Implementierung einer mobilen Bezahlungslösung unter Java 2 Micro Edition

Kennzahl J151
Matrikel-Nr.: 9651146

Diplomarbeit

Eingereicht von
Klaus Brosche

am Institut für
Informationsverarbeitung und Informationswirtschaft,
Abteilung für Informationswirtschaft
an der WIRTSCHAFTSUNIVERSITÄT WIEN
Studienrichtung: Betriebswirtschaft

Begutachter: o.Univ. Prof. Dkfm. Dr. Wolfgang H. Janko
Betreuer: Univ.-Ass. Dr. Michael Hahsler

Wien, 02. Juli 2004

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, daß ich diese Diplomarbeit selbstständig verfaßt, keine anderen als die angegebenen Hilfsmittel und Quellen verwendet, mich auch sonst keiner unerlaubten Hilfsmittel bedient und diese Diplomarbeit weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Klaus Brosche 02. Juli 2004

Inhaltsverzeichnis

Zusammenfassung

Die vorliegende Diplomarbeit beschreibt die Grundlagen von bestehenden mobilen Bezahlungslösungen, die für die Neuentwicklung eines Bezahlungssystems für mobile Endgeräte, wie zum Beispiel einem Mobiltelefon, unter Java 2 Micro Edition notwendig sind.

Die gesamte Arbeit ist in zwei Bereiche geteilt. Der erste Teil beschäftigt sich mit den theoretischen Grundlagen von Bezahlungssystemen und beschreibt die zum Einsatz kommenden Technologien, wie zum Bespiel Irda oder digitale Signaturen. Im zweiten Teil wird die praktische Umsetzung der mobilen Bezahlungslösung unter Java 2 Micro Edition gezeigt.

Das Ergebnis der Diplomarbeit ist eine Software, entwickelt mit der Programmiersprache Java 2 Micro Edition, die das Empfangen von Zahlungsanfragen, sowie die Prüfung und das Erstellen einer digitalen Signatur dieser Anfragen, erlaubt. Um das Testen dieser Applikation zu ermöglichen, mußten auch Java Programme zum Simulieren der beteiligten Parteien, wie zum Beispiel des Kassensystems, erstellt werden. Während diese Diplomarbeit geschrieben wurde, mußte ich erkennen, daß im Moment noch keine Java 2 Micro Edition Standards für die Kommunikation per Infrarotschnittstelle vorhanden sind. Aus diesem Grund wurde die Applikation für die Ausführbarkeit auf einem Siemens SL45i Mobiltelefon programmiert und ist nicht auf ein anderes mobiles Endgerät portierbar. Die erstellte Software ist für Zahlungstransaktionen in der realen Welt ausgelegt, das heißt zum Beispiel für Zahlungen in einem Supermarkt. Der Prototyp zeigt, daß eine solche Bezahlungslösung grundsätzlich unter Java 2 Micro Edition lauffähig ist und daß die fehlenden Standards einer Marktreife im Wege stehen.

Summary

This dissertation submitted for a diploma describes the basics of existing mobile payment systems, which are necessary to implement a new mobile payment system on a small device like a cellular phone and how to write a prototype of such a payment system under Java 2 Micro Edition.

It is devided up into two parts. The first part handles the theory of payment systems, like discussing several terms and describing existing communication protocols and technologies, which are used. The second part is the practical one, which shows how such a implementation is done.

The result of this dissertation submitted for a diploma is a Java 2 Microedition software, which can receive payment requests by using the Irda Port, verify these payment requests and sign them with a digital signature. For testing purposes I also developed a Java software which simulates the payment terminal in a shop as well as a Java software, which simulates the clearing house for these payment transactions. During the development process of this dissertation submitted for a diploma I had to learn that there are still no Java 2 Micro Edition standards for the communication through the Irda inteface available. Due to this missing standard the developed software is written for a Siemens SL45i mobile phone and does not run on other mobile devices. The developed prototyp covers payment transactions in the real world, like payment in a supermarket. Before such a payment system could be available in the real world, some standards have to be passed.

Teil I

Grundlagen

Kapitel 1

Einführung

Der Einzug mobiler Technologien in unseren Lebensalltag hat bereits stattgefunden. In unserem täglichen Leben nutzen wir diese Technologien in verschiedensten Formen: zum Beipiel als Kommunikationsmittel, als mobiles Büro, als Online-Informationsbeschaffungsinstrument oder auch als Spieleplattform.

Der durch den Informations-Technologie-Hype im Jahr 2000 geprägte Begriff ”Mobile Commerce” hat mittlerweile einen eher bitteren Nachgeschmack, da zahlreiche Realisierungsversuche bisher scheiterten. Eine der vielen Definitionen für den Begriff Mobile Commerce ist [WEBAGENCY, 2002] : ”Mobile Commerce ist ein Konzept zur Nutzung von Informations- und Kommunikationstechnologien zur mobilen Integration und Verzahnung unterschiedlicher Wertschöpfungsketten oder unternehmensübergreifender Geschäftsprozesse und zum Management von Geschäftsbeziehungen”.

Ausgangspunkt meiner Diplomarbeit ist ein Teilbereich des Mobile Commerce, nämlich Mobile Payment. Mobile Payment beschäftigt sich mit der Möglichkeit mittels mobiler Endgeräte Zahlungstransaktionen durchzuführen. Eine genauere Beschreibung dieses Begriffs wird in einem späteren Kapitel folgen. Die bisherigen Realisierungen dieser Zahlungsform nützen das mobile Endgerät zumeist nur zum Anstoßen einer Transaktion (zum Beispiel PayBox) und nicht als mobile Geldbörse, wie zum Beispiel das Zahlungsystem ”Quick” aufgebaut ist. Das heißt die Transaktionslogik ist bei den bisherigen Systemen losgelöst vom mobilen Endgerät zu betrachten.

Genau hier möchte ich mich von den am Markt befindlichen Systemen unterscheiden und ein System planen, daß den Transaktionsprozeß mehr zum mobilen Endgerät hin verlagert.

Ziel meiner Diplomarbeit ist es einen Prototypen zu entwickeln, der es ermöglicht sicher und einfach mit mobilen Endgeräten zu bezahlen. Die folgenden Kapitel beschreiben alle Grundlagen für eine solche Implementierung. Nicht alle beschriebenen Technologien werden auch tatsächlich Einzug in den endgültigen Prototypen finden, da sie entweder aus technischen Gründen nicht realisierbar sind oder der Implementierungsaufwand den Leistungsumfang einer Diplomarbeit bei weitem übersteigen würde.

Die Herausforderung, die dieses Thema bietet, ist: ”Kann ein solches Bezahlsystem überhaupt mit den am Markt befindlichen Technologien unter Verwendung von Java 2 Micro Edition realisiert werden?”

Kapitel 2

Der Zahlungsverkehr

Das Kapitel Zahlungsverkehr soll als allgemeine Einführung in die gesamte Thematik der Abwicklung von Zahlungstransaktionen dienen und einige Begriffe wie ”elektronischer Zahlungsverkehr” oder ”Transaktionskosten” genauer beschreiben.

2.1 Geld als Zahlungsmittel

Geld wird als Mittel zum Austausch von Gütern (zum Beispiel Sachgüter oder Dienstleistungen) verwendet und ist somit die Grundlage unseres Wirtschaftens.

Vor der Einführung des Geldes wurde der Handel durch den direkten Tausch von Gütern abgewickelt. Das Geld als Ersatzgut hat also den Charakter einer Ware.

Grundsätzlich gibt es drei verschiedene Arten von Geld:

1. Münzgeld
   ist sicherlich die älteste Form von Zahlungsmitteln, abgesehen von Tauschwaren im Tauschhandel. Die Entstehung reicht zurück in das 7. Jahrhundert vor Christus. Der Unterschied zu den anderen Geldformen ist, daß Münzgeld sehr oft einen tatsächlichen Wert hat, und zwar den Wert des Edelmetalles, aus dem es besteht. Als Beispiel kann man die Silber- und Goldmünzen, ausgegeben von der österreichischen Nationalbank, nennen, welche nicht nur einen ideelen sondern auch den Wert des Gold- oder Silbermarktpreises besitzen und ebenfalls im täglichen Zahlungsverkehr gültig sind. Münzen werden vor allem im Bereich der kleineren Geldeinheiten verwendet. Aufgrund des geringeren Wertes und der hohen Prägekosten, beziehungsweise des Materialwertes, müssen die Sicherheitsmerkmale nicht in

extrem hohen Masse ausgeführt sein, da eine Fälschung sehr kostenintensiv wäre.

2. Banknoten
   bestehen aus einem papierähnlichen Material und überzeugen durch das geringe Gewicht und die gute Transportfähigkeit. Im Unterschied zu dem beschriebenen Münzgeld ist der Materialwert eines Geldscheines gering, der Tauschwert jedoch hoch. Durch diesen Umstand muß eine sehr hohe Fälschungssicherheit gegeben sein.

3. Buchgeld
   Formen von Buchgeld sind Verbindlichkeiten und Forderungen, die in ”Büchern” vermerkt sind. Das beste Beispiel hierfür ist ein Girokonto. Bei einer Einzahlung von Banknoten auf dieses Konto werden keine Waren ausgetauscht, sondern nur die entstandene Forderung vermerkt. Weitere Beispiele sind die Überweisung (=Weitergabe einer Forderung) und die Kreditgewährung (=Eingehen einer Verbindlichkeit).

Funktionen des Geldes:

- Akkumulatorische Funktion: Geld als Zahlungsmittel hat die Eigenschaft der Unverderblichkeit. Es kann daher gehortet - gespart (=akkumuliert) werden.
- Zahlungsmittelfunktion: Geld wird durch die Übereinkunft von Menschen, es als Zahlungsmittel zu verwenden, zum Geld. Das bedeutet im konkreten Fall, daß ein Staat durch eine Gesetzeslegung ein Gut zum offiziellen Zahlungsmittel erklärt. Dieses Geld muß innerhalb des Staates als Tauschmittel angenommen werden.
- Rechenfunktion: Das Geld stellt einen Wert dar und kann daher zum Vergleich von Warenpreisen eingesetzt werden. Ebenfalls ermöglicht es die Planung von Wirtschaftsvorgängen, zum Beispiel die Budgetierung oder die Kostenrechnung.
- Zirkulatorische Funktion: Darunter wird die Möglichkeit verstanden, Geld als Tauschmittel gegen Güter zu verwenden. Es entsteht dadurch ein Geld- - Warenstrom.

