schlüssel für die Übertragung nicht die Stärke des Passworts überschreiten. Dieses Passwort ist eben die MRZ. Die MRZ hat nun zwar eine theoretische Entropie von höchstens 73 Bit aber in der Praxis wird eine Entropie von unter 40 Bit bis 50 Bit erreicht.

Um nun Angriffe auf das Protokoll zu erschweren, kann die Ausführung des Protokolls künstlich verzögert werden. Dies kann zwar den direkten Angriff erschweren aber wenn die Daten aufgezeichnet werden und offline versucht wird den Schlüssel zu brechen, ist diese Methode auch kein Schutz. Da sich nun die Rechenleistung immer mehr erhöht muss der Zugriffsschutz mit BAC spätestens 2015 ersetzt werden da er sonst über die Lebensdauer des Reisepasses gebrochen werden kann [Jen09].

Chip (PICC)		**Terminal (PCD)**
$z = \mathbf{E}(K_\pi, s)$	$\xrightarrow[z]{D_{PICC}}$	$s = \mathbf{D}(K_\pi, z)$
$\tilde{D} = \mathbf{Map}(D_{PICC}, s)$		$\tilde{D} = \mathbf{Map}(D_{PICC}, s)$
Generiere ein flüchtiges Schlüsselpaares		
$(\widetilde{SK_{PICC}}, \widetilde{PK_{PICC}}, \tilde{D})$		$(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, \tilde{D})$
	$\xleftrightarrow[\widetilde{PK_{PICC}}]{\widetilde{PK_{PCD}}}$	
$K = \mathbf{KA}(\widetilde{SK_{PICC}}, \widetilde{PK_{PCD}}, \tilde{D})$		$K = \mathbf{KA}(\widetilde{SK_{PCD}}, \widetilde{PK_{PICC}}, \tilde{D})$
	$\xleftarrow{T_{PCD}}$	$T_{PCD} = \mathbf{MAC}(K_{MAC}, \widetilde{PK_{PICC}})$
$T_{PICC} = \mathbf{MAC}(K_{MAC}, \widetilde{PK_{PCD}})$	$\xrightarrow{T_{PICC}}$	

Tabelle 3.2: Ablauf des PACE Protokolls.

3.3.2 Password Authenticated Connection Establishment

Das Password Authenticated Connection Establishment (PACE) Protokoll wird in [Jen09] und [Bun08] beschrieben. Das PACE-Protokoll wurde vom Bundesamt für Sicherheit in der Informationstechnik (BSI) als Ersatz für das in Zukunft unsichere BAC Protokoll entwickelt (Siehe auch 3.3.1).

Das Protokoll leitet aus einem Passwort mit geringer Stärke, Sitzungsschlüssel mit hoher Entropie ab. Mit diesen starken Schlüsseln wird die nachfolgende Kommunikation verschlüsselt.

PACE ist so konzipiert das es flexibel und zukunftssicher ist. Es können Sitzungsschlüssel für verschiedene symmetrische Verschlüsselungsverfahren abgeleitet werden. Dadurch, dass bei PACE der Sitzungsschlüssel nicht vom Passwort abhängig ist, bietet PACE einen sehr guten Schutz vor einem Offline Angriff, aber gleich wie auch bei BAC sind zusätzliche Mechanismen gegen Brute-Force-Angriffe erforderlich. Dies können zum Beispiel eine Verzögerung des Protokolls oder das Zulassen nur einer geringen Anzahl von Versuchen sein. In Tabelle 3.2 ist der Ablauf des PACE Protokolls dargestellt.

- Als Grundlage für das PACE Protokoll dient ein statischer Domainparameter D_{PICC}.
- Eine vom Chip generierte Zufallszahl (s) wird mit dem Passwort (K_π) verschlüsselt und zusammen mit Domainparametern (D_{PICC}) zum Terminal geschickt. Dieses entschlüsselt die Zahl wieder und erhält so auch die generierte Zufallszahl s.
- Das Terminal und der Chip nutzen eine Mapping-Funktion um die Zufallszahl auf einen Erzeuger ($\tilde{D}$) für die verwendete asymmetrische Kryptographie abzubilden.
- Mit Hilfe des Erzeugers wird ein Diffie-Hellman Schlüsselaustausch durchgeführt. Dabei wird von Terminal als auch von der Karte ein flüchtiges Schlüsselpaar erzeugt und jeweils der öffentliche Teil zur Gegenstelle geschickt.

- Aus den ausgetauschten Schlüsseln wird nun ein gemeinsamer geheimer Wert berechnet (K).
- Aus dem im vorigen Schritt berechneten Wert K werden die Sitzungsschlüssel (K_{MAC} und K_{ENC}) abgeleitet.
- Im letzten Schritt werden noch Autorisierungstokens ausgetauscht die jeweils von der Gegenstelle überprüft werden.

3.3.3 Extended Access Control

Bei BAC hat man sobald man Zugriff auf das Dokument hat auch Zugriff auf die darauf gespeicherten Daten. Werden nun sensible Daten auf dem Reisepass gespeichert, wie zum Beispiel Fingerabdrücke oder das Bild der Iris dann können diese Daten auch ausgelesen werden. Um nun solch sensible Daten zu schützen wurde das Extended Access Control (EAC) Protokoll entwickelt. Das Protokoll ist in [Bun08] beschrieben. Bei diesem Protokoll muss sich das Terminal authentifizieren und bestätigen welche Zugriffsrechte es für die Daten hat.

Beim EAC Protokoll müssen mehrere Schritte ausgeführt werden. Diese werden in den folgenden Abschnitten genauer beschrieben.

Passive Authentication

Hash Werte über die einzelnen Datengruppen sind in einem Security Object gespeichert und mit einer Signatur versehen. Zur Überprüfung muss dieses Security Object ausgelesen werden, das Zertifikat des Unterschreibers geholt und (anhand eine Certificate Revocation List (CRL) und eines vertrauenswürdigen Country Signing Chip Authentication (CA) Zertifikats) überprüft werden. Mit diesem Zertifikat kann nun die Signatur über das Security Object überprüft werden. Werden nun Daten ausgelesen können die Hash Werte der ausgelesen Daten mit denen im Security Object verglichen werden und somit eine Manipulation festgestellt werden. Passive Authentication schützt nur gegen Manipulation der Daten aber nicht gegen ein Klonen der ganzen Daten.

Active Authentication

Mit der Sicherheitsfunktion Active Authentication wird das Klonen durch Einführung eines chipindividuellen Schlüsselpaares verhindert.

In der Datengruppe 15 wird der öffentliche Teil des chipindividuellen Schlüssels abgelegt. Dieser ist durch Passive Authentication geschützt. Der zugehörige private Schlüssel ist im gesicherten Speicher des Chips abgelegt und kann nicht ausgelesen sondern nur intern vom Machine Readable Travel Document (MRTD) verwendet werden.

Um nun zu beweisen, dass der Chip wirklich in Besitz des privaten Schlüsselteils ist schickt das Terminal eine Zufallszahl an die Karte. Diese signiert diese Zufallszahl und schickt sie zurück. Durch Überprüfen der Signatur mit dem öffentlichen Schlüssel wird sichergestellt das der Chip im Besitz des privaten Schlüssels ist und daher nicht geklont sein kann.

Access Control

Zugriffskontrolle ist nicht nur aus Gründen des Datenschutzes eingesetzt sondern auch um das Klonen des Chips zu erschweren. Das MRTD schützt die darauf gespeicherten Daten gegen unautorisierten Zugriff durch geeignete Zugriffskontrollmechanismen.

- Bei wenig sensiblen Daten die auch aus anderen Quellen einfach zu erhalten sind wie zum Beispiel das Passbild kommt BAC zum Einsatz.
- Sensible Daten wie zum Beispiel Fingerabdrücke dürfen nur für autorisierte Terminals zur Verfügung stehen. Dies wird durch EAC sichergestellt.

BAC überprüft nur ob physikalischer Zugriff auf das Dokument vorhanden ist. Hingegen wird bei EAC zusätzlich sichergestellt das der Terminal auch die nötigen Berechtigungen hat um auf die Daten zuzugreifen.

Chip Authentication

Chip Authentication baut eine Secure Messaging Verbindung zwischen dem MRTD und dem Terminal auf. Dies geschieht mit Hilfe eines statischen Schlüsselpaares welches am Chip gespeichert ist. CA ist eine Alternative zu der in [Int06] beschrieben Active Authentication und erlaubt den Terminal die Echtheit des Chips festzustellen, hat aber zwei Vorteile gegenüber Active Authentication.

- Challenge Semantics werden vermieden, da die produzierten Daten nicht übertragbar sind.
- Es werden starke Session Schlüssel erzeugt mit denen die weitere Kommunikation verschlüsselt wird.

Bei CA gibt es zwei verschiedene Versionen. Im Folgenden wird nur auf jene Version eingegangen, die auch bei der Arbeit behandelt wurde. In Tabelle 3.3 ist der etwas vereinfachte Ablauf des CA Protokolls dargestellt.

- Der Chip hat ein statisches Schlüsselpaar gespeichert. Der öffentliche Teil des Schlüssels wird im ersten Schritt zum Terminal übertragen.
- Das Terminal generiert sich seinerseits ein flüchtiges Schlüsselpaar und schickt den öffentlichen Schlüssel zurück zur Karte.
- Der Terminal und die Karte leiten sich nun mittels Diffie-Hellman Schlüsselaustausch einen gemeinsamen geheimen Wert ab.
- Aus diesem Wert werden nun die Schlüssel für die weitere Verschlüsselung der Kommunikation abgeleitet (K_{Enc} und K_{MAC}).

Um die Echtheit des öffentlichen Kartenschlüssels (PK_{PICC}) festzustellen muss eine Passive Authentifizierung durchgeführt werden.

Chip (PICC)		**Terminal (PCD)**
Statisches Schlüsselpaar $(SK_{PICC}, PK_{PICC}, D_{PICC})$		
	$\xrightarrow[D_{PICC}]{PK_{PICC}}$	
		Wähle Flüchtiges Schlüsselpaar $(\widetilde{SK_{PCD}}, \widetilde{PK_{PCD}}, D_{PICC})$
	$\xleftarrow{\widetilde{PK_{PCD}}}$	
$K = \mathbf{KA}(SK_{PICC}, \widetilde{PK_{PCD}}, \widetilde{D_{PICC}})$		$K = \mathbf{KA}(\widetilde{SK_{PCD}}, PK_{PICC}, \widetilde{D_{PICC}})$
$K_{MAC} = \mathbf{KDF_{MAC}}(K)$		$K_{MAC} = \mathbf{KDF_{MAC}}(K)$
$K_{Enc} = \mathbf{KDF_{Enc}}(K)$		$K_{Enc} = \mathbf{KDF_{Enc}}(K)$

Tabelle 3.3: Ablauf des Chip Authentication Protokolls.

Chip (PICC)		**Terminal (PCD)**
Wähle eine Zufallszahl r_{PICC}	$\xrightarrow{r_{PICC}}$	
	$\xleftarrow{s_{PCD}}$	$s_{PCD} = \mathbf{Sign}(SK_{PCD}, ID_{PICC} \|\| r_{PICC} \|\| \mathbf{Comp}(\widetilde{PK_{PCD}}))$
$\mathbf{Verify}(PK_{PCD}, s_{PCD}, ID_{PICC} \|\| r_{PICC} \|\| \mathbf{Comp}(\widetilde{PK_{PCD}}))$		

Tabelle 3.4: Ablauf des Terminal Authentication Protokolls.

Terminal Authentication

Terminal Authentication (TA) erlaubt es dem MRTD zu überprüfen ob der Terminal die Berechtigungen hat auf sensible Daten zuzugreifen. Da nach dieser Überprüfung auch auf sensible Daten zugegriffen werden kann, muss die nachfolgende Kommunikation SM geschützt erfolgen.

Wie auch beim CA Protokoll gibt es auch hier zwei Varianten. Im Tabelle 3.4 wird die erste Version vereinfacht dargestellt. Der Ablauf ist wie folgt:

- Das Terminal sendet eine Zertifikatskette zur Karte. Dies sind aufeinander aufbauende Zertifikate wobei der Anfang der Kette mit dem im Chip gespeicherten öffentlichen Schlüssel der Country Verifying Certificate Authority (CVCA) verifiziert werden kann. Das Ende der Kette ist ein Zertifikat des Terminals.
- Die Karte verifiziert die Zertifikatskette und überprüft die Zugriffsberechtigungen die in den Zertifikaten angegeben sind. Aus dem Terminal Zertifikat wird der öffentliche Schlüssel des Terminals extrahiert (PK_{PCD}).

Chip (PICC)		**Terminal (PCD)**
Eindeutiger Chip Identifier PK_{ID}		Öffentlicher Sektor Schlüssel (PK_{Sector}, D)
	$\xleftarrow{\substack{PK_{Sector} \\ D}}$	
$I_{ID}^{Sector} = \mathbf{H}(\mathbf{KA}(SK_{ID}, PK_{Sector}, D))$	$\xrightarrow{I_{ID}^{Sector}}$	

Tabelle 3.5: Ablauf des Restricted Identification Protokolls.

- Die Karte generiert eine Zufallszahl r_{PICC} und schickt diese zum Terminal.
- Das Terminal signiert nun mit seinem geheimen Schlüssel diese Zufallszahl zusammen mit einer eindeutigen Identifikation der Karte (ID_{PICC} welche die Dokumentnummer in der MRZ ist) und einem Fingerabdruck des im CA Protokolls ausgetauschten flüchtigen öffentlichen Schlüssel des Terminals $\widetilde{PK_{PCD}}$. Diese Signatur wird als Antwort zurückgeschickt.
- Die Karte verifiziert mit dem aus den Zertifikaten extrahierten öffentlichen Schlüssel ob die empfangene Signatur korrekt ist und gewährt bei einem positiven Ergebnis Zugriff auf die im Zertifikat angegeben sensiblen Daten.

3.3.4 Restricted Identification

Mit Restricted Identification (RI)([Bun08]) ist es möglich eine sektorspezifische eindeutige Identifizierung eines MRTD vorzunehmen. Dies wird dazu benutzt um den MRTD innerhalb eines Sektors wiederzuerkennen. Dieses Verfahren hat folgende Eigenschaften:

- Innerhalb eines Sektors ist die Identifizierung eindeutig.
- Zwischen zwei Sektoren ist es unmöglich das MRTD zu verknüpfen.

Um RI durchführen zu können muss Chip Authentication und Terminal Authentication vorher erfolgreich durchgeführt worden sein. In Tabelle 3.5 ist der Anlauf des RI Protokolls dargestellt.

- Das Terminal sendet den öffentlichen Sektorschlüssel PK_{Sector} zusammen mit den Domainparametern D zur Karte.
- Die Karte berechnet sich nun eine eindeutige Zahl zur Identifikation der Karte in der gewünschten Domain. Mit Hilfe des eindeutigen Chip Identifiers PK_{ID} und eines Diffie-Hellman Schlüsselaustausches wird ein Wert abgeleitet und dieser mit Hilfe einer Hashfunktion in den Sectoridentifier umgewandelt. Dieser wird nun an das Terminal zurückgeschickt und kann zur Identifizierung der Karte verwendet werden.

3.4 Bankanwendungen

Bei Bankanwendungen ist es sehr wichtig dass sie nicht nur in einem Land funktionieren sondern das es auch internationale Standards gibt um die Interoperabilität unter den Ländern zu gewährleisten. Aus diesem Grund schlossen sich drei große Zahlkartenorganisationen zusammen und entwickelten eine gemeinsame Spezifikation den Europay MasterCard und VISA-Standard (EMV) [LLC11].

3.4.1 EMV

Der vereinfachte Ablauf einer Europay MasterCard und VISA (EMV) kompatiblem Transaktion ist in Abbildung 3.4 dargestellt.

- Im ersten Schritt wird die richtige Anwendung auf der Karte ausgewählt und die Daten der Anwendung, wie zum Beispiel welche Protokolle unterstützt werden, werden ausgelesen.
- Im nächsten Schritt wird überprüft ob die Daten auf der Karten nicht manipuliert wurden und die Karte nicht kopiert wurde. Diese Vorgänge sind genauer im Abschnitt Static Data Authentication und Abschnitt Offline Dynamic Data Authentication beschrieben.
- Als nächstes wird verifiziert ob der Inhaber der Karte wirklich der rechtmäßige ist (wie zum Beispiel in Abschnitt PIN Verschlüsselung beschrieben).
- Danach überprüft das Terminal die Transaktion und kann sie ablehnen, erlauben oder nur mit Verbindung zum Server zur Überprüfung erlauben.
- Wenn die Überprüfung durch das Terminal erfolgreich war wird eine ähnliche Überprüfung auch auf der Kartenseite durchgeführt (siehe auch Abschnitt Anwendungskryptogramm). Auch bei dieser Überprüfung kann die Transaktion verweigert, erlaubt oder nur in Verbindung mit einer zusätzlichen Überprüfung am Server erlaubt werden.
- Wenn eine Überprüfung am Server vorgesehen ist muss das Terminal online gehen um vom Server die Transaktion bestätigen zu lassen (Herausgeber Authentifizierung).
- Wird die Transaktion von allen beteiligten Stellen bestätigt, wird sie danach durchgeführt.

Static Data Authentication

Bei Static Data Authentication (SDA) werden die statisch auf der Karte gespeicherten Daten ausgelesen und anhand einer digitalen Signatur, die ebenfalls auf der Karte gespeichert ist, überprüft das Terminal ob die Daten manipuliert wurden. Dadurch können unautorisierte Änderungen an Daten nach der Personalisierung festgestellt werden.

SDA setzt eine Zertifizierungsstelle voraus welche die öffentlichen Schlüssel der Herausgeber der Karten signiert. Jedes Terminal muss den öffentlichen Schlüssel dieser Zertifizierungsstelle gespeichert haben.

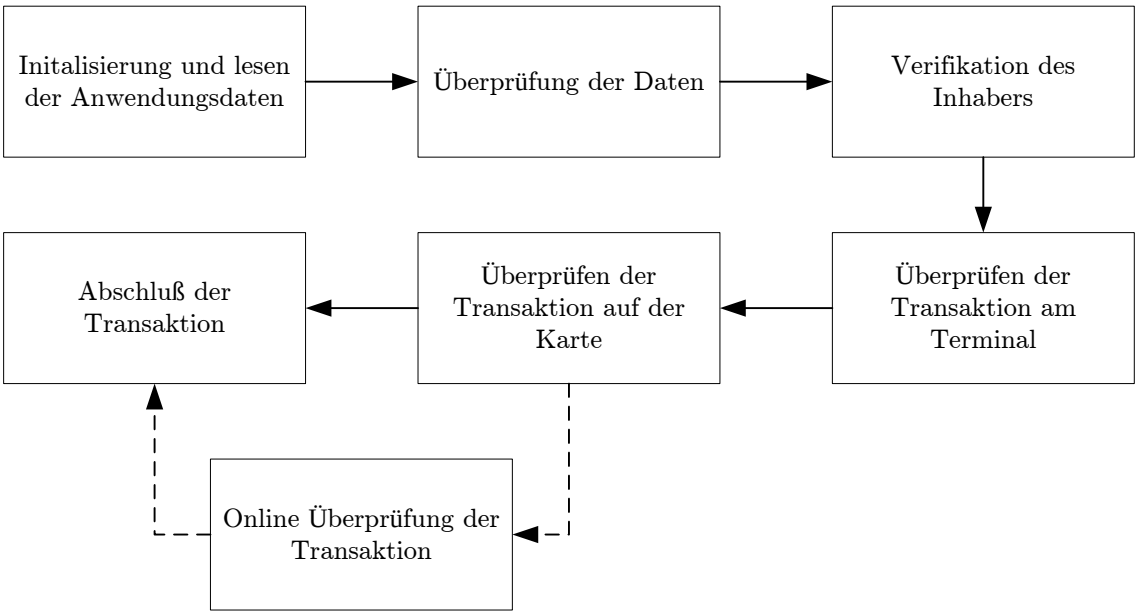

Abbildung 3.4: Ablauf einer EMV Transaktion.

Das Protokoll setzt sich aus drei Schritten zusammen.

- Abfrage des öffentlichen Schlüssels der Zertifizierungsstelle. Auf der Karte ist ein Schlüsselindex der Zertifizierungsstelle gespeichert. Dieser Index zusammen mit der Registered Application Provider Identifier (RID) die sich aus der Application Identifier (AID) ableitet kann den zu verwendeten Schlüssel eindeutig identifizieren.
- Abfrage des öffentlichen Schlüssels des Kartenherausgebers. Dieser ist in Form eines Zertifikats auf der Karte gespeichert. Mit dem öffentlichen Schlüssel der Zertifizierungsstelle wird dieses Zertifikat überprüft und der Schlüssel extrahiert.
- Verifikation der signierten statischen Anwendungsdaten, die auf der Karte gespeichert sind, durch das Terminal. Mit Hilfe des öffentlichen Schlüssels des Kartenherausgebers wird aus den signierten Anwendungsdaten der Hashwert extrahiert. Durch Berechnung des Hashwertes über die Daten und Vergleich des Ergebnisses mit dem extrahierten Wert kann eine Manipulation der Daten festgestellt werden.

Bei diesem Protokoll wird die Signatur über die Daten schon bei der Personalisierung vom Kartenherausgeber berechnet und aufgespielt. Daher ist für diesen Ablauf auf der Chipkarte keine kryptographische Berechnung nötig.

Offline Dynamic Data Authentication

Offline Dynamic Data Authentication (DDA) dient um Fälschungen einer Karte zu verhindern. Das Protokoll baut auf ein Signaturschema mit öffentlichen Schlüsseln auf. Es werden Daten die sich auf der Karte befinden zusammen mit Daten die vom Terminal kommen signiert.

Bei Offline DDA existieren zwei Varianten dies sind DDA und Combined Dynamic Data Authentication/Application Cryptogram Generation (CDA). Diese zwei Methoden unterscheiden sich hauptsächlich in den Daten die Signiert werden. Wie auch bei SDA ist auch bei DDA eine Zertifizierungsstelle nötig die die öffentlichen Schlüssel der Herausgeber signiert.

Bei der Personalisierung der Karte wird nun vom Kartenherausgeber für jede Karte ein eigenes Schlüsselpaar generiert und auf der Karte gespeichert. Der private Schlüssel wird gesichert gespeichert und der öffentliche Schlüssel wird mit dem privaten Schlüssel des Herausgebers signiert und als Zertifikat auf der Karte gespeichert. Bei diesem Protokoll werden folgende Schritte durchgeführt:

- Identifizierung welches Zertifikat verwendet werden soll. Dies erfolgt wie auch bei SDA anhand des Schlüsselindex und der RID. Aus diesem Zertifikat wird der öffentliche Schlüssel der Zertifizierungsstelle extrahiert.
- Aus dem Zertifikat mit dem öffentlichen Schlüssel der Ausstellungsstelle welches auf der Karte gespeichert ist wird der öffentliche Schlüssel extrahiert. Des Weiteren wird das Zertifikat mit dem öffentlichen Schlüssel der Zertifizierungsstelle überprüft.
- Das Zertifikat mit dem öffentlichen Schlüssel der Karte wird geladen und der öffentliche Schlüssel extrahiert. Es wird mit Hilfe des öffentlichen Schlüssels des Kartenherausgebers überprüft ob das Zertifikat gültig ist.
- Das Terminal schickt ein Kommando mit den Daten die signiert werden sollen zur Karte. Diese Daten enthalten unter anderem kartenspezifische Information aber auch eine vier Byte Zufallszahl.
- Die Karte signiert nun diese Daten mit ihrem privaten Schlüssel und sendet die Signatur zurück zum Terminal.
- Der Terminal überprüft mit Hilfe des öffentlichen Schlüssels der Karte ob die Signatur korrekt ist.

PIN Verschlüsselung

Ein PIN ist eine Geheimnummer die nur dem Karteninhaber bekannt sein soll. Sie dient dazu dass man sich gegenüber der Karte authentifizieren kann. Da die Kommunikation zwischen dem Terminal und der Karte drahtlos erfolgt, könnte der PIN ausgespäht werden. Daher ist es notwendig dass diese Daten verschlüsselt übertragen werden. Dies erfolgt durch asymmetrische Kryptographie mit einem öffentlichen und einem privaten Schlüssel. Dieser Schlüssel kann der gleiche sein der bei DDA verwendet wird oder ein eigenes Schlüsselpaar welches gleich gespeichert ist, wie das für DDA verwendete Schlüsselpaar. Für das Protokoll muss nun auch der öffentliche Schlüssel extrahiert werden. Dies erfolgt gleich wie für den öffentlichen Kartenschlüssel bei DDA. Der Ablauf der gesicherten PIN Übertragung ist in Tabelle 3.6 dargestellt.

- Der PIN wird vom Karteninhaber im gesicherten Terminal eingegeben.
- Der Terminal fordert von der Karte eine Zufallszahl (r_{PICC}) an. Die Karte generiert diese Zufallszahl und sendet sie zum Terminal.

Chip (PICC)	**Terminal (PCD)**
	Eingabe des PINs
Wähle ein Zufallszahl r_{PICC}	
	$\xrightarrow{r_{PICC}}$
	Wähle ein Zufallszahl r_{PCD}
	$e_{PCD} = \mathbf{E}(PK_{PICC}, PIN \| \| r_{PICC} \| \| r_{PCD})$
	$\xleftarrow{e_{PCD}}$
$PIN^{'} \| \| r^{'}_{PICC} \| \| r^{'}_{PCD} = \mathbf{D}(SK_{PICC}, e_{PCD})$	
Überprüfe ob $r^{'}_{PICC} = r_{PICC}$	
Überprüfe ob der PIN korrekt ist	

Tabelle 3.6: Ablauf des PIN Verschlüsselung Protokolls.

- Der Terminal generiert seinerseits eine Zufallszahl (r_{PCD}) und verschlüsselt mit dem öffentlichen Schlüssel der Karte (PK_{PICC}) die beiden Zufallszahlen zusammen mit dem PIN und schickt dieses Kryptogramm zur Karte.
- Die Karte entschlüsselt die empfangenen Daten mit ihrem privaten Schlüssel (SK_{PICC}) und überprüft ob die von ihr gesendete Zufallszahl mit der wieder empfangenen zusammenstimmt.
- Die Karte überprüft ob der empfangene PIN korrekt ist.

Anwendungskryptogramm

Das Anwendungskryptogramm wird von der Karte erzeugt und sagt aus ob die Transaktion genehmigt wird oder nicht. Dabei kann die Karte noch eine Risikoabschätzung tätigen und anhand dieser entscheiden welche Antwort sie gibt. Es gibt drei Arten von Anwendungskryptogrammen.

- Das Application Authentication Cryptogram (AAC) sagt aus das die Transaktion verweigert wird.
- Beim Transaction Certificate (TC) wird bestätigt das die Transaktion ohne Verbindung mit dem Server durchgeführt werden kann.
- Kommt ein Authorisation Request Cryptogram (ARQC) als Antwort muss eine Verbindung mit dem Server zur Verifikation aufgebaut werden.

