Digitale Signatur mit Chipkarten an der FH Ravensburg-Weingarten

Diplomarbeit

Max Kliche
Angewandte Informatik

Fachhochschule Ravensburg-Weingarten

Erstkorrektor:
Prof. Dr. rer. nat. W. Ertel
Zweitkorrektor:
Prof. Dr. rer. net. M. Hulin

31. Januar 2003

Inhaltsverzeichnis

1. Einleitung

Geschäftliche Vorgänge werden zunehmend mit Hilfe des Internets abgewickelt.

Die eindeutige Identifizierung der Kommunikationspartner muss dabei gewährleistet sein. Die Möglichkeit, eine fremde Identität anzunehmen oder eine Unterschrift zu fälschen, muss ausgeschlossen werden. Insbesondere muss eine nachträgliche Abänderung eines unterschriebenen Textes von den Kontrollmechanismen bemerkt werden.

Hierfür dient die elektronische oder digitale Signatur.

Im Sinne des Signaturgesetzes sind "elektronische Signaturen" Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen. [SIGG]
Eine Chipkarte oder Smartcard bietet eine geeignete Hardware für die elektronische Signatur.

In der vorliegenden Arbeit geht es um die Integration von Chipkarten in das Online Bestellwesen der FH Ravensburg-Weingarten.

Für die digitale Signatur ist eine zentrale Kontrollstelle, eine Certification Authority (CA), unbedingt notwendig. Für das Bestellwesen-System wurde daher eine eigene CA aufgebaut.

Zur Veranschaulichung der Funktionsweise von Chipkarten, wurde ein Programm entwickelt. Mit Hilfe dieses Programmes ist es möglich die Kommunikationsabläufe von einzelnen Kommunikationsschritten darzustellen.

2. Aufgabenstellung

Die Aufgabenstellung der vorliegenden Arbeit kann sich in zwei Teile aufspalten lassen:

1. die Darstellung der Funktionsweise von Chipkarten
2. die Implementierung einer digitalen Signatur mit Chipkarten in das Bestellwesen der FH Ravensburg-Weingarten.

Die Implemetation war unabhängig von einem Betriebssystem zu lösen und durfte das bestehende System nicht beeinträchtigen. Daher wurde Java als plattformunabhängige Sprache gewählt.

Dazu waren der Chipkartenmarkt zu untersuchen und die für die FH Ravensburg-Weingarten brauchbare Karten und Leser zu finden.

3. Elektronische Signatur

Die elektronische oder digitale Signatur wird in der zunehmend vernetzen elektronischen Welt immer wichtiger.

Zur technischen Realisierung wird ein Verfahren benötigt, das die Zuordnung eines elektronischen Textes zu seinem Verfasser eindeutig und nicht manipulierbar ermöglicht.
Darüberhinaus muss dieses Verfahren rechtssicher sein. Die Anforderungen des Signaturgesetzes sind zu beachten [SIGG].

Im Sinne des Signaturgesetzes sind "elektronische Signaturen" Daten in elektronischer Form die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen.

Für das Bestellwesen können eine qualifizierte elektronische Signaturen eingesetzt werden.

Nach dem Signaturgesetzes sind: "qualifizierte elektronische Signaturen,

- **auf einem zum Zeitpunkt ihrer Erzeugung gültigen qualifizierten Zertifikat beruhen und**
- **mit einer sicheren Signaturerstellungseinheit erzeugt werden."**

Eine elektronische Signatur auf einem mathmatisch kryptographischen System, einem Public-Key-Verfahren.

Bei diesem Verfahren wird ein "Schlüsselpaar" benötigt. Einer der Schlüssel ist geheim und darf von dem Unterzeichner nicht weitergegeben werden. Die Weitergabe des geheimen Schlüssels käme einer Unterschriftsvollmacht gleich. Der zweite Schlüssel ist nicht geheim, sondern öffentlich. Dieser wird vom Empfänger benutzt, um die Authentizität des Absenders des Dokumentes zu überprüfen.

Es ist ein System erforderlich, dass dem Benutzer jederzeit die Möglichkeit gibt, die eindeutige Herkunft eines öffentlichen Schlüssels zu überprüfen.

Weiterhin spielt die Handhabung eine große Rolle. An den Vorgang:
`Dokument ausdrucken, unterschreiben, wegschicken`
hat man sich in langen Jahren gewöhnt. Ein neues Verfahren darf daher nicht komplizierter sein, um angenommen zu werden.

Die elektronische Signatur ist mittlerweile weit verbreitet und hat sich in der Anwend-

barkeit verbessert. In vielen E-Mail Programmen ist eine entsprechende Funktion eingebaut oder kann einfach nachgerüstet werden.

Schwierigkeiten bereitet noch immer die Speicherung des geheimen Schlüssels. Bei der Speicherung auf einem Computer besteht die Gefahr, dass dieser nicht immer verfügbar ist. Die Verwendung von Disketten hat die Nachteile, dass diese leicht kopiert werden können und unhandlich sind.

Diese Probleme können mit Hilfe von **Chipkarten** gelöst werden. Diese sind klein und robust, kopiersicher und können bequem die erforderlichen Datenmengen aufnehmen.

Weiterhin haben die Chipkarten den Vorteil, dass die Verschlüsselung direkt auf ihnen stattfinden kann, da es sich um einen eigenen kleinen Computer handelt.

Die Anzahl der an Computern installierten Chipkartenlesern nimmt ständig zu. So hat zum Beispiel der Marktführer bei Computerverkäufen aus Deutschland im Jahr 2002 Computer mit integriertem Chipkartenleser verkauft. Es gibt Tastaturen mit eingebautem Chipkartenleser, kleine USB Chipkartenleser und vieles mehr.

Alles in allem bietet die elektronische Signatur mit Chipkarten ein interessantes Entwicklungsfeld, das seinen Höhepunkt noch lange nicht erreicht hat.

3.1. Public Key und Signaturverfahren

3.1.1. Kryptographie

“Ein Verschlüsselungsverfahren darf nicht durch die Geheimhaltung des Verfahrens, sondern nur von dem verwendeten Schlüssel abhängen”, Kerkof [ERTEL].

Die moderne Kryptographie legt sehr viel Wert auf diesen Satz. Bei der Ausschreibung für ein Verfahren, welches der “Advanced Encryption Standard” (AES) werden sollte, war dies eine der wichtigsten Forderungen.

Moderne Chiffrier- und Verschlüsselungsalgorithmen basieren auf mathematischen Methoden. Daher ist, wenn von einem Schlüssel gesprochen wird, eine Zahl oder Bitfolge gemeint.

Die Verschlüsselungslehre unterscheidet zwischen zwei Verfahren, dem Symmetrischen und dem Public-Key Verfahren.

3.1.2. Symmetrische Verfahren

Symmetrische Verschlüsselungsverfahren sind der Kryptographie schon seit langem bekannt. Es gibt nur einen Schlüssel, welchen beide Kommunikationsteilnehmer verwenden. Es wird mit dem selben Schlüssel ver- und entschlüsselt. Der Schlüssel kann frei gewählt werden, sofern er zum verwendeten Verfahren passt.
Ein Beispiel ist in Abbildung 1 dargestellt.

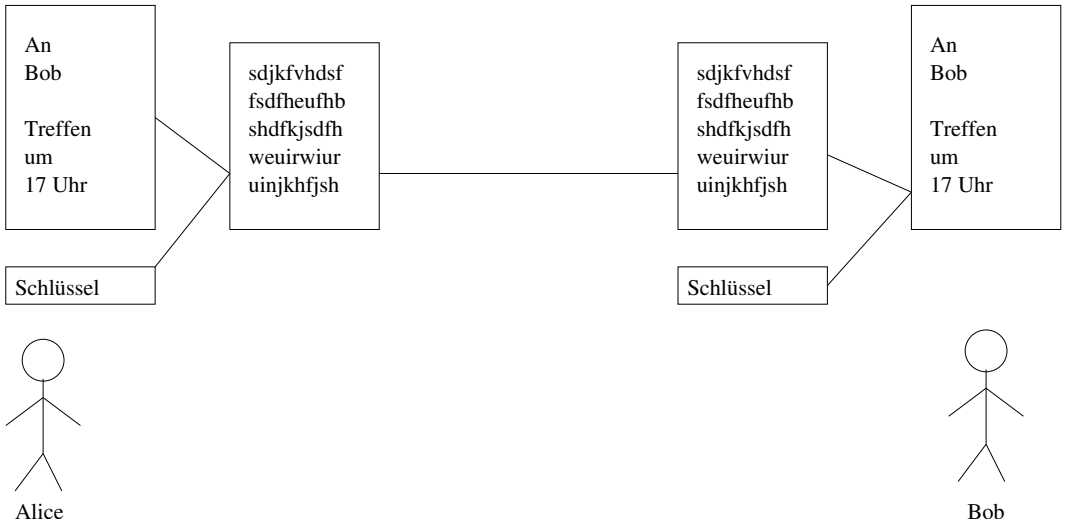

Abbildung 1: Symmetrische Verschlüsselung

Beispiele für symmetrische Verfahren sind:

- DES – Data Encryption Standard

  Basiert auf 56 Bit Schlüsseln. Wird heutzutage nicht mehr als sicher angesehen, da die Schlüssellänge von 56 Bit zu gering ist. (1999 benötigte die distributed.net Gruppe weniger als 24 Stunden, um eine mit DES Verschlüsselte Nachricht zu entschlüsseln) [DNETC].

  DES wurde von IBM entwickelt und der NSA zur Prüfung vorgelegt. Da die NSA an wichtigen Stellen etwas geändert hat, wird oft auf eine Hintertür oder einen "Zweitschlüssel" geschlossen was aber noch nicht bewiesen werden konnte.

- 3DES – 3fach DES

  eine Weiterentwicklung des DES Algorithmus, basierend auf der dreifachen Anwendung eines DES Verfahrens mit zwei Schlüsseln. Dadurch steigt die Schlüssellänge auf 112 Bit an.

- AES – Advanced Encryption Standard

  Dieser Standard wurde 2002 verabschiedet und ist nach einer transparenten Diskussion entstanden. Zum AES wurde schließlich das Verfahren von Rijandal gewählt [ERTEL]. Bei AES beträgt die Schlüssellänge zwischen 128 und 256 Bit.

- IDEA – International Data Encryption Standard

  Der IDEA wurde 1990 von Xuejia Lai und James Massey entwickelt. Dieses Verfahren gilt als sehr sicher. Die Schlüssellänge beträgt hier 128 Bit.

Symmetrische Verfahren arbeiten in der Regel sehr schnell. Dies liegt an der vergleichsweise einfachen Mathematik, die – im Beispiel DES – nur aus xor Verknüpfungen und Transformationen besteht. Dies ist in einem Computer einfach und effizient zu implementieren.

Da bei einem symmetrischen Verfahren derselbe Schlüssel für die Ver- und Endschlüsselung benutzt wird, muss mit jedem Kommunikationspartner ein eigener Schlüssel vereinbart werden. Falls der Schlüssel einer dritten Person bekannt wird, kann diese Person ohne Schwierigkeiten die Daten dechiffrieren.

Symmetrische Verfahren eignen sich daher nur, um z.B. eine Datenverbindung auf einem unsicheren Kanal zu verschlüsseln, wobei der Schlüssel vorher auf einem sicheren Wege ausgetauscht werden mußte.

3.1.3. Public-Key-Verfahren

Die Public-Key- oder asymmetrischen Verfahren arbeiten – im Gegensatz zu den symmetrischen Verfahren – mit Hilfe eines öffentlichen Schlüssels und einem geheimen Schlüssel. Den öffentlichen Schlüssel (Public-Key) darf jeder ansehen. Der geheime oder private Schlüssel darf niemandem außer dem Schlüsselinhaber selbst zugänglich sein.

Während bei den symmetrischen Verfahren der Schlüssel frei wählbar ist, wird bei den asymmetrischen Verfahren ein Schlüsselpaar mit Hilfe einer mathematischen Funktion erzeugt.

Mit einem Public-Key-Verfahren ist es möglich

- einen Text an eine Person veschlüsselt zu verschicken, wobei **nur** der Empfänger den Text lesen kann

  Hierfür ist der Text mit dem öffentlichen Schlüssel des Empfängers zu verschlüsseln. Der Text kann dann nur noch mit dem privaten Schlüssel des Empfängers entschlüsselt werden.

  Auch der Sender selbst kann den Text nicht mehr entschlüsseln!

- einen Text elektronisch zu signieren wobei **jeder** die Möglichkeit hat, die Signatur zu überprüfen.

  Hierfür erstellt man mit Hilfe des geheimen Schlüssels eine Unterschrift unter das Dokument, welche durch den öffentlichen Schlüssel überprüft werden kann.

Die Sicherheit solcher Verfahren liegt in der Schwierigkeit, bestimmte mathematische Funktionen umzukehren.

Es werden zum Beispiel Primzahlen benutzt. Zwei grosse Primzahlen zu erzeugen und diese miteinander zu multiplizieren, ist einfach. Eine Primfaktorzerlegung dieses Produktes vorzunehmen ist dahingegen nur durch Ausprobieren möglich.
Bekannte Public Key Verfahren sind:

- DSA – Data Signature Standard

  Der DSA, Data Signature Standard, ist ein von der NSA entwickelter Algorithmus aus dem Jahre 1991. Dieser Algorithmus eignet sich nur zum Signieren, nicht zum Verschlüsseln. Dieser Algorithmus wird im Online Bestellwesen bislang für die Signatur eingesetzt.

- RSA

  Ein von **Ron Rivest, Adi Shamir** und **Leonard Adleman** 1978 entwickeltes Verfahren, siehe Anhang B und [ERTEL]. Seit September 2000 ist es lizenzfrei benutzbar. Das RSA Verfahren eignet sich sowohl zur Verschlüsselung als auch zur Bildung einer Signatur.

  Die bekannten E-Mail Verschlüsselungsprogramme `pgp` und `gpg` benutzen unter anderem das RSA-Verfahren. Dadurch ist das Verfahren sehr weit verbreitet worden.

Ein Nachteil eines Public-Key-Verfahrens ist, dass die Umsetzung der Algorithmen in Computern relativ langsam ist. Auf einer Chipkarte wird daher, wenn diese RSA-Systeme anbieten, ein Co-Prozessor eingesetzt.

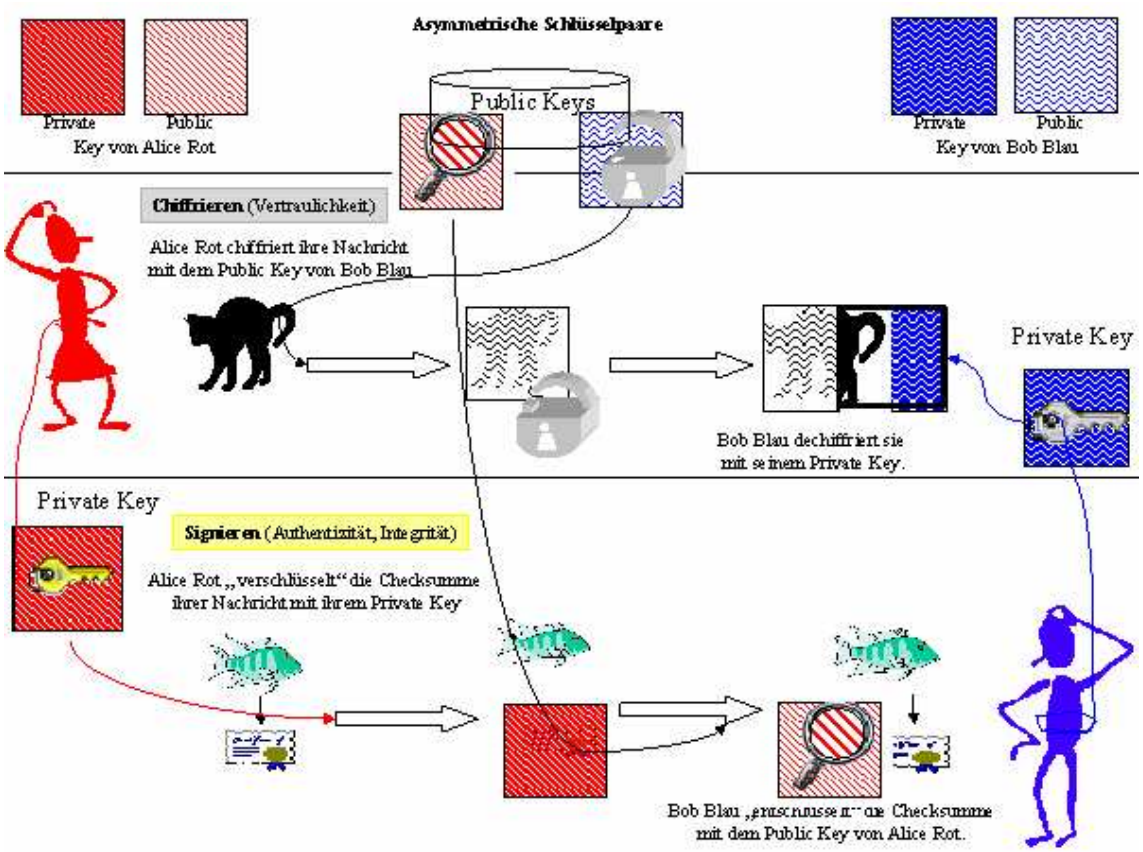

Abbildung 2: Public Key

Bild von http://www.urz.uni-heidelberg.de/Netzdienste/betrieb/security/auth/asym.html

3.1.4. Signaturverfahren

Die Hauptaufgabe eines Signaturverfahrens ist die Sicherstellung der **Integrität des Textes und der Identität des Benutzers**. Die Signatur muss einem Text und einem Benutzer eindeutig zuzuordnen sein.