2.2 Zahlungsformen

Die soeben beschriebenen Formen und Funktionen der Zahlungsmittel bilden die Grundlage unseres Wirtschaftssystems. Um den Austausch dieser Zahlungsmittel durchführen zu können, werden verschiedenste Übertragungsmöglichkeiten

benötigt.

Diese Übertragungsmöglichkeiten von Geld können grundsätzlich in drei Gruppen eingeteilt werden:

1. Bargeldverkehr:
   Die häufigste Form des Zahlungsverkehrs im privaten Bereich ist der Austausch von Gütern gegen Bargeld. Das Einzahlen auf ein Girokonto oder Sparbuch und somit das Überführen von Bargeld in Buchgeld gehört ebenfalls zu dieser Form des Geldverkehrs.

2. Buchgeldverkehr oder bargeldloser Zahlungsverkehr:
   Das Buchgeld selbst zählt zwar zu den staatlich anerkannten Zahlungsmitteln, der Buchgeldverkehr also zum Beispiel eine Überweisung oder eine Bezahlung per Scheck sind jedoch kein Zahlungsmittel, sondern nur eine Geldanweisung oder ein Geldersatzmittel. Das sogenannte Plastikgeld (=Kreditkarten) wird ebenfalls nicht als Zahlungsmittel anerkannt, da die Eigenschaft von Geld (muß von jedem angenommen werden) fehlt. Vielmehr könnte eine Kreditkarte als Identitätsnachweis für den Kunden eines Kreditkartenunternehmens gesehen werden.

3. Elektronischer Zahlungsverkehr:
   Der Übergang vom bargeldlosen Zahlungsverkehr zum elektronischen ist ein fließender, da auch der elektronische Zahlungsverkehr ein bargeldloser ist. Man könnte sogar sagen, der elektronische Zahlungsverkehr ist die Erweiterung des Buchgeldverkehrs. Das elektronische Zahlungsmittel ist jedoch, wie bereits erwähnt, an sich kein Zahlungsmittel, da es nicht von jedermann angenommen werden muß.
   Eine besondere Ausprägung ist das sogenannte ”E-Payment”. Auch wenn es nur wie die englische Übersetzung des Begriffs ”elektronischer Zahlungsverkehr” klingt, so wird darunter eine besondere Ausprägung, nämlich die Abwicklung des Zahlungsverkehrs unter Verwendung des Internets, verstanden. Diese Form beinhaltet sowohl traditionelle Bezahlungsmethoden wie zum Beispiel Kreditkarte oder Lastschriftverfahren, als auch vollkommen neue Methoden wie zum Beispiel virtuelles Geld.

Zahlungsverfahren im Internet

Die folgenden Erklärungen bezüglich der Zahlungsverfahren und ihren Transaktionskosten beschränken sich natürlich nicht nur auf die Gegebenheiten des Internets, sondern sind ebenfalls in unserer ”realen Welt” gültig. Das Beispiel Internet wurde nur verwendet, da es die Gegebenheiten sehr klar darstellen kann und auch

Ausgangspunkt für die Entwicklung von neuen Zahlungformen war und ist.

Das Internet als eine weltweit nutzbare Kommunikationsplattform, ist die ideale Umgebung Waren aller Art zum Verkauf anzubieten. Um das Angebot für den Konsumenten oder Geschäftspartner ideal gestalten zu können, gibt es für diesen Zweck speziell abgestimmte Bezahlungsformen.

Die im Internet hauptsächlich genutzten Formen sind Zahlung auf Rechnung, Bankeinzug, Scheckkarte und Kreditkarte.
Gerade diese ”traditionellen” Zahlungssysteme verursachen aufgrund ihrer meist aufwendigen Abwicklung sehr hohe Nebenkosten. Diese Nebenkosten werden nachfolgend auch als Transaktionskosten bezeichnet. Im folgenden werden die entstehenden Transaktionskosten der verschiedenen Zahlungssysteme beschrieben:

- Sendung auf Rechnung:
  Die über das Internet bestellten Waren werden versandt und zusätzlich wird eine Rechnung für den Käufer ausgestellt, welche dieser innerhalb einer gestellten Frist zu begleichen hat. Grundsätzlich sind hier die Transaktionskosten sehr gering. Die Kosten setzen sich aus den Druckkosten der Rechnung, den Versandkosten plus den Bearbeitungskosten zusammen. Die Versandkosten für die Rechnung selbst können allerdings durch ein Beilegen dieser zur Ware minimiert werden.

  Jedoch gilt zu bedenken, daß das WWW ein anonymer Raum ist und der Verkäufer nur eine Adresse als Sicherheit hat. Wenn sich der Besteller einer Ware in Österreich befindet, so kann die Richtigkeit dieser Adresse noch relativ leicht überprüft werden (Mißbrauch auch hier nicht ausgeschlossen). Was jedoch wenn der Empfänger aus Asien stammt und eine Ware in Österreich bestellt? Durch dieses Sicherheitsrisiko können sich die Kosten eines Mißbrauchs bis auf den Preis der Ware plus Versandkosten plus Bearbeitungskosten,.. erhöhen.

- Lieferung per Nachnahme:
  Bei dieser Art des Waren- und Geldtransfers wird das Geld direkt bei Lieferung der Ware vom Überbringer einkassiert. Partner für den Händler sind hierbei die Post- und Paketdienste. Kosten für eine Lieferung per Nachnahme sind mit größer als vier Euro anzusetzen. Vom Sicherheitsaspekt gesehen, ist diese Form des Austausches Ware gegen Geld als ideal zu betrachten, da eine Übergabe der Ware nur durch Bezahlung des Rechnungsbetrages stattfindet. Einzig bei einer Annahmeverweigerung durch den Kunden

kommt es zu einem Kostenaufwand in Höhe der Versandkosten plus der Bearbeitungskosten.

- Lastschrift:
  Beim Lastschriftverfahren gibt der Kunde bei Bestellung der Ware seine Bankkontodaten bekannt. Der Händler erhält damit die Möglichkeit einen Abbuchungsauftrag gegen den Kunden einzurichten. Die Kosten einer Transaktion belaufen sich in Höhe der Überweisungskosten, sind also als gering einzuschätzen. Einzig bei einer Rückweisung der Zahlung durch die Bank oder durch mißbräuchliche Angaben, erhöhen sich die Kosten wie im Falle der Sendung auf Rechnung. Zusätzlich zu den beschrieben Risiken muß der Konsument davon überzeugt werden, daß seine Kontodaten sicher aufbewahrt werden. Dies stellt sicherlich oftmals eine Hemmschwelle für den Konsumenten dar.

- Kreditkarte
  Die Kreditkarte ist für den Konsument eine sehr sichere Methode zum Einkaufen, da eine berechtigte Rücküberweisung einer Fehlbuchung innerhalb einer Frist zumindest in Österreich leicht möglich ist. Ein Beispiel hierfür wäre die Verbuchung eines Betrages auf eine als gestohlen gemeldete Kreditkarte. Diese kann durch einen Einspruch bei der Kreditkartengesellschaft rückgängig gemacht werden. Anders ist die Situation für den Händler, dieser trägt grundsätzlich die Kosten für einen Zahlungsausfall durch eine wie oben beschriebene Fehlbuchung.
  Für den Händler besteht in diesem Zusammenhang nur die Möglichkeit die Kreditkartendaten durch eine Online-Prüfung beim Kreditkarteninstitut zu verifizieren. Diese Prüfung verursacht allerdings zusätzliche Kosten. In den USA besteht weiters die Möglichkeit die Kreditkartendaten mit den Adressdaten des Kunden gemeinsam zu prüfen.

  Kosten bei einer Bestellung mit Kreditkarte entstehen sowohl für den Händler (in Höhe von ca. drei % vom Umsatz bzw. eines Mindestbetrages) als auch für den Konsumenten im Umfang der Jahresgebühr.

  Beim Lastschriftverfahren könnten die Kosten für die Kontogebühr natürlich auch angeführt werden. Jedoch mit dem Unterschied, daß man heutzutage für fast jede Art des Zahlungsverkehrs, zum Beispiel Lohn- und Gehaltsüberweisung, sowieso ein Konto benötigt und ein solches bei den meisten Kunden daher vorhanden ist.
  Als negativer Aspekt der Kreditkartenzahlung ist jedoch die durchaus vorhandene Hemmschwelle des Kunden seine Kreditkartendaten bekannt zu

geben, zu erwähnen.

Zusammenfassend noch einmal ein Vergleich der traditionellen Bezahlungsformen:

Zahlung per	Rechnung	Nachnahme	Lastschrift	Kreditkarte
Sicherheit für den Kunden	+	+	+	+
Sicherheit für den Händler	-	+	+/-	+/-
Kosten für den Kunden	keine	Nachnahme-gebühr	keine	Karten-jahres-gebühr
Kosten für den Händler	$>$ 2 Euro	$>$ 3 Euro	$>$ 3 cent bei; Ablehnung 4 Euro	$>$ 1 Euro

Bei den Kosten für den Händler wurden Durchschnittswerte für Ausfälle, nach [WI, 2002], verwendet.

Aus diesen ”traditionellen” Zahlungsverfahren haben sich sowohl zur Optimierung des Zahlungsvorganges, als auch zur Minimierung der Transaktionskosten diverse E-Payment - Methoden entwickelt. Die Definition für eine E-Payment - Methode besagt, daß die Abwicklung eines Zahlungsvorganges durch die einmalige Eingabe der Daten durch den Kunden, ohne ein weiteres aktives Mitwirken, abgewickelt wird. Im Gegensatz dazu benötigen traditionelle Verfahren, wie zum Beispiel das Senden per Nachnahme, sehr wohl ein weiteres Mitwirken des Kunden.