Der Ablauf ist wie folgt:

- Das Terminal sendet die Daten der Transaktion zur Karte. Diese beinhalten unter anderem den Betrag, das Datum, den Type der Transaktion und ein Zufallszahl.

- Die Karte leitet sich einen 16-byte Anwendungskryptogramm Session Schlüssel (SK_{AC}) aus dem Anwendungskryptogramm Master Schlüssel (MK_{AC}) und dem Transaktionszähler ab.
- Die Karte generiert mit Hilfe eines Message Authentication Code (MAC)-Algorithmus unter Verwendung von SK_{AC} eine 8 Byte Anwendungskryptogramm über die Daten die sie vom Terminal empfangen hat und über interne Kartedaten wie den Transaktionscounter.

Herausgeber Authentifizierung

Wird nun von der Karte eine Onlineüberprüfung angefordert muss vom Server des Kartenherausgebers eine Antwort geschickt werden. Diese Antwort nennt sich Authorisation Response Cryptogram (ARPC). Dazu gibt es 2 Methoden.

ARPC Methode 1

- Der 8 Byte ARQC wird mit dem Antwortcode verknüpft.
- Mit Hilfe des Session Schlüssels (SK_{AC}) wird eine TDES Verschlüsselung über die Daten gerechnet. Das Ergebnis der Verschlüsselung ist die Antwort die zurück zur Karte geschickt wird und von dieser verifiziert werden kann.

ARPC Methode 2

- Hier werden die Daten des ARQC zusammen mit einem Karten Status Update Wert und proprietären Authentifizierungsdaten mit einem MAC Algorithmus unter Zuhilfenahme des Session Schlüssels (SK_{AC}) auf einen 4 Byte Wert gebracht.
- Der ARPC ist nun das Ergebnis des MACs zusammen mit den Karten Status Update und den proprietären Daten.
- Diese Daten kann die Karte wieder verifizieren.

Kapitel 4

Stand der Technik

In diesem Kapitel wird auf verschiedene andere Arbeiten eingegangen, die sich mit ähnlichen Themen beschäftigen. Diese Arbeiten werden kurz beschrieben. Des Weiteren wird dargelegt warum ein eigenes Design und eine eigene Implementierung benötigt wurde und nicht auf die bestehenden Arbeiten zurückgegriffen wurde.

4.1 Leistungsvorhersage bei Programmen auf verschiedenen Plattformen

In [BEG+06] wird versucht, eine Vorhersage über die Leistungssteigerung einer Anwendung bei Portierung auf eine andere Plattform zu gewinnen. Es werden etliche Mikroarchitektur unabhängige Charakteristiken der Anwendung gemessen und mit einer vorher erstellten Benchmark Sammlung korreliert. Dieser Ansatz wird gewählt, da es zu schwierig und zu teuer ist eine Anwendung, an der man interessiert ist, auf viele Plattformen zu portieren. Als Charakteristiken werden solche Werte ausgewählt, die unabhängig von der verwendeten Mikroarchitektur sind. Also sind die Werte unter anderem unabhängig von der Cache-Größe, Größe der Verzweigungs-Vorhersage und der Prozessorkern Konfiguration. Insgesamt werden 47 verschiedene Werte für die Korrelation verwendet. Diese werden im Folgenden kurz erklärt.

- Instruktions-Mix

  Als Instruktions-Mix werden die Prozentzahl der Speicher und Ladeoperationen, Ablaufkontrolle, Arithmetischen Operationen, Integer Multiplikationen und Gleitkommaoperationen herangezogen.

- Instruktions Level Parallelisierung

  Instruktions Level Parallelisierung (ILP) [Rot11] ist die Möglichkeit, eine Anwendung parallel auszuführen. Der Ansatz misst den Geschwindigkeitsgewinn bei einer idealen Parallelisierung.

- Register Transfer Charakteristik

  Es werden einige Charakteristiken aufgenommen, die mit Registerzugriffen zu tun haben.

  - Durchschnittliche Anzahl der Eingangsoperanten einer Operation.
  - Das Verhältnis der Lese und Schreibzugriffe auf Register.
  - Die durchschnittliche Anzahl von Instruktionen zwischen dem Schreiben eines Registers bis es wieder gelesen wird.

- Working Set

  Es wird gezählt, wie viele verschiedene 32 Byte bzw. 4 Kilobyte Blöcke bei der Ausführung eines Programmes benötigt werden. Dies erfolgt getrennt für Daten und Codebereich.

- Data Stream Strieds

  Data Stream Strieds ist der Unterschied in der Speicheradresse zwischen zwei aufeinanderfolgenden Speicherzugriffen.

- Verzweigungsvorhersage

  Als weitere Charakteristik wird die Möglichkeit der Vorhersage von Verzweigungen herangezogen.

Die oben beschrieben Charakteristiken werden für 26 verschiedene Benchmarks auf 36 verschiedenen Plattformen bestimmt. Die erhaltenen Daten werden mittels einer Transfermatrix in einen mehrdimensionalen Raum transferiert. Für diese Transformation werden drei verschiedene Methoden mit verschiedenen Transfermatrizen angewandt. Bei der ersten Methode werden die Daten nur normalisiert, in der zweiten wird eine Hauptkomponentenanalyse auf die Daten angewandt und der dritte Versuch wird über genetische Algorithmen unternommen.

Die Anwendung deren Performance auf den verschiedenen Systemen bestimmt werden soll, wird analysiert und die gleichen Charakteristiken wie für die Benchmarks bestimmt. Diese Charakteristiken werden nun mit Hilfe der Matrix die schon für die Benchmarks verwendet wurde in den Vergleichsraum transformiert. Nun werden die drei Benchmarks bestimmt, die der Anwendung am ehesten entsprechen und anhand der gewichteten Leistungswerte dieser Benchmarks ergeben sich Leistungswerte für die Anwendung auf den verschiedenen Plattformen. Es wurde festgestellt, dass die gemessenen Geschwindigkeitsgewinne sehr gut mit der Abschätzung übereinstimmten.

Im Gegensatz zu dieser Masterarbeit wird in [BEG+06] nur auf Personal Computer eingegangen wogegen sich diese Arbeit mit Smart Card Prozessoren beschäftigt. Für Personal Computer stehen viel mehr verschiedene Benchmarks als für Smart Cards zur Verfügung. Daher könnte die hier vorgestellte Arbeit nicht verwendet werden. Aus dieser Arbeit werden aber die Ansätze von der Mikroarchitektur unabhängigen Charakteristik übernommen.

4.2 MESURE Tool

In [BCP09] wird das MESURE Tool vorgestellt. Das ist ein Benchmark für Java Card Plattformen. Zum Zeitpunkt der Arbeit gab es noch keine Lösung am Markt, die zur Evaluierung einer Smart Card, die ein Java Card OS implementiert, verwendet werden kann. In [BCP09] wird eine generelle Benchmark Lösung vorgeschlagen. Die Lösung gliedert sich

in einzelne Schritte, die wichtig für die Messung der Performance sind. Außerdem werden die erhaltenen Ergebnisse validiert. Der Benchmark baut auf Java Card 2.2 auf kann aber auch für Java Card 3.0 Classic Edition verwendet werden.

Es wird nur auf Eigenschaften eingegangen, die für die normale Verwendung der Java Card Anwendung von Belangen sind. Das heißt, dass unter anderem nicht auf das Installieren, wieder löschen und personalisieren eingegangen wird. Mit dem Framework lassen sich drei verschiedene Levels analysieren.

- Das Virtuelle Maschine Level: Es werden die Zeiten für die einzelnen Instruktionen der VM bestimmt aber auch darunterliegende Mechanismen wie zum Beispiel das Lesen und Schreiben des Speichers.
- API Level: Die Performance der zur Verfügung gestellten Schnittstellen, wie zum Beispiel APIs für Kryptographie, wird analysiert. Es werden verschiedene Methoden von Java Card und Global Plattform betrachtet.
- Java Card Runtime Environment: In diesem Level werden Funktionen wie zum Beispiel des Transaktionsmanagement oder der Methodenaufruf im Applet betrachtet.

Das Testframework wurde so angelegt das man die Messung mit jedem beliebigen Lesegerät durchführen kann. Um das Rauschen, das durch die ungenauen Reader und dem Testrechner verursacht wird, zu eliminieren werden statistische Methoden angewandt.

Der Test setzt sich aus zwei getrennten Bereichen zusammen, die miteinander kommunizieren. Der erste Teil übernimmt die Steuerung und läuft am Terminal. Die einzelnen Tests auf der Karte werden angesteuert und die Ausführungszeiten gemessen. Der zweite Teil ist das Applet das auf der Karte läuft und dort die Befehle vom Terminal ausführt. Der ganze Benchmark setzt sich aus fünf Schritten zusammen (Abbildung 4.1).

- Kalibrierung: In diesem Schritt werden die optimalen Parameter für die Messungen ermittelt um die gewünschte Genauigkeit in einer akzeptablen Zeit zu erreichen. Die Parameter sind zum Beispiel die Anzahl der Wiederholungen.
- Bench: Die eigentliche Messung der einzelnen Zeiten für die verschiedenen Werte, die von Interesse sind.
- Filter: Der Filter fasst die gemessen Werte statistisch zusammen und eliminiert dadurch das Rauschen und andere Messungenauigkeiten.
- Extractor: In diesem Schritt werden die gewünschten Daten aus den gemessenen und danach gefilterten Daten extrahiert.
- Profiler: Im letzten Schritt wird ermittelt wie sich die Karte für die verschiedenen Anwendungsdomains eignet. Zu diesem Zweck wurden Anwendungen aus den drei Domains analysiert, daraus Gewichtungsfaktoren für die Ergebnisse des Benchmarks ermittelt und danach ein Leistungswert für die einzelnen Anwendungsdomains ermittelt.

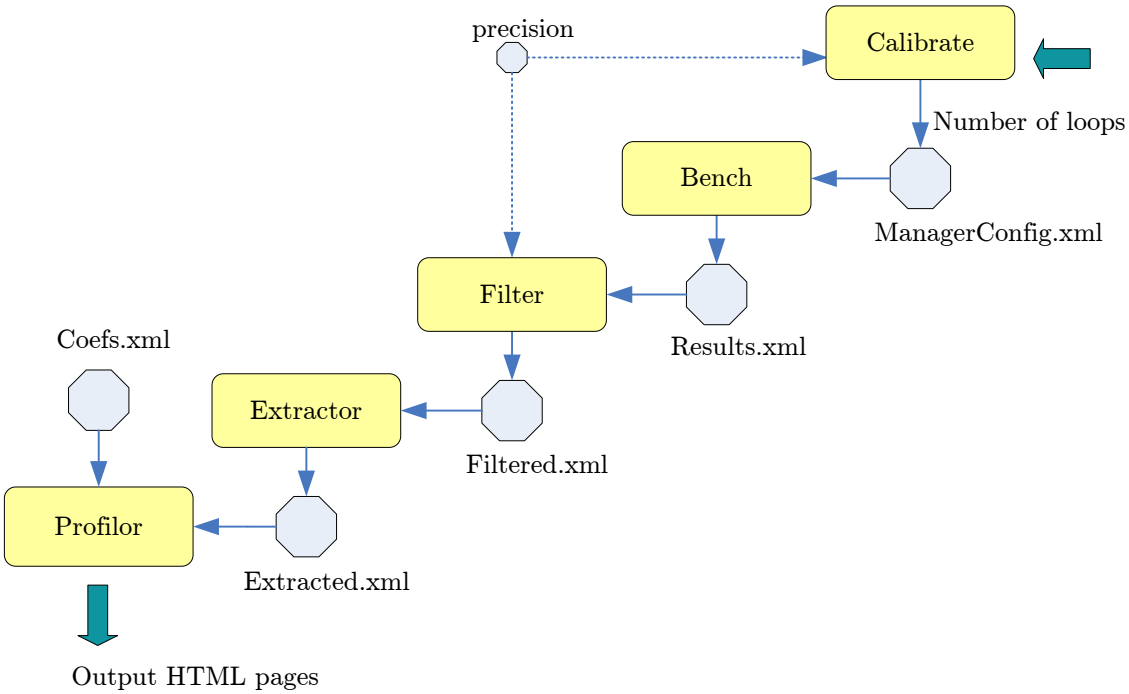

Abbildung 4.1: Benchmark Framework MESURE [BCP09].

Die Anwendungsdomains sind:

- Bank-Anwendungen
- Transport-Anwendungen
- Identitäts-Anwendungen

Mit einen normalen Kartenlesegerät wurde zwar eine Abweichung gegenüber der Messung mit einem speziellen Lesegerät mit exakter Zeitmessung festgestellt aber der Testaufbau lieferte zufriedenstellende Ergebnisse.

Diese Arbeit teilt die Anwendungen auch für die drei Anwendungsdomains und analysiert diese. Da aber die Performance des Java Card Betriebssystems analysiert wurde und nicht die Anforderungen der Anwendungen, erfüllte die Arbeit von [BCP09] die Anforderungen dieser Masterarbeit nicht.

4.3 Java Card im Vergleich zur nativen Implementierung

In [Fis06] wird untersucht, wie sich die Leistungsfähigkeit von Programmen auf Java Card zu einer nativen Implementierung verhält. Es wird versucht auf folgenden drei Fragen einzugehen [Fis06].

- Wie viel schneller ist eine Native-Karte gegenüber einer Java-Karte mit gleicher Hardware-Plattform?
- Ist dieses Verhältnis konstant, oder schwankt es je nach Art des Benchmarks?

- Kann man, wenn man die Performance einer Java-Karte kennt auch auf die Performance der entsprechenden Native-Version schließen?

Es werden eine native und eine in Java Card implementierte Version eines Benchmarks miteinander verglichen. Dabei wird versucht die native Implementierung möglichst ähnlich dem Java Bytecode zu implementieren, da bei einer nativen Implementierung prozessorspezifische Optimierungen einen klaren Vorteil für die native Implementierung bringen würden. Die Java Card Anwendung besteht aus mehreren Unterprogrammen in denen verschachtelte Schleifen durchlaufen werden.

Zugriffe auf den nichtflüchtigen Speicher fanden wegen der Komplexität der nativen Implementierung nicht statt. Auch Zugriffe auf Kryptographiefunktionen werden nicht behandelt, da keine Unterschied zu erwarten ist, da die Operationen nicht in der Java Ebene verarbeitet werden, sondern an die Kryptographieprozessoren weitergegeben werden.

Es werden verschiedene Hardwareplattformen mit je einer nativen und einer Java Card Implementierung betrachtet. Bei diesen Betrachtungen wird versucht ob das Performanceverhältnis zwischen der Java Card und der nativen Implementierung unabhängig von der Hardwareplattform ist.

Es konnte festgestellt werden das der Faktor zwischen der nativen und der Java Card Implementierung bei zwei verschiedenen Plattformen einen anderen Wert hat. Die eine Plattform hatte einen Faktor von 14 die zweite einen Faktor von 22. Der Unterschied wird vermutlich auf die verschiedenen Versionen der Java VM zurückzuführen sein. Daraus folgt das bei unbekannten Karten nur eine vage Schätzung abgegeben werden kann.

Auch diese Arbeit könnte nicht für die gestellte Aufgabenstellung herangezogen werden, da hier zum Vergleich nur einfache Testprogramme herangezogen wurden und nicht echte Anwendungsfälle. In [Fis06] wurde aber auch ebenso wie in dieser Masterarbeit angenommen, dass es bei den Kryptographiefunktionen keinen signifikanten Unterschied zwischen der nativen und der Java Card Implementierung geben wird, da die Berechnungen in Hardware ausgeführt werden.

4.4 Benchmarking von Java Cards

Ein Benchmarking von Java Cards wird in [Erd04] durchgeführt. Es wird versucht verschiedene Smart Cards mit Java Card Betriebssystem auf verschiedene Anforderungen hin zu analysieren.

Anhand von Daten einer Expertenbefragung wird eine Benchmarking Performance Metrik erarbeitet. Anhand dieser Metrik sollen aussagekräftige Benchmarks ausgeführt werden. Es wird auch eine Methodik zur Auswertung der Ergebnisse vorgestellt.

Aus den Expertenmeinungen haben sich mehrere Interessensschwerpunkte gebildet. Aus diesen wurden exemplarisch die Folgenden ausgewählt.

- Ausführungszeit der Kryptographie.
- Laufzeit einzelner Java Operationen wie zum Beispiel eine Division oder einer Programm Schleife.
- Speicherauslastung und Speicherzugriffszeiten.
- Energieverbrauch der Karte bei den einzelnen Tests.

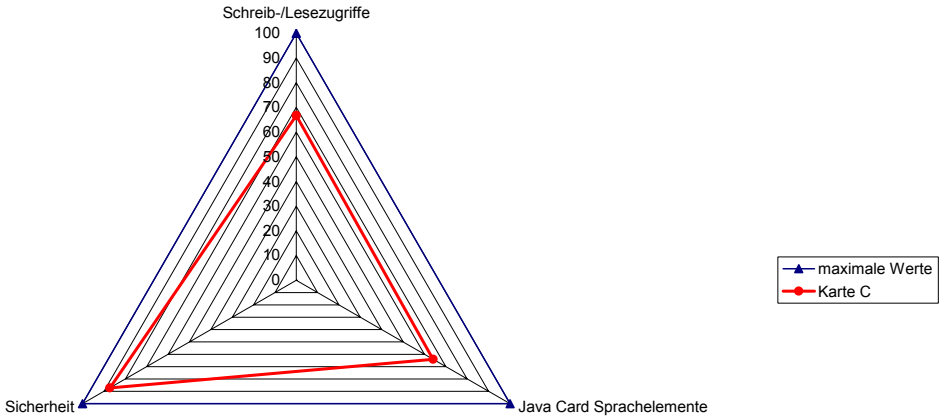

Abbildung 4.2: Performance einer Karte in den verschiedenen Funktionsklassen [Erd04].

Auch in dieser Arbeit wird darauf eingegangen das für verschiedene Anwendungen einer Smart Card die optimale Anordnung der Hardware unterschiedlich ist. Weiters werden die Anwendungen auch in unterschiedliche Anwendungsbereiche unterteilt. Diese sind unter anderem:

- Banking
- Authentication
- Java Card
- Personalisierung

Für jede dieser Anwendungen wird eine Formel erstellt. In diese Formeln gehen die verschiedenen Zeiten, welche die Unterfunktionen einer Chipkarte benötigen, ein. Die Zeiten werden mit der Anzahl der Aufrufe multipliziert und die ganzen Daten aufsummiert.

Ähnlich geschieht es bei den verschiedenen Funktionen einer Smart Card. Diese Funktionen sind in dieser Arbeit zum Beispiel:

- Sicherheit (AES, DES)
- Java Card Sprachelemente
- Schreib und Lesezugriffe

Es wird für alle Funktionsklassen eine mittlere Ausführungszeit pro Hardwareplattform ermittelt. Dies geschieht anhand eines Testapplets für jede Funktionsklasse. Dieses Testapplet wird auf der jeweiligen Plattform ausgeführt und die Zeit der Ausführung wird gemessen.

Der Plattform auf der eine Funktionsklasse am schnellsten ausgeführt wird, werden 100 Punkte zugeordnet und alle anderen Plattformen bekommen proportional dazu Werte zugeteilt. Das Ergebnis ist ein Spinnennetz in dem die Performance einer Smart Card in den verschiedenen Funktionsklassen dargestellt ist. Solch ein Spinnennetz ist in Abbildung 4.2 dargestellt.

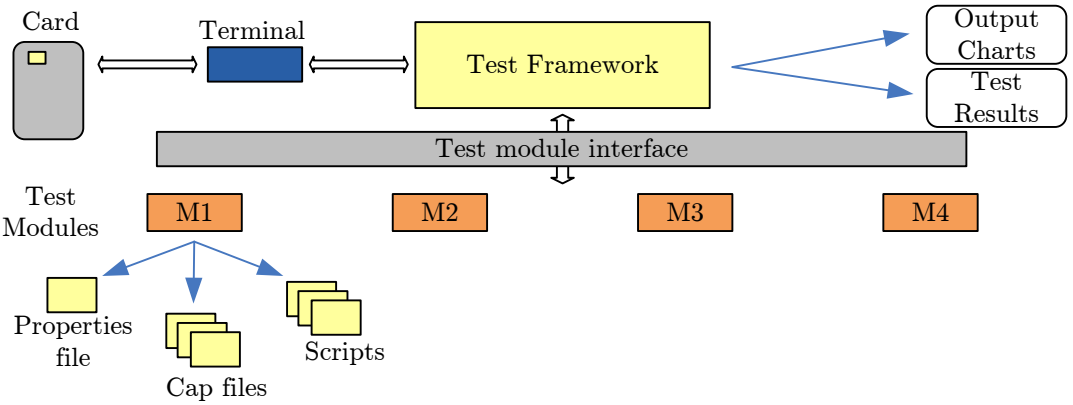

Abbildung 4.3: Test Framework Architektur [Reh05].

In diesem Beispiel erreicht die dargestellte Karte in der Funktionsklasse Sicherheit 87,2 Punkte. Die beste Karte in diesem Bereich erreicht 100 Punkte und war daher um ca. 13% schneller als die betrachtet Karte.

Als Ergebnis der Arbeit werden die analysierten Karten in einem Spinnendiagramm gegenübergestellt. Aus diesem Diagramm ist ersichtlich welche Karte welche Anforderungen am besten erfüllt.

Auch in der Arbeit von [Erd04] wird der Schwerpunkt auf Java Cards gelegt. Außerdem können nur die Zeiten für die gesamte Ausführung analysiert werden, wobei bei dieser Masterarbeit auch ein tieferer Einblick in das System möglich sein soll.

4.5 Java Card Performance Test Framework

Ziel der Arbeit von [Reh05] ist es ein Framework zu designen und zu bauen, das es ermöglicht, Performance Tests auf Java Cards durchzuführen. Anhand dieser Tests sollen Performance Graphen von verschiedenen Java Card VMs generiert werden. Das Framework stellt die Grundfunktionalität, um eine Performancemessung durchzuführen, zur Verfügung. Da Byte Code Performance nicht direkt gemessen werden kann, werden Möglichkeiten diskutiert wie diese Messungen indirekt ausgeführt werden können.

4.5.1 Testframework

Das Testframework arbeitet als Plugin für ein bestehendes Entwicklungsframework für Java Cards. In Abbildung 4.3 ist die Architektur des Testframeworks dargestellt.

Im Zentrum ist der eigentliche Kern des Frameworks. Von diesem Punkt aus wird der ganze Ablauf gesteuert. Über das Test Modul Interface wird beim Starten geprüft, welche Testmodule vorhanden sind bzw. welche ausgeführt werden sollen.

Jedes dieser Testmodule hat eine Properties Datei welche Daten zur Konfiguration des Tests beinhaltet. Anhand dieser Datei erkennt das Testframework welche CAP Dateien mit Implementierungen für den speziellen Testcase auf die Karte geladen werden müssen, und welche Scriptdateien danach ausgeführt werden müssen.

Die CAP Dateien beinhalten den Teil des Tests, der auf der Karte ausgeführt wird.

Diese müssen für die einzelnen Testfälle vom Anwender generiert werden oder es sind bereits bestehende Java Card Anwendungen von denen die Performance analysiert werden soll.

In denn Scriptdateien ist der hostseitige Teil der Test definiert. In diesen Dateien werden die Daten die zur Karte geschickt werden definiert und auch definiert welche Zeiten gemessen und mitprotokolliert werden.

Das Framework lädt nun die entsprechenden CAP Dateien auf die Karte, führt die zugehörigen Script Dateien aus, protokolliert die Daten mit und erstellt aus den Ergebnissen Diagramme.

4.5.2 Opcode Performance Analyse

Bei der indirekten Performance Messung von Opcodes wird zuerst von einer begrenzten Anzahl (n) von Opcodes ausgegangen. Anhand dieser Opcodes werden mehrere ($\geq n$) unterschiedliche Kombinationen von Opcodesequenzen generiert. Die Ausführungszeit dieser Opcodesequenzen wird ermittelt und anhand eines linearen Gleichungssystems kann die Ausführungszeit für die einzelnen Opcodes ermittelt werden. Nach Ermittlung der begrenzten Anzahl von Zeiten werden neue Opcodes zu den Testsequenzen hinzugefügt. Die neuen Sequenzen werden ausgeführt, die Zeiten gemessen und mit diesen Daten wiederum Gleichungssysteme gelöst. Dieser Ablauf wird so lange ausgeführt bis die Ausführungszeit für alle Opcodes ermittelt ist.

Das Hauptproblem dabei ist die linear unabhängigen Opcodesequenzen zu generieren da diese Sequenzen auch gültige Java Card Sequenzen sein müssen.

In [Reh05] werden Opcodes von Java Cards analysiert. Mir diesen Opcodes wird ein Vergleich von verschiedenen Java Card VMs durchgeführt. Daher ist diese Arbeit auch nicht geeignet um zu analysieren ob die verschiedenen Anwendungen einer Domain gleiche Anforderungen an die Hardware haben.

4.6 Entscheidung für ein eigenes System

Alle in diesem Kapitel vorgestellten Arbeiten beschäftigen sich mit der Performance von Computern bzw. der Java Card VM. Der Ansatz für meine Arbeit ist aber ein anderer. Es soll anhand von echten Anwendungen ermittelt werden ob die Anforderungen an die Hardware bei Anwendungen aus der gleichen Anwendungsdomain gleich sind.

Ein großer Unterschied zwischen den hier vorgestellten Arbeiten zu meiner Arbeit ist, dass sich alle Arbeiten mit echten Smart Cards beschäftigen und daher auch Einschränkungen hinnehmen müssen. Zum Beispiel können Zeiten nur von außen gemessen werden und daher können die Zeiten der einzelnen Hardwarekomponenten nicht genau ermittelt werden.

Ein weiterer Vorteil meiner Arbeit ist, dass auch die zeitliche Abfolge der Zugriffe auf die einzelnen Hardwarekomponenten erfasst werden kann und sich daher weitere Analysemöglichkeiten ergeben.

Aus dem hier angeführten Gründen wurde ein eigens Design gewählt. Dieses wird im nächsten Kapitel präsentiert.

Kapitel 5

Design

Im Kapitel 4 wurden Arbeiten vorgestellt die sich mit ähnlichen Aufgaben beschäftigen wie diese Masterarbeit. Es wurde dargelegt warum ein eigenes Design nötig ist. Dieses Design wird nun präsentiert.

Die Darstellung der gesamten Funktionalität eines Systems in einem Diagramm kann zu einer sehr komplexen Abbildung führen. Eine solche Abbildung kann leicht missinterpretiert werden. Daher wurde bei dieser Arbeit ein Ansatz gewählt, wo das Design aus mehreren verschiedenen Sichtweisen auf das System dargestellt wird. Ein solcher Ansatz ist auch in [Kru95] beschrieben.