3.1.5. Hash-Verfahren

Um die Integrität eines Textes zu gewährleisten wird der Text mit Hilfe eines Einweg-Hash Verfahrens "komprimiert". Der erzeugte Hash-Wert wird zur Signaturbildung herangezogen.

Der Hash-Wert eines Textes ist, ausser bei sehr kleinen Texten, kürzer als der Text selbst. Da die Signaturbildung von der Länge des Textes abhängt, dauert eine Signatur des Hash-Wertes nicht so lange wie die Signatur des Ausgangstextes. Dies ist besonders bei Chipkarten wichtig, da hier die Rechengeschwindigkeit gering ist.

Bekannte und zur Zeit benutzte Hash Verfahren sind:

- MD5

  MD5 erzeugt einen 128 Bit langen Hash-Wert.

- SHA1

  SHA1 erzeugt einen 160 Bit langen Hash-Wert.

Ein Hash-Wert ist eine mit einem Einwegverfahren aus einem Text erzeugte Sequenz.
Man kann aus der Sequenz den Text nicht mehr rekonstruieren.
Ein Text liefert immer die selbe Sequenz.

Ein Hash-Verfahren muss weiterhin folgende Bedingungen erfüllen:

- Aus der Sequenz darf nicht mehr auf den Text zu schliessen sein
- bei minimal unterschiedlichen Eingangstexten darf der Hash-Wert **nicht** ähnlich sein und aus den Veränderungen dürfen keine Rückschlüsse auf die Unterschiede der Eingangstexte möglich sein.
- Es muss möglichst schwer sein, zwei Texte zu finden, die zu einem identischen Hash-Wert führen.

Die oben genannten Verfahren erfüllen diese Bedingungen.

3.1.6. Identitätsprüfung

Bei einem Public-Key-Verfahren ist die Identitätsprüfung durch die öffentlichen Schlüssel gegeben. Nur der öffentliche Schlüssel des Signaturerstellers kann die Signatur erfolgreich testen.

Um sicherzustellen, dass das richtige Schlüsselpaar verwendet wird, muss ein Abgleich der Schlüssel gemacht werden. Der Schlüsselabgleich verhindert das Einschleusen von falschen Schlüsseln durch eine dritte Person.

Da Schlüssel für Public-Key-Verfahren (bei RSA) 1024 Bit und länger sind, ist ein Abgleichen des Schlüssels mühsam. Um dies zu erleichtern, vergleicht man nur die Hash-Werte der Schlüssel, die sogenannten "Fingerprints".

```
pub  1024D/C297C9E2 2001-05-19 Max Kliche <max@kliche.org>
Key fingerprint = 1E3E D5D6 2A42 97D3 1D78  6BF2 A8F8 D37C C297 C9E2
sub  1024g/C69F73DC 2001-05-19
```

3.1.7. Web of trust

In einer größeren Umgebung oder bei unpersönlichen Kontakten ist es nicht mehr möglich, die Schlüssel persönlich abzugleichen.

Als Lösung bietet sich ein verteiltes Schlüsselmanagement an.

- "Web of trust"

  Das "web of trust" oder Vertrauensnetz funktioniert nach dem Motto "dein Freund ist auch mein Freund".

  Es wurde bei der Entwicklung der E-Mail Verschlüsselungssoftware `pgp` eingeführt.

  Bei dem "web of trust" werden Schlüssel mit einem zusätzlichem Feld ausgestattet. In dieses Feld kann ein Benutzer mit Hilfe seines geheimen Schlüssels ein Vertrauensverhältnis eintragen. Dies wird Schlüsselsignieren genannt.

  Die Vertrauensverhältnisse bei `pgp` sind:

  - 1 = Don't know
  - 2 = I do NOT trust

    Diese Stufe soll an Schlüssel verteilt werden, bei denen nicht sicher ist, wer die Schlüssel generiert hat und woher sie stammen.

- 3 = I trust marginally
- 4 = I trust fully

  Diese Stufe ist für Schlüssel, bei denen ein persönlicher Abgleich der Fingerprints unternommen wurde, vorgesehen.
- 5 = I trust ultimately

  Hier würde dem Besitzer des Schlüssels sogar eine Bankvollmacht eingeräumt werden. Diese Stufe sollte sehr selten verwendet werden.

In Abbildung 3 haben die mit einem Pfeil verbundenen Personen, ihre Schlüssel auf Stufe 4 gegenseitig zertifiziert.

Wenn “Alice” von “Bob” den Schlüssel von “Carol” bekommt, kann Alice mit Carol in Kontakt treten. Hier ist das Vertrauensverhältnis dann auf Strufe 3.

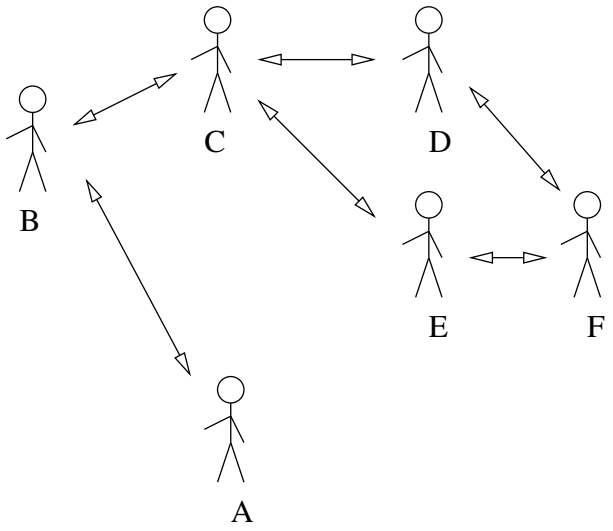

Abbildung 3: Web of trust

- Hierachical Trust

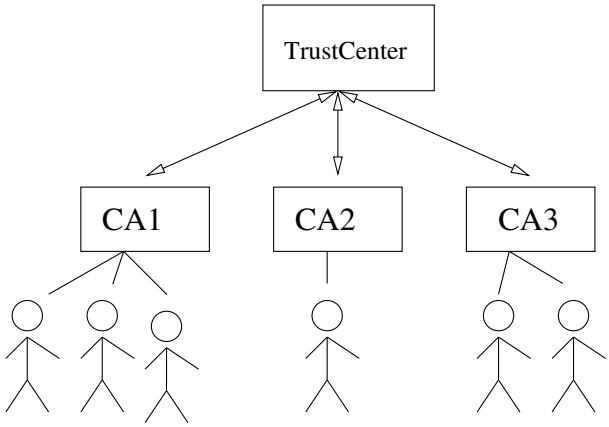

Abbildung 4: Certification Authority

Bei einem Hierachical Trust wird eine Stelle eingeführt, der alle Benutzer vertrauen müssen, eine sogenannte **Certification Authority (CA)**.

Die CA stellt Zertifikate aus, die eindeutig einen Schlüssel mit einem Benutzer in Zusammenhang bringen. Diese Zertifikate tragen auch eine Kennung der CA, damit das Zertifikat nicht gefälscht werden kann.

Für die Digitale Signatur sind in Deutschland mehrere Certification Authority Stellen (TrustCenter) eingerichtet worden. Bei diesen TrustCentern kann gegen eine Gebühr ein Schlüssel zertifiziert werden. Ein Beispiel für ein solches TrustCenter ist Teletrust. [1]

Das Deutsche Forschungsnetz (DFN) betreibt in Hamburg ebenfalls ein TrustCenter. Dieses TrustCenter kann von den Hochschulen in Deutschland als Certification Authority verwendet werden.

Eine Optimierung ist ähnlich dem Web of Trust aufgebaut. Eine CA kann zum Beispiel, eine firmeninterne CA authentifizieren. Diese kann dann Schlüssel zertifizieren. Dabei wird das Vertrauensverhältnis, wie im Web of Trust, verringert.

[1] Ein für E-Mail Verschlüsselung nutzbares Zertifikat ist unter

`http://www.trustcenter.de/homeframes/homeframes_de/privatanwender.htm`

kostenlos zu bekommen.

3.2. Public-Key mit Chipkarten

Die Geheimhaltung des privaten Schlüssels ist die wichtigste Forderung an Public-Key-Verfahren. Durch den Einsatz einer Chipkarte kann der Schlüssel auf dieser sicher gespeichert werden.

Darüberhinaus wird die Signatur direkt auf der Chipkarte vorgenommen. Durch dies muss der Private Schlüssel die Karte niemals verlassen.

Die Schlüsselerzeugung findet direkt auf dem Rechenchip der Karte statt. So ist selbst bei der Schlüsselerzeugung keine Möglichkeit gegeben, an den Schlüssel zu gelangen.

Moderne Chipkarten arbeiten mit Zertifikaten zusammen, die von einer CA ausgestellt sind. Diese Zertifikate werden benutzt, um ein einheitliches Format der Schlüssel zu erreichen.

Abbildung 5: Chipkarte

4. Chipkarten

4.1. Überblick

Die Entwicklung der Chipkarten begann Anfang der 50er Jahre in den USA mit der Einführung der Kreditkarte. Diese Karte war eine schlichte Plastikkarte auf der lediglich der Inhaber und der Firmenname des Kreditinstitutes vermerkt waren. Diese diente als einfacher, stabiler Papierersatz.

Später bekamen diese Karten eine Hochdruckprägung, die nicht so leicht kopierbar war. Die Karten dienten aber weiterhin nur als fester Datenspeicher. Die einmal auf der Karte aufgebrachten Daten, wie Name und Kreditkartennummer, konnten nicht mehr geändert werden.

Mit Einführung des Magnetstreifens auf den Kartenrückseiten änderte sich dies. Die Magnetstreifen sind les– und schreibbare Datenspeicher ähnlich einer Diskette. Jetzt konnten die Daten der Chipkarte auch im nachhinein noch geändert werden. So kann zum Beispiel der Kontostand auf der Karte gespeichert und nach einer Kontobewegung angepasst werden.

Die Weiterentwicklung führte zu Prozessor und Speicherkarten mit einem integriertem Computerchip.

Diese verdienen erstmals den Namen **Chipkarte** .

Mittlerweile begegnet man Chipkarten überall. Bekannte Beispiele sind die Krankenkassenkarte, die Bank-Kunden-Karte, der Studentenausweis oder die Telefonkarte. Sicherlich wird die Anzahl der Applikationen, die Chipkarten, benötigen in Zukunft weiter ansteigen.

4.2. Hardware

Eine Chipkarte besteht in der Regel aus Kunststoff, welches biegsam und haltbar ist[2]. In diese Plastikkarte ist an einer durch den ISO-7816 Standard festgelegten Stelle ein Computerchip eingelassen (Abbildung 5 und Abbildung 6).

Es gibt drei Größen von Chipkarten:
Die Handy-SIM Karte im ID-000 Format als kleinste, das weniger gebräuchliche ID-00 Format, eine Zwischengröße, und das weit verbreitete ID-1 Format, wie man es von Telefonkarten kennt. Um die verschiedenen Formate kompatibel zu machen, gibt es für die kleineren Formate Einsetzadapter.

In ISO-7816 sind acht Kontaktflächen vorgesehen, siehe Tabelle 1. Von den acht Kontakten sind zwei RFU, "reserved for further use"[3]. Auf vielen Karten werden die zwei Kontaktflächen, die für zukünftige Erweiterungen vorgesehen sind, weggelassen. Daher finden sich Karten, auf denen nur sechs Kontaktflächen vorhanden sind. Die Chipkarte in Abbildung 5 besitzt 8 Kontakte.

1	Vcc	Versorgungsspannung	5	GND	Masse
2	RST	Eingang Reset	6	Vpp	Programmierspannung
3	CLK	Eingang für Takt	7	I/O	Ein/Ausgabe
4	RFU	–	8	RFU	–

Tabelle 1: Kontaktbelegung

Über die Kontakte findet die gesamte Kommunikation zwischen Terminal und Chipkarte statt.
Kontaktlose Chipkarten übermitteln die Information über eine in die Karte integrierte Antenne.

[2]es werden immer neue Verfahren erprobt. In den Niederlanden wurde auch schon Holz testweise verwendet.

[3]Die Programmierspannung wird nicht mehr benutzt, da die Chipkarten selbst eine Ladungspumpe besitzen. Eine neue Belegung des Kontaktes wird aus Kompatibilitätsgründen nicht vorgenommen.

4.2.1. Unterteilung der Chipkarten

Chipkarten können in drei unterschiedliche Systeme eingeteilt werden

- Speicherkarten

  Speicherkarten sind relativ "einfache" Karten. Diese werden zur Speicherung von Daten verwendet, die keine besondere Sicherheit benötigen. Die Krankenkassenkarte ist ein solches Beispiel. Auf diesen Karten befindet sich in der Regel kein Schutz der gespeicherten Daten vor Manipulation.

  So ist es möglich, mittels eines Kartenlesers und -schreibers Daten auf der Krankenkassenkarte zu manipulieren [Lkka].

  Ein anderes Beispiel ist die Telefonkarte, die aber vor dem Wiederbeschreiben geschützt ist. Es fehlt die Ladungspumpe, um die Schreibspannung für das EEprom auf die nötigen 24 Volt anzuheben.

  Eine Speicherkarte besteht meistens nur aus einem EEprom und einer kleinen Recheneinheit, die nur zur Datenverwaltung dient.

- Prozessorkarten

  Die Prozessorkarten haben eine komplexere Recheneinheit als die Speicherkarten. Dies ermöglicht es, ein Betriebsystem auf einer solchen Chipkarte zu implementieren.

  Beispiele für Betriebssysteme von Prozessorkarten sind:

  - MFC

    Dieses Betriebssystem wird auf der in dieser Arbeit verwendeten Comcard MFC 4.3 eingesetzt.
  - StarCos
  - Cyberflex
  - TCOS

    Diese System wird auf den Signaturkarten der Firma Teletrust benutzt.

  Bis vor wenigen Jahren wurden die Prozessorkarten noch für jede Anwendung komplett in Assembler neu programmiert.

Mittlerweile werden die Betriebsysteme auch in Hochsprachen entwickelt. Dies führte zu Kosteneinsparungen, da die Entwicklungszeit geringer geworden ist.

Weiterhin muss aber nach einer Änderung des Systems das ROM der Karte neu designed werden, da das Betriebssystem im ROM abgelegt ist.

Die Chipkartenbetriebssysteme werden immer komplexer, so dass immer weniger teure Änderungen nötig sind. Die meisten Betriebssysteme unterstützen mittlerweile kryptographische Funktionen wie die Digitale Signatur oder Verschlüsselungen.

Die Funktionen des Betriebssystems werden in der ISO-7816 genormt. Eine Chipkarte unterstützt in der Regel nur einen Teil dieser Funktionen.

- Programmierbare Karten

  Programmierbare Karten haben ein System, das weitgehend frei programmierbar ist. Die BasicCard oder auch eine JavaCard sind Beispiele.

  Bei einer solchen Karte gibt es keine vorgefertigten Befehle. Befehle für beispielsweise die Speicherung von Daten müssen eigens implementiert werden.

  Bei einer programmierbaren Chipkarte werden die ausführbaren Befehle im EEprom abgelegt. Da das EEprom langsamer als eine ROM Schaltung ist, sind diese Chipkarten bei Ausführung der Befehle langsamer als Prozessorkarten.

  Eingesetzt werden programmierbare Karten entweder um eine Anwendung zu testen oder, da die Entwicklungskosten durch das Wegfallen der ROM Programmierung niedrig sind, für kleine Chipkartenserien einer Spezialanwendung.

Karte	Speicher
BasicCard 3.9	8 KByte
BasicCard 5.4	16 KByte
Comcard MFC 4.3	32 KByte
Cryptofelx	8 Kbyte

Tabelle 2: Speicherkapazitäten von Chipkarten

Eine Chipkarte besitzt eine Speicherkapazität von ca. 8 KByte bis ca. 64 KByte. In Tabelle 2 sind Beispielwerte angegeben.

Die angegebenen Speichergrößen müssen richtig interpretiert werden. Bei der Comcard MFC 4.3 stehen die 32 KByte vollständig für die Datenspeicherung zur Verfügung, da es sich um eine Prozessorkarte handelt.
Bei der BasicCard wird dieser Speicher auch verwendet, um die Erweiterungen des Betriebssystems unterzubringen. So benötigen die für die Signatur wichtigen Elliptischen Kurven ca. 8 KByte Platz im EEprom. Ein AES Verschlüsselungsmodul belegt ca. 4 KByte.

Das EEprom der Chipkarte kann nach Herstellerangaben bis zu 100000 mal beschrieben werden. Dies sollte für alle Applikationen ausreichen. In den meisten Fällen ist eine Chipkarte vorher verlorengegangen oder andersweitig defekt.