Der interessanteste Punkt bezgülich des E-Payments ist, daß die verwendeten Methoden nicht nur auf eine Bezahlung in der virtuellen Welt des Internets beschränkt sind, sondern sich genauso gut in der realen Welt einsetzen lassen. Hintergrund dieser Verknüpfung zwischen realer und virtueller Welt ist die Möglichkeit der Abwicklung sämtlicher elektronischer Zahlungsformen über die Infrastruktur und Technologien des Internets.

Zur Verdeutlichung dieser Aussage kann der Bezahlungsvorgang mittels Kreditkarte in der realen Welt herangezogen werden: Nach Auswahl eines Produktes in einem Geschäft bezahlt der Kunde an der Geschäftskassa mittels Kreditkarte. Die Karte wird durch ein Gerät gezogen, welches die Kartendaten einliest und sofort

über eine Online-Verbindung mit dem Kreditkarteninstitut prüft, ob diese Karte gültig ist. Nach einer Prüfung wird der eigentliche Zahlungsvorgang aktiviert. Weder Händler noch Kunde müssen ein weiteres Mal aktiv bei diesem Vorgang mitwirken. Sowohl der Händler, als auch der Kunde sehen als Zahlungsbestätigung die Buchung des Betrages auf ihrem Konto. Es sind daher alle Eigenschaften der E-Payment-Definition erfüllt worden.

POS

Ein wichtiger Begriff der im Zusammenhang mit elektronischen Zahlungsmitteln oft verwendet wird ist POS. Dies steht für ”Point of Sale” und bezeichnet jenen Ort, an dem ein Verkauf und der dadurch ausgelöste Zahlungsvorgang stattfindet. Ein POS kann sich sowohl auf die reale, als auch die virtuelle Welt beziehen. POS in der realen Welt ist zum Beispiel die Kassa eines Lebensmittelgeschäftes mit all ihren Zahlungsmöglichkeiten wie Bankomatkassa, Barbezahlung, und so weiter. Der POS in der virtuellen Welt ist die virtuelle Kassa, an der zum Beispiel die Daten der Kreditkarte eingegeben werden.

Gliederung von elektronischen Zahlungsformen

Eine grundsätzliche Einteilung in Funktionen und Abrechnungsformen von elektronischen Zahlungsformen ist wie folgt möglich:

1. Online - Offline:
   Auf technischer Ebene ergibt sich die Unterscheidung zwischen Online-Systemen und Offline-Systemen. Online bedeutet in diesem Zusammenhang, daß der elektronische Zahlungsverkehr durch eine Verbindung, zum Beispiel über das Internet oder eine Direkteinwahl über das Telefon, des POS-Systems (zur Wiederholung: Point of Sale ist jeder Ort an dem ein Verkauf und damit verbundener Zahlungsverkehr stattfindet; dies kann sowohl im Lebensmittelgeschäft an der Kassa sein, als auch eine Internetseite sein) und der Verrechnungsstelle, zum Beispiel Kreditkartengesellschaft oder Bankinstitut, erfolgt. Vorteile einer Onlineverbindung sind die Möglichkeit die Daten des Zahlungspartner auf ihre Gültigkeit zu überprüfen, sowie die sofortige Abwicklung einer erfolgreichen Transaktion. Nachteile sind die erforderlichen Investitionen für den Aufbau einer derartigen Infrastruktur, sowie die Kosten für die jeweilige Onlineverbindung. Bei den Offlinesystemen erfolgt die eigentliche Zahlungstransaktion erst nach Abrechnung des POS-Inhabers mit der Verrechnungsstelle. Diese Verrechnung erfolgt ganz unterschiedlich, bei manchen Systemen täglich oder auch zu einem späteren Zeitpunkt. Ein Beispiel für eine spätere Abrechnung wären Automatengeschäfte, bei denen eine Bezahlung per elektronischer Geldbörse

möglich ist. Es lohnt sich oft nicht die auf einen Chip zwischengespeicherten Daten durch einen Mitarbeiter täglich zur Verrechnungsstelle zu bringen. Es stehen Mitarbeiterkosten den Automateneinnahmen gegenüber. Der größte Nachteil liegt sicherlich in dem Risko, daß die Daten des Zahlungsauftraggebers nicht direkt am POS überprüft werden können, und daher die Möglichkeit eines Zahlungsausfalles für den POS-Betreiber besteht.

2. Macro - Micro:
   Die zweite Unterscheidungsmöglichkeit von elektronischen Zahlungsverfahren liegt in der Anwendbarkeit. Diese kann sich auf den Bereich Macropayment oder Micropayment konzentrieren. Macro und Micro definieren hier den Betrag einer Zahlungstransaktion (siehe auch Punkt Micropayment 2.4). Diese Unterscheidung knüpft sehr stark an die im ersten Punkt erklärten Unterschiede zwischen Online-System und Offline-System an, da bei Bezahlung von Kleinstbeträgen eher ein größeres Risiko für einen Zahlungsausfall eingegangen werden kann, als bei höheren Beträgen. Auch wären die zusätzlichen Kosten für eine Onlineverbindung bei Transaktionen von geringem Wert nicht unbedingt gerechtfertigt. Steigt jedoch der Wert der Transaktion, wird sich der POS-Betreiber überlegen, ob die Kosten für eine sichere Zahlungsform nicht doch gerechtfertigt sind.

3. Vorauszahlung - Kreditleistung:
   Die letzte Form der Unterscheidung betrifft sowohl den POS-Betreiber, als auch den Konsumenten und auch hier könnnen wieder Parallelen, zwischen den vorher erwähnten Unterscheidungsmerkmalen, gezogen werden. Zahlungsverfahren die eine Vorauszahlung benötigen, werden auch als Prepaid-Systeme bezeichnet. Beispiele hierfür sind das Aufladen von elektronischen Geldbörsen oder das Kaufen von sogenannten Prepaidkarten. Man könnte dies auch als Kreditleistung des Konsumenten bezeichnen. Nachteil für den Konsument ist der Zinsverlust für den investierten Betrag. Diese Systeme eignen sich am Besten für Bereiche des Micropayments, da der Konsument eher gewillt ist kleine Beträge in ein solches System zu investieren. Ein Beispiel für eine typische Kreditleistung ist der Einkauf mittels Kreditkarte. Der Betrag eines solchen Einkaufs wird erst mit Ende des Monats beim Konsumenten abgebucht. Er erhält daher einen kurzfristigen Kredit bis zur tatsächlichen Abbuchung. Auch hier ist die Verknüpfung zu den schon erwähnten Punkten Macropayment und Online-System leicht zu verstehen, da ein POS-Betreiber nur für größere Beträge dazu bereit sein wird, eine Kreditleistung zu erbringen und diese mit einem sicheren System zu transferieren.

Der Zusammenhang dieser drei Unterscheidungsmerkmale ist sehr leicht zu bemerken und kann zusammengefasst wie folgt dargestellt werden:

- kleine Beträge: Vorauszahlung des Konsumenten → billigeres Offline-System.
- größere Beträge: Kreditleistung des POS-Betreibers → teureres Online-System.
- Ein weiterer Schluß ist → höhere Zahlungssicherheit bedeutet höhere Kosten.

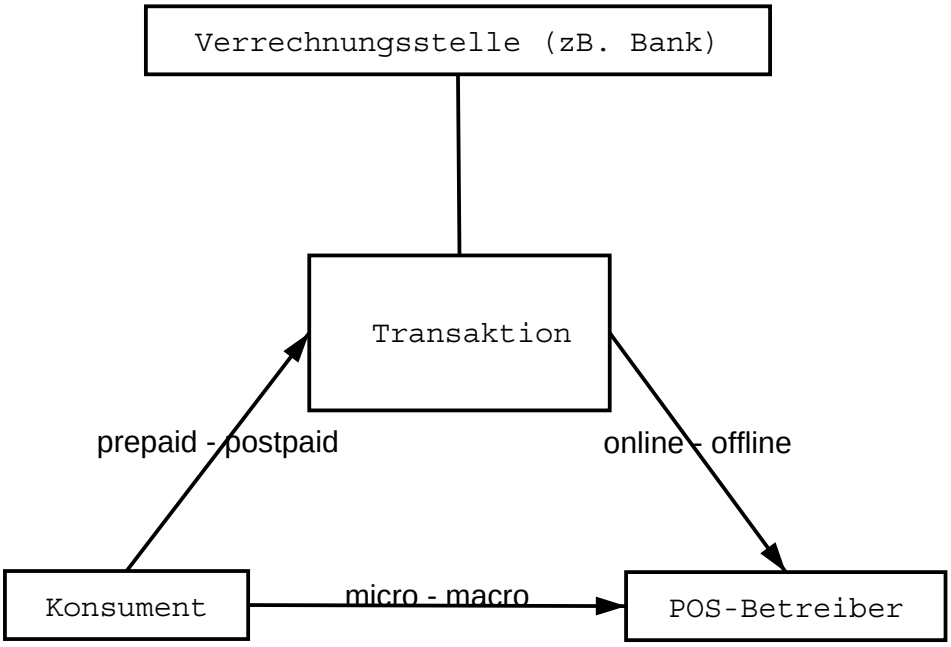

Abbildung 2.1: Zahlungstransaktion

2.3 M-Payment

Die zwei aktuellen Schlagwörter ”M-Commerce” und ”M-Payment” werden gerade im Zusammenhang mit Inovationen im Bereich der mobilen Kommunikationstechnologien sehr oft verwendet. Ist der Begriff des Mobile Commerce (M-Commerce) noch ein sehr schwammiger, weil eine Vielzahl von Definitionen erhältlich sind, so kann man sich von Mobile Payment (M-Payment) schon ein genaueres Bild machen.

Beispiele für die Definition des Mobile Commerce (Auszüge):

- Lehman Brothers 2000: [Brothers, 2000]
  ”the use of mobile hand-held devices to communicate, inform, transact and entertain using text and data via connection to public networks.”

- Durlacher 1999: [Ltd, 1999]
  ”any transaction with a monetary value that is conducted via a mobile telecommunication network.”