In den Abbildungen 5.1 und 5.2 wird die Idee hinter der Arbeit präsentiert. Bei dieser Sichtweise wird auf die Überlegungen hinter der Arbeit eingegangen. Abbildung 5.3 stellt den Ablauf der Analyse dar. Es werden die einzelnen Arbeitsschritten und deren Abfolge beschrieben. Bei der nächsten Darstellung (Abbildung 5.4) werden die einzelnen Software-Module vorgestellt und diese beschrieben. Diese Sichtweise bildet die Sicht eines Entwicklers auf das System ab. In der Ein/Ausgabe-Sichtweise (Abbildung 5.5 und Abbildung 5.6) wird das Zusammenspiel der einzelnen Dateien dargestellt. Es wird gezeigt welche Dateien als Eingabedaten für die Verarbeitung dienen und welche Ergebnisdaten produziert werden. In der Prozesssichtweise (Abbildung 5.7) wird beispielhaft ein Teilabschnitt einer Anwendungssimulation dargestellt. Es wird dargestellt wie die Funktionalitäten zusammenspielen und wie die Abläufe im Modell sind. Als letzter Punkt wird noch ein beispielhafter Anwendungsfall dargestellt (Abbildung 5.8). Dieser zeigen wie das ganze System verwendet werden kann.

5.1 Konzept

Dieser Abschnitt beschreibt die Idee hinter der Arbeit. Es werden die grundsätzlichen Überlegungen beschrieben. Das Konzept teilt sich in zwei Bereiche auf. Der erste Bereich ist die Analyse von bestehenden Anwendungen. Der zweite Teil ist die Abschätzung bei einer neuen Anwendung.

Abbildung 5.1 stellt die Idee hinter der Analyse von bestehenden Anwendungen dar. Es gibt verschiedene Anwendungen für eine Smart Card. In der Abbildung sind das A 1 bis A 5. Diese Anwendungen haben unterschiedliche Spezifikationen und stellen daher auch unterschiedliche Bedingungen an die Hard- und Software. Die eine Anwendung erfordert

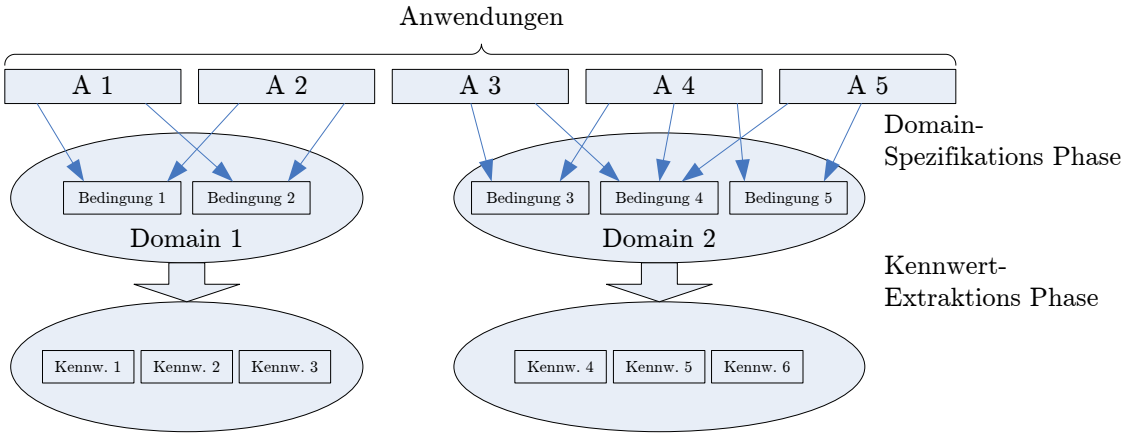

Abbildung 5.1: Konzept der Analyse der unterschiedlichen Anwendungsdomänen.

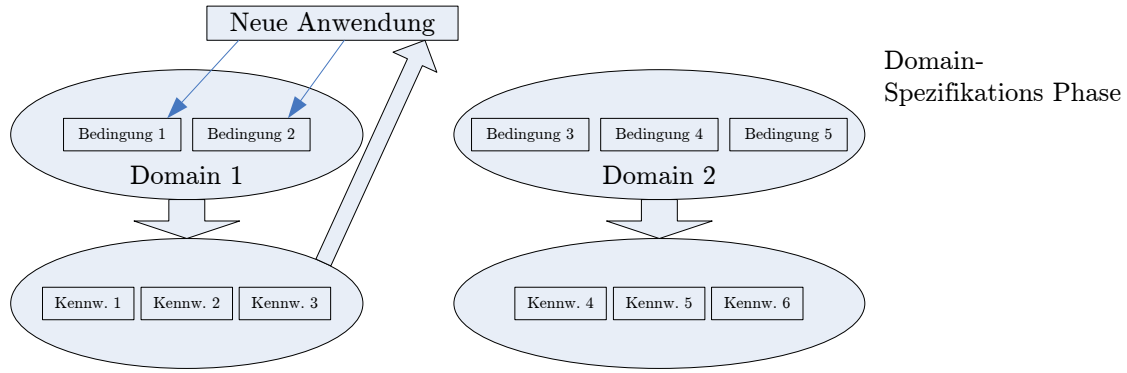

Abbildung 5.2: Konzept der Abschätzung einer neuen Anwendung.

zum Beispiel die Möglichkeit, asymmetrische Kryptographie zu berechnen und eine andere Anwendung muss Daten im nichtflüchtigen Speicher ändern.

Anhand dieser Bedingungen werden die verschiedenen Anwendungen in Anwendungsdomains eingeteilt (Domain-Spezifikations Phase). Die verschiedenen Bedingungen der Anwendungen die sich in einer gemeinsamen Anwendungsdomains befinden, ergeben die Bedingungen der Anwendungsdomain. Diese Bedingungen müssen von der Hardware abgedeckt werden.

Mit Hilfe von Simulationen und Auswertungen (Kennwert-Extraktions Phase) wird ermittelt, welche konkreten Anforderungen an die Hardware gestellt werden. Es werden Kennwerte ermittelt, wie zum Beispiel wie oft der nichtflüchtige Speicher programmiert wird oder wie viele Kryptographieoperationen ausgeführt werden.

In Abbildung 5.2 wird das Konzept hinter der Abschätzung der Kennwerte einer neuen Anwendung eingegangen. Eine neue Anwendung stellt, wie auch die zuvor analysierten Anwendungen, bestimmte Bedingungen an die Hard- und Software. Anhand dieser Bedingungen kann die Anwendung einer der existierenden Anwendungsdomains zugeordnet werden (Domain-Spezifikations Phase).

Wenn nun alle Anwendungen einer Domain die gleichen Anforderungen und damit die gleichen Kennwerte haben, können nur anhand der Zuordnung zu einer Anwendungsdo-

main, die Kennwerte der neuen Anwendung abgeschätzt werden.

Bei der Analyse für diese Arbeit werden alle Anwendungen in die Anwendungsdomains eingeteilt und analysiert. Anhand der Kennwerte wird untersucht ob zwischen den Anwendungsdomains unterschieden werden kann bzw. ob alle Anwendungen in einer Domain ähnliche Kennwerte haben.

5.2 Ablaufdarstellung

In der Ablaufdarstellung wird der zeitliche Ablauf der Analyse dargestellt. Es wird erklärt welche Schritte zur Analyse und Zuordnung einer Anwendung nötig sind. In Abbildung 5.3 ist der Ablauf der Analyse dargestellt.

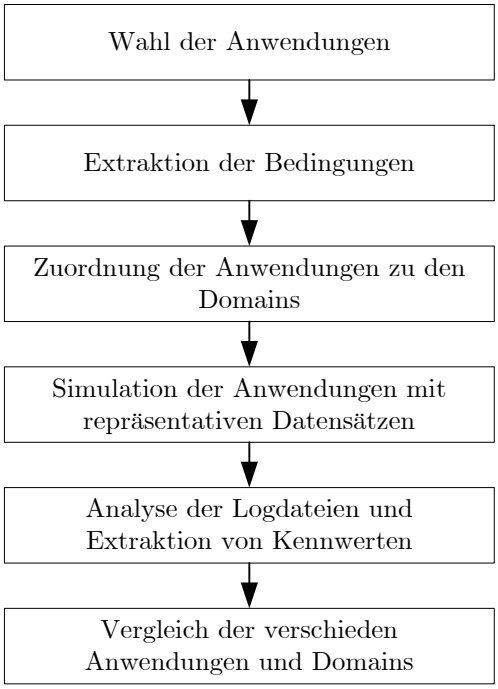

Abbildung 5.3: Ablauf der Analyse.

Im Ersten Schritt werden die Anwendungen die betrachtet werden gewählt. Diese Anwendungen sollen ein möglichst breites Spektrum von verschiedenen Bereichen abdecken um repräsentative Daten zu erhalten.

Aus diesen Anwendungen werden abstrakte Bedingungen an die Hardware extrahiert. Diese Bedingungen können einfach aus der Spezifikation der Anwendung abgeleitet werden. Bei einem Reisepass der die Daten verschlüsselt überträgt können solche Bedingungen zum Beispiel ein großer beschreibbarer Speicher zum Speichern des Passbildes bei der Personalisierung und Unterstützung von symmetrischer Kryptographie zur gesicherten Übertragung der Daten sein.

Im nächsten Schritt werden die Anwendungen anhand der zuvor ermittelten abstrakten Bedingungen Anwendungsgruppen (Domains) zugeteilt. Diese Domains fassen die Eigenschaften der Anwendungen zusammen und können als Übergruppe angesehen werden. Alle

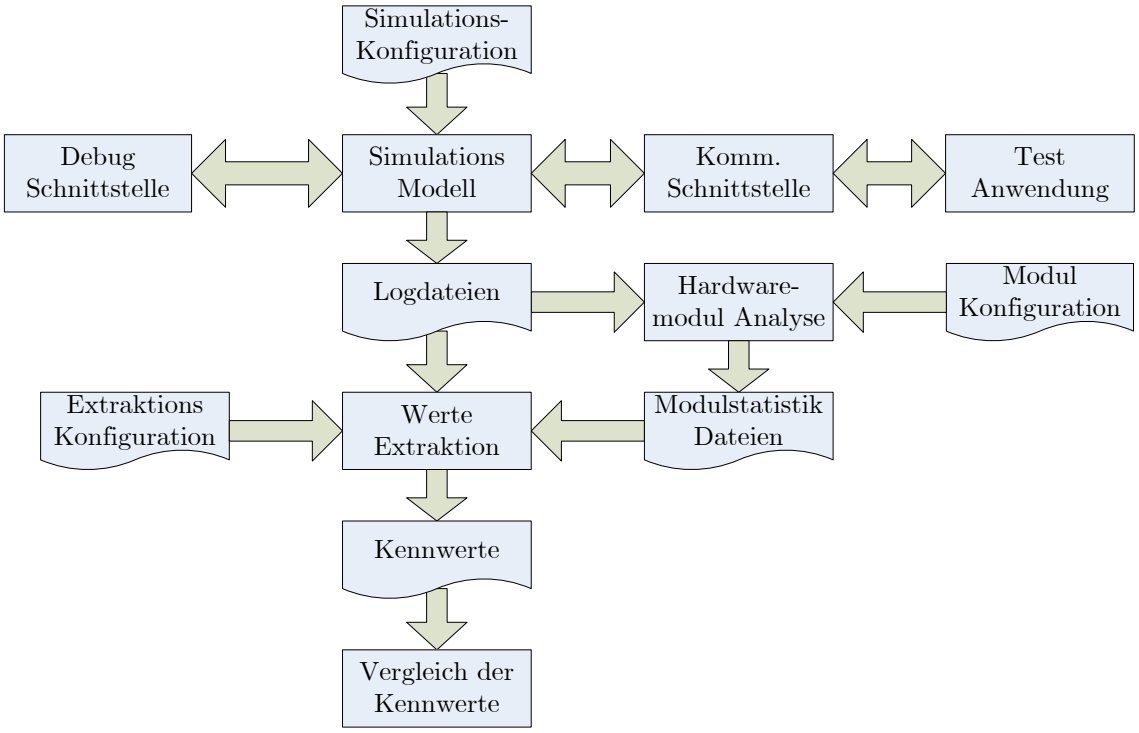

Abbildung 5.4: Entwicklungssichtweise.

Anwendungen innerhalb einer Domain haben annähernd die gleichen abstrakten Bedingungen an die Hardware.

Die gewählten Anwendungen werden nun mit Hilfe des Modells eines Smart Card Prozessors simuliert. Zu diesem Zweck werden echte Anwendungsfälle ausgeführt, wie zum Beispiel das Auslesen eines Reisepasses. Mit Hilfe dieser Anwendungsfälle werden, durch das Modell, Logdateien über die Nutzung der Hardware generiert.

Diese im vorigen Schritt generierten Daten werden analysiert. Es werden daraus Kennwerte über die Verwendung und Aktivitäten der Hardwaremodule extrahiert.

Die Schritte, beginnend von der Simulation bis zum Erstellen von Kennwerten werden für jede Anwendung die analysiert wird, wiederholt.

Nach Abschluss der Simulationen und Auswertung der Daten, können die Kennwerte zwischen den einzelnen Anwendungen und Domains verglichen werden. Dadurch kann gezeigt werden, ob es möglich ist anhand einer Domainzuordnung auf die Hardwareanforderungen zu schließen.

5.3 Entwicklungssichtweise

In diesem Abschnitt wird die Sichtweise eines Entwicklers auf das ganze Projekt dargestellt. Dabei wird das Projekt in mehrere Softwaremodule geteilt und diese werden einzeln beschrieben.

In Abbildung 5.4 ist die Sichtweise auf das Projekt aus Sicht der Entwicklung dargestellt. Im Zentrum steht ein Simulationsmodell eines Smart Card Prozessors, welches

durch eine Konfigurationsdatei parametrisiert wird und von einer Testanwendung ihre Stimuli bekommt. In der Konfigurationsdatei sind die Parameter für das Simulationsmodell definiert. Solche Parameter sind zum Beispiel die Konfiguration der Speichergröße des Modells oder welcher ROM und EEPROM Inhalt beim Modell für die Simulation verwendet werden sollen. Die ROM und EEPROM Inhalte werden beim Starten des Modells in die entsprechenden Hardwaremodule geladen.

Um mit dem Simulationsmodell wie mit einer echten Smart Card, von einer Anwendung aus, zu kommunizieren ist es notwendig eine Schnittstelle dafür zur Verfügung zu stellen. Über dieses Kommunikationsschnittstelle können die Kommandos zur Karte übertragen und die Antworten von der Karte empfangen werden, welche die Anwendung, die den Ablauf einer üblichen Transaktion mit der Karte steuert, benötigt. Für die Anwendung (also zum Beispiel ein Tool zum Auslesen eines Reisepasses) sieht die Kommunikation aus, als ob sie mit einer echten Karte interagiert.

Über eine Debugschnittstelle ist es möglich, während des Ablaufs einer Anwendung weitere Debugausgaben zu erhalten und auch in den Ablauf einzugreifen, bzw. den Inhalt des Speichers zu sichern, um diesen Speicherinhalt später zu analysieren oder wieder in den Speicher zu laden.

Bei der Abarbeitung der Anwendung bei der Simulation werden vom Simulationsmodell Logdateien erstellt. Diese Dateien speichern die Zugriffsstatistik auf die einzelnen Hardwaremodule, alle Speicherzugriffe die auf die Speicher des Simulationsmodells erfolgen und auch die aktiven Zeiten der Hardwaremodule. Aus diesen Logdateien werden Modulstatistiken für die Zeiten ermittelt, an denen die einzelnen Hardwaremodule aktiv sind. Diese Analyse wird mit Hilfe der Modulkonfigurations-Datei konfiguriert. Das Ergebnis dieser Operation sind die Modulstatistikdateien.

Aus den Modulstatistikdateien werden zusammen mit den Logdateien die Kennwerte der Anwendung extrahiert. Dieses Modul ist über die Extraktionskonfigurationsdatei parametrisierbar.

Anhand dieser Kennwerte kann nun eine Anwendung mit anderen Anwendungen verglichen werden und eine Aussage über die Ähnlichkeit der beiden Anwendungen getroffen werden.

5.4 Ein/Ausgabe-Sicht

Im folgenden Abschnitt wird auf das Zusammenspiel der einzelnen Dateien eingegangen. Das beinhaltet welche Eingangsdaten benötigt und verarbeitet werden und welche Ausgangsdaten dadurch generiert werden. In Abbildung 5.5 sind die Eingangs- und Ausgangsdateien und deren Zusammenhang für die Simulation am Simulationsmodell dargestellt. Abbildung 5.6 stellt die Dateien für die Auswertung der vom Simulationsmodell generierten Daten dar. Auf diese beiden Abschnitte wird nun eingegangen.

5.4.1 Simulation

Die Eingangs- und Ausgangsdaten der Simulation beschreiben, welche Dateien beteiligt sind um Anwendungen zu Simulieren und Ergebnisse zu erhalten, welche dann weiter analysiert werden können.

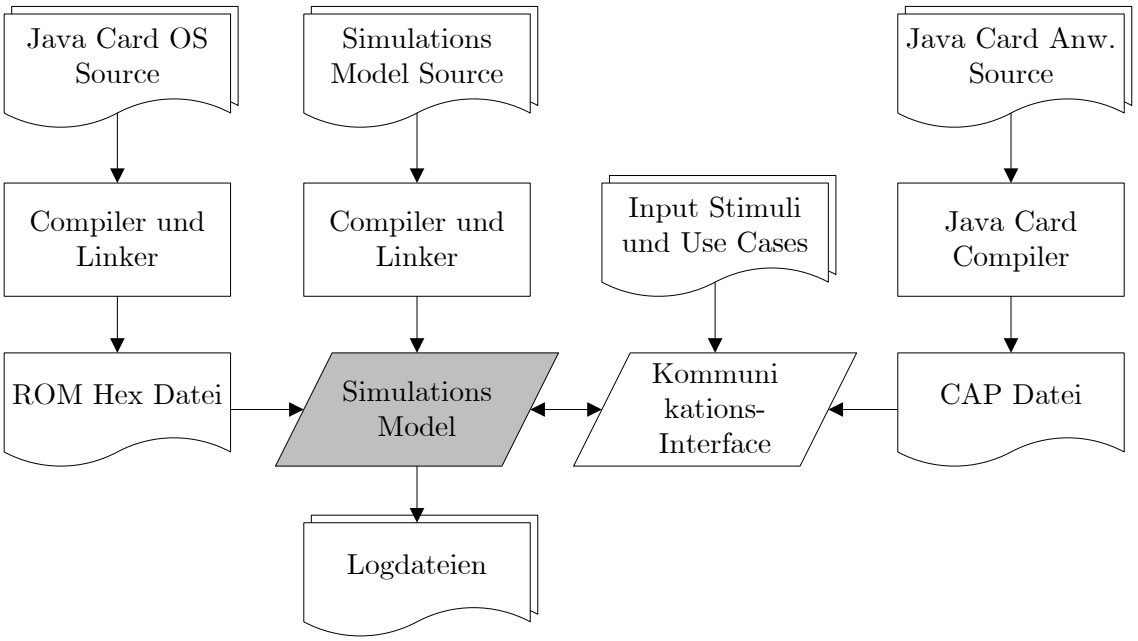

Abbildung 5.5: Ein/Ausgabe des Simulationsmodells.

Simulationsmodell

Im Zentrum von Abbildung 5.5 steht das Simulationsmodell. Das Modell ist ein Abbild eines echten Chipkartenprozessors mit der zugehörigen Peripherie. Auf diesem Modell können reale Anwendungen, wie sie sonst auf einer Chipkarte laufen, simuliert werden. Durch das Modell lassen sich die inneren Zustände der Hardware und die Aktivitäten, die die Anwendung erzeugt, erfassen.

Der Quellcode des Simulationsmodells wird mit Hilfe eines Compilers übersetzt und danach mit einem Linker zum ausführbaren Simulationsmodell zusammengelinkt.

Java Card Betriebssystem

Eine weitere Komponente, die für diese Arbeit wichtig ist, ist ein Java Card Betriebssystem. Der Vorteil eines Java Card Betriebssystems liegt darin, dass Anwendungen dafür leicht in einer Hochsprache entwickelt werden können und es schon existierende Anwendungen dafür gibt.

Das Java Card Betriebssystem besteht meist aus mehreren Schichten die zusammenspielen. Zum einem werden die systemnahen Funktionen und die Java Virtual Maschine meist in Assembler geschrieben. Abstrahierte Funktionen wie zum Beispiel die Application Programming Interface (API) Funktionen können in einer Hochsprache wie C oder Java geschrieben sein. Weiters müssen API Funktionen für die verschiedenen Operationen aus der Virtuellen Maschine zur Verfügung gestellt werden. Der Java Card Operating System (OS) Quellcode besteht also aus einer größeren Anzahl von Dateien, die in den verschiedensten Programmiersprachen implementiert sind.

Die verschiedenen Dateien des Java Card Betriebssystems werden mit Hilfe einer Toolchain, die aus verschiedenen Compilern und Linkern besteht, in den ROM Code des

Betriebssystems umgewandelt.

Dieser ROM Code ist der Code, der bei einem echten Mikrochip bei der Produktion als Inhalt des ROMs vorliegt. Die Daten liegen als Intel Hex Datei vor und beinhalten den ganzen ausführbaren Code der VM und aller API Funktionen. Diese Intel Hex Datei wird bei der Initialisierung der Simulation in den ROM Bereich des Speichers der Simulation geladen. Von diesem Bereich aus wird dann das Programm abgearbeitet.

Java Card Anwendungen

Auf dem Simulationsmodell läuft die Java VM und auf dieser werden Java Card Anwendungen aus den verschiedenen Domains ausgeführt. Diese Java Card Anwendungen bestehen wieder aus ihren Quellcode Dateien. Diese Quellcode Dateien sind normaler Java Source aber mit gewissen Einschränkungen bezüglich des Funktionsumfangs wie zum Beispiel eingeschränkte Typen. In diesen Dateien können auch die API Funktionen des Java Card Betriebssystems aufgerufen werden.

Die Java Dateien werden mit Hilfe eines Java Compilers in Java Bytecode übersetzt und danach in eine Java Card CAP Datei umgewandelt. Diese Datei ist vergleichbar mit einer ausführbaren Java Anwendung. Sie bildet die Kartenseite der Anwendung.

Kommunikation

Um die Simulation auch mit Stimulidaten zu versorgen ist ein Kommunikations-Interface nötig. Dieses Interface stellt eine Standardschnittstelle für die Anwendungen die analysiert werden zur Verfügung. Weiters können über das Interface die CAP Dateien auf die VM, die auf der simulierten Hardware läuft, geladen werden um dort installiert und ausgeführt zu werden.

Logdateien

Als Ergebnis der Simulation einer Anwendung werden verschiedene Daten abgespeichert. Da auch die inneren Aktivitäten des Modells beobachtbar sind, können viele verschiedene Daten über die Aktivitäten der Anwendung erhoben werden. Dies sind zum Beispiel die Aktivitäten der einzelnen Hardwarekomponenten. In 5.4.2 wird näher auf diese Logdateien eingegangen.

5.4.2 Auswertung

In Abbildung 5.6 ist die Abarbeitung der Daten bei der Auswertung und die zugehörigen Eingangs und Ausgangsdaten dargestellt.

Eingangsdaten der Auswertung

Die Eingangsdaten werden wie in 5.4.1 beschrieben von der Simulation generiert. Unter anderem teilen sich diese Daten in die Zugriffe auf die einzelnen Speicherstellen, die Aktivitäten der einzelnen Hardwaremodule und dem Verlauf des Stackspeichers, der auf der Hardware benötigt wir, auf. Es können auch noch weitere Daten über die Aktivitäten mitgespeichert werden um eine möglichst flexible Analyse der Ergebnisse zu erlauben. Die wichtigsten Daten werden kurz beschrieben.

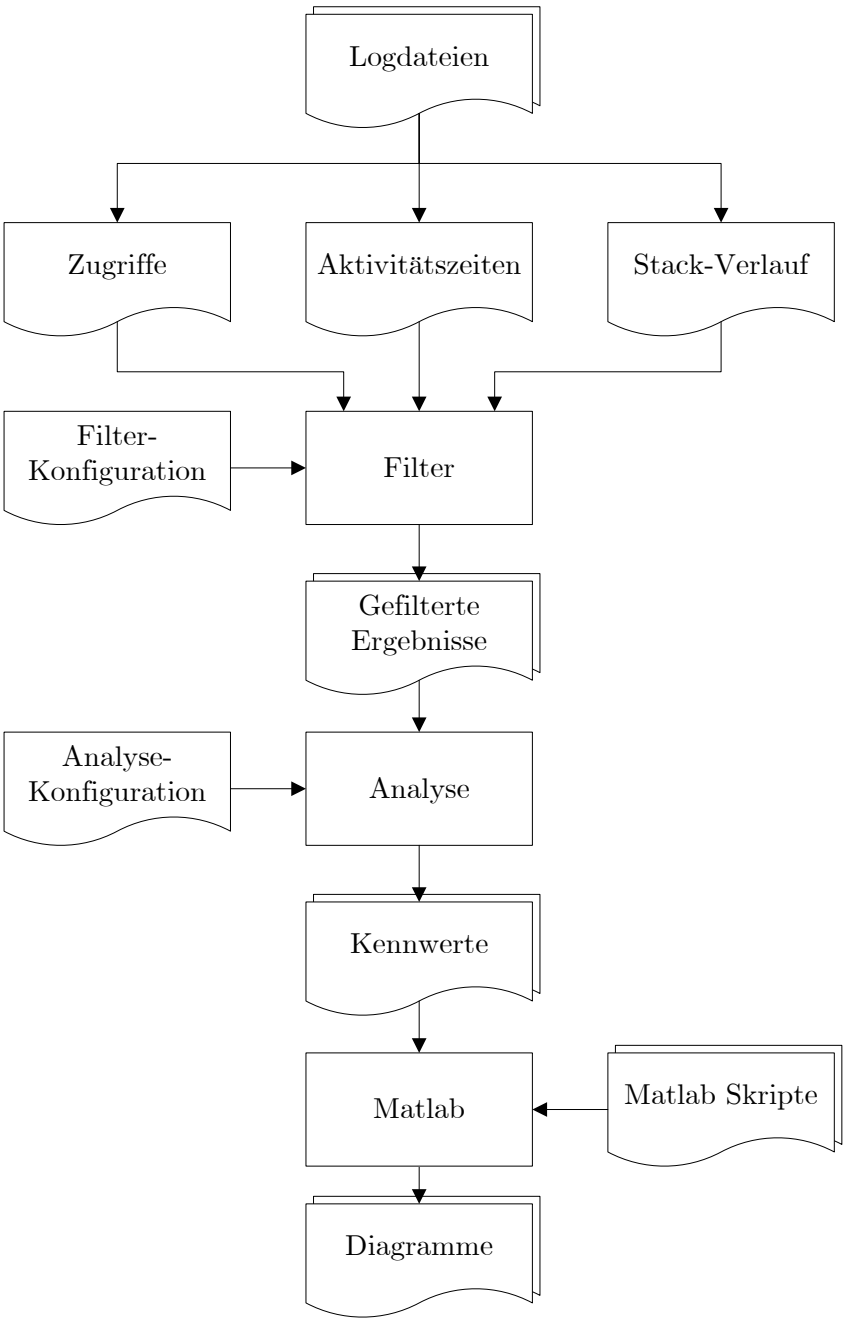

Abbildung 5.6: Ein/Ausgabe der Auswertung.