Der RAM Speicher einer Chipkarte ist wesentlich kleiner. Dieser beträgt nur wenige KByte, bei der BasicCard zum Beispiel genau 1 KByte.

Programme, die auf einer Chipkarte laufen sollen, müssen daher sparsam mit dem Speicherverbrauch umgehen. Um komplexe Abläufe auf einer Chipkarte dennoch realisieren zu können, werden dem Prozessor Co-Prozessoren zur Seite gestellt.
Es gibt Co-Prozessoren für zum Beispiel RSA, mathematische Berechnungen oder Zufallszahlengeneratoren.

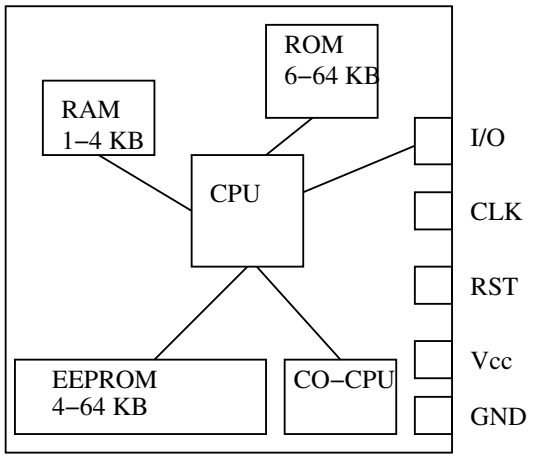

Abbildung 6: Schematischer Aufbau

In Abbildung 6 erkennt man, dass eine Chipkarte keinen Taktgeber besitzt. Der Takt wird extern vom Terminal bezogen.

Weiterhin verfügt eine Chipkarte über keine eigene Spannungsquelle. Die Versorgungsspannung von 5 Volt bezieht die Karte vom Terminal.

Um die für die Beschreibung der EEproms erforderliche Spannung von 24 V zu erreichen haben Chipkarten eine Ladungspumpe integriert.

Damit die Chipkarte ordnungsgemäß funktioniert, muss das Terminal eine genau festgelegte (ISO-7816) Einschaltsequenz benutzen. Die Reihenfolge dafür ist:

1. Masse
2. Versorgungsspannung
3. Takt
4. Reset

Erst mit dem Senden des Reset Signals fängt die Chipkarte an zu arbeiten. Sie sendet den ATR (siehe Kapitel 4.3.1).

Wird diese Reihenfolge nicht beachtet, darf eine Chipkarte aus Sicherheitsgründen keine Daten liefern.

Informationen zum Aufbau und der Funktionsweise siehe [RANKL99].

4.3. Befehle

4.3.1. ATR

Nachdem die Spannungsversorgung und der Takt an die Chipkarte, wie in Abschnitt 4.2.1 beschrieben, vom Terminal angelegt worden ist, wird das Resetsignal gesendet. Auf dieses Signal reagiert die Chipkarte mit dem **ATR**, dem “answer to reset”.

Der ATR hat eine Länge von maximal 33 Byte, welche aber selten völlig ausgenutzt wird. Meistens ist der ATR nur wenige Bytes lang. Dies ist vor allem bei Applikationen wichtig, die eine schnelle Reaktion der Chipkarte benötigen, zum Beispiel eine Mautkarte für LKW, die beim Durchfahren einer Induktionsschleife Daten senden und empfangen soll.

Der ATR unterscheidet sich ja nach Hersteller, Betriebsystem und Chipkarte. Daher kann dieser zur Identifikation von Chipkarten verwendet werden.

Im Programm “Bestellwesen” wurde in der entwickelten Klasse “FHSmartCard” (siehe Kapitel 8.1.1) diese Möglichkeit verwendet, um das Programm zu verschiedenen Chipkarten kompatibel zu machen.

```
//comcard IBM MFC 4.3
 if( atr.equals
   ("3BEF00FF813166557E0B000449424D2043434330303100F6"))
...
//  Basiccard 5.4
 if( atr.equals
   ("3BFB1300FFC0803180755A43352E34205245562041A5"))
...
//Basiccard   5.4 t=1 Protokoll
 if( atr.equals
   ("3BF71100FF81318075543D302041545262"))
...
```

Listing 1: ATR Unterscheidung

4.3.2. ATR-Aufbau

Der Aufbau des ATR ist in ISO-7816-3 definiert.

Datenelement	Bezeichnung
TS	The Initial Character
T0	The Format Character
TA1, TB1, TC1, TD1...	The Interface Characters
T1, T2, ... TK	The Historical Characters
TCK	The Check Character

Tabelle 3: ATR Datenelemente

Im **Initial Character TS** (siehe Tabelle 3) wird die "Convention" der Übertragung definiert. Dies ist die Festlegung, ob ein High Pegel der Übertragung eine binäre 1 oder binäre 0 darstellen soll. Der TS hat nur zwei mögliche Codierungen: 3B oder 3F. Die "Direct-Convention" wird mit 3B codiert, die "Inverse-Convention" mit 3F. In Deutschland üblich ist die 3B Convention.

Der **Format Character T0** dient der Mitteilung, was und wieviel als nächstes übertragen wird. Das Feld ist als Bitfeld aufgebaut. Die ersten vier Bit geben an, welche der Interface Characters übertragen werden. Dieses Feld ist, genauso wie der Initial Character, obligatorisch.

Die folgenden **Interface Characters** sind optional. Welche gesendet werden, steht in den ersten Bits des T0.
Die Interface Character regeln mit der Chipkarte Übertragungsraten aus. Hierzu wird in den ersten Bits der Multiplikator des Standardtaktes von 5 MHz angegeben. Dadurch ist eine höhere Datenübertragung von und zur Chipkarte möglich.

Das **TB1** Feld ist heutzutage nicht mehr nötig. Es diente der Signalisierung der Speicherart. Da heutzutage alle Chipkarten EEprom anstelle von Eprom besitzen, wird dieses Feld ignoriert.

Das Feld **TC1** gibt die Zahl an, um die die Schutzzeit verlängert wird. Bei dem Wert FF wird die Schutzzeit, die normalerweise zwei Takte beträgt, auf einen Takt heruntergesetzt. Das bewirkt eine ca. zehnprozentige Geschwindigkeitssteigerung.

Weitere wichtige Felder sind die **Historical Characters**. In diesen Feldern werden die Betriebssystemversion und der Herstellername kodiert. Anhand dieser Angaben kann dann die Chipkarte identifiziert werden.

Datenbanken über verschiedene ATR Strings findet man unter [muscle].

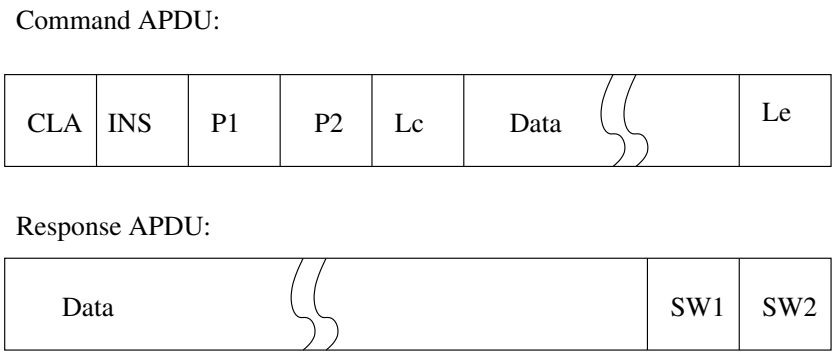

Abbildung 7: APDU

4.4. APDUs

Die Übertragung von Daten zwischen Terminal und Chipkarte auf der physikalischen Ebene wird von den T=0 oder T=1[4] geregelt. Auf der logischen Ebene werden für die Übertragung **APDUs** verwendet.

APDU steht für "application protokoll data unit". Die APDUs sind nach ISO-7816-4 so aufgebaut, dass sie unabhängig von dem darunterliegenden Protokoll sind.

Es gilt weiterhin, dass die Chipkarte keine Daten von sich aus versendet. Das Terminal muss immer erst eine Anfrage stellen, bevor die Chipkarte reagiert.

Eine APDU besteht aus maximal 262 aneinanderhängenden Bytes. Davon sind 6 Bytes für Steuerdaten vorgesehen und 254 Bytes für Daten (siehe Abbildung 7). Mehr Daten können nicht ohne Unterbrechung an die Chipkarte gesendet werden.

Die Antwort der Chipkarte ist maximal 256 Bytes lang, da hier zwei Bytes (SW1 und SW2) zu den zurückgesendeten Daten hinzukommen.

Die APDUs sind, wie in Abbildung 7 zu sehen, aus aneinanderhängenden Feldern aufgebaut. Jedes Feld ist ein Byte lang, mit Ausnahme des "Data" Feldes. Mit Hilfe des TestApplets (siehe Kapitel 7) kann man sich mit den APDUs vertraut machen.

Die Bedeutung der Felder:

- **CLA** und **INS**

  Das CLA-Byte gibt die Klasse des Befehles an. Mit diesem Feld können verschiedene Sicherheitsstufen oder Anwendungen strukturiert werden. Das INS-Byte ist die Kodierung für den Befehl in dieser Klasse.

[4] in Deutschland auch T=14

- **P1** und **P2**

  Dies sind immer zu setzende Felder, die Optionen für den Befehl enthalten. Wenn der Befehl keine Optionen benötigt, wird 00 gesendet.

- **Lc**

  Dieses Feld gibt die Länge der nachfolgenden Daten an. Hier sieht man, dass die maximale Datenlänge eines Befehls an die Chipkarte 254 Byte (FF) sein kann.

- **Data**

  Die eigentlichen Daten.

- **Le**

  Hier wird die Anzahl der als Antwort erwarteten Bytes angegeben.

Der Aufbau der APDU unterscheidet zwischen vier Fällen (siehe Abbildung 8)

1. Case 1

   Das einfache Senden eines Befehls. Die Chipkarte schickt nur die Statusbytes SW1 und SW2 zurück.

2. Case 2

   Hier wird das Le Feld zusätzlich gesetzt. Die Chipkarte bekommt dadurch mitgeteilt, wie viele Antwort Bytes erwartet werden. Dazu kommen noch die beiden Statusbytes.

3. Case 3

   Ein Befehl wie Case 1 mit angehängtem Datenfeld.

4. Case 4

   Dieser Fall ist eine Kombination aus Case 3 und Case 2.

Beispiel für eine Case 3 APDU, gesendet an die BasicCard:

```
public void verifyPin(String PIN)
  {
  byte cmd[]={(byte)0x23,(byte)0x00,(byte)0x00,(byte)0x00};
  command.setLength(0);
  command.append(cmd);
  command.append((byte)PIN.length());
  command.append(PIN.getBytes());
  sendCMD("verifyPIN");
  }
```

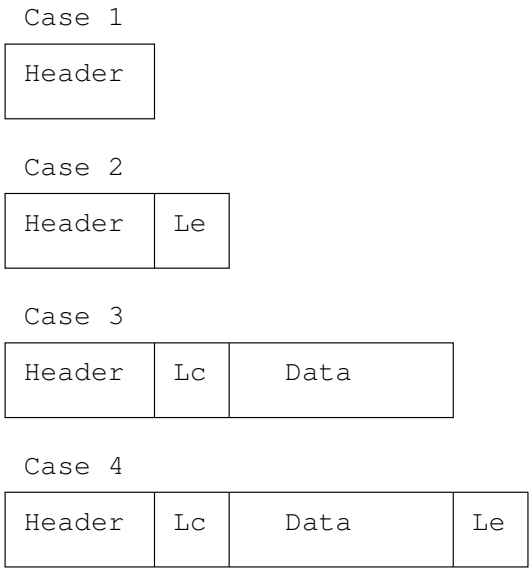

Abbildung 8: APDU Cases

In der ISO-7816 sind eine Reihe von Befehlen für Chipkarten aufgeführt. Dazu gehören:

CLA	INS	Kommando	
00	E2	APPEND RECORD	Erweiterung einer “Liner Fixed” Datei
00	24	CHANGE CHV	Ändern der PIN
00	E0	CREATE FILE	Anlegen einer neuen Datei

Tabelle 4: APDUs nach ISO-7816

Die Antworten der Chipkarte bestehen aus den zurückgeschickten Daten und den beiden Statusbytes. Ein Rückgabewert von $0x90\ 0x00$ steht für eine erfolgreiche Ausführung des Befehls.

Der Rückgabewert ist nach ISO 7816-4 wie in Abbildung 9 aufgebaut.

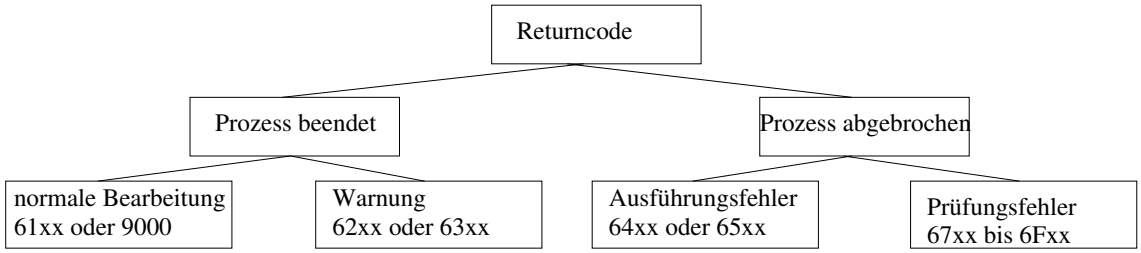

Abbildung 9: Return Codes nach ISO 7816-4

Die Fehlercodes unterscheiden sich trotz der Norm ISO-7816. Bei den vorliegenden Karten gab es u.a. folgenden Unterschied:

- Die BasicCard liefert bei einer Anfrage mit einem Le Feld, das nicht zur Antwort passt, in SW2 die “expected length”
- Die Comcard MFC 4.3 liefert nur die Fehlermeldung, “0x61 0x00 Weitere Daten vorhanden”, nicht aber die Anzahl der Bytes.

4.5. Geschwindigkeit von Chipkarten

Bei einer interaktiven Anwendung, wie zum Beispiel dem Webformular des "Online Bestellwesens", ist es wichtig, dass die Antworten in einer akzeptablen Zeit erscheinen.

Bei der Gestaltung von dialogbasierten Systemen sollte eine Reaktion des Programmes nicht länger als eine Sekunde dauern [Preim].
Zur Verdeutlichung der Chipkartengeschwindigkeit dient ein kleines Experiment: die Erzeugung von 1000 Zufallszahlen auf einer Chipkarte. Die Daten in Tabelle 5 entstanden aus diesen Versuchen.

Karte	Zeit
ComCard	1 Min 2 Sek
BasicCard 5,4	2 Min 20 Sek
PC	ca. 0.2 Sek

Tabelle 5: Zeiten Zufallszahlen

Zeiten für die Erzeugung von 1000 Zufallszahlen mit unterschiedlichen Medien.

In einem weiteren Experiment wurde die Geschwindigkeit der auf der Comcard MFC 4.3 integrierten SHA-1 Implementation gemessen. Tabelle 6 zeigt den Anstieg der Rechen- und Kommunikationszeit bei unterschiedlichen Textlängen.

Textlänge	Dauer
1 Byte	1,87 Sekunden
11 Bytes	1,89 Sekunden
241 Bytes	2.38 Sekunden
17 KByte	55 Sekunden
170 KByte	7 Minuten 46 Sekunden

Tabelle 6: Hash Zeiten

Diese Daten wurden mit Hilfe
der ComCard MFC 4.3 und einem Towitoko Reader
unter Linux ohne eingeschaltete Verschlüsselung erstellt.

4.6. Datenspeicherung

Die Speicherung von Daten auf einer Chipkarte findet im EEprom statt. Bei einer Chipkarte nach ISO-7816 gibt es ein zur Datenspeichung vorgesehenes Filesystem. Dieses Filesystem ist hierachisch aufgebaut, ähnlich dem Unix Filesystem.

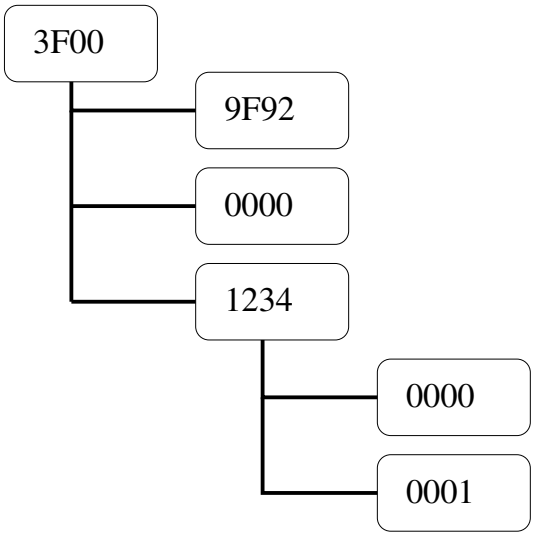

Abbildung 10: Filesystem

Es wird zwischen EF (Elementary Files) und DF (Dedicated Files oder auch Directory Files) unterschieden.