Man könnte daraus eine ganz grobe Definition von Mobile Commerce ableiten und sagen, es handelt sich um den entgeltlichen Transfer von Dienstleistungen aller Art. Mobile Payment kann man daher als die Möglichkeit definieren Dienstleistungen mobil zu bezahlen.

Um die Möglichkeit des mobilen Bezahlens zu schaffen, muß die Infrastruktur der elektronischen Zahlungssysteme auf die der mobilen Endgeräte angepasst werden. Neue mobile Technologien wie schnellere Übertragungsgeschwindigkeiten mittels GPRS und UMTS und eine Erweiterung der Prozessorgeschwindigkeiten sowie die Vergrößerung der Speicherkapazitäten lassen auf zukünftige benutzerfreunliche Zahlungssysteme hoffen.

Derzeitige Lösungen finden durch ihre umständliche Bedienbarkeit nur sehr eingeschränkte Akzeptanz. Genauere Beschreibungen folgen im nächsten Kapitel 2.4 Micropayment unter dem Punkt 2.4.1 ”Bestehende elektronische Zahlungssysteme”.

2.4 Micropayment

”Micropayment” ist gerade im Zusammenhang mit dem Internet ein in letzter Zeit sehr oft benutzter Term. Eine Vielzahl von Anbietern elektronischer Zahlungsformen schmücken sich mit diesem Wort. Micropayment beschäftigt sich mit der Möglichkeit elektronische Zahlungsvorgänge so zu optimieren, daß selbst bei Zahlungen von Kleinstbeträgen die Transferkosten in einem für den Verkäufer rentablen Bereich bleiben. Wenn man von Micropayment-Zahlungvorgängen spricht, so wird ein Betrag von einem Cent bis zu cirka 10 Euro verstanden. Der Höchstbetrag von 10 Euro ist kein internationaler Standard für die Begriffserklärung des Micropayments, vielmehr gibt es verschiedenste Ansichten bis zu welchem Betrag von Micropayment gesprochen werden kann. Allerdings sollte man hier beachten, daß nicht der Höchstbetrag ausschlaggebend für die Theorie des Micropayments ist, sondern vielmehr die Handhabung der Geringstbeträge, also im Bereich von Cent-Beträgen.

Ein Beispiel eines Micropayment-Geschäftes soll die Notwendigkeit von Micropayment-Zahlungsformen verdeutlichen:

In einem Lebensmittelgeschäft werden im Bereich der Kassa Süßigkeiten ab einer Preisklasse von 30 Cent angeboten. Die Positionierung läßt darauf schließen, daß die Kunden während sie auf die Abwicklung des Bezahlvorganges an der Kassa warten auf diese günstigen Artikel aufmerksam gemacht werden und sich dazu entschließen sollen, diese Produkte zusätzlich zu den sonstigen schon ausgewählten Waren zu kaufen.

Werden nun von einem Kunden zusätzlich zu seinem schon bestehenden Warenkorb noch Süßsigkeiten im Wert von 30 Cent ausgewählt und

1. bar bezahlt, so entstehen für den Betreiber folgende Kosten: anteilige Personalkosten für die Abrechnung der Tageslosung, sowie den Transport des Betrages zu einem Bankinstitut. (Die Abrechnung beinhaltet auch die Zählung des Kassenstandes)

2. mit Bankomatkarte bezahlt, so entstehen für den Betreiber folgende Kosten: durchschnittlich cirka 1 Euro für die Abrechnung der Bezahlung und die Online-Kosten oder zusätzliche anteilige Personalkosten bei Offline-Verrechnung.

Werden nun von einem Kunden nur Süßigkeiten mit einem Wert von 30 Cent gekauft und

1. bar bezahlt, so entstehen für den Betreiber die schon in Beispiel eins erwähnten anteiligen Personalkosten.

2. mit Bankomatkarte bezahlt, so entstehen für den Betreiber wiederum durchschnittliche Kosten von cirka 1 Euro. Dies bedeutet einen negativen Umsatz von 70 Cent.

Genau diese Transferkosten gilt es mittels geeigneter Zahlungsformen zu reduzieren. Diese Beispieltransaktion könnte genauso gut im Internet stattfinden und sich statt auf Süßigkeiten, auf eine kostenpflichtige Zusatzinformation beziehen.

2.4.1 Bestehende elektronische Zahlungssysteme

Dieser Abschnitt gibt einen Vergleich der gängigsten elektronischen Zahlungssysteme sowie einen Überblick über die verwendeten Sicherheitsstrukturen. Weiters

wird die Brauchbarkeit der einzelnen Systeme für die Bezahlung von Kleinstbeträgen gezeigt. Die Unterscheidung baut auf der bereits im Punkt 2.2 ”Gliederung von elektronischen Zahlungsformen” getroffenen Einteilung auf.

Kreditkarten

Die Abwicklung eines elektronischen Zahlungsvorganges mittels Kreditkarte wurde bereits unter Zahlungsverfahren im Internet 2.2 erklärt. Ebenfalls wurden die Sicherheitsrisiken, die bei der Übermittlung der Kreditkarteninformationen entstehen, erwähnt. In den USA besteht die Möglichkeit für den Händler, die Karteninformationen mit den Adressdaten des Kunden auf Übereinstimmung zu prüfen, in Europa jedoch besteht ein solches System nicht. Um auch außerhalb der USA eine Sicherheit bei Kreditkartenzahlungsvorgängen zu ermöglichen wurde der Secure Electronic Transaction (SET) Standard geboren. An der Entwicklung waren hauptsächlich begteiligt: Visa, Eurocard, IBM, Microsoft, Netscape [Janssen, 2004].

Sicherheit mittels Secure Electronic Transaction Der SET Standard soll sowohl den Konsumenten, als auch den Händler vor Mißbrauch schützen, indem beide Identitäten eindeutig durch eine Zertifikatsstruktur nachgewiesen werden (siehe Kapitel 3.5 ”Public Key Infrastructure oder kurz PKI”). Zusätzlich muß beim Kunden eine spezielle Software installiert werden, die das vom Kreditkartenaussteller bestätigte Zertifikat speichert und das Händlerzertifikat prüfen kann. Möchte der Kunde eine Rechnung im Internet zahlen, so bekommt er diese vom Händler signiert übermittelt. Nach Prüfung dieser Rechnung wird sie vom Kunden signiert an den Händler zusammen mit den Kreditkarteninformationen zurückgesandt. Dieser leitet die Informationen zusammen mit der von ihm signierten Rechnung an das SET Payment Gateway weiter. Dort werden die Daten geprüft und bei einer positiven Abarbeitung wird die erfolgreiche Transaktion an den Händler rückgemeldet. Dieser kann anschließend mit der Bearbeitung des Auftrages fortfahren. Bei der Übertragung der Daten über das Internet wird ein Verschlüsselungsschutz durch den SSL-Standard (Secured Socet Layer) garantiert.

elektronische Münzen

Elektronische Münzen sind genau wie das Buchgeld (siehe Kapitel 2.2 Zahlungsformen) kein staatlich anerkanntes Zahlungsmittel. Es sind vielmehr virtuelle Münzen die auch im Wert nicht dem der richtigen Geldmünzen gleichen müssen. Diese virtuellen oder digitalen Münzen werden bei einem ”Händler” gekauft und können dann in bestimmten Geschäften oder auch virtuellen Geschäften verwendet werden. Diese Geschäfte tauschen die virtuellen Münzen wieder in ein anerkanntes

Zahlungsmittel um. Sinn dieser elektronischen Geldeinheiten ist es, durch im voraus bezahlte Beträge eine Grundlage für das Micropayment zu schaffen.

E-coins Eine Bespielimplementierung für elektronische Münzen bietet www.ecoins.net [eCoin Incorporated, 2004]. Am System sind drei Parteien beteiligt:

1. Kunde:
   Der Kunde muß sich einen sogenannten Walletmanager auf seinem PC installieren. Zum Aufladen dieses Walletmanagers werden beim Betreiber des Systems E-coins eingekauft und auf dem Rechner verschlüsselt gespeichert. Bezahlt werden diese E-coins per Kreditkarte. Trifft ein Kunde im Internet auf eine kostenpflichtige Seite, die dieses System unterstützt so wird der Walletmanager aktiviert und der Kunde kann den geforderten Betrag bezahlen.

2. Händler:
   Der Händler muß keine spezielle Software installieren. Er verlagert die kostenpflichtigen Informationen auf einen sicheren Bereich des Webservers. Die Zahlungstransaktion des Kunden findet direkt beim sogenannten Broker statt. Dieser ermöglicht dem Kunden den geschützten Bereich des Händlers für die Informationsabfrage zu betreten. Es entsteht daher eine Anonymität zwischen Händler und Kunden. Für jede Bezahlung über dieses Sytem wird ein Prozentsatz vom Rechnungsbetrag durch den Broker abgezogen.

3. Broker:
   Der Broker ist der Betreiber des Systems. Er unterhält für jeden Kunden und auch für jeden Händler ein eigenes Verrechnungskonto. Bei einer Bezahlung wird der Betrag vom Kundenkonto auf das Händlerkonto überwiesen. Aufgabe des Brokers ist die Ausgabe der E-coins, die Prüfung der eingehenden Zahlungsaktion beziehungsweise die Richtigkeit der E-coins, die Transaktion auf das Händlerkonto und die Weiterleitung des Kunden zur Informationsquelle des Händlers.

Ein E-coin ist 16 Byte groß und unterteilt sich in folgende Bereiche:

- E-coin Typ: 2 Byte
- interne Kennzeichnung: 1 Byte
- Datumsstempel und Ablaufdatum: 5 Byte
- Identifizierungsmerkmal: 8 Byte

Ob dieses System jemals eingesetzt wurde, konnte ich leider nicht herausfinden, da auf der HomePage von www.ecoins.net unter dem Bereich ”Beispielhändler” keine wirkliche Implementierung durch eine Firma aufscheint.

elektronische Geldbörsen - Smart Cards

Bei elektronischen Geldbörsen wird das virtuelle Geld auf einen Smart Card Chip gespeichert der auf einer Plastikkarte angebracht ist und kann über diverse Terminals aufgeladen und transferiert werden. In Österreich ist ein solches System unter dem Namen Quick im Einsatz. Vorteile von elektronischen Geldbörsen ist die Möglichkeit das virtuelle Geld sicher auf einem Chip speicher zu können. Es werden dabei asymtrische Verschlüsselungtechniken verwendet (asymetrische Verschlüsselungsverfahren Kapitel 3.3). Eine genaue Beschreibung befindet sich im Kapitel 5.5 ”Funktionsweise der elektronischen Geldbörse”.