Bei allen Speicherzugriffen der CPU werden der Zeitpunkt des Zugriffs, die Speicheradresse, die Zugriffsart und der Inhalt des Speichers mitprotokolliert. Diese Daten können dadurch weiterverarbeitet werden und es können auch die verschiedensten Statistiken zu den Zugriffen daraus erstellt werden.

Der Verlauf des Stacks, der auf der Hardware genutzt wird, wird ebenfalls aufgezeichnet. Aus diesen Daten kann eine Statistik der Stack Nutzung wie zum Beispiel der maximale Stackverbrauch ermittelt werden.

Als weiterer wichtiger Punkt werden die Aktivitäten der einzelnen Hardwaremodule aufgezeichnet. Also zum Beispiel in welchen Zeitraum das EEPROM-Programmieren von sich geht oder wann eine Kommunikation mit der Testanwendung stattfindet.

Zu jedem Zeitpunkt an dem es zu einer Aktivitätsänderung kommt, also zum Beispiel beim Starten einer EEPROM Programmierung oder beim Beenden dieser Programmierung, wird der aktuelle Zustand des Systems gespeichert. Dieser beinhaltet die Aktivitäten des Systems bis zum aktuellen Zeitpunkt. Ein Beispiel dafür ist wie oft welcher CPU Opcodes bis zu diesem Zeitpunkt ausgeführt wurde. Diese Daten können später ausgewertet werden, um daraus einen genaueren Einblick in die Aktivitäten der CPU und des Systems zu erhalten.

Da ein Ablauf einer Anwendung meist über mehrere Abschnitte im Lebenszyklus einer Smart Card geht, wie zum Beispiel die Personalisierung oder das Auslesen, können auch diese Abschnitte mitgespeichert werden. Zum Lebenszyklus einer Smart Card siehe 1.1.5.

Filterung

Aus der großen Menge an Daten kann mit Hilfe einer Verkettung von Filtern ein eingeschränkter Datensatz erzeugt werden. Durch die frei konfigurierbaren Filter können verschiedene Arten von Analysen über die Daten der Anwendung durchgeführt werden.

Die Filter sind über eine Datei konfigurierbar. In dieser Datei wird eingestellt in welcher Reihenfolge und nach welchen Kriterien die Daten gefiltert werden. Als Filteroptionen stehen folgende Optionen zur Verfügung:

- Abschnitt: Bei dieser Option wird nach einem vorgegebenen Abschnitt im Lebenszyklus gefiltert. Zum Beispiel werden nur alle Zugriffe, Aktivitäten und der Stackverlauf während der Personalisierung als Ergebnis ausgegeben.

- Zugriffsart: Dabei können die Zugriffe auf den Speicher nach der Art des Zugriffs gefiltert werden. Mögliche Zugriffsarten sind lesender Zugriff, schreibender Zugriff und Zugriffe bei denen ein neuer Opcode zur Ausführung in die CPU geladen wird.

- Speicherbereich: Als letzte Filtermöglichkeit kann ein Speicherbereich angegeben werden. Dies kann zur Filterung der Zugriffe auf die einzelnen Hardwaremodule verwendet werden indem zum Beispiel der Speicherbereich für das ROM angegeben wird.

Um nun zum Beispiel alle schreibenden Zugriffe auf das EEPROM während des Auslesens zu erhalten, muss eine Kette mit den drei richtigen Filtertypen und Parametern hintereinander geschaltet werden.

Analyse

In diesem Schritt werden die zuvor für jede Anwendung ermittelten und gefilterten Daten einer Auswertung unterzogen um Kennwerte daraus zu extrahieren mit denen die einzelnen Anwendungen miteinander verglichen werden können. Die Auswertung ist flexibel gestaltet und es kann konfiguriert werden was alles analysiert werden soll. Bei der Analyse können zum Beispiel folgende Kennwerte ausgewertet werden:

- Stackverbrauch: Der maximale sowie der durchschnittliche Stackverbrauch können ermittelt werden. Diese Werte können zum Vergleich zwischen den Anwendungen herangezogen werden.
- Opcode Analyse: Es wird analysiert, welchen Anteil verschiedene CPU Operationen wie zum Beispiel der Zugriff auf Speicher an der gesamten Aktivität der CPU haben. Dies kann auch während der Aktivität von Modulen ausgewertet werden. Ein Beispiel dafür ist wie sich die CPU Operationen aufteilen während das EEPROM programmiert wird.
- Speicherzugriffs Analyse: Die vorher gefilterten Speicherzugriffe können nach verschiedenen Kriterien analysiert werden.
  - Anzahl der Zugriffe auf die einzelnen Speicherstellen. Anhand dieser Analyse kann festgestellt werden, ob bestimmte Speicherzellen einer stärkeren Belastung ausgesetzt sind als andere.
  - Zugriffsdifferenz: Wie viele Speicherzugriffe liegen zwischen zwei Zugriffen auf die gleiche Speicherstelle. Dieser Wert ist interessant für Caching Strategien.
  - Array Größe: Auf wie viele Speicherstellen wird hintereinander sequenziell zugegriffen.
  - Sprungverhalten: Wie verhalten sich die Adressen aufeinanderfolgender Speicherzugriffe zueinander. Aus diesem Verhalten kann man schließen wie das Ausführungsverhalten des Codes ist. Also ob größtenteils lineare Codeausführung erfolgt oder ob es viele Sprünge in der Ausführung gibt.
  - Cache Größe: Analyse über die Auswirkung von verschiedenen Größen des Programmcaches. Wie oft können die Daten aus dem Cache geladen werden und wie oft müssen die Daten nachgeladen werden.

Die Ergebnisse der gewählten Analysen liegen wieder als Dateien vor und können zur Weiterverarbeitung gebraucht werden.

Darstellung

Um nun die Ergebnisse der einzelnen Anwendungen besser miteinander vergleichen zu können ist es sinnvoll die Anwendungen graphisch gegenüber zu stellen. Zu diesem Zweck werden die Analyseergebnisse mit Hilfe von Matlab und entsprechenden Script-Dateien zur Anzeige der Kennwerte dargestellt. Als Ergebnis dieser Operation werden Diagramme generiert.

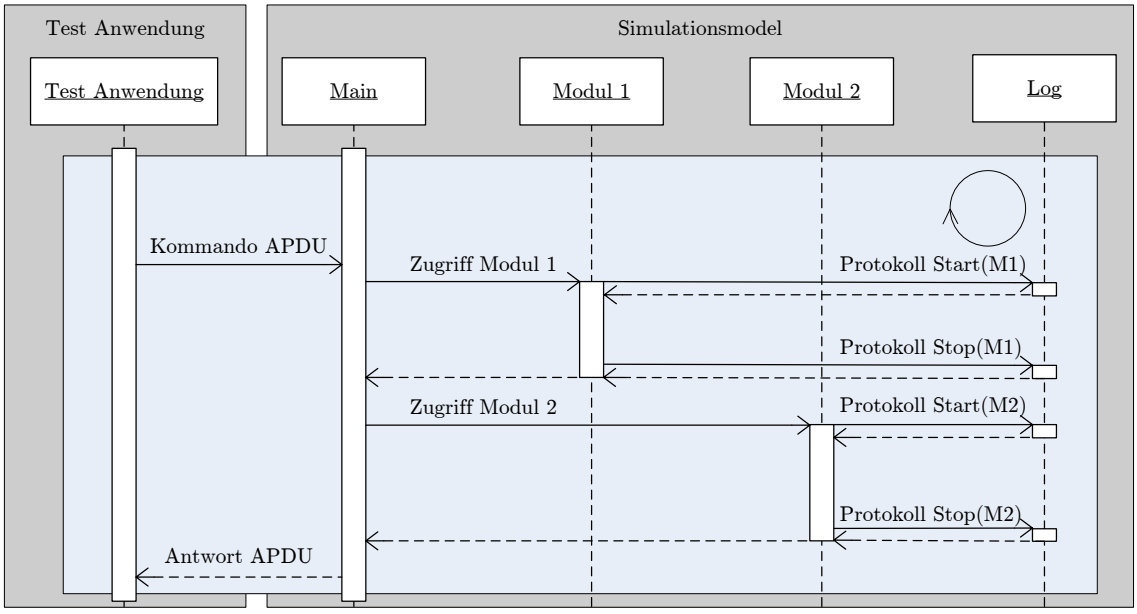

Abbildung 5.7: Prozess-Sichtweise.

5.5 Prozess-Sichtweise

In der Prozess-Sichtweise sind das Zusammenspiel und die zeitliche Abfolge während einer Simulation dargestellt. In Abbildung 5.7 ist ein Ausschnitt der Testanwendung und der Simulation inklusive der Datenaufzeichnung dargestellt.

Der Ablauf erstreckt sich über zwei separate Anwendungen. Links ist die Anwendung dargestellt, die den Ablauf der Befehle die zum Simulationsmodell geschickt werden generiert und entsprechend auf die Antworten des Modells reagiert. Diese Anwendung kann zum Beispiel das Auslesen eines Reisepasses mit allen dazu nötigen Schritten durchführen. Die zweite in dieser Abbildung dargestellte Anwendung ist das Simulationsmodell. Das Modell empfängt die Kommandos, arbeitet diese ab und sendet die Antwort zurück. In der beispielhaften Darstellung besteht das Simulationsmodell nur aus dem Hauptzweig, zwei unterschiedlichen Hardwaremodulen und einer Funktionalität um den Ablauf zu protokollieren.

Die Testanwendung generiert einen Befehl, den die Simulation ausführen soll. Dieser Befehl wird an das Modell gesendet. Der Prozessor des Simulationsmodells beginnt die Verarbeitung des Kommandos. Wird nun der Zugriff auf ein Hardwaremodul benötigt wird dem Modul signalisiert, dass es seine Abarbeitung starten soll. Das Modul informiert sobald es gestartet wird die Loggingfunktionalität darüber das es aktiviert wurde. Die Loggingfunktionalität protokolliert diese Aktivität mit dem aktuellen Zustand des Systems und das Modul startet seine Aufgabe. Wenn die Abarbeitung des Moduls abgeschlossen ist, wird wieder die Loggingfunktionalität informiert und diese protokolliert das die Aktivität des Hardwaremoduls beendet wurde und auch den Zustand des Systems. Danach wird die Verarbeitung des Kommandos im Hauptzweig fortgesetzt bis entweder wieder ein Hardwaremodul benötigt wird oder bis die Abarbeitung des Befehls abgeschlossen ist. Ist der ganze Befehl abgearbeitet wird die Rückantwort gesendet.

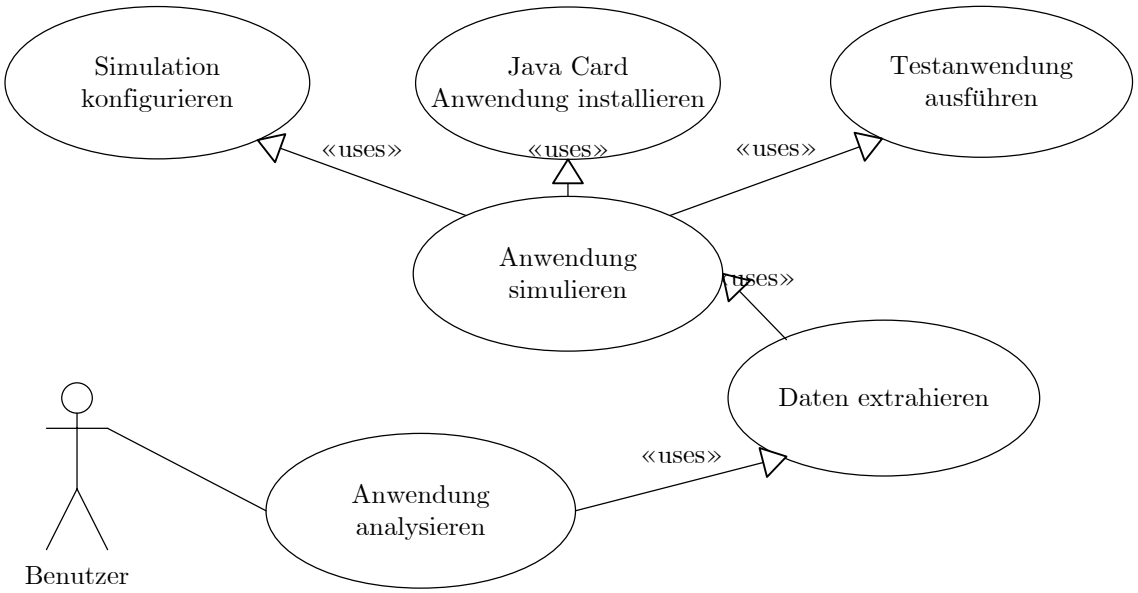

Abbildung 5.8: Anwendungsbeispiel.

Nachdem die Testanwendung die Antwort empfangen hat kann sie das nächste Kommando generieren und zur Simulation senden. Dieser Vorgang wird so lange wiederholt bis die ganze Anwendung durchlaufen ist. Durch die Logging Funktionalität im Simulationsmodell werden die ganzen Aktivitäten und Systemzustände der Hardwaremodule aufgezeichnet.

5.6 Anwendungsbeispiel

Anhand von Anwendungsfällen wird das Design noch verdeutlicht. In Abbildung 5.8 ist ein beispielhafter Anwendungsfall dargestellt. Beim diesem Anwendungsfall wird eine Anwendung analysiert und die Kennwerte der Anwendung ermittelt. Der Anwendungsfall lässt sich in fünf Arbeitsschritte unterteilen. Diese werden nun kurz beschrieben.

Im Zentrum steht auch hier die Simulation der Anwendung. Damit diese Simulation durchgeführt werden kann müssen mehrere Bedingungen erfüllt sei. Als erstes muss das Simulationsmodell richtig konfiguriert sein. Dabei müssen die Speicherkonfigurationen und sonstigen Parameter für das Modell gesetzt werden. Um die Testanwendung auch ausführen zu können muss die Java Card Anwendung auf das Simulationsmodell geladen und dort installiert und personalisiert werden. Bei der Simulation wird die nun die Kartenseite der Anwendung auf der Simulation ausgeführt und die Anwendung die auf die Karte zugreift führt ihren Ablauf aus. Zum Beispiel kann sie die Daten eines Reisepasses auslesen. Aus den Daten die von der Simulation generiert wurden, werden die Kennwerte, die eine Aussage über die zu untersuchende Anwendung erlauben, extrahiert.

Bei einer Anwendungsanalyse werden nun die zuvor beschrieben Schritte ausgeführt und die extrahierten Kennwerte können je nach Anforderung mit bereits bestehenden Anwendungen verglichen oder zur Erzeugung von Kennwerten für eine Anwendungsdomain herangezogen werden.

Kapitel 6

Implementierung

Da sich diese Arbeit über viele unterschiedliche Module erstreckt und für diese Module teils extrem unterschiedliche Anforderungen und Aufgaben existieren, wurden diese Module je nach Aufgabenstellung in verschiedenen Entwicklungsumgebungen und Programmiersprachen entwickelt.

In diesem Kapitel wird nun auf die Implementierung der einzelnen Module eingegangen. Es wird ausgeführt in welcher Programmiersprache und in welcher Entwicklungsumgebung die Module implementiert wurden und es wird auf wichtige Details der Implementierung eingegangen.

Die Module werden in vier große Gruppen unterteilt, die getrennt betrachtet werden. Diese Gruppen sind:

- Das Simulationsmodell zur Simulation der Anwendungen.
- Die Kommunikationsschnittstelle zwischen dem Simulationsmodell und der Testanwendung.
- Die Analysesoftware zur Auswertung der Simulationsergebnisse.
- Die Software zur graphischen Darstellung der Analyseergebnisse.

6.1 Simulationsmodell

Das Simulationsmodell ist der Kern der ganzen Simulation. Ausgehend von einem bereits bestehenden Grundgerüst eines Simulationsmodells das auf Basis von SystemC geschrieben war, wurden die gewünschten Anforderungen implementiert. Da das Grundgerüst schon vorhanden war, musste auf die bestehende Umgebung aufgebaut werden. Diese Umgebung bestand aus einem Microsoft Visual C++ 2003 Projekt und einer proprietären Erweiterungsbibliothek für SystemC. In der Implementierungsphase wurde nun das Modell mit korrekten Zeitinformationen erweitert und mit einer Funktionalität zur Erfassung der inneren Zustände ausgestattet.

Bei der Implementierung der Funktionalität, um die inneren Zustand des Systems zu erfassen, wurde auf große Flexibilität geachtet. Es wurden drei verschiedene Funktionalitäten zum Überwachen der Aktivitäten implementiert. Dies sind:

- Aufzeichnung aller Speicherzugriffe.

- Aufzeichnung der Stackaktivitäten.
- Aufzeichnung der Hardwaremodulaktivitäten.

Auf die Implementierung dieser unterschiedlichen Loggingfunktionalitäten wird in Folge eingegangen.

6.1.1 Logging der Speicherzugriffe

Um eine Offlineanalyse der Speicherzugriffe zu ermöglichen, müssen diese Zugriffe mitgespeichert werden. Da beim Simulationsmodell eine MMU integriert ist, erfolgen die meisten Speicherzugriffe über dieses Hardwaremodul. Nur die Zugriffe auf die Register, die sich direkt in der CPU befinden werden nicht über die MMU ausgeführt. Aus diesem Grund wurde in die CPU des SystemC Modells die zusätzliche Funktionalität zur Speicherung der Zugriffe implementiert. An den Stellen, wo schreibend oder lesend auf die MMU oder die CPU-Register zugegriffen wird, wird eine Funktion eingeklinkt um die Zugriffe in eine Datei zu speichern. Diese Funktion übernimmt die aktuelle Simulationszeit, die Adresse, den Zugriffstype und die Daten die geschrieben werden sollen oder gelesen wurden. Diese Daten werden dann für jeden Speicherzugriff in eine Datei gespeichert. Insgesamt mussten vier Funktionen um diese Funktionalität erweitert werden.

6.1.2 Logging des Stackzugriffs

Um den Verlauf des Stacks analysieren zu können wird der Stackverlauf mitgespeichert. Wie auch bei den Speicherzugriffen wird bei jeder Funktion die auf den Stack des Simulationsmodells zugreift eine zusätzliche Funktion aufgerufen, die das Speichern dieser Werte übernimmt. Bei dieser Funktion werden nur der aktuelle Simulationszeitpunkt und die aktuelle Speichertiefe übergeben. Diese Werte werden für jeden Stackzugriff in eine Datei abgelegt. Dadurch ist der Verlauf der Stacknutzung ersichtlich.

6.1.3 Modulaktivitäten

Um die Aktivitäten der Hardwaremodule mitzuspeichern wurde ein anderer Ansatz gewählt. Um zu ermitteln, was während der Aktivität eines Moduls geschieht, ist es notwendig die internen Zustände des Simulationsmodells während der Modulaktivität abzuspeichern. Um das zu erreichen wird der Zustand bei der Aktivierung des Moduls und beim Beenden der Aktivität des Moduls abgespeichert. Die Daten die bei jeder Aktivitätsänderung eines Hardwaremoduls abgespeichert werden sollen, können aus verschiedenen Hardwaremodulen kommen. Um das zu realisieren wurde ein Logginginterface implementiert. Bei diesem Interface müssen die gewünschten Daten registriert werden.

Die Funktionalität zum Logging der Modulaktivitäten wurde als Singleton implementiert damit von allen Hardwaremodulen aus einfach auf die gleiche Instanz zugegriffen werden kann. Diese Klasse stellt nun verschiedene Funktionen zur Verfügung.

- Registrieren einer Variable die mitgespeichert werden soll.

  Um eine Variable zu registrieren, wird ein Interface zur Verfügung gestellt, welches ein Zeiger auf die Variable und der Name, unter dem die Variable geführt werden

soll, übergeben wird. Diese Daten werden in der Logginginstanz gespeichert. Eine solche Variable kann zum Beispiel ein Zähler für die übertragenen Daten sein.

- Registrieren eines ganzen Arrays mit Variablen die gespeichert werden sollen.

  Sollen mehrere Variablen auf einmal registriert werden, kann das mit Hilfe einer speziellen Registrierungsfunktion erfolgen. Dieser Funktion müssen ein Zeiger zum Array, die Größe des Arrays, ein Name unter dem das Array gespeichert wird und optional noch eine Beschreibung für jedes Element des Arrays übergeben werden. Dadurch ist es zum Beispiel möglich alle ausgeführten Opcodes einfach zu erfassen, indem bei der Analyse des Opcodes einfach das Arrayelement mit dem richtigen Index erhöht wird.

  Auch die bei dieser Funktion übergebenen Daten werden in der Logginginstanz gespeichert.

- Ausgabe der aktuellen Werte aller registrierten Variablen.

  Zur Ausgabe der registrierten Werte stellt die Logginginstanz eine Schnittstelle zur Verfügung. Dabei werden alle registrierten Variablen und Arrays über ihre Zeiger abgefragt und die Namen, inklusive des aktuellen Wertes, in eine Zeichenkette geschrieben.

- Signalisieren das ein Hardwaremodul aktiviert wurde.

  Wird ein Hardwaremodul gestartet, wird dieses Interface der Logginginstanz aufgerufen. Dabei werden der Name des Moduls und der aktuelle Simulationszeitpunkt übergeben. Diese Daten speichert die Logginginstanz intern bis ihr signalisiert wird, dass die Aktivität des Moduls beendet ist. Gleichzeitig werden auch die internen Systemzustände mit dem aktuellen Zeitpunkt versehen abgespeichert, um sie später verarbeiten zu können.

- Signalisieren das die Aktivität eins Hardwaremoduls beendet wurde.

  Beim Beenden einer Aktivität eines Moduls wird der Logginginstanz über dieses Interface die Simulationszeit und der Name des Moduls übergeben. Der Zeitpunkt an dem das Modul gestartet wurde, wird anhand der beim Starten des Moduls übergebenen Daten ermittelt. Die Start und Endzeitpunkte werden gespeichert. Auch der aktuelle Zustand des Systems wird mit einem Zeitstempel versehen abgespeichert.

Um diese Funktionen zur Verfügung stellen zu können, müssen die aktuell aktiven Hardwaremodule und die Variablen die beobachtet werden, intern abgespeichert werden. Beim der Aktivierung eines Moduls wird das der Logging Instanz signalisiert und die Daten werden in eine Liste aufgenommen. Wird nun die Aktivität des Modul beendet, werden aus der Liste die zuvor gespeicherten Daten für das Modul abgerufen, die Daten aus der Liste entfernt und die Informationen in eine Datei gespeichert.

6.2 Kommunikation

Um mit dem Simulationsmodell kommunizieren zu können, wurde eine Schnittstelle benötigt die von beliebigen bereits existierenden Anwendungen benutzt werden kann. Es

gab bereits einen existierenden PC/SC Treiber für einen Smart Card Reader. Ausgehend von diesem Treiber wurde eine Verbindung zwischen der PC/SC Schnittstelle und dem SystemC Modell hergestellt. Es handelt sich dabei um einen virtuellen Smart Card Reader. Dieser virtuelle Reader besteht aus drei Teilen.

- Treiber Seite:

  Auf der Treiberseite wurde die Kommunikation mit dem echten Smart Card Reader gegen eine vereinfachte Kommunikation, mit der Readerseite des virtuellen Readers, über TCP/IP ausgetauscht. Über dieses Interface werden die einzelnen Kommandos zum virtuellen Reader geschickt und dieser kommuniziert mit dem SystemC Modell.

- Reader Seite:

  Als Gegenstelle zum Treiber stellt der virtuelle Reader einen Server Socket zur Verfügung auf den sich der Treiber verbinden kann. Über diesen Socket werden die Befehle zur Kommunikation mit der SystemC Simulation der Karte und zur Konfiguration der Schnittstelle entgegengenommen. Der Virtuelle Reader übersetzt die Kommandos die zur Karte geschickt werden sollen im SystemC Signale die vom Modell als die Signale die vom analogen Empfangsteil kommen interpretiert werden. Diese Signale werden synchron zum 13.56 MHz Träger generiert. Die Signale die die Rückantwort der Karte darstellen, werden vom virtuellen Reader analysiert und dieser sendet die Antwort über TCP/IP zurück zum Treiber.

- Karten Seite:

  Bei der SystemC Simulation werden die Signale die vom virtuellen Reader generiert werden in der Empfangseinheit wieder in Daten übersetzt. Dabei werden die einzelnen Signale gleich wie bei der echten Hardware mit einer State Maschine analysiert und abgearbeitet. Die Antworten die über die kontaktlose Schnittstelle gesendet werden sollen, werden wiederum in Hardwaresignale übersetzt und diese zum virtuellen Reader gesendet.

Die bestehenden Anwendungen verbinden sich mit den PC/SC Treiber um über dieses Interface mit einer Karte zu kommunizieren. Der Treiber übersetzt diese Daten in Befehle für den virtuellen Reader und sendet die Befehle über TCP/IP zum Reader. Die Befehle werden in Hardwaresignale übersetzt. Das Simulationsmodell dekodiert diese Signale und arbeitet die Befehle ab. Die Rückantwort wird wieder in Hardwaresignale übersetzt und zum virtuellen Reader gesendet. Dieser dekodiert seinerseits die Daten, überprüft die Antwort auf Fehler und sendet die Antwort über TCP/IP zurück zum PC/SC Treiber der die Antwort an die Testanwendung weitergibt.

6.3 Auswertung

Da durch das SystemC Modell eine große Menge an Daten produziert wird ist eine Nachanalyse der Daten wichtig. Diese Analyse erfolgt wie im Design-Kapitel beschrieben mit Hilfe von mehreren unterschiedlichen Anwendungen. Auf die Implementierung dieser Anwendungen und wie ihr Zusammenspiel funktioniert, wird in diesem Abschnitt eingegangen.

Das Zusammenspiel der unterschiedlichen Anwendungen und deren Abfolge und Parameter sind mit einer Script Datei konfiguriert. In dieser Datei muss angegeben werden in welchem Verzeichnis die Daten der Simulation zu finden sind. Die Skriptdatei ruft danach alle Analyse- und Filterfunktionen für die unterschiedlichen Abschnitte im Lebenszyklus und die unterschiedlichen Hardwaremodule auf. Welche Anwendungen bei der Analyse zusammenspielen und was diese Anwendungen machen wird nun erklärt.

1. Aufteilung in die Abschnitte im Lebenszyklus der Karte.

   Da die Analyse der Anwendungen über mehrere Abschnitte im Lebenszyklus eine Java Card Anwendung geht, werden zuerst die bei der Simulation generierten Daten auch in diese Abschnitte aufgeteilt. Das hat den Vorteil, dass mit kleineren Datensätzen gearbeitet werden kann was die Analyse beschleunigt und auch den Arbeitsspeicherbedarf bei der Analyse verringert.