Bei den EF wird weiterhin zwischen **Working-EF** und **Internal-EF** unterschieden. Die Working EF sind die Dateien, in denen Daten abgelegt werden, die für die externe Kommunikation bestimmt sind. Die Internal-EF beinhalten die Betriebssystem Daten und auch die geheimen Schlüssel. Die Internal-EF werden durch das Betriebssystem der Chipkarte besonders geschützt. Es ist nicht möglich, mit Fileoperationen auf die Daten zuzugreifen.

Jedes File, egal ob EF oder DF, besitzt eine Kennung. Diese besteht aus zwei Bytes, dem Dateinamen.

Bei Chipkarten nach ISO-7816 sind einige Dateinamen bereits für die Schlüsselverwaltung oder das Betriebssystem reserviert (siehe Tabelle 7).

Das Wurzelverzeichniss wird als Master File bezeichnet und hat die Kennung $3F00$.

In einem Directory kann eine Kennung nur einmal vorkommen.

Um eine Datei zum Lesen oder Schreiben auszuwählen, ist in ISO-7816 das Select-File

FileID	Funktion
3f 00	Master File
00 00	PIN
01 00	SO-PIN
00 11	Management Keys
00 01	Keys für “Externel Authentificate”
2F 00	ATR String
FF FF	Reserved

Tabelle 7: Auswahl der bei der Comcard MFC 4.3 reservierten Filenamen

Kommando (CLA = $0x00$ INS = $0xA4$) vorgesehen.

Es gibt zwei Arten des SELECT-FILE

- **Explizit**

  Hier wird durch den Befehl nur ein Zeiger auf das File mit der FileID, die im Kommando angegeben wird, gesetzt.

  Um den Dateiinhalt zu lesen, muss ein READ Kommando an die Chipkarte geschickt werden.

- **Implizit**

  Hier wird bei einem Zugriff auf eine Datei der Inhalt direkt gelesen. Dieses Verfahren hat den Vorteil, einen Befehl einzusparen.

  Der Nachteil dieses Verfahrens ist, dass es nicht möglich ist, ein File zu selektieren, welches nicht im aktuellen Verzeichnis ist. Weiterhin unterliegt das implizite Select-File Kommando strengeren Sicherheitskriterien als das explizite.

Ein Beispiel für ein Kommando, das die Datei 1234 im Master File selektiert:

$$0x00 \;\; 0xA4 \;\; 0x08 \;\; 0x0C \;\; 0x02 \;\; 0x12 \;\; 0x34$$

also CLA = $0x00$, INS = $0xA4$, P1 = $0x08$ (absoluter Pfad), P2 = $0x0C$ (keine File-Info). Danach folgen die Länge des Dateinamens und der Dateiname 1234 selbst.

Obwohl der Standard die Funktionsweise so vorschreibt, unterstützen nicht alle Karten dieses Kommando.

Von den Testkarten unterstützt es die Comcard MFC 4.3 . Auf der BasicCard muss dieses Kommando erst mit Hilfe eines Basic-Programmes nachimplementiert werden.

4.6.1. Typen von Elementary Files

Jedes EF hat eine interne Struktur. Diese Struktur ist je nach Anwendung individuell für ein EF wählbar. Es kann auf einer Chipkarte mehrere Filetypen geben. Der FileTyp wird beim Anlegen des Files angegeben und kann im Nachhinein nicht mehr geändert werden.

Die verschiedenen Filetypen im einzelnen:

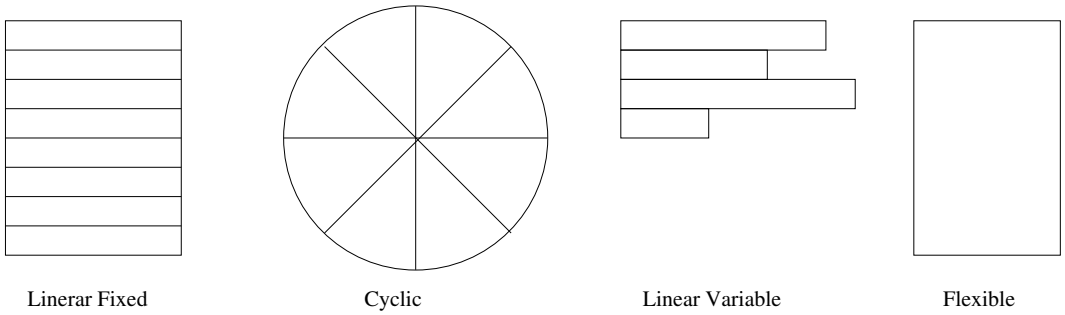

Abbildung 11: Filetypen

- **transparent/flexible**

  Die transparente Datenstruktur ist eine einfache Bytekette. Auf die Bytes kann mit Hilfe eines Offsets zugegriffen werden. Das Lesen und Schreiben ist byteweise möglich.

- **linear fixed**

  Diese Dateistruktur ist eine Verkettung von gleich langen Datensätzen, auch Records genannt. Ein Record ist eine Verkettung aus einzel Bytes. Auf die Datensätze kann mit Hilfe von Offsetangaben direkt zugegriffen werden. Der Offset wird immer in ganzen Record-Schritten angegeben. Allerdings muss man bei Änderungen nur den ganzen Record neu schreiben. Ein Schreiben von einzelnen Bytes ist nicht möglich.

  In einem “linear fixed File” können maximal 254 Records abgelegt werden, da für die Offsetangabe in dem Select-File Kommando nur 1 Byte vorgesehen ist.

- **linear variable**

  Im Gegensatz zu den linear fixed Files können die einzelnen Records bei der “linear variable” Struktur unterschiedliche Längen aufweisen. Für die Angabe der Länge wird ein zusätzliches Informationsbyte in jedem Record benötigt. Dennoch überwiegt bei flexiblen Daten, wie Namen, die Platzeinsparung.

- **cyclic**

  Cyclic Files basieren auf der Linear Fixed Dateistruktur. Die Records haben auch hier die gleiche Länge. Die Daten sind aber cyclisch angeordnet.

Für spezielle Anwendungen gibt es Chipkarten mit angepassten Datentypen. Dies können Datenbanken oder Executable Files sein.

Die Sicherheit der Dateien hängt wesentlich von dem Betriebssystem der Chipkarte ab. Die Dateien haben einen Dateiheader, in dem die Zugriffsbedingungen definiert sind. Die Sicherheitseinstellungen unterscheiden zwischen Applikationen und verschiedenen Benutzerstufen (PIN, SO-PIN).

4.6.2. Datenspeicherung auf der BasicCard

Bei der BasicCard gibt es mehrere Möglichkeiten, Daten zu speichern.
Es ist die Erstellung eines BasicCard Programmes erforderlich , um Funktionen für die Datenspeicherung anzubieten.
Mit Hilfe eines eigenen Befehles können Daten direkt in eine Variable geschreiben werden, die im EEprom abgelegt wird. Für so ein Verfahren sind nicht viele Resourcen auf der Karte erforderlich.
Die andere Möglichkeit ist, die File-IO Library der BasicCard zu benutzen. Diese Lösung benötigt aber Resourcen (ca 1,5 Kb) auf der Karte, unabhängig von den gespeicherten Daten.

In der vorliegenden Arbeit wurden auf der BasicCard nur wenige Daten gespeichert. Daher wurde auf die Filesystem Library verzichtet. Eine Sicherheitsüberprüfung der Daten wird vorgenommen. Der Zugriff auf die gespeicherten Daten ist nur nach der erfolgreichen PIN-Eingabe möglich.

```
Eeprom UserName  As String*254 = ""
Eeprom UserPass  As string*254 = ""
Command &H23 &H70 GetName(Message$)
  If Not (PINVerified) Then SW1SW2 = swPINRequired : Exit
  Message$=UserName
End Command

Command &H23 &H71 SetName(Message$)
  if (dontchange) then SW1SW2 = swAllreadySet : Exit
  If Not (PINVerified) Then SW1SW2 = swPINRequired : Exit
  UserName=Message$
  dontchange=true
End Command

Command &H23 &H72 GetPassword(Message$)
  If Not (PINVerified) Then SW1SW2 = swPINRequired : Exit
  Message$=UserPass
End Command

Command &H23 &H73 SetPassword(Message$)
  If Not (PINVerified) Then SW1SW2 = swPINRequired : Exit
  UserPass=Message$
End Command
```

Das Beispiel zeigt die Implementation der Befehle zum Speichern und Auslesen des Namens und des Passworts im EEprom der BasicCard. Der Zugriff auf die Daten kann nur nach erfolgreicher PIN-Authentifizierung erfolgen.

4.6.3. Datenspeicherung auf der Comcard MFC 4.3

Die Comcard MFC 4.3 verfügt über ein Filesystem nach ISO-7816. Zusätzlich bietet der Treiber `cccsigit` Funktionen, die dem PKCS#15 Standard entsprechen.

Interessant sind für die Anwendung die Plain-Text Files und die Schlüsselspeicher. In den Plain-Text Files werden der Benutzername und das Passwort für das Bestellwesen gespeichert.

Beim Anlegen eines Files werden die Zugriffsrechte mit angegeben.

Diese FileID dient dem System später zum Wiederfinden der Daten

In Abbildung 12 ist die Anordnung der Daten auf der ComCard dargestellt. In den EF_KEY_ Dateien werden die Directoryinformationen, wie FileIDs, gespeichert.

Es kann auch, wie das unten stehende Beispiel zeigt, ohne Kenntniss der FileID auf die Daten zugegriffen werden. Hierfür durchsucht das Betriebssystem die Dateien nach den angegebenen Inhalten oder Attributen.

Das Suchen mit Hilfe des PKCS-Wrappers nach dem Usernamen:

```
 Data dataObjectTemplate = new Data();
 // set the data object's label
dataObjectTemplate.getLabel().setCharArrayValue
            ("Bestellwesenuser".toCharArray());
 // create object
 session.createObject(dataObjectTemplate);
 session.findObjectsInit(dataObjectTemplate);
 Object[] foundDataObjects = session.findObjects(1);
```

Bei dieser Suchmethode muss jedes File nach den Attributen durchsucht werden. Dies ist zwangsläufig langsamer, als die direkte Auswahl der Datei.

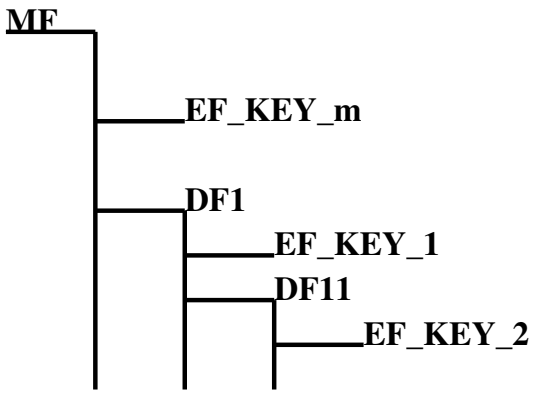

Abbildung 12: Filesystemaufbau der Comcard MFC 4.3

Eine Suche läuft intern auf der Karte folgendermaßen ab:

1. Wenn die FileID des gesuchten Files nicht bekannt ist, wird diese berechnet.
2. Im aktuellen Directory wird nach der FileID gesucht.
3. Ist diese nicht in diesem Verzeichnis, wird in die nächst höheren Ebene gewechselt.
4. Die Suche wird solange fortgesetzt, bis
   - das gesuchte File gefunden ist
   - das Masterfile erreicht ist. Dann wird eine Fehlermeldung ausgegeben.

4.7. Sicherheit

4.7.1. PIN

Eine Chipkarte darf Daten und Aktionen nur dann ausführen, wenn sichergestellt ist, dass der richtige Benutzer mit der Karte arbeitet. Die Karte darf also nur vom Besitzer bedient werden können.
Um die Authentizität des Benutzers sicherzustellen, wurde die PIN, "Personal Identification Number", eingeführt.

Die PIN ist in der Regel eine 4-8 stellige Zahlenfolge, die nur dem Besitzer der Karte bekannt sein darf. Die meisten Karten unterstützen ein Komnando, mit dem die PIN geändert werden kann.

In den meisten Fällen beginnt eine Kommunikationssession mit der PIN-Eingabe. Wird diese richtig eingegeben, wechselt die Chipkarte in den "Authentifiziert Modus" (siehe Abbildung 16).

Dieser Modus erlaubt es, Befehle auszuführen, die sonst nicht möglich wären.

Bei einer programmierbaren Karte wie der BasicCard muss die PIN Überprüfung selbst implementiert werden.

```
EEProm PIN=1234
authentifiziert=no
funktion dummy
  if authentifiziert==no
  than
    exit
  else
    tuwas
  endif
function pinpruefung (eingabedaten)
 if PIN==eingabedaten
 than
    authentifiziert=yes
 else
    exit
 endif
```

Mit diesem Programmcode kann man dies erreichen. Die Variable PIN steht im EEprom, ist also fest auf der Karte gespeichert. Die Variable "authentifiziert" wird im RAM gehalten.
Da die Variable im RAM gehalten wird, ist sie nach einem Kontaktverlust der Chip-

karte wieder im Ausgangszustand.

Die PIN ist die einzige Möglichkeit, die Zuordnung “Chipkarte–Benutzer” herzustellen. Daher darf die PIN nicht weitergegeben werden.

Ein Sicherheitsproblem ist wärend der vorliegenden Arbeit aufgefallen. Einige Applikationen setzen die Chipkarte, nachdem die PIN eingegen worden ist, nicht in einen sicheren Zustand zurück. So kann es unter Umständen zu einem Missbrauch kommen.

Daher gilt immer: **Chipkarten nie länger im Chipkartenleser lassen, als nötig!**

4.7.2. SO-PIN

Die SO-PIN, Security Officer-PIN, dient zur Initialisierung der Chipkarte. Diese Funktion ist bei Chipkarten, die als Signaturkarten angeboten werden, vorhanden. Mit dieser PIN kann die Karte gelöscht, die User-PIN zurückgesetzt oder einfach auch gesperrt werden.

Die SO-PIN muss daher ebenfalls geheim bleiben. Zur zusätzlichen Sicherheit ist die SO-PIN wesentlich länger als die normale PIN.

4.7.3. Andere Entwicklungen

Damit die Chipkarte eindeutig einer Person zugeordnet werden kann, werden neuerdings andere Methoden wie Fingerabdruckscanner oder Hautwiderstandsmessungen untersucht.

4.7.4. Hardwaresicherheit

Eine Chipkarte muss die Daten sicher verwahren. Es darf für niemanden die Möglichkeit geben, ohne Berechtigung an die Daten zu gelangen.
Die erforderlichen Anstrengungen, um widerrechtlich an Daten der Chipkarte zu gelangen, müssen wesentlich teurer sein, als die dadurch beschafften Daten.

Auf einer Chipkarte werden viele verschiedene Verfahren zur Datensicherheit eingesetzt. Viele dieser Verfahren basieren auf Geheimhaltung von Produktionseigenschaf-

ten.

Dies ist kein Widerspruch zum “Kerkofschen Prinzip”, da eine Chipkarte als Schlüssel angesehen wird und nicht als Verfahren.

Beispiele sind:

- Verwinkelte und nutzlose Leiterbahnen im Chipdesign

  Diese Methode erschwert die Auflösung der Strukturen, etwa mit einem Elektronenmikroskop.

- konstanter Stromverbrauch

  Es wird dafür gesorgt, dass der Stromverbrauch einer Chipkarte unabhängig von den eingegebenen Daten ist. Durch diese Methode kann nicht aus eventuellen Stromschwankungen auf Schlüssel oder Rechenalgorithmen geschlossen werden.

- Zufälliges Speichern im EEprom

  Beim Speichern von Daten werden zufällige Speicherseiten im EEprom ausgesucht. Dadurch bleibt verborgen, welche Stellen im EEprom die relevanten Daten enthalten.

Sollte es trotz aller Schwierigkeiten gelingen, an Daten der Chipkarte zu gelangen, muss dafür Sorge getragen werden, dass der dadurch verursachte Schaden gering bleibt. So werden bei Chipkarten, die als Schlüsselspeicher verwendet werden, sogenannte diversifizierte Schlüssel verwendet. Aus einem Hauptschlüssel heraus werden die Anwendungsschlüssel abgeleitet.
Wird ein Anwendungsschlüssel bekannt, kann aus dem Hauptschlüssel ein neuer generiert werden.

Weitere Informationen dazu und zum Aufbau der Chipkarten siehe [RANKL99] [IX].

4.8. Chipkartenleser

4.8.1. Klassen von Terminals

Chipkartenleser oder Terminals lassen sich in drei Sicherheitsklassen einteilen.
Klasse 1 ist die unsicherste Klasse.

- Klasse 1

  Ein Terminal dieser Klasse hat keine Tastatur oder Eingabemöglichkeit, ausser der seriellen Verbindung. Diese Leser sind weit verbreitet, da Banken diese ihren Kunden zum Einstieg in das HBCI-Homebanking anbieten.

  Wegen der geringen Sicherheitsanforderungen, sind diese Leser sehr kostengünstig.

  Als Beispiel sei der Towitoko Micro 130 genannt.