Paybox

Ein in Österreich und Deutschland relativ weit verbreitetes System ist ”Paybox”. Durch eine Verknüpfung unterschiedlichster Technologien bietet es den Vorteil sowohl im Internet, als auch in der realen Welt einsetzbar zu sein. Der Schlüssel zu diesem System ist das Mobiltelefon. Kunden die keines besitzen, sind davon ausgeschlossen. Allerdings ist bei einer Mobiltelefonmarktdurchdringung von über 90% der österreichischen Bevölkerung [derstandard.at, 2004] der verbleibende Kundenanteil zu vernachlässigen.

Die Funktionsweise zur Abwicklung von Transaktionen, egal ob im Internet oder der realen Einkaufswelt, hat immer den gleichen Ablauf [www.paybox.at, 2004]:

1. Bekanntgabe der Telefonnummer:
   Zur Aktivierung eines Zahlungsvorganges muß die eigene Telefonnummer bekanntgegeben werden. Dies kann sowohl in einem Internetformular, als auch in einem Taxi geschehen. Einzig bei einer Überweisung auf ein anderes Bankkonto muß der Empfänger selbst und nicht der Aufraggeber eingetragen werden.

2. Anruf von Paybox und Ansage der Rechnungsdetails:
   Anschließend wird der Auftraggeber, also der Inhaber der Telefonnummer, von Paybox angerufen und es werden ihm die Details der Transaktion vorgelesen.

3. Autorisierung der Zahlung durch PIN-Eingabe:
   Stimmt der Angerufene mit den Transaktionsdetails überein, so kann er die Zahlung durch Eingabe eines PIN-Codes aktivieren. Eine Stornierung des Auftrages ist zu diesem Zeitpunkt leicht möglich. Nach erfolgreicher Eingabe des Codes wird die Zahlung beim Empfänger bestätigt.

4. Abbuchung vom Bankkonto:
   Die Verrechnung der Zahlungen erfolgt über ein Bankkonto innerhalb der üblichen Fristen, beziehungsweise kann der Händler ein längeres Zahlungsziel gewähren.

Sowohl für den Kunden, als auch für den Händler entstehen Kosten. Der Kunde muß eine Jahresgebühr von Euro 15,- entrichten, der Händler einen drei prozentigen Umsatzanteil. Zusätzlich fallen noch Kosten für die Infrastruktur an.
Dies zeigt deutlich, daß Paybox nicht als Micropayment Zahlungsmethode gedacht ist.

Der Einsatz von Paybox in der realen Welt scheint durch die komplizierte Abarbeitung nicht zielführend zu sein. Ein Einkauf im Supermarkt und eine anschließende Zahlung mittels Paybox würde die anderen wartenden Kunden wahrscheinlich sehr verärgern, da es einige Zeit benötigt bis der Kunde seine Telefonnummer angegeben, sich beim anschließenden Anruf die Rechnungsdetails angehört, die Autorisierung mittels PIN-Code durchgeführt und auf die Bestätigung an der Kassa abgewartet hat.
Es gibt jedoch bereits eine Anzahl von sinnvollen Einsatzmöglichkeit für dieses System außerhalb des Internets, zum Bespiel die Zahlung des Parkscheines oder die Zahlung einer Taxirechnung.

Mehrwert oder Premium SMS Dienste

Das derzeitige Einsatzgebiet dieser Lösung beschränkt sich auf das Versenden von diversen Informationen, Klingeltönen, Mobiltelefongrafiken und Spielen auf das mobile Endgerät. Hierfür wird vom Mobiltelefon aus eine SMS an eine Telefonnummer gesendet. Die Informationen werden anschließend mittels verschiedenster Technologien wie zum Beispiel WAP-Push oder SMS an den Teilnehmer zurückgesandt. Die Abrechnung erfolgt über die Monatsrechnung des Mobiltelefons.
Nachteil dieser Dienste ist oft die unzureichende Information über die genauen Kosten. Zwar müssen die Kosten für den jeweiligen Dienst angegeben werden, jedoch finden sich diese oft im Kleingedruckten und sind für den Konsumenten daher schlecht sichtbar.

Dieses System ist ebenfalls für das Internet brauchbar. Es kann Seiten, die durch einen Code geschützt und kostenpflichtig sind geben. Möchte ein Konsument den Inhalt dieser Seite sehen so sendet er ein Premium SMS an den Betreiber und erhält im Gegenzug ein SMS mit dem gewünschten Code. Nach Eingabe dieser Kennung kann er die Informationen einsehen. Nachteil dieser Möglichkeit ist, daß

der Konsument die Kennung an Dritte weitergeben kann und dem Betreiber daher mögliche Einnahmen entgehen würden.

Zur Zeit wird dieses System nur im Bereich des Micropayment genutzt. Eine Ausweitung auf den Macrobereich wird wahrscheinlich aus Sicherheitsgründen nicht stattfinden, da ein Bezahlungssystem, das nur den Absender als Sicherheitsmerkmal verwendet, nicht als ausreichend geschützt gilt. Es gab zum Beispiel bereits erfolgreiche Versuche eine SIM Karte zu fälschen [www.heise.de, 2002]. Entscheidende Merkmale wie die Verschlüsselung der Daten und ein Signieren der Rechnung fehlen hierbei.

Zusammenfassender Überblick zu den Zahlungssystemen

Es soll noch einmal überblicksweise durch die Tabelle verdeutlicht werden, welche Zahlungssysteme nur im Internet und welche auch in der realen Welt, das heißt in jedem Supermarkt, benutzt werden können. Zusätzlich wird gezeigt, ob sie mit Hilfe von mobilen Endgeräten eingesetzt werden können.

Zahlungsmethode	Internet	reale Welt	mobil Internet	mobil reale Welt
Kreditkarte	+	+	o	-
elektr. Münzen	+	-	-	-
elektr. Geldbörse	-	+	-	-
Paybox	+	+	+	+
Premium SMS	o	o	+	o

2.4.2 Welche Anforderungen entstehen für eine mobile Zahlungslösung?

Dieses Kapitel hat sich mit den verschiedenen Möglichkeiten von elektronischen Zahlungen sowie deren Vor- und Nachteile beschäftigt. Im letzten Teil wurde gezeigt, daß keines der existierenden Systeme die bestehenden Anforderungen an ein Zahlungssystem abdecken kann. Es müßte daher möglich sein, durch eine Verknüpfung dieser Technologien eine Lösung zu erstellen, die alle Anforderungen abdeckt:

- Sicherheit durch Zertifikate und Verschlüsselung
- einfache Bedienbarkeit
- einfache Implementierbarkeit für Händler
- Nutzung bestehender Infrastrukturen

- Abdeckung des Micro - Macrobereiches
- Nachvollziehbarkeit der Ausgaben für den Kunden
- Transaktionsschnelligkeit
- Offline - Online
- Internet - Reale Welt
- Anwendbarkeit mit und ohne mobilen Endgerät
- Anonymität - zumindest gegenüber dem Händler

Inwieweit sich eine Durchführung eines solchen Vorhabens mittels der in der Diplomarbeit zum Einzatz kommenden Technologie realisieren lässt, steht offen.

Kapitel 3

Verschlüsselung

Gerade im Bereich von Bezahlung, sei es in der ”realen” oder in der ”virtuellen” Welt, wie zum Beispiel im Internet, ist Datensicherheit stets eine heikle Frage. Im folgenden sollen die wichtigsten Methoden und Standards dargestellt werden. Die hier beschriebenen Verfahren werden im zweiten Teil dieser Arbeit zur sicheren Abwicklung des Bezahlvorganges benötigt.

3.1 Datenintegrität

Um sicherzugehen, daß übersendete Daten auch wirklich die Daten sind, die abgeschickt wurden, ist es notwendig, die Integrität festzustellen. Verschlüsselung alleine ist in diesem Fall nicht ausreichend, da die Daten zwar nicht angesehen werden können aber es trotzdem möglich ist, den Bytecode, in dem sich die Daten befinden, durch ein sogenanntes ”Bit-flipping” (Austausch von 0 gegen 1 und umgekehrt) zu verändern. Falls es nicht notwendig ist, die Daten geheimzuhalten, bietet sich die Möglichkeit, die Daten sowohl im Klartext (unverschlüsselt), als auch verschlüsselt zu senden. Der Empfänger ist dann in der Lage beide Daten zu vergleichen und eine etwaige Veränderung festzustellen. Eine sehr gute und schnelle Möglichkeit die Datenintegrität festzustellen bieten sogenannte Hashfunktionen.

Nachfolgend eine Definition der Hashfunktion:

Der Hashwert ist die komprimierte Version einer Datei. Man kann sich den Hashwert als den Fingerprint einer Datei vorstellen. Dieser Hashwert wird mit Hilfe einer Hashfunktion berechnet. Die Berechnung funktioniert nur in eine Richtung, das heißt, es kann zwar ein Hashwert berechnet werden, aus diesem jedoch keine Datei zurück berechnet werden.

Eine Datei die zusammen mit dem Hashwert verschlüsselt und versandt wurde kann somit durch Entschlüsselung Neuberechnung des Hashwertes und anschließendem Vergleich der beiden Hashwerte als integer eingestuft werden.

Ein Standard zu Berechnung eines Hashwertes bietet der SHA-1 (Secure Hash Standard), dieser berechnet, egal wie groß die Datei ist, einen 160 bit langen Wert. Der einzige Angriffspunkt bei dieser Art der Datenintegritätsprüfung ist es, eine Datei zu finden, die genau den gleichen Hashwert liefert.