   Die Übergabeparameter dieser Anwendung sind die Eingabedaten mit den Aktivitäten der Hardwaremodule, der Dateiname der Ausgabedatei und der Abschnitte der extrahiert werden soll.

   Aus der Datei in denen die Aktivitäten und Abschnitte gespeichert sind, werden der Start und Endzeitpunkt des gewünschten Abschnitts ermittelt. Dies geschieht indem die Datei Zeile für Zeile nach dem Abschnitt durchsucht wird. Sind die Zeitpunkte bekannt werden nur jene Modulaktivitäten in die Ausgabedatei gespeichert, die in diesem Abschnitt vorkommen.

2. Extraktion der Systemaktivitäten.

   Bei der zweiten Anwendung werden aus den zuvor erzeugten Aktivitätszeiten der einzelnen Abschnitte und den Daten, die bei jeder Änderung einer Modulaktivität abgespeichert wurden, Werte extrahiert, die Aussagen darüber erlauben was während der Aktivität eines Hardwaremoduls geschieht.

   Diese Anwendung wird mit mehreren Parametern konfiguriert. Es wird angegeben wo sich die Dateien befinden, die bei den Hardwaremodulaktivitäten abgespeichert werden, die Datei mit den Aktivitätszeiten der Module ist angegeben, das Hardwaremodul für welches die Analyse erfolgen soll und die Datei in welche die Ergebnisse gespeichert werden sollen.

   Um eine größtmögliche Flexibilität zu erreichen, kann eine Kombination von Modulen angegeben werden für die die Analyse erfolgen soll und es kann auch angegeben werden welche Module ausgeschlossen werden sollen. Dadurch ist es zum Beispiel möglich als reine CPU Operationen jene Zeiten zu analysieren an denen sonst kein Modul aktiv ist.

   Es können sehr viele Aktivitätszeiten der einzelnen Module vorliegen. Dabei kann es auch zu Überlappungen von Zeiten kommen. Damit eine effiziente Extraktion der Ergebnisse möglich ist, müssen die Daten zu einem gemeinsamen Zeitverlauf umgewandelt werden. Dabei werden zuerst die benötigten Abschnitte und jene Abschnitte die ignoriert werden sollen ermittelt. Dabei wird auch auf Überlappungen geachtet und gegebenenfalls die überlappenden Abschnitte zu einem vereinigt. Beim nächsten Schritt werden die Ausschnitte die betrachtet werden sollen und jene die

ignoriert werden sollen zusammengefügt. Dabei kann es vorkommen das Segmente die betrachtet werden sollen durch die Segmente die ignoriert werden abgeschnitten werden, in zwei Segmente geteilt werden oder ganz gelöscht werden.

Nachdem die Zeiten ermittelt wurden, zwischen welchen das Verhalten des Systems analysiert werden soll, können für die jeweiligen Start und Endzeitpunkte der Zustand des Systems aus den Dateien gelesen werden und durch Bildung der Differenz die Operationen, die während des zu betrachtenden Zeitraums passieren, berechnet werden. Dies geschieht für jedes Zeitsegment und die daraus resultierenden Werte werden aufsummiert und danach in die Ausgabedatei abgespeichert.

3. Analyse der Opcodes.

   Die dritte Anwendung im Analyseablauf analysiert die Verwendung der unterschiedlichen Opcodes. Dabei wird auf die zuvor erzeugten Dateien aufgebaut. Diese werden als Parameter übergeben. Der zweite Parameter ist der Name der Datei in der die Ergebnisse gespeichert werden sollen. Um wiederum eine große Flexibilität zu erreichen können die einzelnen Opcodeklassen über eine Konfigurationsdatei angegeben werden. Dabei werden die einzelnen Opcodes der zugehörigen Klasse zugeordnet und die Dauer des Opcodes in Zyklen angegeben.

   Bei der Analyse werden nun die einzelnen Opcodes aus der Konfigurationsdatei eingelesen. Wie oft die einzelnen Opcodes ausgeführt wurden, wird aus der im zweiten Schritt generierten Datei geladen. Die Daten die aus den beiden Dateien geladen wurden, werden miteinander verknüpft und aus diesen Daten die prozentuelle Aufteilung in die einzelnen Opcodeklassen ermittelt. Das Ergebnis der Opcode Analyse wird in die Ausgabedatei gespeichert.

4. Extraktion ausgewählter Kennzahlen.

   Beim diesem Schritt können aus den im zweiten Schritt erzeugten Daten wiederum Werte, wie zum Beispiel die Anzahl der Programmierzyklen des nichtflüchtigen Speichers extrahiert werden. Auch diese Anwendung ist frei konfigurierbar, wobei die Eingabedatei, die Ausgabedatei und die Werte die extrahiert werden sollen angegeben werden können.

5. Ermittlung der Modulzeiten.

   Mit dieser Anwendung werden die Zeiten die die einzelnen Hardwaremodule benötigen, ermittelt. Zur Ermittlung können zwei Methoden zum Einsatz kommen. Bei der Ersten werden aus den Dateien, die im zweiten Schritt erzeugt wurden, die Anzahl der CPU Zyklen, die mit den unterschiedlichen CPU Frequenzen ausgeführt wurden, extrahiert. Die Zyklen werden mit ihrer Ausführungsdauer multipliziert und aufsummiert. Die zweite Möglichkeit ist, die einzelnen Zeitsegmente wie auch in Schritt zwei zu ermitteln und aus ihrer Summe die Modulzeit zu erhalten.

6. Filterung der Speicherzugriffe.

   Nach der Analyse der CPU Aktivitäten und Modulzeiten erfolgt eine Filterung und Analyse der Speicherzugriffe.

   Die Speicherzugriffe werden nach verschiedenen Kriterien gefiltert um danach einzeln analysiert werden zu können. Für diese Filterung wurde ein Filter implementiert

der für unterschiedliche Filtermethoden konfigurierbar ist. Anhand der Programmparameter kann übergeben werden, welcher Filter verwendet werden soll, was die Eingangsdaten und Ausgangsdaten sind und welche Filterparameter zum Einsatz kommen.

Der Ablauf dieser Anwendung ist so, dass die Filterparameter analysiert werden und anhand der Filterart die entsprechende Filterfunktion ausgeführt wird. In dieser Filterfunktion erfolgt die Filterung indem die Eingangsdaten, mit den Informationen über die Speicherzugriffe, eingelesen werden, verglichen wird ob die Filterkriterien zutreffen und wenn das der Fall ist die Daten in eine Ausgabedatei geschrieben werden.

7. Analyse der Speicherzugriffe.

   Nachdem die Dateien für die Speicherzugriffe gefiltert wurden, können sie weiteranalysiert werden. Als Parameter für die Analyse der Speicherzugriffe wird die Datei angegeben, in der die zuvor gefilterten Zugriffe gespeichert sind. Bei der Analyse werden mehrere verschiedene Werte ermittelt. Bei allen Werten die bei der Analyse ermittelt werden, werden alle Speicherzugriffe geladen und jeder Zugriff analysiert.

   - Zugriffszahl auf die verschiedenen Speicheradressen:
     Bei der Ermittlung der Zugriffe auf die einzelnen Speicherstellen werden die verschiedenen Zugriffsarten über den ganzen analysierten Zeitbereich aufsummiert und am Ende ausgegeben. Um diese Analyse effizient zu ermöglichen wurden für jede Speicheradresse mehrere Zähler angelegt und je nach Zugriffsart der entsprechende Zähler erhöht.
   - Array Größe:
     Bei der Ermittlung der Arraygrößen werden die direkt aufeinander folgenden Zugriffe analysiert und anhand der Abfolge ermittelt, ob sequenziell zugegriffen wird. Ist die Abfolge nicht mehr sequenziell wird abgespeichert auf wie viele Adressen bis zu diesem Zeitpunkt sequenziell zugegriffen wurde. Diese Analyse erfolgt für alle Speicherzugriffe im betrachteten Zeitraum.

   Für die Zugriffe auf den Programmspeicher werden zusätzlich zu den generellen Analysen auch noch folgende Daten analysiert.

   - Schleifengröße:
     Anzahl der Zugriffe bis wieder auf die gleiche Speicherstelle zugegriffen wird. Für die Implementierung dieser Analyse wird bei jeder Speicherstelle mitgespeichert wann der letzte Zugriff erfolgte. Wird nun erneut auf die Speicherstelle zugegriffen, kann die Differenz zwischen den Zugriffen ermittelt werden. Wie oft die verschiedenen Differenzen vorkommen wird gespeichert. Am Ende der Analyse werden diese Daten in eine Datei ausgegeben.
   - Cache:
     Bei der Analyse des Cacheverhaltens wird mit verschiedenen Cachegrößen untersucht wie oft Daten im Cache gefunden werden und wie oft die Daten aus dem Speicher geladen werden müssen. Als Cachemodell wurde ein einfaches

Modell angenommen, bei dem immer ein ganzes Segment also zum Beispiel 16 Byte auf einmal geladen werden, auch wenn nur ein Bytezugriff erfolgt. Ist nun der nächste Zugriff auch in diesem 16 Byte Segment, dann wird kein Speicherzugriff benötigt sondern die Daten können direkt aus dem Cache geladen werden. Die Analyse erfolgt für unterschiedliche Segmentgrößen und die Anzahl der Hits und Misses wird nach erfolgter Analyse für die verschiedenen Größen in eine Datei ausgegeben.

- Sprung Type:
  Bei der Analyse der Sprungbereiche wurde die Adressänderung von aufeinanderfolgenden Speicherzugriffen betrachtet. Je nach dem um wie viele Bytes sich die Adresse ändert, werden die verschiedenen Adressierungsarten detektiert. Es wird gezählt wie oft welche Adressierungsart vorkommt. Auch bei dieser Analyse werden am Ende die Anzahl der Sprünge bei den verschiedenen Adressierungsarten in eine Datei ausgegeben.
- Sprungweite:
  Die Analyse der Sprungweite ergibt sich daraus wie sich aufeinanderfolgende Zugriffe verhalten. Es wird die Differenz zwischen aufeinanderfolgenden Zugriffen ermittelt. Für jede dieser Differenzen wird aufsummiert wie oft sie vorkommt. Diese Daten werden wieder nach erfolgter Analyse ausgegeben.

Durch die Skriptdateien können die Analysen automatisiert ablaufen und es werden alle benötigen Kennzahlen erzeugt.

6.4 Darstellung

Bei der Analyse der Daten wurde eine große Menge an Daten für die unterschiedlichen Anwendungen generiert. Um die Anwendungen nun miteinander vergleichen zu können ist es nötig eine geeignet Darstellung zu finden. Bei einer optischen Darstellung sind die Unterschiede leicht ersichtlich bzw. Abläufe einfach darzustellen. Daher wurde für die Darstellung eine optische Ausgabe gewählt.

Da viele Kombinationen von Vergleichen nötig sind, ist es nicht sinnvoll die Erzeugung der Diagramme händisch durchzuführen. Aus diesem Grund wurde ein automatisiertes Verfahren gewählt, bei dem die Vergleichsdiagramme automatisch generiert werden.

Als Grundlage für diese Generierung wurde Matlab herangezogen da es die Fähigkeit besitzt große Datensätze zu verarbeiten und diese Verarbeitung automatisiert werden kann. In dieser Entwicklungsumgebung wurde die graphische Darstellung der Analyse implementiert. Die Daten werden von den bei der Analyse generierten Dateien geladen, aufbereitet und graphisch ausgegeben. Die Implementierung der Darstellung teilt sich in mehrere Bereiche auf.

- Darstellung des Zeitbedarfs der einzelnen Hardwaremodule.

  Die bei der Analyse extrahierten Zeiten der Hardwaremodule werden aus den entsprechenden Dateien eingelesen. Für jeden Lebenszyklus der Karte wird ein Tortendiagramm generiert. Die einzelnen Segmente entsprechen dem Zeitbedarf der zugehörigen Hardwaremodule.

- Aufteilung der CPU Opcodes in den verschiedenen Lebensphasen.

  Für die einzelnen Lebensphasen einer Karte wird die Aufteilung der CPU Opcodes während die unterschiedlichen Hardwaremodule aktiv sind gegenübergestellt. Auch hier werden die Daten aus den zuvor generierten Dateien geladen. Aus diesen Daten wird die Anzahl der Zyklen, die für die unterschiedlichen Opcodes benötigt werden extrahiert. Diese Daten werden für alle Lebensphasen geladen, danach auf 100% normiert und graphisch ausgegeben.

- Vergleich der CPU Opcodes bei unterschiedlichen Anwendungen.

  Die zur Darstellung der Opcodes geladenen Daten werden für alle Anwendungen in einer Datenstruktur gespeichert. In einer doppelten Schleife wird eine Analysefunktion aufgerufen in der alle Kombinationen des Lebenszyklus und der Hardwaremodule durchlaufen werden. Für jede dieser Kombinationen wird nun die Aufteilung der CPU Opcodes gegenübergestellt. Dazu ist es nötig die Daten aus den Strukturen zu laden und in das richtige Format für die graphische Darstellung umzuwandeln.

- Vergleich der Stackgröße, des Sprungverhalten, der Sprungbereichs und der Schleifeneigenschaften.

  Bei der Generierung für diese Ausgaben werden aus den zuvor generierten Dateien die gewünschten Daten für die unterschiedlichen Anwendungen geladen. Diese Daten werden normiert und für alle Anwendungen parallel dargestellt.

- Darstellung des zeitlichen Verlaufs der Aktivitäten der Hardwaremodule.

  Der Zeitlich Verlauf einer Anwendung kann einen schnellen Überblick über die Aktivitäten der Anwendung geben. Um auch einen Zusammenhang zwischen den Speicherzugriffen, dem Stackverlauf und der Hardwaremodulaktivität zu erhalten, können diese Aktivitäten nebeneinander dargestellt werden. Damit Details einer solchen Darstellung betrachtet werden können, wurde die Möglichkeit implementiert, in die Darstellung hinein zu zoomen. Dabei ist zu beachten, dass alle Ansichten synchron gezoomt werden müssen. Da Matlab eine solche Funktionalität nicht zur Verfügung stellt, musste sie selbst implementiert werden. In Abbildung 6.1 ist die zeitliche Darstellung eines Anwendungsablaufs dargestellt.

  Welche Dateien zeitlich dargestellt werden sollen, kann beim Start des Matlab Scripts ausgewählt werden. Als erstes kann der Dateiname der Datei mit den Modulaktivitäten angegeben werden. Optional kann eine Datei mit dem Verlauf des Stackspeichers angegeben werden und es können außerdem mehrere Dateien mit unterschiedlichen Speicherzugriffen ausgewählt werden.

  Nach Auswahl der Dateien werden die Daten aus den Dateien mit den Speicherzugriffen und dem Stackverlauf geladen und in Speicherstrukturen abgelegt. Für die einzelnen Parameter einer solchen Zeichenkette gibt es eine eigene Struktur in der sie abgelegt werden Diese Strukturen sind:

  – data: Die Werte die aus der Datei geladen werden. Das können zum Beispiel die aktuelle Stackgröße oder die Adresse des aktuellen Speicherzugriffs sein.
  – time: Die zu den Werten gehörenden Zeitstempel.

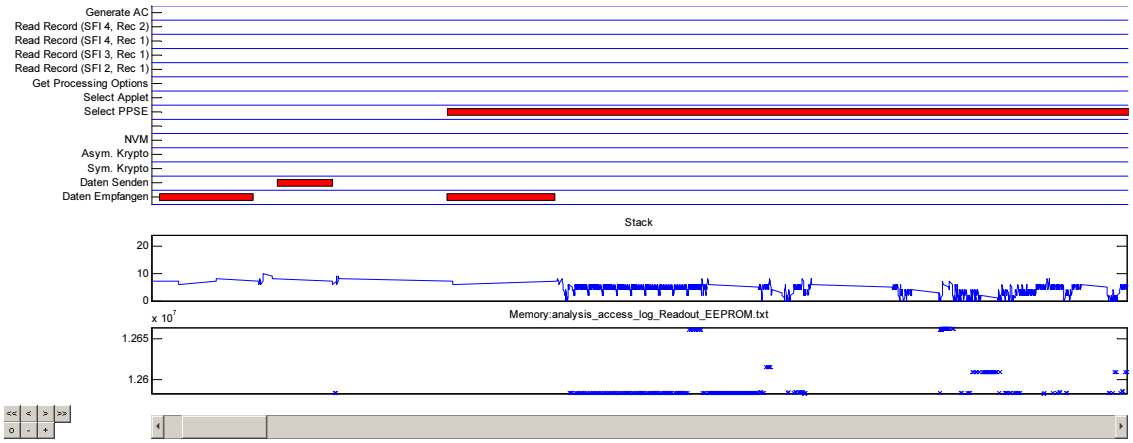

Abbildung 6.1: Darstellung des zeitlichen Verlaufs der Aktivitäten der Hardwaremodule und Speicherzugriffe.

- labels: Die Beschriftung der Diagramme die erstellt werden. Diese werden aus den Dateinamen extrahiert.
- mode: Der Darstellungsmodus des Diagramms. Es wird für den Verlauf des Stacks ein anderer Modus verwendet als für die Speicherzugriffe.

Nachdem diese Datensätze aufgebaut sind werden sie zusammen mit dem Dateinamen der angibt wo die Aktivitätszeiten zu finden sind der Darstellungsfunktion übergeben. Diese übernimmt die Darstellung und auch die Synchronisation des Zooms.

Bei der Darstellung der Modulaktivitäten werden die einzelnen Module zeilenweise dargestellt. Ist das Modul aktiv wird ein Balken dargestellt und bei keiner Aktivität ist der Zeitbereich frei. Mit Hilfe einer Konfigurationsdatei kann die Darstellung der Modulaktivitäten konfiguriert werden. Dabei ist es möglich die Anzeige einzelner Module zu unterdrücken und anzugeben in welcher Zeile welches Modul angezeigt werden soll.

Bei der Darstellung selbst werden zuerst die Datei mit den Hardwaremodulaktivitäten und die Konfiguration für die Darstellung geladen. Im nächsten Schritt werden alle Aktivitäten durchgegangen und anhand der Konfiguration die richtigen Zeilen für die einzelnen Aktivitäten ermittelt. Nachdem alle Daten ermittelt wurden, werden die einzelnen Aktivitäten als Rechtecke in den entsprechenden Zeilen und mit den entsprechenden Zeiten ausgegeben. Als Abschluss werden noch Trennlinien eingefügt und die Achsen richtig skaliert und beschriftet.

Die schon zuvor geladenen Speicher und Stackaktivitäten werden in eigenen Diagrammen im gleichen Fenster ausgegeben und die Skalierung der Achsen passend zu den Aktivitäten gesetzt.

Um nun alle Achsen synchron scrollen zu können wurden Steuerelemente eingeführt mit denen gezoomt und gescrollt werden kann. Dadurch ist es möglich den Zoomfaktor zu erhöhen oder zu verringern und die Zeitachse zu verschieben.

Das ermöglicht es detailliert in die verschiedenen Bereiche beim Ablauf einer Anwendung hinein zu zoomen und dadurch ein gutes Verständnis über das Zusammenspiel der einzelnen Komponenten bei der Ausführung zu erhalten.

Bei all diesen Darstellungen werden die Ergebnisse auch geordnet abgespeichert, um sie für eine weitere Verwendung verfügbar zu haben. Mit dieser Implementierung wurde nun eine Simulation und Analyse für fünf verschiedene Anwendungen durchgeführt. Im nächsten Kapitel wird auf die Ergebnisse der Analyse eingegangen und es werden die erzeugten Diagramme präsentiert.

Kapitel 7

Ergebnisse

Aufgrund der Architektur von typischen Smart Cards und dem Ablauf einer Anwendung wurden die Ergebnisse in folgende Kategorien unterteilt.

- Zeitlicher Ablauf von Anwendungen.
- Der prozentuelle Zeitbedarf der einzelnen Hardwaremodule.
- Die Aufteilung der CPU Opcode in verschiedene Opcode Klassen.
- Statistische Werte über die Stack Tiefe.
- Die Größe der Schleifen im Programmcode.
- Sprungverhalten im Programmfluss der Anwendung.

7.1 Zeitlicher Verlauf

Der zeitliche Verlauf einer Anwendung mit den Aktivitäten der unterschiedlichen Hardwaremodule kann einen schnellen Überblick über eine Anwendung geben. In diesem Abschnitt werden nun solche Verläufe für Bank- und Passanwendungen dargestellt.

Abbildung 7.1 stellt den Ablauf einer Passanwendung die mit EAC geschützt ist beim Überprüfen des Passes dar. Im unteren Bereich ist dargestellt wann die einzelnen Hardwaremodule aktiv sind. Man sieht viele Datenpakete die empfangen und gesendet werden. Nach dem Empfangen bzw. vor dem Senden ist der Koprozessor für symmetrische Kryptographie aktiv um die Daten zu ver- und entschlüsseln. Es ist auch dargestellt wann asymmetrische Kryptographie im Einsatz ist und wenn auf den nichtflüchtigen Speicher geschrieben wird.

Der obere Bereich stellt den Ablauf der einzelnen Befehle bei einer Passanwendung dar. Dadurch ist leicht ersichtlich bei welchen Befehlen welche Hardwaremodule aktiv sind. Ein Beispiel dafür ist die Verifikation der Zertifikate. Für diesen Vorgang müssen zusätzlich zur gesicherten Datenübertragung auch die Zertifikate mit Hilfe von asymmetrischer Kryptographie verifiziert werden. Auch der Non Volatile Memory (NVM) wird verwendet. Zum einen dient er zum Schutz gehen Brute-Force Attacken indem ein Zähler mitgespeichert wird und zum anderen wird bei der Verifikation der Zertifikate anhand des Zertifikatsdatums das interne Datum des Passes aktualisiert.

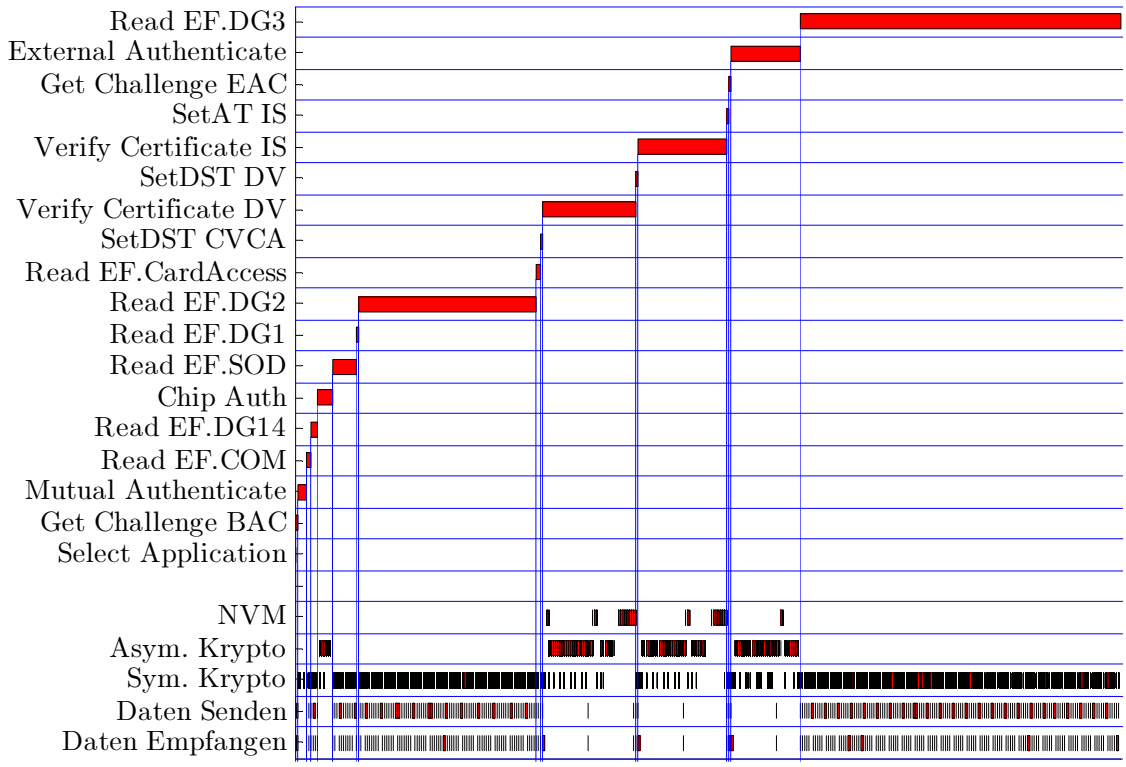

Abbildung 7.1: Zeitlicher Verlauf einer Reisepassanwendung mit EAC.

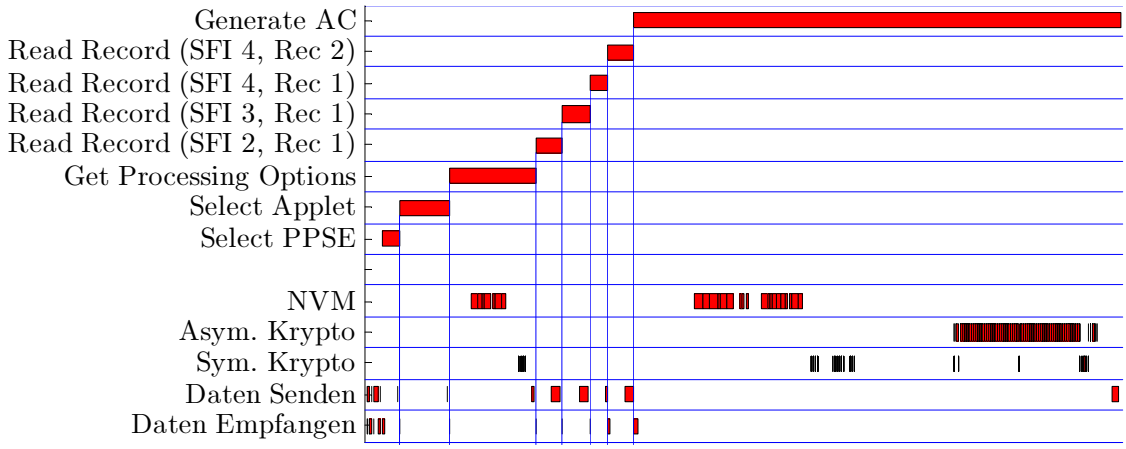

Abbildung 7.2: Zeitlicher Verlauf einer EMV kompatiblen Bankanwendung.

In Abbildung 7.2 ist ein zeitlicher Verlauf einer EMV kompatiblen Bankanwendung dargestellt. Wie auch bei der Passanwendung sind im unteren Bereich die Aktivitäten der einzelnen Hardwaremodule und im oberen Bereich die Befehlsabfolge dargestellt. Bei diesem Beispiel wird die Datenübertragung nicht verschlüsselt und daher kommt auch bei den meisten Befehlen keine symmetrische Kryptographie zum Einsatz. Bei dieser Anwendung ist ersichtlich, dass nur bei zwei Befehlen Daten in den nichtflüchtigen Speicher geschrieben

werden und Kryptographie verwendet wird und zwar beim Befehl *Get Processing Options* und beim Befehl *Generate AC*. Alle anderen Befehle greifen nur lesend auf den Speicher zu und benötigen keine Kryptographie.