- Klasse 2

  Ein Klasse 2 Leser verfügt über eine eingebaute Tastatur. Oft ist dies eine 10-er Tastatur zur PIN Eingabe. Erwähnt werden sollen noch die Cherry PC-Tastaturen mit eingebautem Chipkartenleser.

- Klasse 3

  Ein Klasse 3 Leser verfügt zusätzlich zu der Tastatur noch über ein Display, welches, passende Software und Karten vorausgesetzt, die an die Karte geschickten Befehle in einer lesbaren Form darstellt.

  Diese Art von Chipkartenleser ist teuer und hat sich nicht durchgesetzt. Eine Anwendung muss für einen Klasse 3 Leser angepasst sein. Ist sie das nicht, verhält sich der Leser wie ein Klasse 2 oder 1 Leser. Durch die geringe Verbreitung der Leser wurden viele Anwendungsprogramme nicht angepasst.

Bei Notebooks mit eingebautem Chipkartenleser ist es wichtig, auf die Klasse des Chipkartenlesers zu achten.

Oft werden diese Notebooks als “sicher” angepriesen, obwohl es sich um einen versteckten Klasse 1 Leser handelt. Ein Notebook sollte mindestens ein Klasse 2 Gerät eingebaut haben.

Zum Testen standen zwei Klasse 1 Leser und eine Tastatur mit eingebautem Chipkartenleser zur Verfügung.

Abbildung 13: Towitoko Micro 130

Der Towitoko Chipkartenleser (Abbildung 13) wird an die serielle Schnittstelle angeschlossen. Der zusätzliche PS/2 Anschluss dient der Stromversorgung.

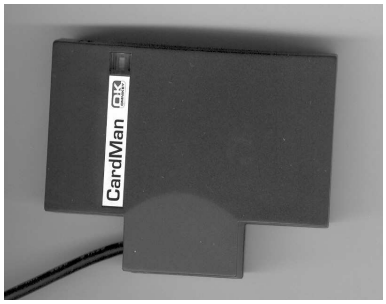

Abbildung 14: Omnikey Cardman 2020

Der Omnikey Leser wird über die USB Schnittstelle angesprochen.Bei der USB Schnittstelle wird der Chipkartenleser über diese mit Strom versorgt. Weiterhin zeichnet sich diese Schnittstelle durch höhere Übertragungsraten aus.
Dies ist in der Anwendung des Bestellwesens zwar nicht zu merken, da nur kurze Nachrichten an die Karte geschickt werden.

Bei der Anwendung des Bestellwesens hat die unterschiedliche Treiber und Betriebssystemauswahl eine größere Rolle gespielt.

5. PKCS

In der Kryptographie wird seit 1991 an dem PKCS, dem “Public-Key Cryptography Standard” gearbeitet. Herausgeber ist die RSA–Security [Lrsa]. Dieser Standard hat sich international bereits für viele Systeme durchgesetzt.

Zur Zeit besteht dieser Standard aus 15 verschiedenen Abschnitten. In Tabelle 8 findet man einen Überblick. Die neuesten Versionen sind unter [Lrsa] zu finden.

Der Standard beschreibt in den einzelnen Abschnitten das Format von Daten und Schlüsseln für den problemlosen Datenaustausch zwischen verschiedenen Kryptosystemen.

PKCS#1	RSA Cryptography Standard
PKCS#2	Note below
PKCS#3	Diffie-Hellman Key Agreement Standard
PKCS#4	Note below
PKCS#5	Password-Based Cryptography Standard
PKCS#6	Extended-Certificate Syntax Standard
PKCS#7	Cryptographic Message Syntax Standard
PKCS#8	Private-Key Information Syntax Standard
PKCS#9	Selected Attribute Types
PKCS#10	Certification Request Syntax Standard
PKCS#11	Cryptographic Token Interface Standard
PKCS#12	Personal Information Exchange Syntax Standard
PKCS#13	Elliptic Curve Cryptography Standard
PKCS#15	Cryptographic Token Information Format Standard

Note: PKCS#2 and PKCS#4 have been incorporated into PKCS#1

Tabelle 8: PKCS Standards

Wichtig für die Digitale Signatur mit Chipkarten sind die Standards

- PKCS#11 - Cryptographic Token Interface Standard

  Dieser Standard beschreibt die Kommunikation der Chipkarte mit dem Benutzerprogramm. PKCS#11 ist auch unter dem Namen “Cryptoki” bekannt.

- PKCS#10 - Certification Request Syntax Standard

  Dieser Standard wird verwendet, um Zertifikate an eine CA zum Unterschreiben und Authentifizieren zu schicken. Er beschreibt das Aussehen der Botschaften und die notwendigen Felder.

- PKCS#15 - Cryptographic Token Information Format Standard

  Hier wird das Speichern von Zertifikaten und Daten auf einer Chipkarte beschrieben.

5.1. Chipkarten und PKCS

In PKCS#11 wird von “Cryptographic Tokens” gesprochen, womit kryptographische Hardware im allgemeinen gemeint ist. Eine Chipkarte zum Verschlüsseln und Signieren ist somit ein Token.

Eine Chipkarte, die den Standard PKCS#11 unterstützt, wird vom Hersteller in der Regel mit einem Treiber, der die Umsetzung übernimmt, ausgeliefert. Nicht jede Chipkarte unterstützt diesen Standard. Für die BasicCard existiert Beispielsweise kein solcher Treiber.

In dem Standard werden Funktionen vorgeschrieben, die nicht alle auf einer Chipkarte Platz finden. So werden zum Beispiel verschiedene Verschlüsselungsverfahren gefordert von denen eine Chipkarte in der Regel nur eines unterstützt.
Bei der Comcard MFC 4.3 (Abbildung 15) wird der Treiber `cccsigit.dll` verwendet, um die Funktionen zur Verfügung zu stellen. Dieser Treiber ist nur für Windows erhältlich. Andere Chipkartenhersteller bieten Treiber auch für andere Betriebssysteme an.

Bei der Entwicklung eines Programmes ist darauf zu achten, dass die verwendeten Chipkarten die geforderten Funktionen unterstützen.

Abbildung 15: Comcard

PKCS#11 unterscheidet zwischen den verschiedenen Sicherheitsmodi der Chipkarte (siehe Abbildung 16).

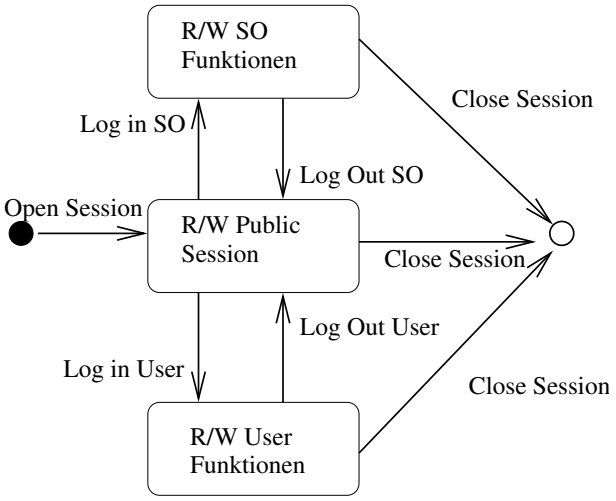

Abbildung 16: Modi bei PKCS#11

5.1.1. Java und PKCS

Es existieren für die PKCS#11 Funktionen zwei verschiedene Java Umsetzungen:

- IBM

  `http://www.alphaworks.ibm.com/tech/pkcs`

- IAIK Universität Graz

  `http://jce.iaik.tugraz.at/products/15_PKCS11_Provider/index.php`

Beide Implementationen sind unter einer freien Lizenz erhältlich. Die IBM Lösung steht unter der IBM eigenen Open Source Lizenz “Alphaworks”, die Lösung von IAIK unter der GPL(GNU GENERAL PUBLIC LICENSE).

Für die Umsetzung der digitalen Signatur in das Bestellwesen der FH Ravensburg-Weingarten wurde der IAIK-PKCS11-Wrapper verwendet, da hierfür von derselben Stelle auch eine Erweiterung der javax.security Module angeboten wird. Diese Erweiterung unterstützt RSA und ECC Verschlüsselungen, die auf den verwendeten Chipkarten eingesetzt werden.

Um auf ein Token zuzugreifen, muss die Library geladen werden.

```
Module pkcs11Module = Module.getInstance("cccsigit");
pkcs11Module.initialize(null);
```

Dieser Codeabschnitt lädt die PKCS#11 Treiber der Chipkarte, so das mit Java darauf zugegriffen werden kann.

Damit die Library geladen werden kann, muss diese im Systempfad des Rechners liegen.

Die Library wird beim Installieren der Chipkartensoftware auf dem Rechner angelegt. Wird keine Software für die Chipkarte installiert, muss die Datei `cccsigit.dll` auf dem Rechner in das Windows Stamm-Verzeichnis (`c:/win`) kopiert werden.

5.2. Erstellung eines x509 Zertifikats

Die Erstellung eines Zertifikates mit Hilfe einer Chipkarte ist eine wichtige Funktion. Hierbei darf der geheime Schlüssel die Chipkarte nie verlassen. Dies wird vom Betriebssystem der Chipkarte kontrolliert.

Um die erforderliche Zertifizierung des Schlüssels von der CA zu erhalten, muss die Chipkarte die Funktionen, ein Schlüsselpaar zertifizierungsfähig zu machen, unterstützen.
Stichworthaft kann man die Erstellung eines x509 Zertifikates in diesen Schritten beschreiben:

1. Generierung eines PKCS#11 Paares auf der Karte
2. Generierung eines PKCS#10 Requestes auf der Karte oder auf dem Rechner
3. senden des Requestes an die CA
4. Zertifizierung des Requestes durch die CA, Generierung des x509 Zertifikats
5. Importierung dieses Zertifikats auf die Chipkarte

5.2.1. Generierung eines Schlüsselpaares

Bei der Generierung eines Schlüsselpaares müssen viele Parameter gesetzt werden. Diese werden später zum Auffinden des Private- oder auch Public-Keys verwendet.

```
 Mechanism keyPairGenerationMechanism =
           Mechanism.RSA_PKCS_KEY_PAIR_GEN;
 RSAPublicKey rsaPublicKeyTemplate = new RSAPublicKey();
 RSAPrivateKey rsaPrivateKeyTemplate = new RSAPrivateKey();

// set the general attributes for the public key
rsaPublicKeyTemplate.getModulusBits().
                setLongValue(new Long(1024));
byte[] publicExponentBytes = {0x01, 0x00, 0x01}; // 2^16 + 1
rsaPublicKeyTemplate.getPublicExponent().
                setByteArrayValue(publicExponentBytes);
rsaPublicKeyTemplate.getToken().setBooleanValue(Boolean.TRUE);
byte[] id = new byte[20];
new Random().nextBytes(id);
rsaPublicKeyTemplate.getId().setByteArrayValue(id);
...
rsaPublicKeyTemplate.getLabel().setCharArrayValue
                     ("bestellwesen".toCharArray());
...
rsaPrivateKeyTemplate.getSensitive().
                     setBooleanValue(Boolean.TRUE);
rsaPrivateKeyTemplate.getToken().
                     setBooleanValue(Boolean.TRUE);
rsaPrivateKeyTemplate.getPrivate().setBooleanValue(Boolean.TRUE);
rsaPrivateKeyTemplate.getId().setByteArrayValue(id);
//byte[] subject = args[1].getBytes();
//rsaPrivateKeyTemplate.getSubject().setByteArrayValue(subject);
rsaPrivateKeyTemplate.getLabel().setCharArrayValue
                      ("bestellwesen".toCharArray());
...
```

In dem obigen Beispiel wird das Label Feld mit dem Wert “bestellwesen” gesetzt. Für die Anwendung ist dies ausreichend. In der Implementation werden dennoch weitere Felder gesetzt. Dadurch ist es möglich, die Schlüssel der Chipkarte auch für E-Mail Signaturen oder Verschlüsselungen zu benutzen.

Das auf diese Weise erzeugte Schlüsselpaar ist ein PKCS#11 Paar, das noch durch die CA zertifiziert werden muss.

5.2.2. Generierung eines PKCS#10 Certificate Request

Um einen PKCS#10 Request zu erzeugen, wird ein PKCS#11 Schlüssel oder Schlüsselpaar benötigt.

Weiterhin sind die persönlichen Daten des Schlüsselbenutzers einzubringen. Diese müssen vom Verwalter der CA überprüft werden.
Zu den persönlichen Daten gehören:

- Vorname, Nachname
- Firma
- Länder-Schlüssel
- E-Mail Adresse.

Diese werden der Karte, und somit dem Zertifikat, in der im x509 Standard vorgesehenen Form übergeben:

```
CN:Max Kliche,O=FH-Weingarten,C=DE,EMAIL=max@kliche.org
```

Die Generierung des Zertifikates geschieht folgendermaßen:

```
RFC2253NameParser subjectNameParser =
          new RFC2253NameParser(nameandmail);
Name subjectName = subjectNameParser.parse();

CertificateRequest certificateRequest = new
           CertificateRequest(subjectPublicKey, subjectName);

Signature tokenSignatureEngine =
new PKCS11SignatureEngine("SHA1withRSA", session,
          Mechanism.RSA_PKCS, AlgorithmID.sha1);

AlgorithmIDAdapter pkcs11Sha1RSASignatureAlgorithmID =
       new AlgorithmIDAdapter(AlgorithmID.sha1WithRSAEncryption);
pkcs11Sha1RSASignatureAlgorithmID.setSignatureInstance
                    (tokenSignatureEngine);

java.security.PrivateKey tokenSignatureKey =
              new TokenPrivateKey(selectedSignatureKey);

certificateRequest.sign(pkcs11Sha1RSASignatureAlgorithmID
                       , tokenSignatureKey);
```

Diese Operationen erzeugen einen “Certificate Request”. Dieser muss jetzt, per Diskette oder E-Mail an die CA zum Signieren geschickt oder übergeben werden.

In dem Request steht der öffentliche Schlüssel. Der geheime Schlüssel bleibt immer auf der Chipkarte.
Ein Beispiel für einen Certificate Request:


```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBjDCB9gIBADBOMR0wGwYJKoZIhvcNAQkBDA5tYXhAa2xpY2hlLm9yZzELMAkG
A1UEBhMCREUxCzAJBgNVBAoTAkZIMRMwEQYDVQQDEwpNYXggS2xpY2hlMIGeMA0G
CSqGSIb3DQEBAQUAA4GMADCBiAKBgHc80uYFtBPEHLkHX5aB6Y6QKDsSh2peGHu1
OSWiKffJ7kLTNwT4f1w0RsoI4FEg1isQiJeTGc3h7Qh1QG5TQ3o/ZO5/VaXnA7eJ
4OPlhdSpc/PtHxr2D3ohEPTRFN4M1X1mxDa5Nu57DCK/ysoA3sxXV2OwCSN4W62v
qN8AggC5AgMBAAGgADANBgkqhkiG9w0BAQUFAAOBgQByVkmig8eCmKYT13weDSJ9
P+NM1UCQ/7eHrUuITR110WaQE3tYHzzVvMe0Idi8ArjU4WYSkAHa5GwGGmHi5rMl
i7QEgo6FMbbZqXp05UWHhANzgaNtL+gCyIosHstTR8HPi0ZUsXLgRpQS/EHYNeVF
RA3M2rF9mVXyPKK3ziUWIg==
-----END NEW CERTIFICATE REQUEST-----
```


Dieses File ist ein ASN1 codiertes Datenobjekt. Dies kann man mit

```
openssl asn1parse -in filename
```

aufschlüsseln. Alternativ liefert `xca` diese Ausgabe:

Abbildung 17: PKCS#10 Zertifikat aufgeschlüsselt mit `xca`

5.2.3. Zertifizierung des Zertifikates durch die CA

Nach dem Überprüfen der persönlichen Daten erstellt die CA ein Zertifikat für den Benutzer.
Dies geschieht mit Hilfe des Programmes `openssl` oder mit dem graphischen Frontend `xca`.
Das erstellte Zertifikat:


```
-----BEGIN CERTIFICATE-----
MIIDOTCCAqKgAwIBAgIBAjANBgkqhkiG9w0BAQQFADBtMQswCQYDVQQGEwJERTET
MBEGA1UEBxMKV2VpbmdhcnRlbjELMAkGA1UECBMCQlcxDzANBgNVBAoTBkZILVdH
VDEMMAoGA1UECxMDZmJlMR0wGwYJKoZIhvcNAQkBFg5yb290QGxvY2FsaG9zdDAe
Fw0wMzAxMjMxODEwMTFaFw0wNDAxMjMxODA3MThaME4xHTAbBgkqhkiG9w0BCQEM
Dm1heEBrbGljaGUub3JnMQswCQYDVQQGEwJERTELMAkGA1UEChMCRkgxEzARBgNV
BAMTCk1heCBLbGljaGUwgZ4wDQYJKoZIhvcNAQEBBQADgYwAMIGIAoGAdzzS5gW0
E8QcuQdfloHpjpAoOxKHal4Ye7U5JaIp98nuQtM3BPh/XDRGygjgUSDWKxCIl5MZ
zeHtCHVAblNDej9k7n9VpecDt4ng4+WF1Klz8+0fGvYPeiEQ9NEU3gzVfWbENrk2
7nsMIr/KygDezFdXY7AJI3hbra+o3wCCALkCAwEAAaOCAQcwggEDMAwGA1UdEwEB
/wQCMAAwgZcGA1UdIwSBjzCBjIAUw2Cb3Pl+gmN7UH9qWNJIj6PxiR+hcaRvMG0x
CzAJBgNVBAYTAkRFMRMwEQYDVQQHEwpXZWluZ2FydGVuMQswCQYDVQQIEwJCVzEP
MA0GA1UEChMGRkgtV0dUMQwwCgYDVQQLEwNmYmUxHTAbBgkqhkiG9w0BCQEWDnJv
b3RAbG9jYWxob3N0ggEBMAsGA1UdDwQEAwIEsDAZBgNVHREEEjAQgQ5tYXhAa2xp
Y2hlLm9yZzARBglghkgBhvhCAQEEBAMCBaAwHgYJYIZIAYb4QgENBBEWD3hjYSBj
ZXJ0aWZpY2F0ZTANBgkqhkiG9w0BAQQFAAOBgQA3Aw506Zv3nuBWZFrS6ccWjZK4
pGBxj9Bm673RZnKQqiBNJlnUT+1uFLVvz7kwZINisb3DCQcRoLEpbIeapB+0LjLT
Yx/CBA6Yu1CTZ6JYHYZiplCOoVeqYy6mSxpqX9G7U3O0siHkTh7no1jl9+noAWAO
8HSdtcJRKSdaDsMXiQ=
-----END CERTIFICATE-----
```


Siehe auch Abbildung 18.