3.2 Symetrische Verfahren

Bei der symetrischen Verschlüsselung wird eine Nachricht mit ein und demselben Schlüssel ver- und entkodiert. Folgende Formel veranschaulicht das Verfahren:

$E_K(M) = C$
$D_K(C) = M$

E .. Verschlüsselung
K .. Schlüssel
M .. Nachricht
C .. verschlüsselte Nachricht
D .. Entschlüsselung

Die verschlüsselte Nachricht kann sicher übermittelt werden. Das größte Risiko hierbei ist die Übertragung des Schlüssels vom Sender zum Empfänger.

3.2.1 DES

Eine der bekanntesten symetrischen Verschlüsselungsverfahren ist der DES (Data Encryption Standard). Dieser Standard ist seit 1977 im Einsatz und wurde unter Leitung der amerikanischen National Bureau of Standards und der amerikanischen Sicherheitsbehörde NSA entwickelt. Die Daten werden hierbei in 64 bit Blöcke aufgeteilt und verschlüsselt. Der Schlüssel hat 56 bits und 8 parity bits. Der gleiche Algorithmus wird für die Ver- und die Entschlüsselung verwendet.

DES ist allerdings nicht als sicher einzustufen, zum Beispiel wurde er 1999 durch

einen Zusammenschluß von ca. 100.000 Computer über das Internet innerhalb von 23 Stunden geknackt.

Für eine Implementierung unter Java 2 Micro Edition (J2ME) eignet sich das DES - Verschlüsselungsverfahren nicht nur aus Gründen der mangelnden Sicherheit nicht, sondern auch durch die Langsamkeit des Algorithmus. Zur Untermauerung dieser Aussage dient die folgende Vergleichstabelle von unterschiedlichen symetrischen Verschlüsselungsverfahren. Alle Tests wurden mit einem Mobiltelefon der Type Nokia 6310i unter Verwendung eines 23 Zahlen langen Textstrings gemacht. Dieses Mobiltelfon verfügt über eine J2ME Implementierung nach MIDP1.0 (siehe J2ME 6.1).

Algorithmus	64 bit key	128 bit key	256 bit key
DES	405 Millisek.	-	-
RC4	113 Millisek.	116 Millisek.	-
RC6	299 Millisek.	301 Millisek.	304 Millisek.

Quelle: [Cervera, 2002]

Auch wenn sich die Unterschiede im ersten Moment nicht als sehr gravierend darstellen, da sie sich alle im Bereich von Millisekunden befinden, so kann es bei einer Vergrösserung des Eingabestrings doch zu erheblichen Geschwindigkeiteinbussen kommen.

3.2.2 Advanced Encryption Standard (AES)

[of Standards and (NIST), 1997]
Nachdem der DES Standard bereits seit über 20 Jahren im Einsatz war und nicht mehr als besonders sicher galt, wurde im Jahr 1997 durch das National Institute of Standards and Technology (NIST) die Ausschreibung für die Neu-Entwicklung eines symetrischen Verschlüsselungsverfahren beschlossen. Der Name für den neu zu entwickelnden Standard lautete Advanced Encryption Standard (AES).

Das Hauptziel dieser Auschreibung war es einen ”Federal Information Processing Standard (FIPS)” zu erstellen der einen oder mehrere Algorithmen spezifiziert und den Schutz von sensiblen Regierungsdaten sicherstellt. Zum Einsatz sollte er daher vor allem im Bereich der US Regierung kommen.

Die Anforderung laut Ausschreibung lauteten: AES soll

- öffentlich publiziert werden
- eine symetrische Blockverschlüsselung haben
- ein Design haben, dass eine Veränderung der Schlüssellänge erlaubt
- in Hard- und Software implentierbar sein
- entweder lizenzfrei oder auf Basis der American National Standards Institute (ANSI) Lizenz erhältlich sein

Für die erste Auswahlrunde bewarben sich folgende Kandidaten:

- CAST-256 Entrust Technologies, Inc. (represented by Carlisle Adams)
- CRYPTON Future Systems, Inc. (represented by Chae Hoon Lim)
- DEAL Richard Outerbridge, Lars Knudsen
- DFC CNRS - Centre National pour la Recherche Scientifique - Ecole Normale Superieure (represented by Serge Vaudenay)
- E2 NTT - Nippon Telegraph and Telephone Corporation (represented by Masayuki Kanda)
- FROG TecApro Internacional S.A. (represented by Dianelos Georgoudis)
- HPC Rich Schroeppel
- LOKI97 Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
- MAGENTA Deutsche Telekom AG (represented by Dr. Klaus Huber)
- MARS IBM (represented by Nevenko Zunic)
- RC6TM RSA Laboratories (represented by Burt Kaliski)
- RIJNDAEL Joan Daemen, Vincent Rijmen
- SAFER+ Cylink Corporation (represented by Charles Williams)
- SERPENT Ross Anderson, Eli Biham, Lars Knudsen

- TWOFISH Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

Durch eine Bewertung nach folgenden Maßstäben und einer weiteren Runde

- Sicherheit
- Recheneffizienz
- Speicheransprüche
- Hard- und Softwareangemessenheit
- Einfachheit
- Flexibilität
- Lizenzanforderungen

wurden MARS, RC6, Rijndael, Serpent, und Twofish als Finalisten gekürt.

Am 2. Oktober 2000 wurde der Rijndael - Algorithmus als Sieger durch das NIST festgelegt. Begündet wurde diese Entscheidung dadurch, daß Rijndael die beste Kombination von Sicherheit, Performanz, Effektivität, Flexibilität und einfacher Implementierung bietet.

AES spezifiziert drei Schlüssellängen:

1. 128-bit d.h. 3.4 $\bullet 10^{38}$ mögliche Schlüssel
2. 192-bit d.h. 6.2 $\bullet 10^{57}$ mögliche Schlüssel
3. 256-bit d.h. 1.1 $\bullet 10^{77}$ mögliche Schlüssel

Eine Beispielimplementierung des RC6 und des Rijndael Algorithmus unter J2ME und Verwendung der Bouncy Castle API befindet sich im Anhang B.

3.3 Asymetrische Verfahren

Das größte Sicherheitsrisiko bei der symetrischen Verschlüsselung ist das sichere Übertragen des Schlüssels. Zur Umgehung dieses Problems bieten sich asymetrische Verfahren an. Hier wird eine Nachricht nicht mit ein und demselben

Schlüssel ver- und entkodiert, sondern kann immer nur mit einem sogennanten Schlüsselpaar ver- und entschlüsselt werden.

Ein solches Verfahren wird auch als Public - Private Key System bezeichnet. Der Public - Key (öffentlicher Schlüssel) ist für jedermann zugänglich. Der Private - Key hingegen muß streng geheim aufbewahrt werden. Zur sicheren Übermittlung einer Nachricht wird diese mittels des öffentlichen Schlüssels des Empfängers verkodiert. Nur der Empfänger selbst kann sie durch Verwendung seines Private Keys wieder entkodieren.

Die mathematische Darstellung zur Verdeutlichung dieses Mechanismus:

$$E_{PuK}(M) = C$$

$$D_{PrK}(C) = D_{PuK}(E_{PuK}(M) = M$$

E .. Verschlüsselung der Nachricht M mit dem Public Key
PuK .. Public Key
M .. Nachricht
C .. verschlüsselte Nachricht
D .. Entschlüsselung von C mit Hilfe des Private Key
PrK .. Private Key

Das am meisten verbreitete asysmetrische Verschlüsselungsverfahren ist das RSA System, benannt nach den Entwicklern Rivest, Shamir und Adleman. Verwendet wird dieser Algorithmus in vielen bekannten Anwendungen zB.: Secure Shell, Netscape Navigator,..

Funktionsweise: Ausgegangen wird vom Satz von Euler. Dieser besagt: Sind m und n zwei teilerfremde Zahlen so gilt:

$$m^{j(n)} MOD n = 1$$

Zur Erklärung: teilerfremde Zahlen sind Zahlen (kleiner gleich der ersten Zahl) deren größter gemeinsamer Teiler (ggT) gleich 1 ist. Zur Verdeutlichung die Definition von j(n) = Die Anzahl der von n teilerfremden Zahlen.

j(10) = 4
der ggT von 10 und 1 = 1
der ggT von 10 und 2 = 2

der ggT von 10 und 3 = 1
der ggT von 10 und 4 = 2
der ggT von 10 und 5 = 5
der ggT von 10 und 6 = 2
der ggT von 10 und 7 = 1
der ggT von 10 und 8 = 2
der ggT von 10 und 9 = 1
der ggT von 10 und 10 = 10

bei einer Primzahl sieht das Ergebnis immer wie folgt aus:

j(n) = n-1

Beispiel:

j(11) = 10 (1,2,3,4,5,6,7,8,9,10)

Als Folge ergibt sich die Gleichung: j(pq) = (p-1)(q-1)

Für die Erzeugung des Schlüsselpaares werden zwei sehr große Primzahlen (p, q) benötigt, die cirka in der Größenordnung von 10^{200} liegen sollten. Aus diesen beiden Zahlen wird jetzt das Produkt der teilerfremden Zahlen nach dem eulerschen Satz ermittelt.

$$\varphi(n) = (p-1)(q-1)$$

Die Formel zur Berechnung des Schlüsselpaares lautet:

$$(c \bullet d) MOD \varphi(n) = 1$$

c .. Schlüssel zum Verkodieren
d .. Schlüssel zum Entkodieren

Zur Bestimmung von c muß beachtet werden, daß c teilerfremd zu j(n) ist und daß der ggT(c-1,p-1) und der ggT(c-1,q-1) möglichst klein ist. Um dies zu erreichen, kann entweder eine Primzahl gewählt werden die größer als j(n) ist, oder man bestimmt zufällig eine Zahl c kleiner als j(n), für die gilt: ggT(c, j(n)) = 1. Zur Prüfung dieser Gegebenheit dient der euklidische Algorithmus.

Um nun einen Text verschlüsseln zu können, wird der Text in eine Zahlenkombination umgewandelt (zB ASCII oder UNICODE) und anschließend mit folgender

Formel verkodiert:

$C = M^c mod n$

C .. verschlüsselte Nachricht
M .. Text in Zahlenformat
c .. Schlüssel zum Verkodieren

Das heißt der ASCII Wert eines Buchstabens wird mit dem öffentlichen Schlüssel c potentziert. Der ganzzahlige Rest der Division dieser Zahl durch n ergibt die verschlüsselte Nachricht C.