Vergleicht man nun die beiden Anwendungen rein optisch ist ersichtlich das bei der Passanwendung viel mehr Kommunikation erfolgt als bei der Bankanwendung. Das ist darauf zurückzuführen das bei der Passanwendung viel mehr Daten transferiert werden müssen. Es werden aber bei beiden Anwendungen alle Hardwaremodule benötigt. Auf den Bedarf der einzelnen Hardwaremodule wird im nächsten Kapitel näher eingegangen.

7.2 Zeitbedarf der einzelnen Hardwaremodule

In Abschnitt 7.1 wurde die zeitliche Abfolge der Hardwaremodule dargestellt. Aus diesen Darstellungen ist auch ersichtliche welche Module aktiv sind. In diesem Abschnitt werden nun die Ergebnisse der Analyse des Zeitbedarfs der einzelnen Hardwaremodule im Verhältnis zur gesamten Ausführungszeit dargestellt. Die Ergebnisse werden für die verschiedenen Anwendungen dargestellt. Die Hardwaremodule werden nun kurz beschrieben und wie sich die Zeiten der einzelnen Module ergeben.

- Non Volatile Memory:

  Der Anteil an der Gesamtzeit des NVM setzt sich aus den Zeiten die zum Löschen und Programmieren des nichtflüchtigen Speichers aufgewendet werden müssen zusammen. Die Zeiten die reine Lesezugriffe benötigen, werden dabei nicht berücksichtigt.

- Symmetrische Kryptographie:

  Bei den Zeiten die für die symmetrische Kryptographie benötigt werden, werden nur die reinen Berechnungszeiten berücksichtigt. Die Zeiten für das Laden der Schlüssel und der Daten, sowie für die Konfiguration der Kryptographiekoprozessoren sind dabei nicht inkludiert.

- Asymmetrische Kryptographie:

  Berechnungszeiten für asymmetrische Kryptographie wie zum Beispiel Elliptic Curve (EC) Kryptographie. Diese kommt zum Beispiel beim Überprüfen von Zertifikaten oder beim Erstellen von Signaturen zum Einsatz. Auch hier werden nur die reinen Berechnungszeiten der Kryptographie berücksichtigt.

- Kommunikation:

  Anteil der Kommunikation zwischen der Karte und dem Terminal. Dies sind die Zeiten die für die Übertragung der Daten an die Karte und für die Rückantwort benötigt werden. Die Kommunikation mit der Karte erfolgte bei allen Anwendungen mit der gleichen Datenrate.

- CPU:

  Der Zeitbedarf für CPU Operationen während sonst kein Hardwaremodul aktiv ist. Zu diesen Zeiten zählen auch Datenzugriffe auf die Hardwaremodule. Dies können zum Beispiel Zeiten für das Laden der Daten für die Kryptographiekoprozessoren

oder das Lesen von Daten aus dem nichtflüchtigen Speicher (wie es zum Beispiel beim Auslesen des Passbildes der Fall ist) sein. Nicht zu dieser Zeit zählen die Zeiten bei denen auf andere Module gewartet wird, wie zum Beispiel das Warten bis ein EEPROM Programmiervorgang abgeschlossen ist.

7.2.1 Zeitbedarf bei Passanwendungen

In Tabelle 7.1 ist der zeitliche Anteil der einzelnen Hardwaremodule an der Gesamtausführungszeit von Passanwendungen dargestellt. Es werden zwei verschiedene Passanwendungen analysiert. Dies ist ein Reisepass der mit BAC geschützt ist und ein Reisepass mit EAC Schutz. Auf letzteren sind auch die Fingerabdrücke des Inhabers gespeichert. Des Weiteren wird zwischen der Personalisierung der Anwendung und dem Auslesen des Reisepasses unterschieden. Aus der Tabelle ist sofort ersichtlich dass der Anteil der CPU für alle Anwendungsfälle den größten Teil der Zeit ausmacht. Auf die genauen Werte wird später noch eingegangen.

	Personalisierung BAC	Auslesen BAC	Personalisierung EAC	Auslesen EAC
NVM	28,5%	0,0%	38,5%	4,5%
Sym. Krypto	1,9%	3,3%	1,6%	2,4%
Asym. Krypto	0,0%	0,0%	1,6%	13,7%
Kommunikation	11,1%	18,3%	8,8%	12,4%
CPU	58,5%	78,4%	49,5%	67,0%

Tabelle 7.1: Zeitverbrauch der einzelnen Hardwaremodule bei Passanwendungen.

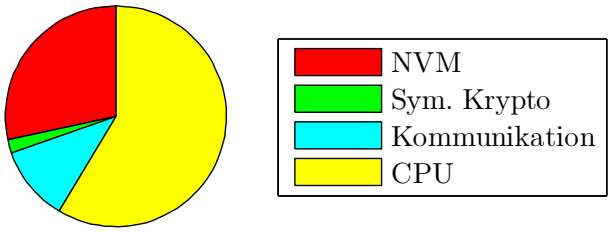

Abbildung 7.3: Zeitverbrauch der einzelnen Hardwaremodule beim Personalisieren der Passanwendung mit BAC.

In Abbildung 7.3 wird der zeitliche Anteil der einzelnen Module der Hardware bei der Personalisierung einer mit BAC geschützten Passanwendung dargestellt. Die Werte sind auch aus in Tabelle 7.1 ersichtlich. Da die Daten und das Bild des Inhabers bei der Personalisierung in den Speicher geschrieben werden müssen, erfolgen Schreiboperationen auf den nichtflüchtigen Speicher. Der Anteil dieser Schreiboperationen beträgt 28,5% der Zeit. Für die Übertragung der Daten vom Terminal zur Karte und die Bestätigungen der Karte, dass die Operationen erfolgreich waren werden an Kommunikationszeit 11,1% der Ausführungszeit benötigt. Da die Übertragung auch kryptographisch mit einem symmetrischen Kryptographieverfahren gesichert ist, werden für diese Ver- und Entschlüsselungsoperationen 1,9% der Zeit benötigt. Der restliche Anteil der Zeit entfällt auf CPU

Operationen wie zum Beispiel Datentransferoperationen. Die CPU Operationen benötigen mit 58,5% den größten Teil der Zeit bei der Personalisierung.

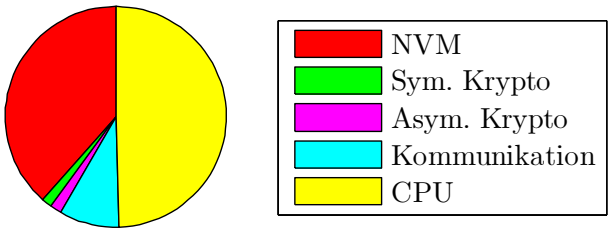

Abbildung 7.4: Zeitverbrauch der einzelnen Hardwaremodule beim Personalisieren der Passanwendung mit EAC.

Abbildung 7.4 stellt die zeitlichen Anteile der einzelnen Hardwarekomponenten bei der Personalisierung einer mit EAC geschützten Passanwendung dar. Bei einer EAC Anwendung werden mehr Daten in den nichtflüchtigen Speicher geschrieben. Zu den gleichen Daten wie bei einem BAC Reisepass kommen noch der Fingerabdruck und das in der Karte generierte Schlüsselmaterial hinzu. Der Anteil der Zeit der für das Schreiben des nichtflüchtigen Speichers benötigt wird ist mit 38,5% etwas größer als bei einem BAC Reisepass. Die Zeitanteile der Kommunikation, symmetrischen Verschlüsselung und der CPU Operationen gehen im Vergleich zum Schreiben des Speichers zurück. Das Verhältnis dieser drei Zeiten zueinander ist aber ähnlich wie bei der Personalisierung eines BAC Reisepasses. Obwohl das Schreiben des nichtflüchtigen Speichers einen größeren Zeitanteil benötigt als bei einem BAC Reisepass, benötigen die CPU Operationen mit 49,5% dennoch den größten Anteil der Personalisierungszeit. Bei der Personalisierung eines EAC Reisepasses wird auch die asymmetrische Kryptographie benötigt. Mit einem Anteil von 1,6% der Zeit werden hier die geheimen Schlüsseldaten der Karte generiert.

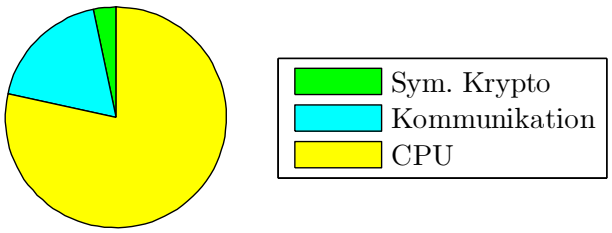

Abbildung 7.5: Zeitverbrauch der einzelnen Hardwaremodule beim Auslesen der Passanwendung mit BAC.

Die zeitliche Aufteilung der einzelnen Hardwaremodule beim Auslesen eines mit BAC geschützten Reisepasses ist in Abbildung 7.5 dargestellt.

Beim Auslesen eines Reisepasses werden die Schritte durchgeführt, wie sie bei der normalen Überprüfung eines Passes durchgeführt werden. Der größte Anteil an der Zeit wird mit 78,4% in der CPU verbraucht. Diese Zeit setzt ich unter anderem aus der Kommandoanalyse, dem Laden der Daten aus dem Speicher und dem Laden der Koprozessoren für die Verschlüsselung zusammen. Die restliche Zeit benötigt die Kommunikation zwischen Terminal und Pass mit 18,3% der Auslesezeit und die Sicherung dieser Daten durch symmetrische Kryptographie (3,3%).

Beim Auslesen werden keine Daten im nichtflüchtigen Speicher geändert und es kommt auch keine asymmetrische Kryptographie zum Einsatz. Daher benötigen diese beiden Hardwaremodule auch keine Zeit.

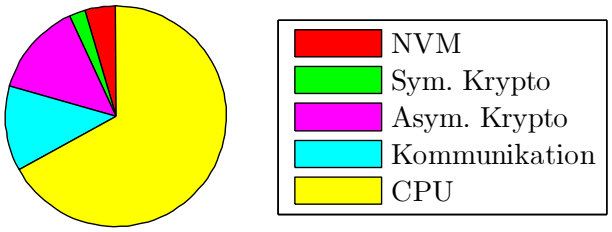

Abbildung 7.6: Zeitverbrauch der einzelnen Hardwaremodule beim Auslesen der Passanwendung mit EAC.

Das Auslesen eines Reisepasses der mit EAC geschützt ist hat eine zeitliche Aufteilung wie in Abbildung 7.6 dargestellt. Im Vergleich zu einem Pass mit BAC verkleinert sich der Anteil von CPU, Kommunikation und symmetrischer Kryptographie. Das Verhältnis der drei Zeiten der Hardwaremodule zueinander bleibt aber annähernd gleich. Der relative Rückgang der Zeit ist darauf zurückzuführen das auch andere Hardwaremodule aktiv sind. Dies sind in diesem Fall die asymmetrische Kryptographie und das Schreiben des nichtflüchtigen Speichers.

Der Anteil der asymmetrischen Kryptographie mit 13,7% ist nicht zu vernachlässigen und ist auf den Diffie Hellman (DH) Schlüsselaustausch beim der Chip Authentication und die Verifikation der Terminal Zertifikate zurückzuführen. Für Details siehe Abschnitt 3.3.3. Beim DH Schlüsselaustausch muss mit Hilfe asymmetrischer Kryptographie der gemeinsame Schlüssel berechnet werden. Beim Terminal Authentication muss für die Verifikation der Zertifikate auch asymmetrische Kryptographie eingesetzt werden. Im letzten Schritt der Terminal Authentication muss die Signatur die vom Terminal generiert wurde mit Hilfe asymmetrischer Kryptographie verifiziert werden.

Das Schreiben auf den nichtflüchtigen Speicher, welches sich mit 4,5% zu Buche schlägt hat zwei Ursachen. Zum einem ist das Überprüfen der Zertifikate bei der Terminal Authentication gegen Brute Force Attacken geschützt und daher wird ein Zugriffszähler mitgespeichert. Ein weiterer Grund für ein Schreiben auf den nichtflüchtigen Speicher ist, dass der Reisepass intern ein geschätztes Datum gespeichert hat und dieses Datum, anhand der Zertifikate die verifiziert werden, upgedated wird.

7.2.2 Zeitbedarf bei Bankanwendungen

Die Zeitaufteilung der einzelnen Hardwaremodule bei den drei untersuchten Bankanwendungen ist in Tabelle 7.2 dargestellt. Es wird jeweils der Zeitbedarf der Hardwaremodule bei der Personalisierung und bei einer Referenztransaktion angegeben. Die drei Bankanwendungen sind eine EMV kompatible Anwendung, und die Open Source Anwendungen Java Purse (JP) und Java Purse Crypto (JPC).

Generell zeigt sich für Bankanwendungen das der größte Zeitanteil bei der Personalisierung für das Schreiben des nichtflüchtigen Speichers benötigt wird. Der zweite große Anteil beim Personalisieren ist der Zeitbedarf der CPU. Die anderen Module werden bei der Personalisierung nur zu einem geringen Teil beansprucht.

Bei einer Referenztransaktion wird auch ein großer Teil der Zeit für das Schreiben des nichtflüchtigen Speichers und für die CPU verwendet. Geringe Anteile an Kommunikation und Kryptographie vervollständigen den Zeitbedarf. In Folge wird auf die einzelnen Anwendungen noch genauer eingegangen.

	Perso. EMV	Trans. EMV	Perso. JP	Trans. JP	Perso. JPC	Trans. JPC
NVM	48,3%	15,5%	66,3%	54,2%	53,5%	41,9%
Sym. Krypto	0,5%	0,5%	0,1%	0,1%	0,8%	0,9%
Asym. Krypto	4,3%	16,2%	0,0%	0,0%	0,0%	0,0%
Kommunikation	2,3%	7,5%	2,8%	6,6%	1,6%	3,9%
CPU	44,6%	60,3%	30,8%	39,1%	44,1%	53,3%

Tabelle 7.2: Zeitverbrauch der einzelnen Hardwaremodule bei Bankanwendungen.

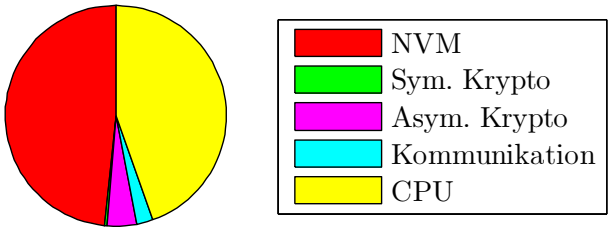

Abbildung 7.7: Zeitverbrauch der einzelnen Hardwaremodule beim Personalisieren der EMV kompatiblen Bankanwendung.

Bei der Personalisierung der ersten Bankanwendung die EMV kompatibel ist, werden wie auch in Abbildung 7.7 ersichtlich, 48,3% der Zeit für das Schreiben des nichtflüchtigen Speichers verwendet. Diese Zeit wird benötigt um die Daten des Inhabers und Schlüsselmaterial auf der Karte zu speichern. Die asymmetrische Kryptographie die zur Generierung des Schlüsselmaterials zum Einsatz kommt benötigt 4,3% der Gesamtzeit. Da auch Daten gesichert abgelegt werden, müssen diese verschlüsselt werden. Dies erklärt denn Zeitanteil von 0,5% der auf die symmetrische Kryptographie entfällt. Um die Daten vom Terminal zur Karte zu übertragen und auch die Rückantworten zu übertragen werden 2,3% der Personalisierungszeit benötigt. Die restlichen 44,6% der Personalisierungszeit entfallen auf CPU Operationen bei denen die Befehle analysiert und abgearbeitet werden.

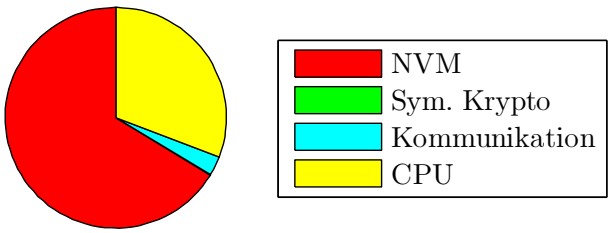

Abbildung 7.8: Zeitverbrauch der einzelnen Hardwaremodule beim Personalisieren der Java Purse Bankanwendung.

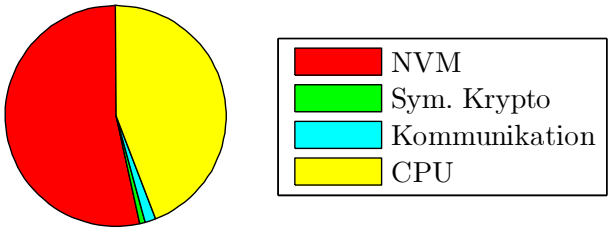

Abbildung 7.9: Zeitverbrauch der einzelnen Hardwaremodule beim Personalisieren der Java Purse Crypto Bankanwendung.

Die Zeitbedarf bei der Personalisierung der Java Purse Bankanwendung (Abbildung 7.8) teilt sich hauptsächlich in CPU Zeit und Zeit für das Schreiben des nichtflüchtigen Speichers auf. Das Schreiben stellt mit 66,3% denn größten Anteil. Hier werden, wie auch bei der EMV kompatiblen Bankanwendung, die Daten des Inhabers und Schlüsselmaterial abgespeichert. Auf die einzelnen CPU Operationen entfallen 30,8%der Zeit welche sich aus der Verarbeitung der einzelnen Kommandos inklusive des Ladens der Koprozessoren ergeben. Für die Übertragung der Daten zwischen Terminal und Karte werden 2,8% der Personalisierungszeit benötigt. Die restlichen 0,1% der Zeit entfallen auf symmetrische Kryptographieoperationen. Diese kommen zum gesicherten Speichern von wichtigen Daten zum Einsatz.

Die dritte Bankanwendung die betrachtet wurde ist die Open Source Anwendung Java Purse Crypto. Der Zeitbedarf der einzelnen Hardwaremodule bei der Personalisierung dieser Anwendung ist in Abbildung 7.9 dargestellt. Bei dieser Implementierung werden zum Speichern der Daten des Inhabers und der Schlüssel im nichtflüchtigen Speicher 53,5% der Personalisierungszeit benötigt. Die Operationen der CPU schlagen sich mit 44,1% zu Buche. Bei der Übertragung der Daten werden 1,6% der gesamten Personalisierungszeit benötigt. Auch bei dieser Anwendung werden die Daten gesichert abgespeichert, wofür die symmetrische Kryptographie benötigt wird. Diese kryptographischen Operationen nehmen die restlichen 0,8% der Zeit für sich in Anspruch.

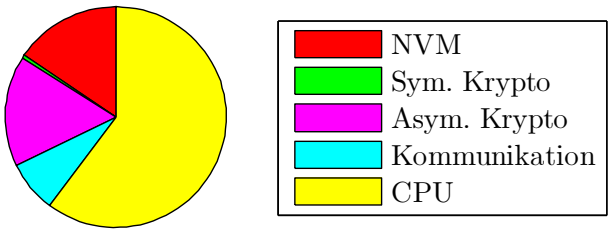

Abbildung 7.10: Zeitverbrauch der einzelnen Hardwaremodule bei einer Transaktion der EMV kompatiblen Bankanwendung.

Die Zeitliche Aufteilung der verschiedenen Hardwaremodule bei einer Referenztransaktion der ersten Bankanwendung, die EMV kompatibel ist, ist in Abbildung 7.10 dargestellt. Bei dieser Bankanwendung wird für eine Referenztransaktion mit 60.3% die meiste Zeit in der CPU verbraucht. Diese Zeit setzt sich unter anderem aus der Bearbeitung der Befehle, die vom Terminal kommen, dem Laden der Daten aus dem nichtflüchtigen Speicher und dem Laden der Schlüssel und Daten für die kryptographischen Koprozessoren zu-

sammen. Mit 16,2% benötigt die asymmetrische Kryptographie den zweitgrößten Anteil, da bei dieser Bankanwendung eine digitale Signatur über die Transaktion berechnet wird (Details siehe Abschnitt 3.4.1). 15,5% der Transaktionszeit werden für die Speicherung der Daten im nichtflüchtigen Speicher benötigt. Dies sind zum einen die Daten über die Transaktion und auch Daten, die zum Schutz der Transaktion gespeichert werden. Für die Übertragung der Daten zwischen Terminal und Karte werden 7,5% der Zeit benötigt. Mit 0,5% benötigt die symmetrische Kryptographie die zur Sicherung der gespeicherten Daten eingesetzt wird, den geringsten Anteil.

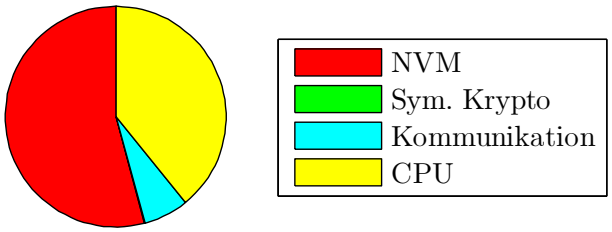

Abbildung 7.11: Zeitverbrauch der einzelnen Hardwaremodule bei einer Transaktion der Java Purse Bankanwendung.

Für einer Referenztransaktion der Bankanwendung Java Purse ist die zeitliche Aufteilung der Hardwaremodule in Abbildung 7.11 dargestellt. Bei dieser Banktransaktion wird der größte Teil der Zeit durch das Schreiben des nichtflüchtigen Speichers verbraucht (54,2%). Die Zugriffe auf den Speicher dienen zur Absicherung der Transaktion und zum Abspeichern des neuen Guthabens. Die Operationen der CPU benötigen mit 39,1% den zweitgrößten Anteil. Auch bei dieser Transaktion wird mit der CPU das Kommando abgearbeitet und die Koprozessoren mit Daten versorgt. Für die Kommunikation die zwischen Terminal und Karte stattfindet werden 6,6% der Zeit benötigt. Die restlichen 0,1% der Zeit die für symmetrische Kryptographie benötigt werden, sind auf die gesicherte Speicherung von Daten zurückzuführen.

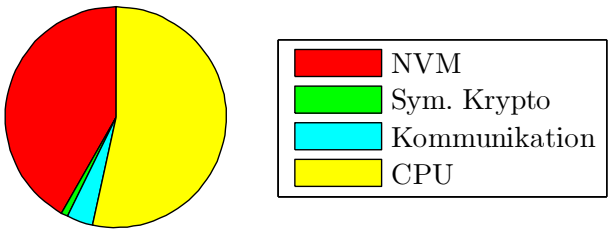

Abbildung 7.12: Zeitverbrauch der einzelnen Hardwaremodule bei einer Transaktion der Java Purse Crypto Bankanwendung.

Bei einer Referenztransaktion der Bankanwendung Java Purse Crypto ist die zeitliche Aufteilung der Hardwaremodule wie in Abbildung 7.12 dargestellt. Bei dieser Bankanwendung wird der größte Teil der Zeit für CPU Operationen verbraucht (53,3%). Wie auch bei den vorigen Anwendungen setzten sich die CPU Operationen aus der Abarbeitung der Kommandos und der Versorgung der verschiedenen Hardwaremodule mit Daten zusammen. Aber auch wie bei den anderen Bankanwendungen wird ein hoher Anteil der Zeit

(41,9%) für das Schreiben des nichtflüchtigen Speichers verwendet. Auch hier wird das neue Guthaben gespeichert und die Transaktion zusätzlich abgesichert. Auf die Kommunikation entfallen 3,9% der gesamten Transaktionszeit und für das geschützte Speichern der Daten benötigt die symmetrische Kryptographie 0,9% der Zeit.

Wenn nun alle Anwendungen betrachtet werden ist keine klare Unterscheidung zwischen den beiden Anwendungsdomains möglich. Bei beiden gibt es eine Anwendung für die asymmetrische Kryptographie benötigt aber die anderen Anwendungen in der gleichen Anwendungsdomain benötigen die asymmetrische Kryptographie nicht.

7.3 CPU Opcode Aufteilung

Ein wesentlicher Faktor von Programmen ist, welche Operationen die CPU ausführt. Daran können sich manche Programme wesentlich unterscheiden. Zum Beispiel wird ein Programm bei dem viele Daten kopiert werden müssen, mehr Datenoperationen haben als ein Programm bei dem Daten berechnet werden.

In diesem Abschnitt wird nun die Aufteilung der CPU Operationen für die einzelnen Smart Card Anwendungen bei einer Referenztransaktion dargestellt. Die Operationen der CPU werden auch für die verschiedenen Aktivitäten der Hardwaremodule getrennt betrachtet. Ein Beispiel dafür ist, welche Operationen die CPU ausführt, während ein kryptographischer Koprozessor aktiv ist.

7.3.1 Opcode Aufteilung bei Passanwendungen

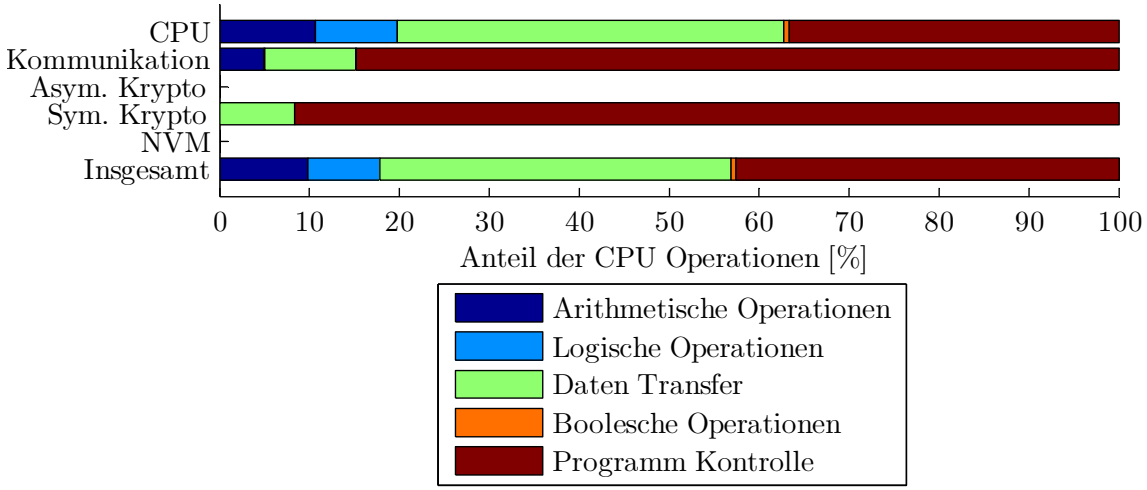

Abbildung 7.13: CPU Instruktionsmix bei Aktivität der einzelnen Hardwaremodule der Passanwendung mit BAC.

In Abbildung 7.13 ist die Aufteilung der unterschiedlichen CPU Operationen während die verschiedenen Hardwaremodule aktiv sind dargestellt. In dieser Abbildung wird die Aufteilung während des Auslesens einer Passanwendung, die mit BAC gesichert ist, dargestellt. Da es beim Auslesen einer BAC Anwendung keine Schreiboperationen auf den

nichtflüchtigen Speicher und keine asymmetrischen Kryptographieoperationen gibt, sind für diese Module keine Daten vorhanden.