Details des Zertifikates

Allgemeines

Name	out
Unterschrieben von:	Bestellwesen
Schlüssel:	nicht verfügbar
Serien Nr.:	02

Besitzer		Aussteller	
Land:	DE	Land:	DE / BW
Locality:		Locality:	Weingarten
Name / DNS:	Max Kliche	Name / DNS:	
Firma	FH	Firma:	FH-WGT
Abteilung:		Abteilung:	fbe
E-Mail:	max@kliche.org	E-Mail:	root@localhost

Gültigkeit

Jan 23 18:10:11 2003 GMT	Jan 23 18:07:18 2004 GMT	gültig

Fingerprint

MD5	4B:DC:78:72:2F:41:6B:A6:15:40:2B:EA:F1:B1:AD:F1
SHA1	4A:1A:90:93:BC:CE:46:55:10:E5:F5:2C:76:2E:2A:EC:DA:96:76:01

X509v3 Basic Constraints: critical:
CA:FALSE
X509v3 Authority Key Identifier:
keyid:C3:60:9B:DC:F9:7E:82:63:7B:50:7F:6A:58:D2:48:8F:A3:F1:89:1F
DirName:/C=DE/L=Weingarten/ST=BW/O=FH-WGT/OU=fbe/Email=root@localhost
serial:01
X509v3 Key Usage:
Digital Signature, Key Encipherment, Data Encipherment
X509v3 Subject Alternative Name:
email:max@kliche.org
Netscape Cert Type:
SSL Client, S/MIME
Netscape Comment:
xca certificate

Abbrechen OK

Abbildung 18: Zertifikat aufgeschlüsselt mit `xca`

5.2.4. Speichern des Zertifikates

Das Zertifikat muss auf der Chipkarte an der richtigen Stelle gespeichert werden, damit die Chipkarte die Beziehung “geheimer Schlüssel – CA” herstellen kann.

Dafür ist das Zertifikat zu untersuchen. Auf der Chipkarte müssen der dazugehörige Public- und Secret-Key gefunden werden.

Dann kann das Zertifikat auf der Chipkarte gespeichert werden.

```
...
\\ analyse
iaik.x509.X509Certificate x509Certificate =
     (iaik.x509.X509Certificate)
certificateFactory.generateCertificate(fileInputStream);
PublicKey publicKey = x509Certificate.getPublicKey();
Object searchTemplate = null;
....
\\suche
....
\\einspielen auf die karte
X509PublicKeyCertificate pkcs11X509PublicKeyCertificate
          = new X509PublicKeyCertificate();
Name subjectName = (Name) x509Certificate.getSubjectDN();
Name issuerName = (Name) x509Certificate.getIssuerDN();
String subjectCommonName = subjectName.getRDN(ObjectID.commonName);
String issuerCommonName = issuerName.getRDN(ObjectID.commonName);
char[] label = (subjectCommonName + "'s␣" +
         ((issuerCommonName != null) ? issuerCommonName + "␣" : "")
         +  "Certificate").toCharArray();

...
```

5.2.5. Signieren mit Hilfe des Zertifikates

Wenn die Schlüssel und Zertifikate auf der Karte gespeichert sind, ist das Signieren möglich.

```
...
Mechanism signatureMechanism = Mechanism.RSA_PKCS;
session.signInit(signatureMechanism, signatureKey);
byte[] signatureValue = session.sign(dataToBeSigned);
```

Vor der Ausführung dieses Codes muss der richtige Schlüssel ausgewählt werden. Die Suche nach dem Schlüssel ist in Kapitel 4.6 beschrieben.

Um die Signatur zu erstellen, muss vorher der Hash-Wert der Daten berechnet werden. Dies geschieht entweder mit dem `SHA-1` oder mit dem `md5` Verfahren. Hier hängt es von der Chipkarte ab, mit welchen Daten diese zusammenarbeitet. Für die Comcard MFC 4.3 wird SHA-1 verwendet.

```
MessageDigest digestEngine = MessageDigest.getInstance("SHA-1");
Mechanism signatureMechanism=Mechanism.RSA_PKCS;
session.signInit(signatureMechanism, signatureKey);
digestEngine.update(signString.getBytes());
byte[] digest = digestEngine.digest();
```

In PKCS#11 wird beim Signieren von Daten erwartet, dass diese bereits in der “gehashten” Form vorliegen. Der Hash-Wert wird auf Grund der höheren Geschwindigkeit auf dem Rechner und nicht auf der Chipkarte erstellt.

5.2.6. Überprüfung der Signatur

Die Überprüfung der Signatur erfolgt ebenfalls auf dem Rechner. Im Bestellwesen der FH Ravensburg-Weingarten ist dies nicht anders möglich, da der Server, der die Signatur überprüft, keine Chipkarte besitzt. (Die Abläufe des Bestellwesens sind in Kapitel 8 und insbesondere in Abbildung 22 dargestellt.)

Für die Überprüfung der Signatur wird die IAIK-Crypto-API verwendet, da die Javax Implementation von SUN die erforderlichen Verfahren nicht anbietet.

```
Signature dsa=Signature.getInstance("SHA1withRSA","IAIK");
// eine Signaturinstanz erzeugen
dsa.initVerify(pubKeyrsa);
// der Signaturinstanz den oeffentlichen Schluessel
// des Users zum Verifizieren uebergeben
// dieser Schluessel kommt aus der Datenbank

dsa.update(this.parseEncodedKey(clientDaten).getBytes());
System.err.println(this.parseEncodedKey(signierdaten));
// die Signierdaten noch mit base64endcoden
byte[] a=decoder.decodeBuffer(this.parseEncodedKey(signierdaten));
System.out.println(new BigInteger(1, a).toString(16));
System.err.println(a.toString());
ok=dsa.verify(a);//signierdaten.getBytes());
// die Signatur ueberpruefen
```

6. Treiber und Software

Zur Kombination und Kommunikation zwischen Chipkarten, Chipkartenleser und PC sind verschiedene Treiber erforderlich.

Auf einen Treiber für die Chipkarte kann verzichtet werden, wenn, wie bei der Implementation der BasicCard in das Bestellwesen oder im Test-Applet, die Chipkarte **direkt mit APDUs** angesprochen wird.

Die Kommunikation mit Chipkarten ist, was die Treiber und Middleware betrifft ähnlich dem OSI-Schichtenmodell aufgebaut. In Abbildung 19 ist dies an drei Beispielen dargestellt. Von diesen drei Modellen wurden die beiden ersten in dieser Arbeit verwendet.

6.1. PC/SC

Die PC/SC-Spezifikation (Personal Computer/Smartcard) [PCSC] regelt die Benutzung von Chipkarten unter Windows. Diese Schnittstelle bietet Funktionen zum Senden von APDUs an. Weitere Aufgaben sind der Verbindungsaufbau zur Chipkarte und die Analyse des ATR.

Die PC/SC Schnittstelle befindet sich in der Schicht direkt über dem Hardwaretreiber. Der Hardwaretreiber des Chipkartenlesers stellt die in der PC/SC-Spezifikation vorgesehenen Funktionen zur Verfügung.

Durch die weite Verbreitung hat sich diese Spezifikation durchgesetzt und es gibt Umsetzungen auf andere Betriebssysteme.

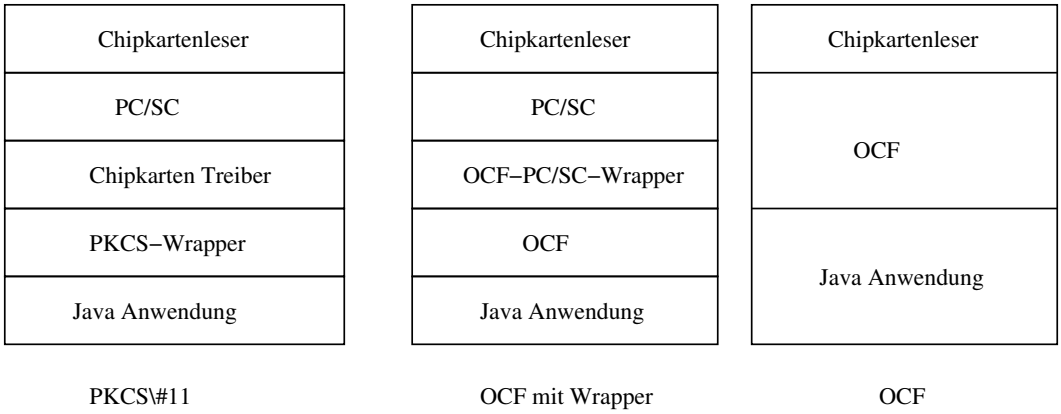

Abbildung 19: Vergleich der Treiber-Strukturen

6.1.1. Windows

Die PC/SC Schnittstelle ist seit Windows 2000 fest in das Betriebssystem integriert. Für andere Windows Versionen bietet Microsoft diese zum Download an.

Die Chipkartenhersteller bieten Treiber für ihre Hardware an. Diese Treiber unterstützen alle den Zugriff der PC/SC Schnittstelle auf die Hardware.

Chipkartenapplikationen unter Windows, wie Homebanking oder SmartCard Login benutzten alle die PC/SC Schnittstelle. Daher kann der Chipkartenleser in der Regel ausgetauscht werden, ohne dass eine Applikation umgeschrieben werden muss.

6.1.2. Linux

Für Linux hat sich unter dem M.U.S.C.L.E-Project (Movement for the Use of Smart Cards in a Linux Environment) eine Gruppe zusammengetan, die Hardwaretreiber für Chipkartenleser und eine PC/SC-Schnittstelle bereitstellt.

Diese “pcsc-lite” genannte Schnittstelle arbeitet als Daemon auf einem Linux System. Dieser Daemon (pcscd) nimmt Anfragen von Applikationen entgegen und reicht diese an den Chipkartenlesertreiber weiter. Die aktuelle Version ist (Januar 2003) pcsc-lite-1.1.1. Auf der Internetseite des Projektes findet man für die gängigen Distributionen RPM-Pakete zum installieren.

Der pcscd benötigt einen Hardware Treiber, der die Kommunikation mit dem Chipkartenleser übernimmt.

Es gibt nur wenige Hersteller von Chipkartenlesern, die auch Treiber für Linux entwickeln. Von den untersuchten Lesegeräten waren das

- **Omnikey**

  Diese Treiber arbeiten als Kernelmodul. Dieses muss mit Hilfe von `modprobe` in den Kernelspeicher geladen werden.

  Kerneltreiber unter Linux laufen in der Regel nur mit der Kernelversion, mit der sie compiliert worden sind. Der von OmniKey zur Verfügung gestellte Treiber funktioniert so nur unter der Linux-Distribution Mandrake mit Kernel 2.4.17-9. Um den Treiber auch auf einem anderem Linux System benutzen zu können, liefert OmniKey auf Anfrage den Quellcode des Treibers aus. Damit ist es möglich, den Treiber an das Vorhandene System anzupassen.

  Dabei traten bei den getesteten Linux Versionen Probleme auf, da die USB

Struktur im Linux Kernel geändert wurde. Damit der Chipkartenleser getestet werden konnte, musste der Treiber umgeschrieben werden.

Die Änderungen wurden an die Entwicklungsabteilung von OmniKey weitergegeben.

- **Cherry**

  Cherry bietet für einige Tastaturen Treiber an. Für die vorliegende Tastatur war leider kein Treiber verfügbar.

- **Towitoko**

  Für den Towitoko Chipkartenleser findet man für Linux auf der Linkliste des M.U.S.C.L.E Projektes einen Verweis auf einen Open-Source Treiber[5].

  Dieser Treiber ist auf dem System zu kompilieren und stellt daraufhin eine Library zur Verfügung.

  Die Library stellt den Kontakt zum Chipkartenleser nicht direkt her, sondern benutzt die serielle USB Schnittstelle oder USB. Wenn der Benutzer Schreib und Leserechte auf die serielle Schnittstelle besitzt, kann der pcscd mit Benutzerrechten auf dem System laufen. Dies ist bei der Verwendung des OmniKey Treibers nicht möglich, da Kernelmodule nur von “root” geladen werden können.

[5] `http://www.geocities.com/cprados/`

6.1.3. Installation des pscsd

Der pcscd sollte beim Systemstart gestartet werden. Bei der Installation der RPM-Pakete werden in den Startscripten entsprechende Links gesetzt.

Wird der pcscd nicht automatisch gestartet, kann dieser auf der Konsole mit

```
pcscd -d -f stdout
```

gestartet werden.

Der pcscd wird duch die Datei `/etc/readers.conf` konfiguriert.

```
# FRIENDLYNAME: Any name
# DEVICENAME: Any name
# LIBPATH: Location of the driver library for your reader
# CHANNELID:
#                0x000001 - COM1
# Towitoko Chipdrive Micro (COM1)
FRIENDLYNAME    "Towitoko Chipdrive Micro"
DEVICENAME      TOWITOKO_CHIPDRIVE_MICRO
LIBPATH         /opt/chipkarte/lib/libtowitoko.so.2.0.0
CHANNELID       0x000001
```

Diese Datei ist entsprechend der Hardware anzupassen. Im obigen Beispiel wird ein Towitoko Chipkartenleser an der ersten seriellen Schnittstelle verwendet.

6.2. OCF

Das Open Card Framework (OCF) ist eine Java Programmbibliothek, welche es ermöglicht, auf Chipkarten zuzugreifen.
Es existieren zwei Möglichkeiten, OCF zu benutzen:

1. Direkter Hardwarezugriff

   Hierfür werden Hardwaretreiber vom OCF bereitgestellt.

   Es gibt allerdings nur wenige Hardwaretreiber [OPENC]. Für die benutzten Chipkartenleser existierten keine, so dass diese Lösung in der Arbeit nicht weiter verfolgt wurde.

2. Zugriff über die PC/SC Schnittstelle

   Das OCF kann mit Hilfe des OCF-PC/SC-Wrappers auf die PC/SC Schnittstelle der Chipkartentreiber zugreifen.

   Diese Lösung wurde bei dem TestApplet und der BasicCard Implementation in das Bestellwesen benutzt.

- Die Verwendung des “pass through” Modus

  Bei diesem Modus wird jede Chipkarte angesprochen. Es werden direkt APDUs an die Chipkarte geschickt.

```
CardRequest cr = new
CardRequest(CardRequest.ANYCARD,
            null,
            PassThruCardService.class);
cr.setTimeout(TIMEOUT);  // set timeout for IFD

SmartCard sc = SmartCard.waitForCard(cr);
  if(sc != null){
   ptcs = (PassThruCardService)
          sc.getCardService(PassThruCardService.class, true);
  cardID=sc.getCardID();
```

Listing 2: Initialisierung des PassThruCardService

In dem TestApplet Jar Paket ist die Klasse “MyChipcard” enthalten. Dort findet sich weiterer Code und Erklärungen zu OCF [kliche].

7. Test Applet

Unter `http://www.wrankl.de` und bei [RANKL99] findet sich ein Chipkartensimulator. Dieses Programm arbeitet ohne eine Chipkarte und ist dafür gedacht, das ein Benutzer sich mit dem Verschicken von APDU-Folgen und dem Filesystem auf Chipkarten vertraut machen kann.

Das Programm simuliert eine Chipkarte in Software. um eine APDU Sendesequenz kennenzulernen. Da die Kommunikation mit einer simulieren Chipkarte stattfindet ist dieses Programm nicht zum Test einer realen Chipkarte geeignet.

Für die vorliegende Arbeit war es erforderlich, den Datenverkehr von und zur Chipkarte zu analysieren und darzustellen.

Dazu wurde ein Chipkarten Test-Applet entwickelt.