Die Entschlüsselung der Nachricht C erfolgt durch folgende Berechnung:

$M = C^d mod n$

In Worten: die entkodierte Nachricht M ist der ganzzahlige Rest aus C zur Potenz von d (Schlüssel zum Dekodieren) durch n.

Die eigentliche Sicherheit des RSA Algorithmus liegt in der praktischen Unmöglichkeit die zwei Primfaktoren p und q, mit einer Größenordnung von $> 10^{100}$ innerhalb eines relevanten Zeitraumes zu berechnen. Daraus folgt ein n der Größenordnung 10^{200} . Mit den heutigen technischen Möglichkeiten wird ein Zeitraum von ca. 10^7 Jahren angenommen. Natürlich verändern sich die technischen Möglichkeiten und damit die Rechnerkapazitäten rapide. Die Annahme hierbei liegt bei Verdopplung der Rechnerkapazität alle 18 Monate. Wie schon bei der Beschreibung des DES Algorithmus erwähnt, liegt die Gefahr in der Möglichkeit ein großes Rechnernetz durch Nutzung des Internets zu schaffen und damit die Rechnerkapazitäten von 100.000enden Computern gleichzeitig zu nutzen. Um diesem entgegen zu wirken bleibt die Möglichkeit die Größe des Schlüssels von den heute meist genutzten 512 bit auf ein Vielfaches zu erhöhen.

Ein Beispiel für einen derartigen Zusammenschluß von Computern über das Internet war der über ca. vier Jahre laufende Versuch, eine mittels RC5-64 verschlüsselte Nachricht zu dekodieren. Der RC5-64-Standard gehört zu den symetrischen Verschlüsselungsverfahren. Die Aufforderung diese Nachricht zu entschlüsseln stellte die Firma RSA Security Corporation und wurde mit einer Belohnung von 10.000 US Dollar prämiert. Der einzige Lösungsweg der sich für die Entschlüsselung dieser mit 64Bit verschlüsselten Nachricht anbot, war das Ausprobieren aller möglichen Schlüssel. Diese Methode wird auch ”Brute Force” genannt. Um diese Rechenkapazität zur Verfügung stellen zu können, bildete

sich eine Internetgemeinschaft von rund 300.000 Mitgliedern aufgeteilt in 3689 Teams. Jedes Mitglied mußte sich ein Programm auf seinem Computer installieren, welches einen kleinen Teil der Aufgabe übernahm. Nach genau 1757 Tagen wurde der richtige Schlüssel gefunden. Es mußten 82 % der möglichen Schlüssel durchprobiert werden um das Ergebnis zu finden. [Schneider, 2002]

Die Geschwindigkeit der Entschlüsselung gegenüber dem des DES - Algorithmus erhöht sich um den Faktor 1000 bei einer reinen Hardwareentschlüsselung und softwareseitig um den Faktor 100.

Andere asymetrische Verschlüsselungsverfahren, wie zum Beispiel das ”Elliptic Curve” Verfahren werden nicht näher beschrieben, da sie für die Implementierung des Systems in Abschnitt zwei nicht von Bedeutung sind.

3.4 Digitale Signaturen

Sinn einer digitalen Signatur ist es, genauso wie bei einer handschriftlichen Unterschrift, die Richtigkeit und Echtheit von Dokumenten sicherzustellen. Bei digitalen Dokumenten wäre das Einfügen der handschriftlichen Unterschrift als Bilddokument nicht sinnvoll, da diese jederzeit von jedem kopiert und mißbraucht werden könnte. Ebenfalls könnte eine Veränderung der unterschriebenen Dokumente nicht festgestellt werden.

Durch die soeben beschriebenen Sicherheitsprobleme ergeben sich folgende Anforderungen an eine digitale Signatur:

- Eindeutigkeit der Unterschrift:
  Die Unterschrift muß eindeutig dem Unterschreiber zugeordnet werden können. Es muß daher auch die Möglichkeit geben diese Eindeutigkeit nachzuprüfen. Zum Beispiel durch Prüfung in einem öffentlichen Register (siehe PKI 3.5).
- Fälschungssicherheit:
  Es muß sichergestellt werden, daß die abgegeben Signatur nicht einfach nur kopiert worden ist (vergleiche Beispiel mit Unterschrift als Bilddokument).
- abhängig von der unterschriebenen Nachricht:
  Aus dem zuvor beschriebenen Punkt ergibt sich, daß die Unterschrift von

der Nachricht abhängig sein muß.

Die beschriebenen Anforderungen können ideal durch ein asymetrisches Verschlüsselungsverfahren gelöst werden. Das Dokument wird vom Aussteller mit seinem Privat Key verschlüsselt und kann daher nur mehr durch Verwendung des öffentlichen Schlüssels entschlüsselt werden. Da nur der Eigentümer den privaten Schlüssel hat, ist sichergestellt, daß diese Nachricht von ihm stammt.

Zur Verdeutlichung die Darstellung als Formel:

$$E_{PuK}(D_{PrK}(M)) = M$$

E..Entschlüsselung
PuK..Public Key
D..Verschlüsselung
PrK..Private Key
M..Nachricht oder sonstiges Dokument

Diese Form des sicheren Unterschreibens von Dokumenten hat einen Nachteil: der Verschlüsselungsvorgang für ein längeres Dokument ist sehr rechenintensiv und es würde, vor allem bei größeren Dokumenten, sehr viel Zeit in Anspruch nehmen diese mittels eines solchen Verfahrens zu verschlüsseln. Die Lösung des Problems liegt in der Verbindung zweier Methoden, nämlich der Erstellung eines Hash-Codes (siehe Datenintegrität 3.1) für die zu unterschreibende Nachricht und der Verschlüsselung dieses Hashwertes mittels asymetrischen Verfahrens. Damit ist sowohl die Eindeutigkeit der Unterschrift als auch die Richtigkeit der übermittelten Daten gewährleistet.

Bei Benützung des RSA Verfahrens ergibt sich folgende Formel:

$$S = D(h(M)) = h(M)^d mod n$$

S..Signatur
D..Verschlüsselung
h..Hashfunktion
M..Nachricht
(siehe Punkt RSA 3.3)

Um nun eine so signierte Nachricht zu verifizieren, benötigt man:

- die Nachricht

- die Signatur
- den öffentlichen Schlüssel
- und natürlich Informationen über die verwendeten Algorithmen der Hashfunktion und des Verschlüsselungsverfahrens

Folgende Schritte werden nun durchgeführt:

1. Der verschlüsselte Hashwert wird mittels öffentlichen Schlüssel des Signaturausstellers entschlüsselt.
2. Der Hashwert der Nachricht (das Dokument) wird mit dem gleichen Hashalgorithmus berechnet.
3. Der entschlüsselte Hashwert und der soeben erstellte Hashwert werden verglichen.

Stellt sich heraus, daß die beiden Hashwerte ident sind so ist sichergestellt, daß dies die signierte Nachricht ist und keine Veränderungen vorgenommen wurden.

formalisiert dargestellt:

E(S)=h(M)

E..Entschlüsselung
S..Signatur
h..Hashfunktion
M..Nachricht oder Dokument

Ein einziges Problem hierbei wurde bisher noch nicht ausreichend erklärt: woher bekommt der Empfänger einer Nachricht den öffentlichen Schlüssel des Ausstellers? Das Kapitel Public Key Infrastructure gibt Antworten auf diese Frage.

3.5 Public Key Infrastructure oder kurz PKI

Bei den asymetrischen Verschlüsselungsverfahren spielt die Veröffentlichung der öffentlichen Schlüssel eine tragende Rolle, da einerseits Signaturen überprüft werden sollen und andererseits Nachrichten und Dokumente so verschlüsselt werden sollen, daß diese nur mehr für den Empfänger lesbar sind (Anwendungsbeispiel ist die sichere Übertragung von Emails mittels PGP - Pritty Good Privacy). Bei

der Bereitstellung von öffentlichen Schlüsseln muß gewährleistet sein, daß dieser öffentliche Schlüssel auch wirklich zu dem Schlüsselpaar des Besitzers gehört. Daraus ergibt sich auch, daß die Identität des Schlüsselinhabers von einer vertrauenswürdigen Stelle zertifiziert werden muß, ähnlich wie es staatliche Stellen bei der Ausstellung eines Reisepasses tun. Organisationen die Identitäten zertifizieren nennt man ”Certification Authorities” oder kurz ”CA”. Um auch sicher gehen zu können, daß diese Organisationen berechtigt sind Identitäten zu zertifizieren müssen auch sie sich bei anderen CAs zertifizieren lassen. Es entsteht daher ein Netzwerk von gegenseitigen Zertifikaten. Diese Form einer Infrastruktur für digitale Identitäten wird für den öffentlichen Raum benötigt. Es können natürlich die gleichen Methoden für den Aufbau einer PKI innerhalb einer Firma oder sonstigen Organisationsform verwendet werden in der die Organisation selbst die Funktionen einer CA übernimmt. Wenn diese Identitäten jedoch nicht nur innerhalb dieser Organisation verwendet werden, so muß sich die Organisation selbst wieder durch eine andere CA zertifizieren lassen.

Zusammenfassung der Aufgaben einer CA:

1. Zertifizierung einer Identität
2. Bereitstellung der öffentlichen Schlüssel
3. Gegenseitige Zertifizierung der CAs

3.5.1 X.509 Zertifizierungsformat

Um die Einheitlichkeit der Zertifikate zu wahren, wurde das Format eines Zertifikates durch die International Standard Organisation (ISO) und die Internationl Telecommunication Union - Telecommunications (ITU-T) standardisiert.