Während kein anderes Modul aktiv ist werden von der CPU größtenteils Datentransferoperationen (>40%) ausgeführt. Etwas unter 40% der Operationen benötigt die Steuerung des Programmablaufs. Rund 10% der Operationen entfallen auf Arithmetische Operationen und etwas unter 10% auf Logische Operationen. Mit kleiner als 1% ist der Anteil der Booleschen Operationen verschwindend klein.

Während der kontaktlosen Kommunikation zwischen Terminal und Karte werden größtenteils Operationen zur Ablaufkontrolle und Datentransferoperationen ausgeführt. Dies kann auf die Abfrage ob die aktuellen Daten schon gesendet sind und das Nachladen der nächsten Daten zurückzuführen sein. Die Arithmetischen Operationen können durch die Zähler für die aktuelle Position beim Senden verursacht werden.

Während die symmetrische Kryptographie aktiv ist, werden nur Daten Transfer Befehle und Befehle zur Kontrolle des Befehlsflusses ausgeführt. Daraus lässt sich schließen, dass während eine symmetrische Kryptographieoperation ausgeführt wird, eine Schleife so lange durchlaufen wird, bis das Hardwaremodul signalisiert, dass es fertig ist.

Wenn über den gesamten Zeitraum einer beispielhaften Reisepasstransaktion die CPU Aktivitäten betrachtet werden, ergibt sich ein ähnliches Bild wie nur bei Betrachtung der reinen CPU Operationen. Der Anteil der Befehle zur Steuerung des Programmflusses steigt etwas an und die Anteile der anderen Operationen sinken dementsprechend.

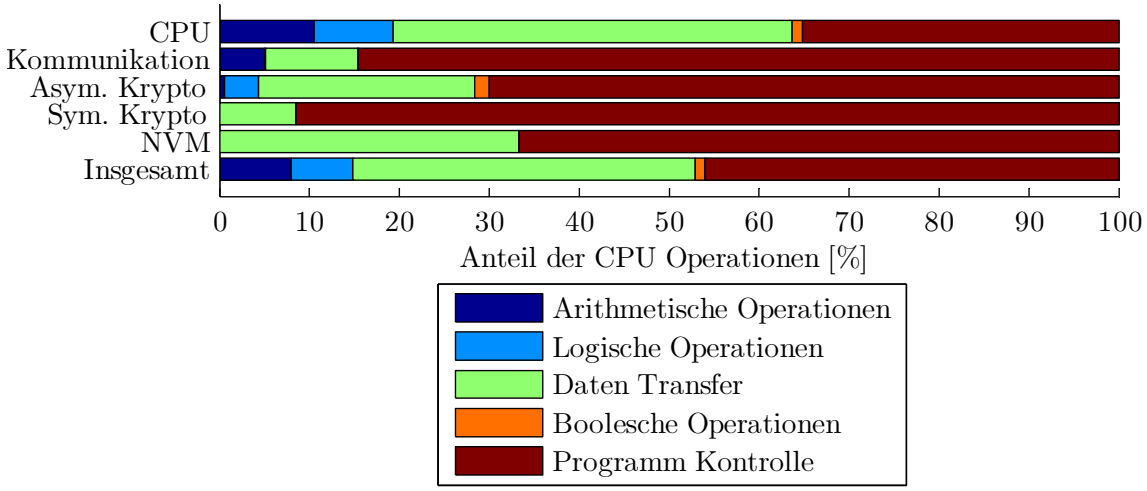

Abbildung 7.14: CPU Instruktionsmix bei Aktivität der einzelnen Hardwaremodule der Passanwendung mit EAC.

Bei einer Passanwendung, die über einen Zugriffsschutz mit EAC verfügt, sieht der Instruktionsmix der CPU während der Aktivitäten der einzelnen Module wie in Abbildung 7.14 dargestellt aus.

Wenn nur die CPU aktiv ist, während asymmetrische Kryptographie berechnet wird und während der Kommunikation ist die Aktivität der CPU annähernd gleich aufgeteilt, wie bei einer mit BAC geschützten Passanwendung (siehe dazu Abbildung 7.13).

Während des Schreibens des nichtflüchtigen Speichers werden von der CPU ca. zu einem Drittel Datentransferoperationen ausgeführt und die restlichen zwei Drittel dienen

der Kontrolle des Programmablaufs. Gleich wie bei den Operationen des Koprozessors für die symmetrische Kryptographie sind auch hier diese CPU Operationen auf das Warten bis der Vorgang abgeschlossen ist zurückzuführen.

Bei asymmetrischen Kryptographischen Operationen sind ca. 70% der CPU Operationen für den Programmfluss zuständig, 25% sind Datentransferoperationen und die restliche rund 5% teilen sich auf Arithmetische, Logische und Boolesche CPU Operationen auf. Anhand dieser Aufteilung kann auch bei der asymmetrischen Kryptographie darauf geschlossen werden, dass die meiste Zeit nur auf die Fertigstellung der Operation gewartet wird.

7.3.2 Opcode Aufteilung bei Bankanwendungen

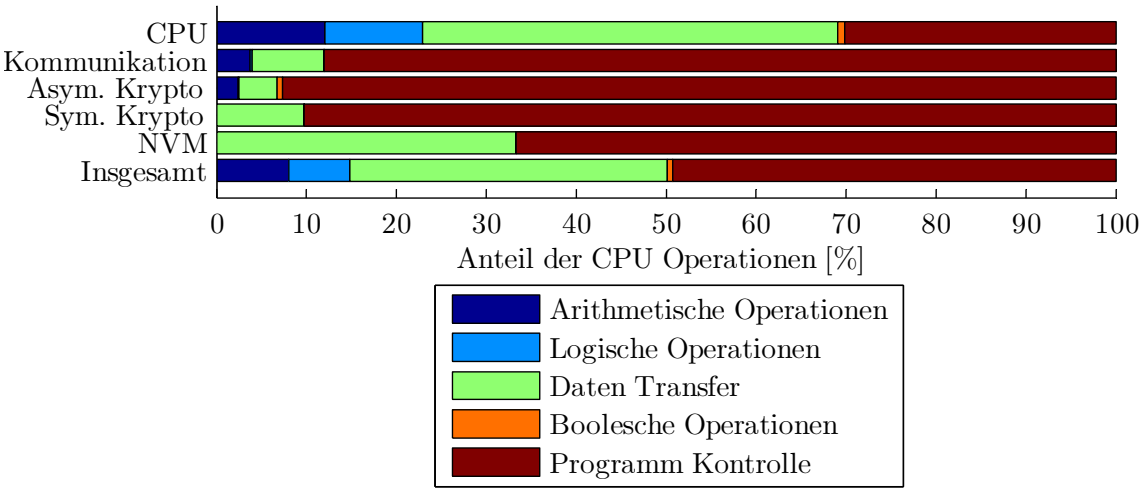

Abbildung 7.15: CPU Instruktionsmix bei Aktivität der einzelnen Hardwaremodule der EMV kompatiblen Bankanwendung.

Abbildung 7.15 stellt die CPU Opcode Aufteilung bei einer Referenztransaktion der ersten Bankanwendung (EMV kompatibel) dar. Wenn man diese Abbildung mit Abbildung 7.14 vergleicht stellt man fest, dass sie sich sehr ähnlich sind. Einen größeren Unterschied findet man bei der asymmetrischen Kryptographie und einen zweiten bei der Kommunikation. Der Unterschied bei der asymmetrischen Kryptographie ist auf die verschiedenen kryptographischen Operationen und die verschiedenen Schlüssellängen, die bei den zwei verschiedenen Anwendungsfällen benötigt werden, zurückzuführen. Bei der Kommunikation erklärt sich der Unterschied aus der unterschiedlichen Übertragungsgröße. Während bei einer Passanwendung große Daten übertragen werden, sind bei Bankanwendungen die Datenpakete kürzer.

Bei der zweiten (Abbildung 7.16) und der dritten (Abbildung 7.17) Bankanwendung ist die Aufteilung der CPU Operationen annähernd gleich. Die Aufteilung der CPU Operationen bei der Programmierung des nichtflüchtigen Speichers und bei der symmetrischen Kryptographie entsprechen denen der ersten Bankanwendung. Der Anteil der Befehle zur Programmflusskontrolle während der Kommunikation ist bei diesen beiden Bankanwendungen etwas höher als bei der EMV kompatiblen Bankanwendung, was mit verschiede-

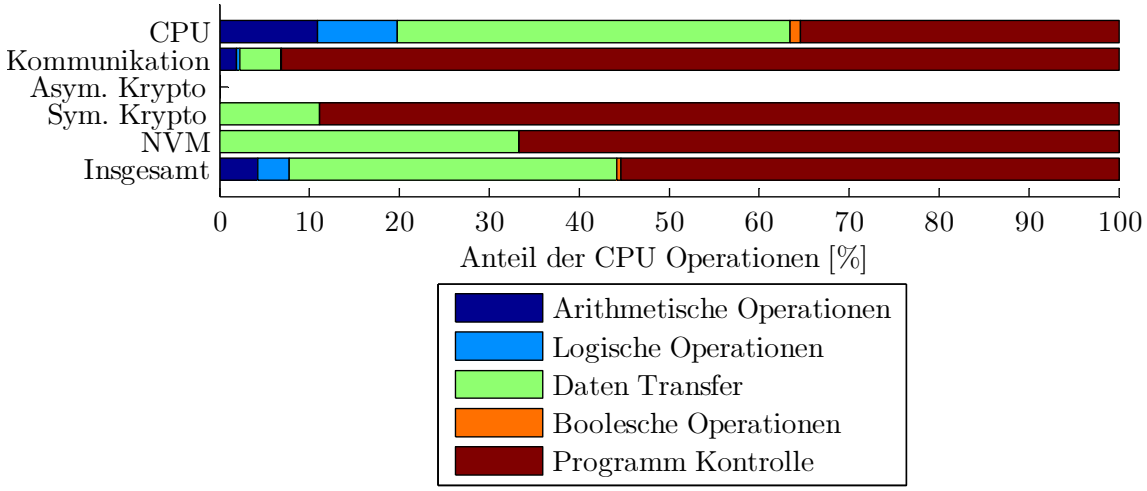

Abbildung 7.16: CPU Instruktionsmix bei Aktivität der einzelnen Hardwaremodule der Java Purse Bankanwendung.

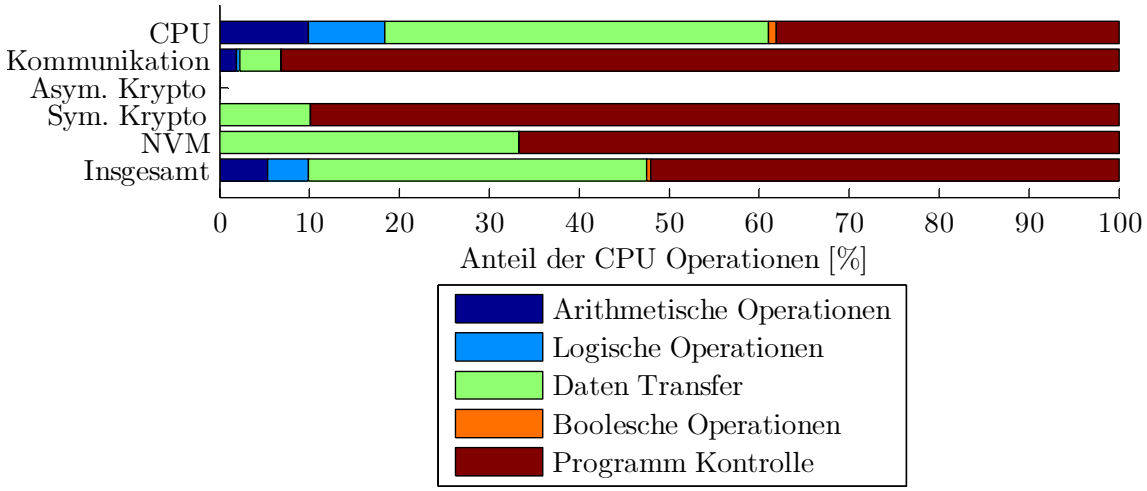

Abbildung 7.17: CPU Instruktionsmix bei Aktivität der einzelnen Hardwaremodule der Java Purse Crypto Bankanwendung.

nen Datenlängen die übertragen werden zusammenhängen kann. Auch während der reinen CPU Operationen ist der Anteil der Kontrollbefehle etwas höher als bei der ersten Bankanwendung wobei aber nicht von einer wesentlichen Verschiebung gesprochen werden kann.

7.3.3 Opcode Vergleich der unterschiedlichen Anwendungen

In Abbildung 7.18 sind verschiedene Anwendungen gegenübergestellt. Es werden jeweils die verschiedenen CPU Operationen während des Ausführens einer Referenztransaktion (bei den Passanwendungen das Auslesen des Passes und bei den Bankanwendungen eine Bezahloperation) dargestellt. Es werden jeweils nur die reinen CPU Operationen dargestellt während kein Hardwaremodul aktiv ist.

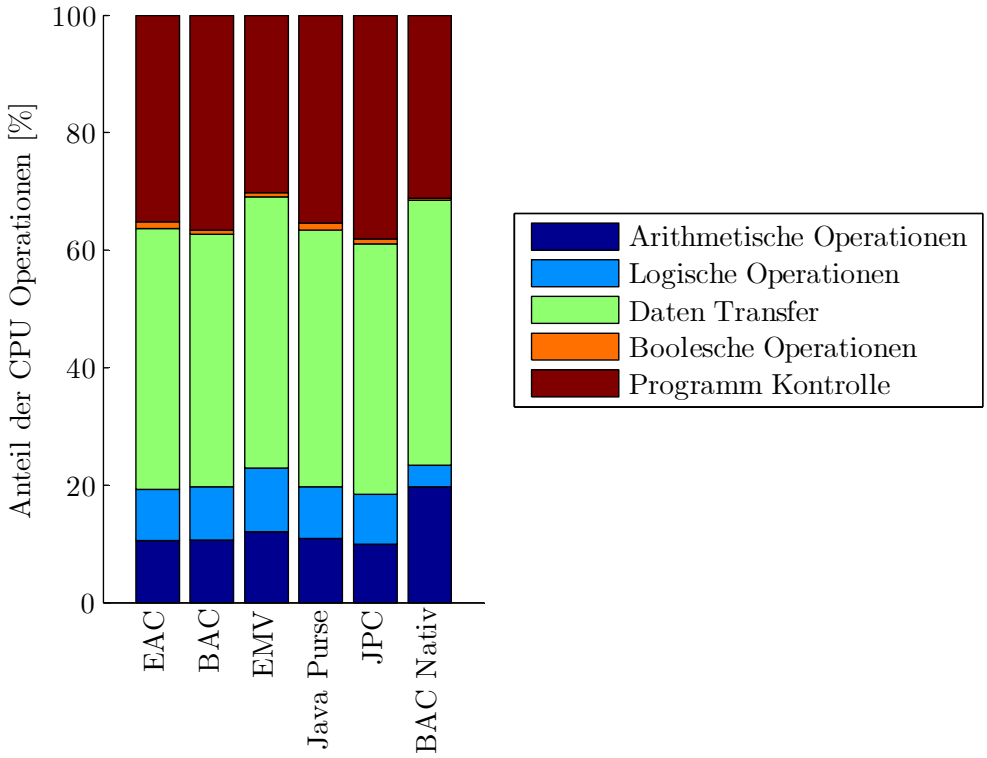

Abbildung 7.18: CPU Instruktionsmix verschiedener Anwendungen bei einer Referenztransaktion.

Diese Darstellung beinhaltet zusätzlich zu den bereits ausführlich diskutierten Anwendungen auch eine native Implementierung einer Reisepassanwendung die mit BAC geschützt ist.

Es ist ersichtlich dass die einzelnen Anwendungen ein ähnliches Profil bei der Aufteilung der CPU Opcodes haben. Der größte Anteil an den CPU Operationen machen die Datentransfer-Operationen mit rund 40% aus. Einen weiteren großen Anteil machen die Befehle zur Kontrolle des Programmflusses aus. Deren Anteil ist aber bei allen Anwendungen etwas geringer als die Datentransfer-Operationen. Einen verschwindend geringen Anteil an den CPU Operationen machen die Booleschen-Operationen aus. Die restliche Anteile an den CPU Operationen teilen sich die Logischen- und Arithmetischen-Operationen wobei die Arithmetischen tendenziell einen etwas größeren Anteil haben.

Wird nun im Vergleich die native Implementierung betrachtet sieht die Aufteilung der CPU Operationen etwas anders aus. Zwar sind die Anteile der Datentransfer Operationen, Booleschen Operationen und der Programmkontrolle vergleichbar, aber der Anteil der Logischen Operationen ist im Gegensatz zu den auf Java Card implementierten Anwendungen viel geringer, dafür ist der Anteil der arithmetischen Operationen größer. Dieser Unterschied ist wahrscheinlich auf die unterschiedlichen Implementierungen der Kommandoabarbeitung und die verschiedenen Sicherheitsmechanismen zurückzuführen.

7.4 Stack Tiefe

Der Stack eines Programmes kann unter anderem einen Rückschluss darauf erlauben wie viele Funktionsaufrufe ineinander verschachtelt sind. In diesem Abschnitt werden nun verschiedene Kennzahlen des Stacks für die verschiedenen Anwendungsfälle präsentiert.

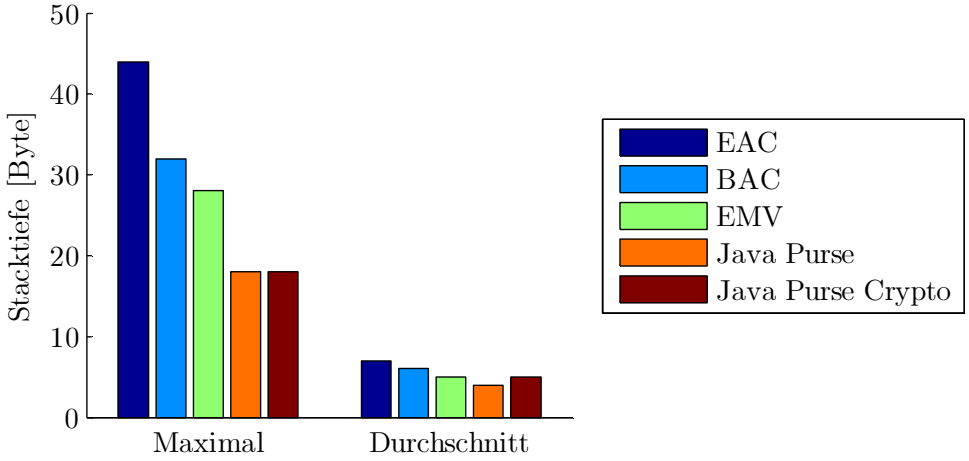

Abbildung 7.19: Stacktiefe verschiedener Anwendungen bei der Personalisierung.

In Abbildung 7.19 ist die Stacktiefe der betrachteten Anwendungen bei der Personalisierung dargestellt. Die linken Säulen repräsentieren die maximale Stacktiefe, die rechten die durchschnittliche Stacktiefe.

Bei der maximalen Stacktiefe haben die beiden Java Purse Anwendungen den gleichen Wert. Die EMV kompatible Bankanwendung benötigt etwas mehr Stack Speicher und bei den beiden Passanwendungen ist der Speicherbedarf noch höher, wobei die EAC Anwendung noch einmal deutlich hervorsticht.

Aus diesen Stacktiefen kann geschlossen werden, dass bei den Bankanwendungen weniger Funktionsaufrufe ineinander verschachtelt sind, hingegen bei den Passanwendungen eine größere Verschachtelung von Funktionen existiert.

Bei der durchschnittlichen Stack Speichertiefe verhält es sich ähnlich wie bei der maximalen Speichertiefe. Die Java Purse Anwendung hat durchschnittlich die geringste Tiefe, gefolgt von der Java Purse Crypto Anwendung und der EMV kompatiblen Bankanwendung. Auch bei der durchschnittlichen Speichertiefe haben die beiden Passanwendungen einen etwas größeren Wert.

Vergleicht man nun die durchschnittliche mit der maximalen Speichertiefe ist ersichtlich, dass der Spitzenwert nur manchmal erreicht wird und den größten Teil der Zeit ein niedriger Stackbedarf und damit eine geringere Verschachtelungstiefe herrscht.

Die Stacktiefe beim Ausführen einer Referenztransaktion bzw. beim Auslesen eines Reisepasses ist in Abbildung 7.20 dargestellt. Die Ergebnisse sind ähnlich wie bei der Personalisierung der Anwendungen. Beim maximalen Stackverbrauch haben die beiden Java Purse Anwendungen wieder den geringsten Wert. Die dritte Bankanwendung (EMV) benötigt einen etwas größeren Stack. Auch hier haben die Reisepassanwendungen den größten Stackbedarf wobei die EAC Anwendung von allen Anwendungen den größten Bedarf hat.

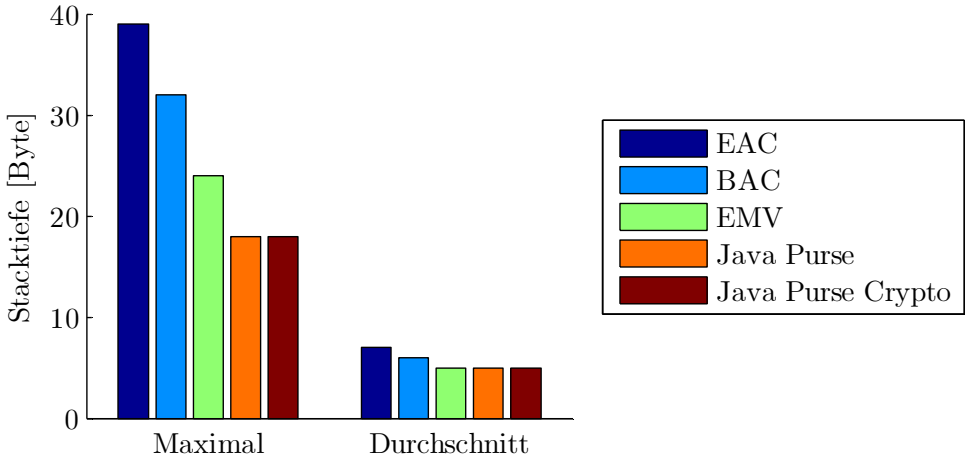

Abbildung 7.20: Stacktiefe verschiedener Anwendungen bei der Referenztransaktion.

Beim durchschnittlichen Stackbedarf liegen alle Bankanwendungen gleich auf und die beiden Passanwendungen benötigen etwas mehr Speicher. Auch aus dieser Abbildung lässt sich ableiten das der maximale Stackbedarf nur manchmal auftritt und das bei den Passanwendungen mehr Funktionen ineinander verschachtelt sind.

7.5 Schleifengröße

Die Anzahl der Befehle die benötigt wird um die gleiche Speicherzelle wieder zu erreichen kann für eine Abschätzung der Größe von Schleifen im Programmfluss herangezogen werden. Anhand solcher Schleifen können auch Abschätzungen über die verschiedenen Cachingstrategien erstellt werden. Bei der Auswertung wurden vier verschiedene Schleifengrößen betrachtet.

- Als kleine Schleifen wurden Schleifen definiert, wenn innerhalb von fünf Speicherzugriffen auf die gleiche Speicherzelle zugegriffen wird. Solche Schleifen können zum Beispiel das Warten in einer Schleife auf eine Hardwarekomponente sein.
- Mittlere Schleifen benötigen zwischen sechs und 64 Zugriffe bis wieder auf die gleiche Speicherzelle zugegriffen wird. Bei solchen Schleifen besteht die Möglichkeit durch einem ausreichend großen Cache die Zugriffe auf den Programmspeicher stark zu reduzieren.
- Bei großen Schleifen werden zwischen 65 und 256 Speicherzugriffe benötigt bis wieder auf die gleiche Speicherstelle zugegriffen wird.
- Nicht als Schleifen werden hier Befehlsabläufe betrachtet die mehr als 256 Zugriffe benötigen, bevor wieder auf die gleiche Stelle zugegriffen wird.

In Abbildung 7.21 ist die Aufteilung in die verschiedenen Schleifengrößen bei der Personalisierung der unterschiedlichen Anwendungen zu sehen. Bei allen Anwendungen sind die meisten Schleifen kleine Schleifen, wo bis eine Adresse wieder aufgerufen wird, maximal fünf Speicherzugriffe in der Zwischenzeit erfolgen. Es werden viele kleine Schleifen

durchlaufen. Daraus kann man schließen, dass oft auf Hardwaremodule gewartet wird. Die beiden Passanwendungen haben einen etwas geringeren Anteil an kleinen Schleifen als die Bankanwendungen.

Bei mittleren Schleifen ist das Bild genau umgekehrt. Die beiden Passanwendungen haben einen größeren Anteil an mittleren Schleifen als die Bankanwendungen.

Die großen Schleifen sind bei allen Anwendung prozentuell ungefähr gleich oft in Verwendung. Eine Ausnahme bildet dabei die Java Purse die einen etwas geringeren Anteil aufweist.

Die restlichen Befehle werden nicht in Schleifen verbracht (oder die Schleifen sind sehr groß). Dabei haben die beiden Passanwendungen einen etwas größeren Anteil an Befehlen die nicht in einer Schleife ausgeführt werden, als die Bankanwendungen.

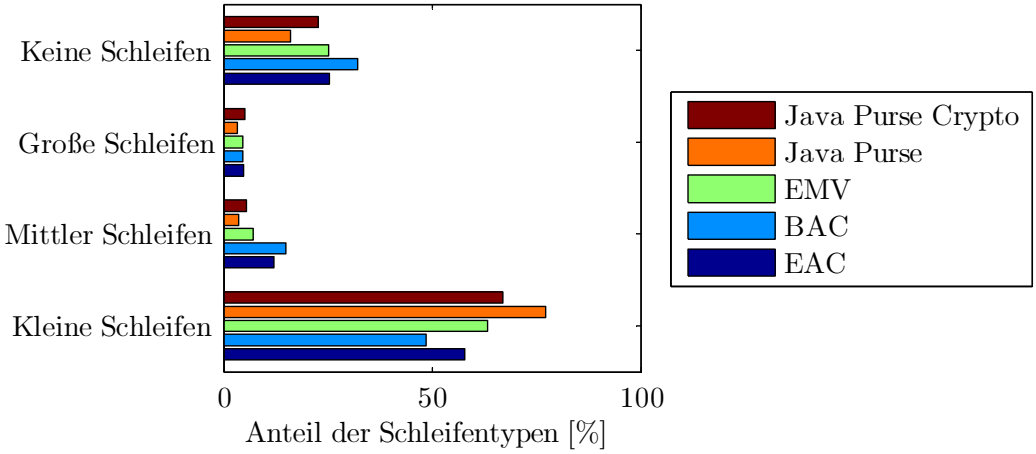

Abbildung 7.21: Schleifengröße verschiedener Anwendungen bei der Personalisierung.