Dieses Programm ermöglicht es, mit einer beliebigen Chipkarte über den Webbrowser zu kommunizieren. Es erwartet eine APDU als Eingabe und gibt die Antwort der Chipkarte als String aus (siehe Bild 20).

7.1. Anforderungen

Die Anforderungen zur Benutzung des Applets sind:

- Chipkartenleser mit PC/SC Treiber
- Chipkarte
- Internetbrowser mit Java Unterstützung

Die Hardware-Treiber müssen auf dem System installiert sein. Unter Linux muss der pcscd laufen.

7.2. Bedienung

Die Bedienung gliedert sich in drei Teile.

- Kontakt zu der Karte herstellen

  Eine Karte wird mit dem System verbunden, sobald man den Connect Button drückt, nicht schon beim Einsetzen der Karte in den Leser. So kann das Applet

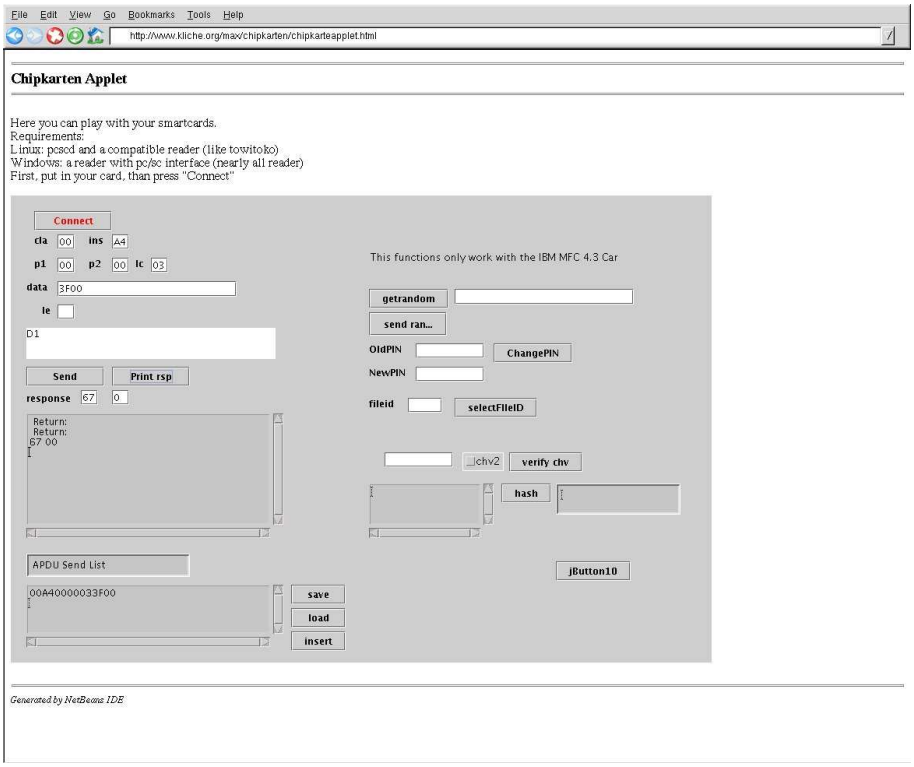

Abbildung 20: Chipkarten Kommunikation

in einem Browserfenster im Hintergrund laufen und eine Chipkartenapplikation kann dennoch auf den Leser und die Chipkarte zugreifen.

Bei dem “Connect” zur Chipkarte wird die OCF Instanz PathThrough initialisiert und erst nach dem Entfernen der Chipkarte aufgehoben.

Um die Karte neu zu initialisieren, genügt es, Connect erneut auszuführen.

- Senden eines Befehls an die Karte

  Ein Befehl setzt sich aus den in Kapitel 4.4 beschriebenen Teilen zusammen.

  Eingabefelder für die einzelnen Teile finden sich in der oberen Hälfte des Applets.

  In diese Felder werden die gewünschten Daten als Hex-Werte eingetragen. Hierbei kann das normalerweise vorangestellte “0x” weggelassen werden.

  Ein Beispiel zur Verdeutlichung:

Der Befehl

$$0x00 \;\; 0xA4 \;\; 0x00 \;\; 0x00 \;\; 0x00$$

zur Selection des Master-Files einer ISO-7816 Chipkarte wird folgendermaßen kodiert:

Feld	Eingabe
cla	00
ins	A4
p1	00
p2	00
lz	00

- Abrufen der Antwort

  Die Statuswerte S1 und S2 werden direkt nach dem Absenden des Kommandos an die Chipkarte in den Feldern eingetragen.

  Um die komplette Antwort zu erhalten, muss der “PrintRsp” Button benutzt werden.

  Die Antwort wird als Hex-String dargestellt.

7.3. Speicherung von APDU Folgen

Mit Hilfe der Buttons

- Load
- Save
- Insert

können APDU-Folgen auf dem PC gespeichert werden. Diese werden in einem Text File abgelegt. So ist es möglich, diese mit Hilfe eines Editors zu bearbeiten.

Die gespeicherten APDU-Folgen können mit Hilfe von Load wieder in die APDU Send List geladen werden. Aus dieser Liste kann eine APDU ausgewählt und mit Hilfe von Insert wieder in die entsprechenden Felder eingefügt werden.

8. Bestellwesen

Das System “Bestellwesen” bot bislang den Benutzern die Möglichkeit, eine Bestellung mit Hilfe einer Diskette und dem darauf abgelegtem Schlüssel zu signieren.

Das System überprüft die Signatur dann mit Hilfe des in einer Datenbank abgelegten öffentlichen Schlüssels, und nur bei einer erfolgreichen Verifikation wird die Bestellung weitergeleitet (siehe Abbildung 22).

Dieses System wurde auf die parallele Nutzung von Chipkarten erweitert. Die alte Lösung musste bestehen bleiben, da die Benutzer noch nicht alle mit Chipkarten und Chipkartenlesern ausgestattet sind.

8.1. Programmablauf

Um eine Signatur mit Hilfe von Disketten oder einer Chipkarte zu erstellen, wird folgendermaßen vorgegangen:

- Disketten

  Es wird der Hash-Wert der Daten auf dem Computer generiert. Der Secret-Key wird von der Diskette geladen. Der Hash-Wert wird auf dem Computer mit Hilfe des `DSA-Algorithmus` signiert.

  Die Signatur wird mit dem auf der Datenbank gespeichertem Public Key vom Server überprüft. Danach wird eine E-Mail an den Empfänger versendet.

- Chipkarten

  Bei der Chipkarte wird der Hashwert des Textes auch im Computer berechnet, dieser Hashwert dann aber auf der Chipkarte zur Generierung der Signatur verwendet. So hat der Rechner nichts mehr mit der Signaturbildung zu tun. Der Secret Key kommt somit niemals in den Speicher des Computers und kann daher nicht von einem unbefugtem Benutzer ausgelesen werden.

8.1.1. Die entwickelten Klassen

Das alte Programm bestand aus zwei Teilen. Der erste Teil wird vom Benutzer kontrolliert. Der zweite Teil vom Server. An dieser Aufteilung sieht man, dass nur der erste Teil mit Chipkarten arbeiten kann. Der Server selbst hat keine Chipkarten.

Damit die Integration in diese Programmstruktur einfach gestaltet werden konnte, wurden die für die Signatur wichtigen Funktionen in eine eigene Klasse ausgelagert. Bislang waren diese in der Hauptklasse.

Die Klassen wurden so entwickelt, dass diese ohne das Bestellwesen funktionieren. Es ist leicht möglich, diese Klassen in ein eigenes Programm zu importieren.

Diese Klasse (`FHSmartCard.java`) bietet unter anderem folgende Funktionen:

- SignData()
- CheckSignData()
- GetPublicKey()
- GetName()
- GetPassword()
- GenerateKeyPair()

Diese Klasse ist eine virtuelle Schnittstelle für weitere Klassen. Von dieser Klasse werden daher, ja nach Anforderung, diese Klassen abgeleitet (siehe Abbildung 21):

- NoChipCard

  Diese Klasse arbeitet mit der Hilfe der Disketten. Hier stehen die alten Funktionen zur Verfügung.

- BasicCard

  Diese Klasse bietet bislang die Funktionen SignData(),GetName() und GetPassword().

- ComCard

  Diese Klasse implementiert die PKCS#11 Lösung, die mit dieser Karte möglich ist und bietet hiermit die volle Funktionalität.

Mit Hilfe dieser Klassen ist es möglich, die Applikation an die verschiedensten Chipkarten oder Hardware anzupassen.

Unterschieden werden die einzelnen Chipkarten anhand des ATR, wie in Kapitel 4.3.1 beschrieben wurde.

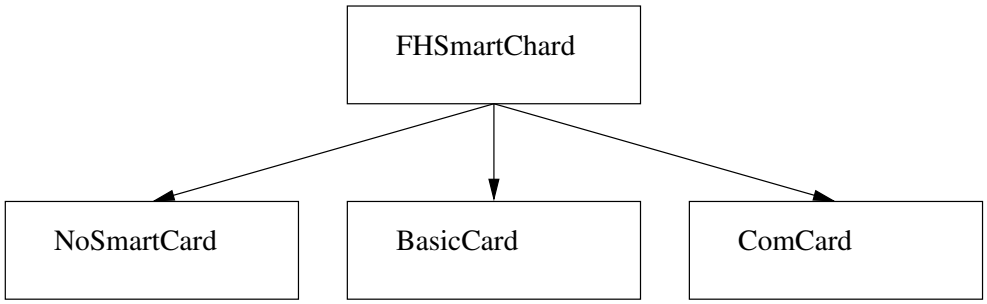

Abbildung 21: Klassen Diagramm

8.1.2. Programmablauf

Der Programmablauf hat sich für den Benutzer gegenüber der alten Lösung nichts wesentliches geändert. Die wichtigste Änderung für den Benutzer ist, dass er bei der Signatur keine "Passphrase" mehr eingeben muss, sondern die PIN der Chipkarte.

Weiterhin kann sich der Benutzer mit Hilfe der Chipkarte an dem System anmelden. Die Benutzerdaten, Username und Passwort, werden von der Chipkarte nach Eingabe der PIN gelesen. Ein Anmelden per Tastatur und Passwort ist weiterhin möglich, da das Auslesen der Daten aus der Chipkarte lange (mehr als 5 Sek.) dauert.

Intern haben sich u.a. folgende Änderungen ergeben:

- Die Signaturüberprüfung muss auf verschiedene Systeme eingehen.

  Die Datenbank wurde um ein Feld erweitert, in dem gespeichert wird, ob und welche Chipkarte verwendet wird. So kann vom Server die richtige Funktion zur Überprüfung aufgerufen werden.

- Die Speicherung des Schlüssels

  Bei einer Chipkarte wird als Schlüssel das Zertifikat eines Benutzers angelegt. Dieses wird nicht, wie in der alten Version, noch einmal zusätzlich verschlüsselt.

8.1.3. Schlüssel generieren

Bei der Schlüsselgenerierung mit einer Diskette war es einem Benutzer möglich, sein Schlüsselpaar zu erzeugen und sofort mit diesem zu arbeiten.

Durch die Einführung der CA für das Bestellwesen, muss der Benutzer sich den Schlüssel erst signieren lassen. Dies führt zu einem Mehraufwand für den Benutzer sowie für den Verwalter der CA. Durch diesen Mehraufwand erhöht sich aber die Sicherheit des Schlüssels und das Vertrauensverhältnis.

Um einen Schlüssel auf der Chipkarte zu generieren, sind mehrere Schritte nötig.

1. Akzeptieren des Zertifikates für das Java Applet

   Das Java Applet muss auf die Hardware des Rechners zugreifen. Dafür ist es mit Hilfe eines Zertifikates signiert worden.

   In Bild 24 ist das zu bestätigende Formular zu sehen.

2. Anmelden an der Datenbank und Generierung des Schlüsselpaares

   Bei der Generierung des Schlüsselpaares wird der Certification-Request in eine Datei gespeichert, die vom Benutzer wählbar ist.

3. Zertifizierung

   Die im letzten Schritt erzeugte Datei muss der CA zur Zertifizierung vorgelegt werden. Die CA erstellt dann das x509 Zertifikat für den Benutzer. Dieses Zertifikat bekommt der Benutzer in Form einer Datei auf Diskette wieder.

4. Speichern des Zertifikates

   Das Zertifikat wird vom Benuzter wieder mit Hilfe des GeyGenerator-Applets auf die Chipkarte gespeichert.

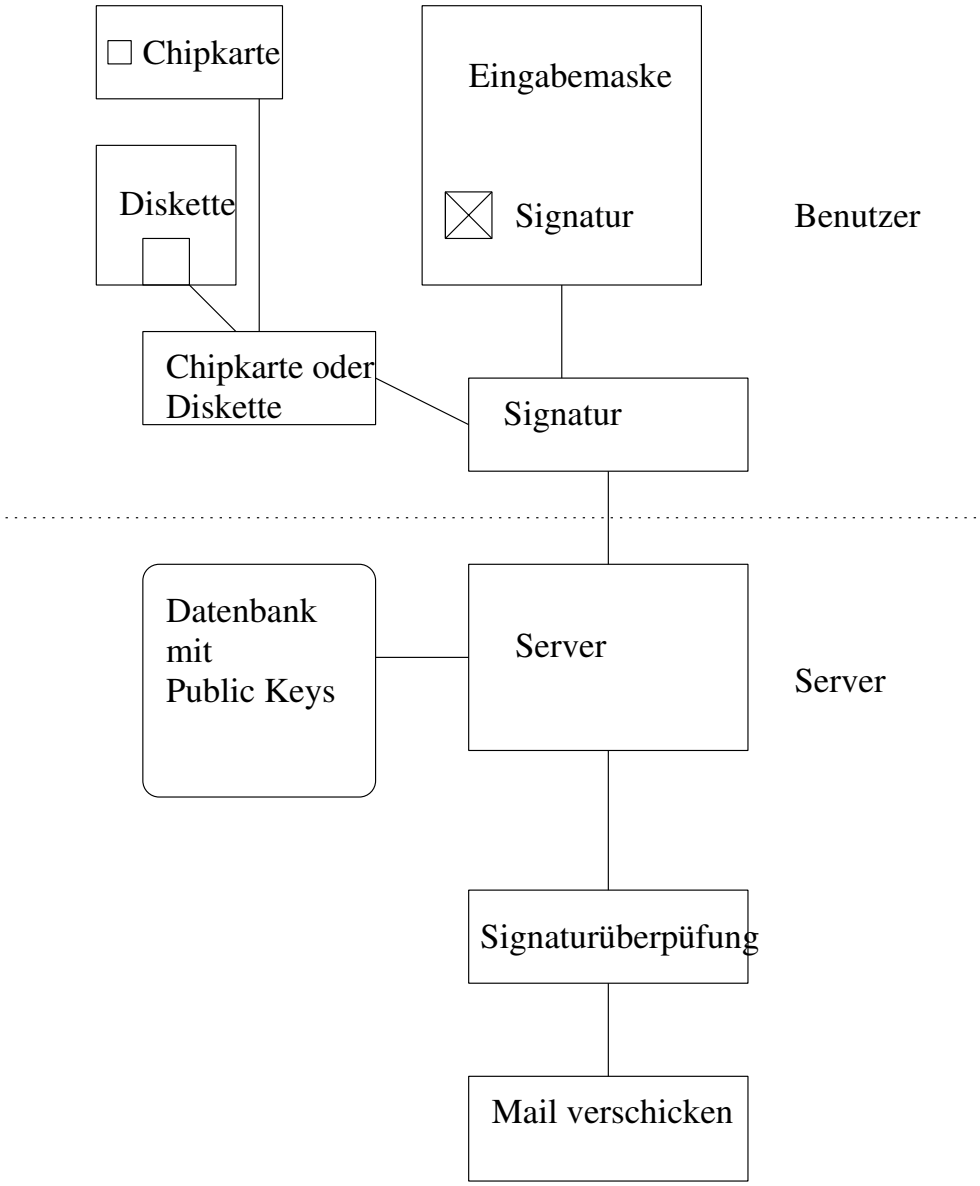

Abbildung 22: Programmablauf Signatur

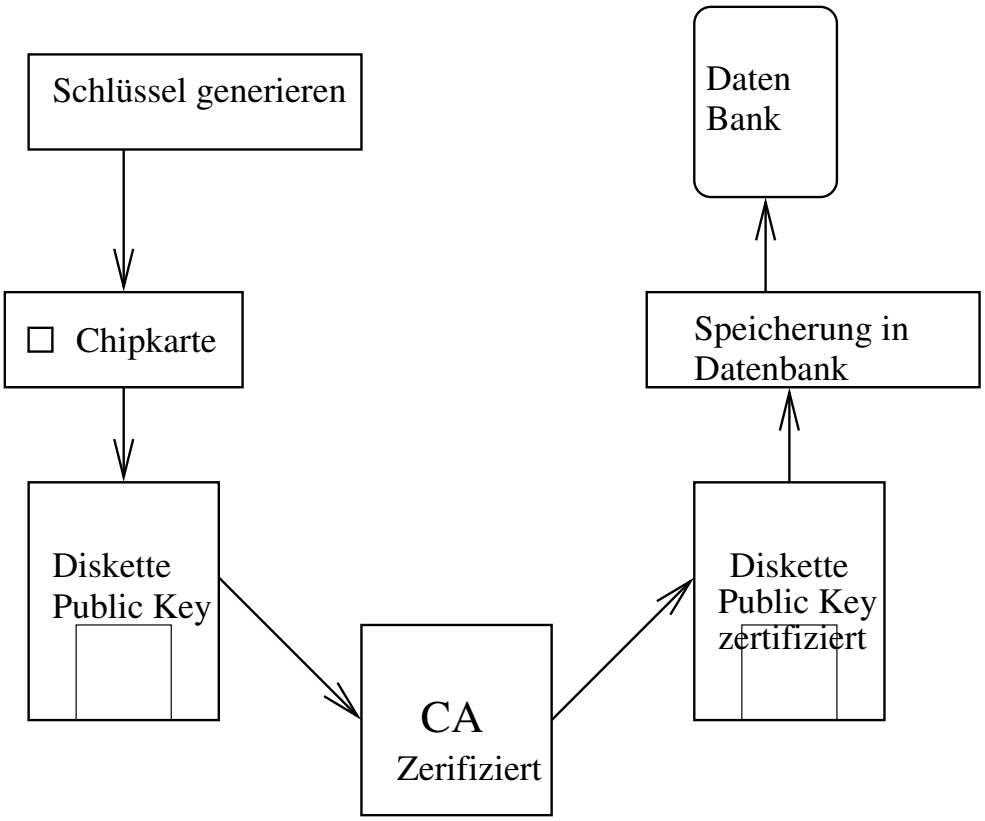

Abbildung 23: Programmablauf Schlüsselgenerierung

Abbildung 24: Applet-Zertifikat

8.2. Signatur

Aus dem Bestellauftrag wird im System ein String zusammengesetzt, der als Dateneingabe für die Signatur dient.