Abbildung 7.22 stellt die unterschiedlichen Schleifengrößen der Anwendungen bei einer Referenztransaktion dar. Im Gegensatz zu der Aufteilung der Schleifengrößen bei der Personalisierung ist bei der Referenztransaktion der Anteil der kleinen Schleifen geringer. Der Anteil der mittelgroßen Schleifen bleibt annähernd gleich und bei den großen Schleifen steigt der Anteil etwas an. Bei allen Anwendungen steigt der Anteil der Befehle, die nicht als Schleifen definiert sind, relativ stark an.

Die unterschiedlichen Anwendungen haben ein annähernd gleiches Verhältnis zueinander wie bei der Personalisierung. Bei den kleinen Schleifen haben die Passanwendungen den geringsten Anteil, wogegen sie bei den mittleren, großen und nicht als Schleifen deklarierten Abfolgen einen größeren Anteil als die Bankanwendungen haben.

7.6 Sprungverhalten

Wie auch bei der Analyse der Schleifen ist auch das Ablaufverhalten des Opcodes interessant. In diesem Abschnitt werden nun die Ergebnisse einer solchen Analyse für die verschiedenen Anwendungen präsentiert.

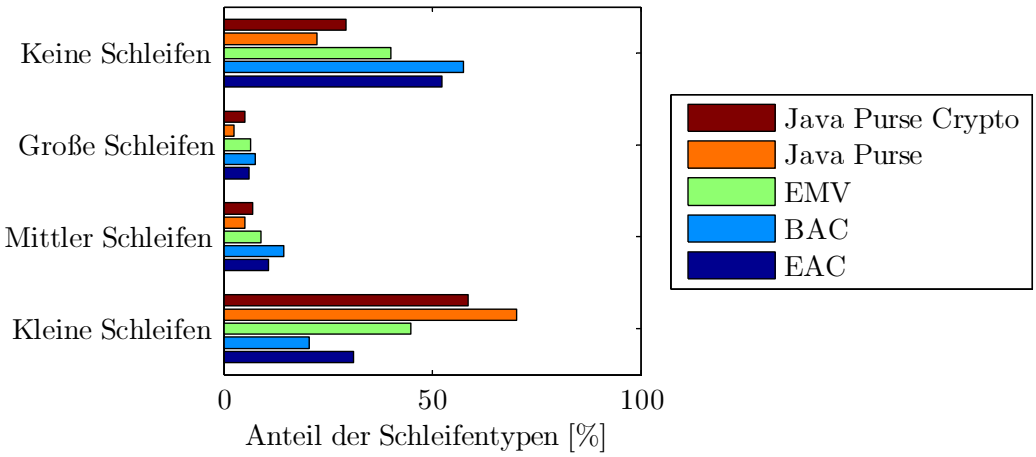

Abbildung 7.22: Schleifengröße verschiedener Anwendungen bei der Referenztransaktion.

7.6.1 Sprungweiten

Bei der ersten Analyse die auf die Sprünge bezogen ist, wurde die Länge der Sprünge ausgewertet. Auch hier wurde der Ablauf in vier verschiedene Kategorien unterteilt.

- Als Linearer Code wird bei dieser Auswertung jener Opcode angesehen, wo der nächste Zugriff auf den Programmspeicher innerhalb von fünf Speicherstellen geschieht. Dabei können schon Sprünge innerhalb der fünf Speicherzellen auftreten. Mit einem geeigneten Cachingsystem könnten diese aber beschleunigt werden.
- Bei kleinen Sprüngen erfolgt der nächste Zugriff innerhalb von sechs bis 64 Speicherzellen.
- Mittlere Sprünge werden gezählt, wenn er nächste Zugriff zwischen 65 und 256 Adressen neben dem aktuellen Zugriff ist.
- Bei großen Sprüngen ist der Adressabstand zwischen dem aktuellen und dem nächsten Opcode größer als 256.

In Abbildung 7.23 ist die Aufteilung in die einzelnen Sprungbereiche für die Personalisierung der unterschiedlichen Anwendungen dargestellt. Aus der Abbildung ist ersichtlich, dass der größte Teil des ausgeführten Codes linearer Code ist und mit geeigneten Cachingstrategien sehr effizient verarbeitet werden kann. Die Aufteilung zwischen kleinen und mittleren Sprüngen ist annähernd gleich und die langen Sprünge sind etwas stärker vertreten. Der Unterschied zwischen den einzelnen Anwendungen ist minimal und daher kein Unterscheidungskriterium.

Abbildung 7.24 stellt die Sprungbereiche bei einer Referenztransaktion dar. Auch bei dieser Abbildung ist über 90% der ausgeführten Opcodes linearer Code und die restlichen Anteile teilen sich die kurzen, mittleren und langen Sprünge wobei die mittleren Sprünge einen etwas kleineren Anteil haben. Auch bei der Referenztransaktion ist kein wesentlicher Unterschied zwischen den verschiedenen Anwendungen ersichtlich.

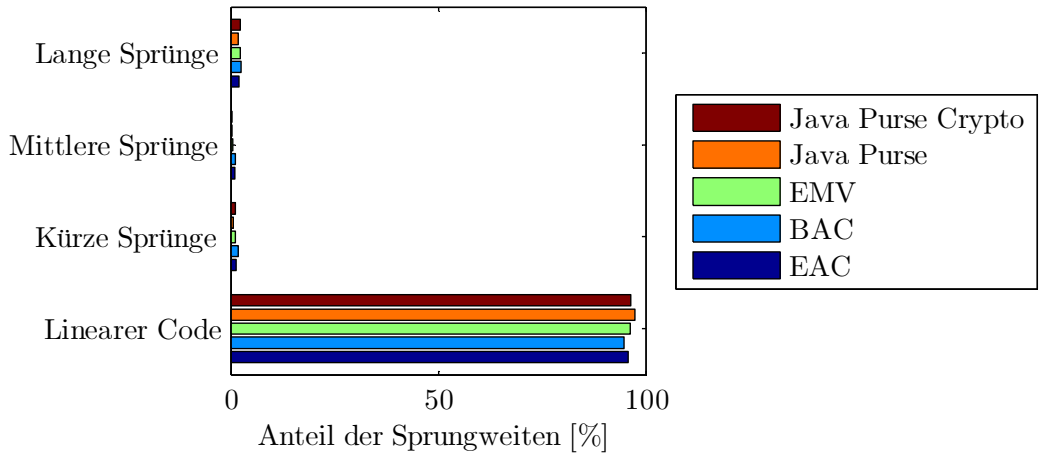

Abbildung 7.23: Sprungweiten verschiedener Anwendungen bei der Personalisierung.

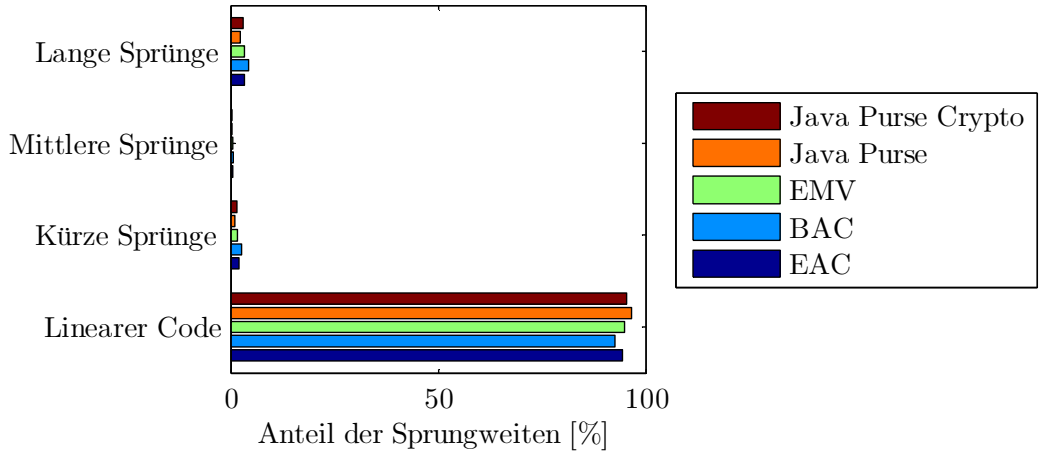

Abbildung 7.24: Sprungweiten verschiedener Anwendungen bei der Referenztransaktion.

7.6.2 Sprung-Adressbereich

Bei der zweiten Analyse der Sprünge wird analysiert in welchen Adressbereiche die Zieladresse liegt. Anhand dieser kann ermittelt werden, welcher Adressierungstype benötigt wird. Es wird zwischen drei verschiedenen Typen unterschieden.

- Short Jumps sind Sprünge innerhalb des 8bit Adressbereichs. Für die Adressierung wird im optimalen Fall nur ein Byte benötigt.
- Jumps sind Sprünge innerhalb des 16bit Adressbereichs. Hier werden für die Adressierung zwei Byte benötigt.
- Bei Long Jumps kann der ganze Adressbereich adressiert werden und es werden je nach Speichergröße drei oder vier Byte zur Adressierung benötigt.

In Abbildung 7.25 sind die Adressbereiche bei Sprüngen der unterschiedlichen Anwendungen bei der Personalisierung dargestellt. Im Schnitt werden 87% der Sprünge im 8bit

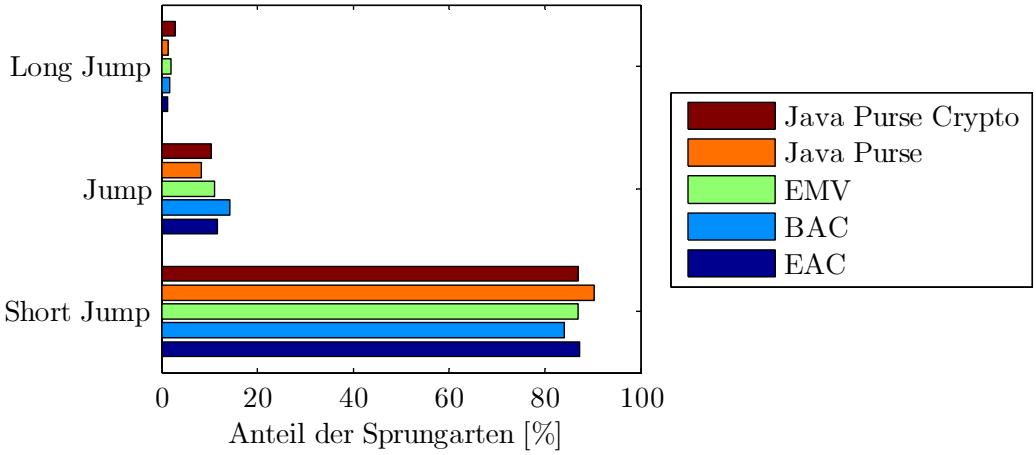

Abbildung 7.25: Sprungbereich verschiedener Anwendungen bei der Personalisierung.

Speicherbereich ausgeführt, wobei die Java Purse einen etwas größeren und die BAC Passanwendung einen etwas geringeren Anteil hat. Die Sprünge im 16bit Segment sind bei den Bankanwendungen etwas geringer als bei den Passanwendungen, aber im Schnitt ca. bei 12%. Die restlichen Sprünge gehen über den ganzen Speicherbereich. Hier haben die Passanwendungen durchschnittlich einen etwas geringeren Anteil als die Bankanwendungen.

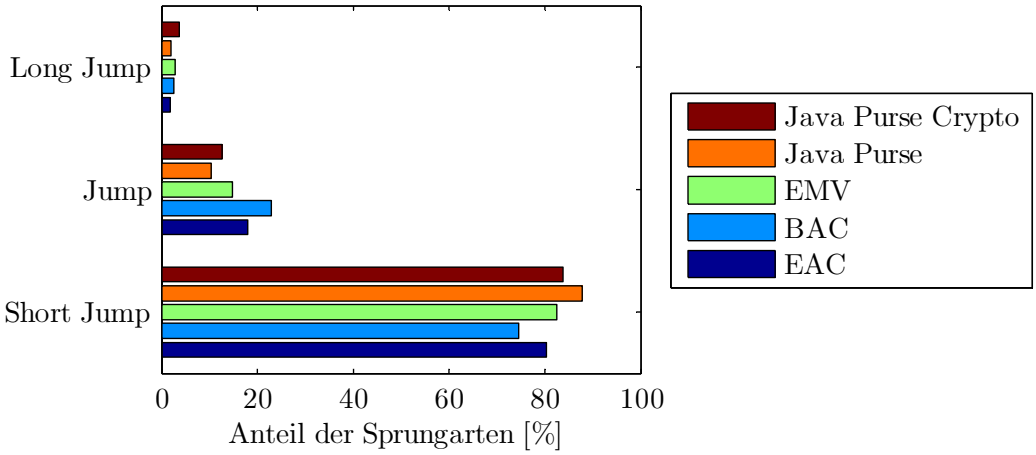

Abbildung 7.26: Sprungbereich verschiedener Anwendungen bei der Referenztransaktion.

Die Aufteilung der Sprünge bei einer Referenztransaktion ist in Abbildung 7.26 dargestellt. Diese Aufteilung ist ähnlich wie bei der Personalisierung nur das der Anteil der Sprünge im 8bit Segment etwas zurück geht und im Gegenzug die Sprünge im 16bit Segment etwas höher sind. Auch das Verhältnis der einzelnen Anwendungen zueinander ist ähnlich. Die Passanwendungen benötigen etwas weniger Sprünge im 8bit Segment als die Bankanwendungen. Dafür benötigen sie mehr Sprünge im 16bit Segment. Auch Sprünge über den gesamten Speicherbereich werden bei Passanwendungen weniger benötigt, als bei Bankanwendungen.

7.7 Zusammenfassung

In diesem Abschnitt wird versucht die verschiedenen Anwendungsklassen anhand der vorher vorgestellten Ergebnisse zu separieren.

Bei der zeitlichen Aufteilung der einzelnen Hardwaremodule zeigt sich, dass bei den Bankanwendungen der Anteil des NVM etwas höher ist als bei den Passanwendungen. Der Anteil der Kommunikation und der symmetrischen Kryptographie ist hingegen bei den Bankanwendungen etwas geringer als bei den Passanwendungen. Bei der asymmetrischen Kryptographie kann hingegen keine klare Aussage getroffen werden. Bei einer Bankanwendung und einer Passanwendung wird asymmetrische Kryptographie benötigt und bei den anderen Anwendungen nicht.

Auch bei der Personalisierung der Anwendungen kann anhand der Anwendungsdomain nicht exakt auf die Anforderungen an die Hardware geschlossen werden. Auch hier wird bei einer Bankanwendung die asymmetrische Kryptographie benötigt und bei den anderen nicht. Bei den Passanwendungen ergibt sich dasselbe Bild.

Anhand der Aufteilung des CPU Opcodes kann auch nicht zwischen den einzelnen Anwendungsklassen unterschieden werden. Die Verteilung der CPU Opcodes während kein Hardwaremodul aktiv ist, ist bei allen Anwendungen annähernd gleich. Wenn die asymmetrische Kryptographie aktiv ist ergibt sich zwar bei der CPU Opcode Verteilung ein unterschiedliches Bild, dies ist aber auf die unterschiedlichen kryptographischen Verfahren zurückzuführen und würde wenn zum Beispiel bei der Passanwendung eine andere Schlüssellänge oder ein anderer Schlüsseltype verwendet würde, wieder anders ausschauen. Nur bei der CPU Verteilung während der Kommunikation kann ein leichter Unterschied zwischen Pass- und Bankanwendungen festgestellt werden. Bei der Personalisierung der Anwendungen kann nur während der Kryptographie ein Unterschied festgestellt werden, der aber wiederum auf die Generierung der unterschiedlichen Schlüssel zurückzuführen ist.

Ein mögliches Unterscheidungskriterium wurde bei der Analyse des Stacks festgestellt. Die beiden Passanwendungen benötigen etwas mehr Stackspeicher als die Bankanwendungen. Beim durchschnittlichen Stackbedarf ist der Unterschied aber nicht so groß wie beim maximalen Stackbedarf.

Bei der Schleifengröße sind bei den Bankanwendungen tendenziell mehr kleine Schleifen und dementsprechend weniger Abfolgen, die nicht als Schleifen gewertet werden, als bei den Passanwendungen.

Beim Sprungverhalten und beim Adressbereich der Sprünge kann nicht zwischen Bank- und Passanwendungen unterscheiden werden.

Kapitel 8

Schlussbemerkung und Ausblick

Mit dieser Arbeit wurde untersucht, ob es möglich ist anhand einer Domain, der eine Smart Card Anwendung zugeordnet werden kann, auf die Anforderungen zu schließen, die an die Smart Card Hardware gestellt werden.

Es wurden für die Arbeit fünf unterschiedliche Anwendungen in zwei Anwendungsdomains betrachtet. Alle Anwendungen sind für Java Card geschrieben und auf der simulierten Smart Card läuft eine Java Virtuell Maschine. Die Anwendungen wurden auf die Karte geladen, installiert und personalisiert. Danach wurde für alle Anwendungen eine Referenztransaktion durchgeführt. Dabei wurden die Aktivitäten der Hardwaremodule, der CPU und die Speicherzugriffe für eine spätere Analyse aufgezeichnet.

Anhand der in Kapitel 5 beschriebenen Auswertung wurden aus den Rohdaten Kennzahlen extrahiert, mit denen die Anwendungen miteinander verglichen worden sind.

Beim Vergleich der Kennzahlen der Anwendungen wurde festgestellt, dass es in manchen Bereichen leichte Unterschiede zwischen den Kennzahlen der Anwendungsdomains gab. Generell sind sich die Kennwerte der unterschiedlichen Anwendungsdomains aber so ähnlich das keine klare Trennung zwischen den Domains möglich ist. Ein weiteres Verhalten das festgestellt wurde ist, dass sich bei manchen Anwendungen auch innerhalb der Anwendungsdomain große Unterschiede ergeben. Ein Beispiel dafür sind die Passanwendungen. Bei diesen Anwendungen benötigt die BAC Version keinen schreibenden Zugriff auf den NVM und keine Unterstützung für die asymmetrische Kryptographie wogegen die EAC Version diese beiden Hardwaremodule benötigt. Aus den hier angeführten Gründen kann keine klare Grenze zwischen den Anwendungsdomains gezogen werden.

Der Grund für dieses Verhalten hat vermutlich zwei Ursachen. Zum einen muss bei der Abarbeitung der Anwendung der Java Bytecode mit der Virtuellen Maschine übersetzt werden. Diese Übersetzung ist zum größten Teil unabhängig von den Anwendungen und daher ist das CPU Profil das sich bei allen Anwendungen ergibt vermutlich das CPU Profil der Virtuellen Maschine.

Ein weiter Grund warum zwischen den einzelnen Anwendungsdomains nicht klar unterschieden werden kann ist, dass auch die Anwendungen innerhalb einer Domain große Unterschiede aufweisen. Ein weiteres Beispiel dafür sind die Bankanwendungen. Bei dieser Anwendungsdomain wird zum Beispiel für eine Anwendung die asymmetrische Kryptographie benötigt und bei den beiden anderen nicht. Auch beim Schreiben des nichtflüchtigen Speichers und beim Anteil der CPU gibt es signifikante Unterschiede.

Aus diesen Erkenntnissen kann geschlossen werden, dass mit den gewählten Kennwer-

ten und einem Java Card Betriebssystem eine klare Unterscheidung zwischen den Anwendungsdomains nicht möglich ist.

Als weiterführende Arbeit sollte analysiert werden welche Auswirkungen die Java Virtuell Maschine auf die Profile der Anwendungen hat. Dazu müssen alle Anwendung als native Implementierung analysiert werden. Dadurch fällt der Einfluss der virtuellen Maschine weg bzw. es kann der Einfluss der virtuellen Maschine ermittelt werden. Möglicherweise könnte auch durch eine andere Wahl der Anwendungsdomains oder Kennwerte eine klare Unterscheidung ermöglicht werden. Um eine statistisch zuverlässige Aussage treffen zu können sollten eine größere Menge an Anwendungen aus den Anwendungsdomains analysiert werden.

Anhang A

Definitionen

AAC	Application Authentication Cryptogram
AES	Advanced Encryption Standard
AFI	Application Family Identifier
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ARPC	Authorisation Response Cryptogram
ARQC	Authorisation Request Cryptogram
ATQB	Answer To Request-B
BAC	Basic Access Control
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Chip Authentication
CAP	Card Application
CC	Common Criteria
CDA	Combined Dynamic Data Authentication/Application Cryptogram Generation
CLA	Class
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CVCA	Country Verifying Certificate Authority

DDA	Dynamic Data Authentication
DES	Data Encryption Standard
DF	Dedicated File
DH	Diffie Hellman
EAC	Extended Access Control
EC	Elliptic Curve
EEPROM	Electrically Erasable Programmable Read-Only Memory
EF	Elemantary File
EMV	Europay MasterCard und VISA
ICC	Integrated Circuit Card
ILP	Instruktions Level Parallelisierung
INS	Instruction
JCF	Java Card Forum
JP	Java Purse
JPC	Java Purse Crypto
kB	Kilobyte=1024 Byte
kbit	Kilobit=1024 Bit
Lc	Length command
Le	Length expected
MAC	Message Authentication Code
MF	Master File
MHz	Megahertz
MMU	Memory Management Unit
MRTD	Machine Readable Travel Document
MRZ	Machine Readable Zone
NVM	Non Volatile Memory
OS	Operating System
P1	Parameter 1

P2	Parameter 2
PACE	Password Authenticated Connection Establishment
PIN	Persönliche Identifikationsnummer
PKI	Public Key Infrastruktur
RAM	Random Access Memory
REQA	Request-A
REQB	Request-B
RF	Radio Frequency
RI	Restricted Identification
RID	Registered Application Provider Identifier
RNG	Random Number Generator
ROM	Read Only Memory
SDA	Static Data Authentication
SIM	Subscriber Identity Module
SM	Secure Messaging
SW1	Status Word 1
SW2	Status Word 2
TA	Terminal Authentication
TC	Transaction Certificate
TDES	Triple DES
USB	Universal Serial Bus
VM	Virtuelle Maschine

Literaturverzeichnis

[BCP09] Samia Bouzefrane, Julien Cordry, and Pierre Paradinas. MESURE Tool to benchmark Java Card platforms. *IJCSI International Journal of Computer Science Issues, Vol. 1, 2009*, 2009.

[BEG+06] Koen De Bosschere, Lieven Eechhout, Andy Georges, Kenneth Hoste, Lizy K. John, and Aashish Phansalkar. Performance Prediction based on Inherent Program Similarity. *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, 2006.

[Bun08] Bundesamt für Sicherheit in der Informationstechnik (BSI). Advanced Security Mechanisms for Machine Readable Travel Documants. Online verfügbar auf `http://www.bsi.de/english/publications/techguidelines/tr03110/TR-03110_v200.pdf`, 2008.

[Cor08] Oracle Corporation. *Integrating E-ID into Next-Generation Citizen Services*, 2008.

[Cri10] Common Criteria. Common Criteria. Online verfügbar auf `http://www.commoncriteriaportal.org`, 2010.

[Erd04] Monika Erdmann. Benchmarking von Java Cards. Master's thesis, Institut für Informatik der Ludwig-Maximilians-Universität München, 2004.

[Fin08] Klaus Finkenzeller. *RFID Handbuch.* Carl Hanser Verlag München, 2008.

[Fis06] Mario Fischer. Vergleich von JAVA- und Native-Chipkarten Toolchains, Benchmarking, Messumgebung. Master's thesis, Institut für Informatik der Ludwig-Maximilians-Universität München, 2006.

[fsIA12] A-SIT Zentrum für sichere Informationstechnologie Austria. Bürgerkarte. Online verfügbar auf `http://www.buergerkarte.at`, 2012. [Stand 07. Jänner 2012].

[Gmb12] PayLife Bank GmbH. www.quick.at - Quick - auf ihrer Maestro Bankomatkarte: Einfach, schnell, kostenlos. Online verfügbar auf `http://www.quick.at`, 2012. [Stand 07. Jänner 2012].

[Int06] International Civil Aviation Organization. Doc 9393 - Machine Readable Travel Documents. Online verfügbar auf `http://www2.icao.int/en/MRTD/Pages/Downloads.aspx`, 2006.

[ISO00] ISO/IEC 7816-9. Identification cards - Integrated circuit cards - Part 9: Additional interindustry commands and security attributes, 2000.

[ISO04] ISO/IEC 7816-4. Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange, 2004.

[ISO08a] ISO/IEC 14443-2. Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 2: Radio frequency power and signal interface, 2008.

[ISO08b] ISO/IEC 14443-3. Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 3: Initialization and anticollision, 2008.

[ISO08c] ISO/IEC 14443-4. Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 4: Transmission protocol, 2008.

[Jen09] Jens Bender and Dennis Kügler. Fortgeschrittene Zugriffskontrolle mit PACE. Online verfügbar auf `https://www.bsi.bund.de/cae/servlet/contentblob/915630/publicationFile/58834/kes0609_pdf.pdf`, 2009.

[Kru95] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12:42–50, November 1995.

[LLC11] EMVCo LLC. EMVCo. Online verfügbar auf `http://www.emvco.com`, 2011. [Stand 3. Februar 2011].

[Ltd12] ARM Ltd. ARM - the architecture for the digital world. Online verfügbar auf `http://www.arm.com`, 2012. [Stand 07. Jänner 2012].

[Mic06] Sun Microsystems. Java card platform specification. Online verfügbar auf `http://java.sun.com/javacard/specs.html`, 2006. [Stand 4. September 2010].

[RE08] Wolfgang Rankl and Wolfgang Effing. *Handbuch der Chipkarten*. Carl Hanser Verlag München Wien, 2008.

[Reh05] Karima Rehioui. Java Card Performance Test Framework. Master's thesis, Université de Nice - Sophia-Antipolis, 2005.

[Rot11] Eric Rotenberg. The journal of instruction-level parallelism. Online verfügbar auf `http://www.jilp.org`, 2011. [Stand 07. Jänner 2012].

[Shi08] Chris Shire. Smart card technology trends. *Smart Cards, Tokens, Security and Applications*, 2008.

[uEm11] Sozialversicherungs-Chipkarten Betriebs und Errichtungsgesellschaft m.b.H. e-card. Online verfügbar auf `http://www.chipkarte.at`, 2011. [Stand 24. August 2011].

[Wik10] Wikipedia. Chipkarte — Wikipedia, Die freie Enzyklopädie. Online verfügbar auf `http://de.wikipedia.org/w/index.php?title=Chipkarte&oldid=75418698`, 2010. [Stand 1. Juli 2010].

[Wik11] Wikipedia. Reisepass — Wikipedia, Die freie Enzyklopädie. Online verfügbar auf `http://en.wikipedia.org/wiki/File:Biometrie-reisepass-deutsch.jpg`, 2011. [Stand 24. August 2011].

[www11] www.mikrocontroller.net. 8051 - mikrocontroller.net. Online verfügbar auf `http://www.mikrocontroller.net/articles/8051`, 2011. [Stand 9. April 2011].