Im Falle der Comcard MFC 4.3 wird die Signatur wie folgt erzeugt:

```
private String pkcsSignData(String signString)
{
Token token_;
try{
 Module pkcs11Module = Module.getInstance("cccsigit");
 pkcs11Module.initialize(null);
 Slot[] slots=pkcs11Module.getSlotList
        (Module.SlotRequirement.TOKEN_PRESENT);
 Slot selectedSlot = slots[0];
 token_=selectedSlot.getToken();
 Session session=
 token_.openSession(Token.SessionType.SERIAL_SESSION,
        Token.SessionReadWriteBehavior.RO_SESSION, null,null);

 session.login(Session.UserType.USER, jPasswordField1.getText()
           .toCharArray());
 // Suchen nach dem Schluessel
 iaik.pkcs.pkcs11.objects.RSAPrivateKey templateSignatureKey=
                 new iaik.pkcs.pkcs11.objects.RSAPrivateKey();

 templateSignatureKey.getSign().setBooleanValue(Boolean.TRUE);
 session.findObjectsInit(templateSignatureKey);
 iaik.pkcs.pkcs11.objects.Object[] foundSignatureKeyObjects
               = session.findObjects(1);

 iaik.pkcs.pkcs11.objects.RSAPrivateKey signatureKey=null;
 signatureKey=(iaik.pkcs.pkcs11.objects.RSAPrivateKey)
               foundSignatureKeyObjects[0];
 session.findObjectsFinal();

 ByteArrayOutputStream dataToBeSignedBuffer =
               new ByteArrayOutputStream(128);

 MessageDigest digestEngine = MessageDigest.getInstance("SHA-1");
 Mechanism signatureMechanism=Mechanism.RSA_PKCS;
 session.signInit(signatureMechanism, signatureKey);
 digestEngine.update(signString.getBytes());
            byte[] digest = digestEngine.digest();
            DigestInfo digestInfoObject = new
```

```
                    DigestInfo(AlgorithmID.sha1, digest);
            byte[] digestInfo = digestInfoObject.toByteArray();
            byte[] signatureValue = session.sign(digestInfo);
            System.out.println("Value: "+new
                    BigInteger(1,signatureValue).toString(16));

            String sigString1=new String();
            BASE64Encoder encoder=new BASE64Encoder();
            sigString1=encoder.encode(signatureValue);
            sigString1=this.parseDecodedKey(sigString1);
            return sigString1;
        }catch(Exception e)
 {System.out.println("Hallo fehler beim signieren mit token");
 e.printStackTrace(); return "false";}
}
```

8.3. Signieren mit der BasicCard

Die BasicCard bietet keine PKCS Schnittstelle an. Daher mussten die Funktionen zur Signatur selbst implementiert werden.

Die BasicCard stellt hierfür Elliptische Kurven nach IEEE P163 zur Verfügung. Diese müssen mit Funktionen im BasicCard Programm versehen werden. Für die vorliegende Arbeit sind die Funktionen

- Generierung eines Schlüsselpaares
- Signatur mit der Karte
- Exportieren des Öffentlichen Schlüssels

entwickelt worden.

Als Beispiel ist die Generierung des Schlüsselpaares auf der BasicCard aufgeführt:

```
Rem  Command &H20 &H01 GenerateKey()
Rem
Rem Diese Funktion darf nur
Rem genau einmal aufgerufen werden.
Rem
Command &H20 &H01 GenerateKey()
  If Not (PINVerified) Then SW1SW2 = swPINRequired : Exit
  If keysgenerated = 1 Then Exit Command
  Call EC167GenerateKeyPair()
  Call EC167SetPrivateKey(EC167PrivateKey)
  keysgenerated=1
  If LibError <> 0 Then SW1SW2 = LibError : LibError = 0
End Command
```

Für die von der BasicCard erzeugten Signaturen gibt es noch keine Möglichkeit der Überprüfung. Daher kann die Chipkarte noch nicht zur Signatur eingesetzt werden.

9. Ausblick

Die Digitale Signatur ist immer noch nicht so häufig verbreitet, wie es eigentlich möglich wäre. Die Benutzung von Signaturprogrammen und Verschlüsselungssoftware ist mittlerweile kein “Hexenwerk” mehr. Programme wie Outlook haben die Signatur und Verschlüsselungstechniken integriert.

Diese Funktionen werden von den Benutzern oft als zu kompliziert und umständlich abgetan und daher nicht benutzt.

Bei dem Online-Bestellwesen ist eine Digitale Signatur, wie auch bei anderen Diensten, die über das Internet abgewickelt werden, ein Muss. Hier darf der Benutzer die Signatur und dadurch seine Unterschrift nicht als zu umständlich ablehnen.

Bei der Implementation in das Bestellwesen wurde daher viel Wert auf eine einfache Handhabung gelegt. Dadurch wird es den Benutzern erleichtert, sich mit der Digitalen Signatur anzufreunden.

Die entwickelten Klassen sind nicht nur für die spezielle Anwendung nutzbar. Somit steht einer Digitalen Signatur mit Chipkarten bei anderen Anwendungen eine Grundlage zur Verfügung.

Für den interessierten Anwender wurde, ein TestApplet entwickelt, das es ermöglicht, den Aufbau und die Funktionsweise von Chipkarten besser zu verstehen.

A. APDU Liste

Auflistung der APDUs und Fehlercodes des in der Arbeit entwickelten BasicCard Programmes

90	00	alles in Ordnung
91	xx	Antwort enthielt xx Bytes. Das le Feld der Anfrage hatte eine andere Länge als die zurückgegebenen Daten
6B	00	NotPersonalized
6B	02	Encryption Required
6B	05	InvalidPIN
6B	06	PINErrorsExceeded
6B	07	AllReadyPersonalized
61	xx	Interner Fehler

Tabelle 9: Fehlercodes

CLA	INS	Beschreibung
20	00	Get Curve
20	01	Generate Key Pair
20	02	Get Public Key
20	03	Sign Message
20	04	SetSharedSecret
20	05	SetSessionKey
20	10	Hash Message
20	11	HashBegin
20	12	AddHash
20	13	EndHash
23	00	VerifyPin
23	01	ChangePin
23	70	GetName
23	71	SetName
23	72	GetPassword
23	73	SetPassword
23	74	GetStrasse
23	75	SetStrasse
23	76	GetTelefon
23	77	SetTelefon
23	78	GetWohnort
23	79	SetWohnort
23	80	SetData1
23	81	SetData2
23	82	SetData3
23	83	SetData1
23	84	GetData2
23	85	GetData3

Tabelle 10: Implementierte APDUs

B. RSA Verfahren

Das RSA Verfahren besteht aus zwei Teilen, der Schlüsselerzeugung und der Anwendung.

1. Es werden zwei große, zufällige Primzahlen p und q gewählt. Die Länge sollte zwischen 384 und 512 Bit liegen.
2. Das Produkt $n = pq$ bestimmen.
3. Auswahl einer kleinen ungeraden natürlichen Zahl e. Die zu $(p-1)*(q-1)$ relativ prim ist. d.h. ggT $(e, \phi(n)) = 1$.
4. Berechnung von d, wobei gilt: $ed = 1$ mod x
5. Der öffentliche Schlüssel ist nun das Zahlenpaar e, n
6. Der geheime Schlüssel ist d, x

Die Verschlüsselung einer Nachricht $M \epsilon Z_n$ funktioniert mit:

$$E(M) = M^e \text{ n}$$

Das Entschlüsseln eines Chiffretextes $C \epsilon Z_n$ entsprechend:

$$D(C) = C^d \text{ mod } n$$

Siehe auch [ERTEL] und
`http://www.sias.de/kryptologie-rsa.html`

B.1. Zufallszahlen

Für die Schlüsselerzeugung benötigt man einen Algorithmus, um große Primzahlen zu erzeugen.

Hierfür werden oft Verfahren verwendet, die auf Zufallszahlen beruhen. Daher bieten Chipkarten einen Hardware Zufallszahlengenerator an.

Bei der Comcard MFC 4.3 wird der auf der Chipkarte integrierte DES Algorithmus benutzt, um die Zufallszahlen zu erzeugen. Der Startwert (seed) wird bei der Personalisierung der Chipkarte eingestellt.

Um dennoch einen immer wechselnden Seed zu erhalten, wird dieser nach einer Zufallszahlberechnung mit einer Zufallszahl überschrieben. Damit das EEprom an dieser

Stelle nicht durch das häufige Schreiben zerstört geht, wird der Seed wert in einem "Cyclic File" (siehe 4.6) abgespeichert. Hierdurch verringern sich die Schreibzugriffe auf eine Speicherstelle.

C. openssl und xca

Da die Integration der Chipkarte in das Bestellsystem eine CA erforderte, wurde mit dem Programpaket `openssl` und `xca` auf dem Linux Rechner, der auch die Anwendung beherbergt, eine eigene CA errichtet.
Diese CA ist bislang von keiner anderen CA zertifiziert worden. Für den Einsatzzweck als CA für das Bestellwesen ist dies auch nicht notwendig. Hier reicht das "self certificated" aus.

C.1. Generierung der Schlüssel

Um mit `openssl` ein Root-CA-Zertifikat zu erzeugen, ist nach einem Anpassen der Datei `openssl.conf` dieser Aufruf erforderlich:

```
openssl genrsa -des3 -out $(SSLETC)/private/CAkey.pem \\
 -rand Zufallsdaten 2048
```

Dieser Schlüssel muss noch unterschrieben werden:

```
openssl req -new -x509 -days 730 -key $(SSLETC)/private/CAkey.pem\\
 -out $(SSLETC)/private/CAcert.pem
```

C.2. Fingerprints

Um festzustellen, dass die CA nicht manipuliert worden ist, muss ein Benuzter die Fingerprints der CA Stelle kennen.
Der X509v3 Authority Key Identifier der eingeführten CA lautet:

```
keyid:C3:60:9B:DC:F9:7E:82:63:7B:50:7F:6A:58:D2:48:8F:A3:F1:89:1F
DirName:/C=DE/L=Weingarten/ST=BW/O=FH-WGT/OU=fbe/Email=root@localhost
serial:01
```

Die Fingerprints der CA sind:

```
MD5:
8D:74:3F:6D:E5:C5:9D:FF:38:96:85:38:42:47:F9:B9
SHA1:
D1:55:73:F8:51:CA:70:93:CE:3D:70:C3:2C:4A:F6:77:26:8B:A1:D9
```

D. Marktübersicht

Um die richtigen Chipkarten für die FH Ravensburg-Weingarten zu ermitteln, ist eine Marktanalyse vorgenommen worden.

Dabei ist aufgefallen, dass Chipkartenhersteller sich mit Demonstrationsprodukten sehr zurückhalten. Es werden teure Entwicklungspakete angeboten oder nur komplette Lösungen.

Einen guten Support hat die Firma ZeitControl gegeben, leider ist die Chipkarte nur bedingt einsetzbar.

Die Hersteller von Chipkartenlesern waren entgegenkommender. Hier wurden mehrere Demonstrationsobjekte zur Verfügung gestellt.

Die Auswahl der Hardware hängt eng mit dem verwendeten Rechner zusammen. Da neue Rechner immer seltener mit einem seriellen Anschluss ausgeliefert werden, ist eine USB Schnittstelle sinnvoll.

In den folgenden Tabellen finden sich die bei der Arbeit benutzten und von Firmen zur Verfügung gestellten Hardware/Produkte.

Hersteller	Produkt und Treiber	Schnittstelle	Bemerkungen
Omnikey	CardMan 2020	USB	Installation unter Linux nur mit Aufwand möglich; Für Linux existiert ein Kerneltreiber, der für das verwendete RedHat 7.3 angepasst werden musste. Für die vorliegende Arbeit ist daher ein Treiber-Patch entstanden, der das Problem mit Kernel > 2.4.19 behebt.
Towitoko	Micro 130	USB Seriell	Teilweise Verbindungsprobleme mit der Chipkarte; ist weit verbreitet durch die HBCI Initiative der Banken.
Cherry	Tastatur	Intern PS/2	Die Treiber, die in den Tests verwendet wurden, sind nicht sehr stabil gewesen. Unter Windows 2000 ist es ab und zu, nachdem die Chipkarte eingeführt wurde, nicht mehr möglich gewesen, Tastatureingaben zu tätigen.

Hersteller	Karte	Speicherplatz	Betriebsystem	Treiber	Kryptoverfahren	Bemerkungen
ZeitControl	BasicCard 3.9	8 KByte	BasicCard	–	DES, Elliptische Kurven auf GF(161)	Eine programmierbare Karte, bei der alles selbst implementiert werden muss. Die Karte eignet sich hervorragend als Studienkarte für schnelle Testzwecke, aber weniger zur Digitalen Signatur.
ZeitControl	BasicCard 5.4	16 KByte	BasicCard	–	AES, Elliptische Kurven auf $GF(2^{167})$	Siehe BasicCard 3.9 Die Elliptischen Kurven Implementation ist nicht weit verbreitet. Daher ist das Verifizieren der Signatur noch nicht möglich.
ComCard	ComCard MFC 4.3	32 KByte	MFC4.3	cccsigit.dll	RSA bis 1024 Bit, DES	Nur für Windows. Die Dokumentation der Karte ist nicht online verfügbar. Die MFC 4.3 Version unterscheidet sich stark von den alten Betriebssystemversionen. So ist die Nutzung von alten Treibern, die es für Linux gibt, nicht möglich. Eine Treiberanpassung an Linux ist nicht geplant.
Gieseke&Devrient	StarCos SPK	16 KByte	StarCos	aetpkss1	RSA bis 1024 Bit	Die Karten konnten nicht getestet werden

Literatur

[bac] http://www.basiccard.de

[DNETC] http://www.distributed.net

[ERTEL] Angewandte Kryptographie, Ertel 2002

[IEEE] http://grouper.ieee.org/groups/1363/
http://grouper.ieee.org/groups/1363/P1363/draft.html

[IX] http://www.heise.de/ix/artikel/2000/12/152/ixtract

[JavaC] http://www.citi.umich.edu/projects/smartcard/webcard

[kliche] http://www.kliche.org/max/chipkarten/index.html

[Lkka] http://www.linuxnetmag.com/de/issue7/m7chipdrive1.html

[Lrsa] http://www.rsasecurity.com/rsalabs/pkcs/

[muscle] u̇rlhttp://www.linuxnet.com

[OPENC] http://www.opencard.org

[PCSC] http://www.pcscworkgroup.com

[Preim] Entwicklung interaktiver Systeme, Preim 1999

[RANKL99] Effling/Rankl Handbuch der Chipkarten

[SCHN] Angewandte Kryptographie

[SIGG] http://www.goinform.de/demo/arbsch/ce/bu/sigg.pdf

[xca] http://www.hohnstaedt.de/xca.html

Weitere Internet-Verweise finden sich auf der erstellen Internet-Adresse:

http://www.kliche.org/max/chipkarten/index.html

5. Danksagung

Ich danke den Herren Prof. Dr. Wolfgang Ertel und Herrn Prof. Ekkehard Löhmann für das gestellte Thema und die wohlwollende Unterstützung dieser Arbeit. Mein besonderer Dank gilt den Firmen ZeitControl, OmniKey, Towitoko und Cherry für die Bereitstellung von Chipkarten und Chipkartenlesern.

6. Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Weingarten, den 31.1.2003

Max Kliche