Die Notation eines Zertifikates sieht wie folgt aus:

Certificate ::=SEQUENCE
version [0] EXPLICIT Version DEFAULT v1,
serialNumber CertificateSerialNumber,
signature AlgorithmIdentifier,
issuer Name,
validity Validity,
subject Name,
subjectPublicKeyInfo SubjectPublicKeyInfo,
issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,

extensions [3] EXPLICIT Extensions OPTIONAL

- version: gibt die Versionsnummer des Zertifikatsstandards an, auf die es sich bezieht und dient der Sicherstellung der Kompatibilität.
- serialNumber: eindeutige Seriennummer des Zertifkats, zu vergleichen mit der Nummer eines Reisepasses.
- signature: die einzelnen Zertifikate werden mit einer digitalen Signatur der CA ”unterschrieben”
- issuer: Name der ausstellenden CA
- validity: die Gültigkeit des Zertifikates von ... bis ...
- subject: Name des Zertifikateigentümers, also der Person (natürlich oder juristisch) für die dieses Zertifikat ausgestellt wurde.
- subjectPublicKeyInfo: der öffentliche Schlüssel plus Informationen über den verwendeten Algorithmus
- issuerUniqueID: Zusatzinformationen zur CA
- subjectUniqueID: Zusatzinformationen zum Eigentümer des Zertifikates
- extensions: allgemeine Zusatzinformationen, zum Beispiel: wie das Zertifikat benützt werden sollte,...

SEQUENCE bedeutet in diesem Zusammenhang nur, daß es sich um eine Liste von Zertifikaten handelt.

In manchen Fällen ist es notwendig ein Zertifikat als ungültig zu erklären. Ein Beispiel hierfür wäre, wenn der private Schlüssel verloren gegangen ist. Die ungültigen Zertifikate werden in einer Liste zusammengefasst, genannt ”certificate revocation list (CRL)”.

Die Liste hat folgendes Format:

CertificateList ::=SIGNED SEQUENCE
version Version OPTIONAL,
signature AlogrithmIdentifier,
issuer Name,

thisUpdate Time,
nextUpdate Time OPTIONAL,
revokedCertificates SEQUENCE OF SEQUENCE
userCertificate CertificateSerialNumber,
revocationDate Time,
crlEntryExtensions Extensions OPTIONAL OPTIONAL,
crlExtensions [0] Extensions OPTIONAL

Feldbeschreibung:

- thisUpdate: Datum der Aktualisierung der CRL.
- nextUpdate: optional; das Datum der nächsten Aktualisierung.
- revokedCertificate: die Liste der ungültigen Zertifikate.
- userCertificate: die Seriennummer des ungültigen Zertifikates.
- revocationDate: Datum der Ungüligkeitserklärung.
- crlEntryExtensions und crlExtensions: siehe nachfolgende Beschreibung.

Der Term ”Sequence of Sequence” bedeutet, daß in einer CRL eine Vielzahl von Zertifikaten für ungültig erklärt werden können. Als Beispiel für optionale Zusatzinformationen der Felder crlEntryExtensions und crlExtensions kann der Grund des Gültigkeitsentzuges beschrieben werden. Folgende Einträge können laut Spezifikation dafür verwendet werden:

- unspecified: Grund ist nicht bekannt.
- keyCompromise: die Geheimhaltung des privaten Schlüssels ist nicht mehr gegeben.
- cACompromise: das Vertrauen in die CA ist nicht mehr gegeben.
- affiliationChanged: Felder des Zertifikates haben sich geändert, wie zum Beispiel der Betreff.
- superseded: das Zertifikat wurde durch ein anderes ersetzt.
- cessationOfOperation: das Zertifikat wird nicht mehr benötigt.
- certificateHold: das Zertifikat wird temporär für ungültig erklärt.

- removeFromCRL: ein CRL Eintrag soll entfernt werden, zum Beispiel weil er temporär für ungültig erkärt wurde.

Durch die ständig wachsende Anzahl der ungültigen Zertifikate vergrößern sich auch die CRL-Daten. Es wäre daher nicht sehr sinnvoll jedesmal die neueste vollständige Version der CRL zu laden. Aus diesem Grund gibt es die Möglichkeit nur die Veränderungen zwischen der letzten Version der CRL und der Neuesten zu beschreiben. Dieses Beschreibungsformat nennt sich delta-CRL.

3.5.2 Verfügbarkeit von Zertifikaten

Wie bereits beschrieben, muß der öffentliche Zugang zu Zertifikaten gegeben sein, damit die Authentizität des Dokumentenaustellers geprüft werden kann. Für diesen öffentlichen Zugang zu Information bedient man sich sogenannter Verzeichnisdienste (Directory Services). Die Abfrage für ein Zertifikat erfolgt dann mittels des LDAP-Protokolls (Lightweight Directory Access Protocol). Das LDAP-Protokoll baut auf TCP/IP auf und speichert Information über sogenannte Entitäten, welche wiederum Attribute haben können und mittels eines eindeutigen Namens identifizert (distinguished name = dn) werden. Durch die hierarchische Anordnung der Einträge entsteht eine Baumstruktur. Die Entitäten sind in Form von Objekten definiert und haben daher spezielle Eigenschaften; zum Beispiel das Objekt Zertifikat, das wie bereits beschrieben zahlreiche formale Bedingungen erfüllen muß.

Ein Bespiel für eine LDAP-Abfrage, die mittels Browser abgesetzt wird:
ldap://rechnername.domainname/o=domainname?userCertificate?sub?(cn=Thomas Muster)

Mittels LDAP können zwar Zertifikate abgefragt werden, die Prüfung des Zertifikates muß jedoch zusätzlich von der anfragenden Stelle vorgenommen werden. Zur Vereinfachung dieses Vorganges, kann mittels des OCSP (Online Certificate Status Protocol) eine Prüfung eines Zertifikates vorgenommen werden. Dabei wird eine Anfrage auf die Gültigkeit eines Zertifikates an eine CA gestellt und anschließend beantwortet.

Folgende Antwortmöglichkeiten bietet OCSP:

- good: das Zertifikat ist gültig.
- revoked: das Zertifikat ist ungültig.

- unknown: es wurde keine Information über das Zertifikat gefunden.

Kapitel 4

Mobile Endgeräte und ihre Kommunikationstechnologien

Schon der Titel der Diplomarbeit verrät, daß sich die zu entwickelnde Bezahlungslösung auf die Bedürfnisse von mobilen Endgeräten konzentriert. Dieses Kapitel soll daher eine Einführung in die Welt der verschiedenartigen Gerätetypen, sowie in die unterschiedlichen Kommunikationstechnologien, sein.

4.1 Mobile Endgeräte

Durch die unterschliedlichen Kundenbedürfnisse werden eine Vielzahl von verschiedensten mobilen Endgeräten am Markt angeboten. Die unscharfen Grenzen bei der Unterscheidung dieser Geräte machen eine genaue Einteilung fast unmöglich und es ist daher nur eine grobe Kategorisierung möglich:

- Mobiltelefon oder Handy (siehe Abb. 4.1):
  Das Handy ist sicherlich das meistgenutzte mobile Endgerät für die Übertragung von Sprache und Kurzmitteilungen (Short Message Service oder SMS). Der Anteil der Österreicher, die das Handy als mobiles Kommunikationsmittel benützen, ist von 81 % [Wirtschaftskammer, 2002] im Jahr 2002 auf über 90 % [derstandard.at, 2004] im Jahr 2004, gestiegen. Diese Marktdurchdringung bedeutet daher eine Marktsättigung für diesen Bereich. Mittlerweile bieten bereits Mobiltelefone zusätzliche Anwendungen, wie zum Beispiel ”Personal Information Management (PIM)” Funktionalitäten oder integrierte Kameras, an.

- Smartphone (siehe Abb. 4.2):
  Wie bereits erwähnt ist die Unterscheidung zwischen den einzelnen Produkten schwierig und verschwommen. Smartphones bieten gegenüber den

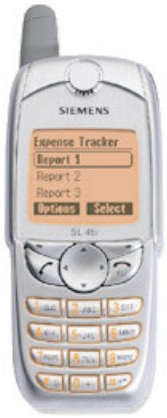

Abbildung 4.1: Handy Siemens SL45i

Mobiltelefonen ein größeres und hochauflösenderes Display sowie erweiterte PIM Funktionalitäten. Das Display ist zumeist auch als Touscreen ausgeführt.

Abbildung 4.2: Smartphone Siemens SX45

- Personal Digital Assistant (PDA) (siehe Abb. 4.3):
  Der PDA verfügt im Normalfall, gegenüber den bisherigen Endgeräten, nicht über die Möglichkeit über ein Mobilfunknetz wie GSM zu kommunizieren. Dafür werden Anwendungen ähnlich einem PC, sowie bessere Rechnen- und Speicherkapazitäten, geboten.

- Notebook:
  Um die Aufzählung zu vervollständigen muß auch das Notebook erwähnt werden, obwohl es nicht Zielgruppe des zu implementierenden Prototypen ist.

Abbildung 4.3: PDA Sharp Zaurus

Idealerweise sollte ein Bezahlungssystem für mobile Endgeräte alle angeführten Systeme unterstützen.

4.2 Kommunikationsstandards

Die nachfolgend beschriebenen Technologien zur Übermittlung von Daten bilden die Basis für die Implementierung einer mobilen Bezahlungslösung. Prinzipiell wäre es natürlich optimal, wenn das Ergebnis der Diplomarbeit eine Vielzahl an existierenden Kommunikationsmitteln unterstützen würde.

4.2.1 Short Message Service (SMS)

SMS ist ein Standard zur Übertragung von kurzen Textnachrichten in Telekommunikationsnetzen. Ursprüglich war dieser Dienst nur in Mobilfunknetzen möglich, mittlerweile können auch im Festnetzbereich kurze Textnachrichten versendet und empfangen werden. Die Nachrichtenlänge ist mit 160 Zeichen beschränkt.

Es gibt bereits zwei Nachfolgestandards:

1. Enhanced Messaging Service (EMS):
   Erlaubt das Versenden von Textnachrichten mit bis zu 255 Zeichen beziehungsweise das Aufteilen einer längeren Nachricht.

2. Multimedia Messaging Service (MMS):
   Eine MMS-Nachricht kann neben Text auch farbige Bilder, Musik und Videos enthalten.

SMS hat sich im Verlauf der letzten Jahre von einem beliebten Kommunikationsmittel unter Jugendlichen zur Basistechnologie von Geschäftsanwendungen entwickelt. Die Einsetzbarkeit dieser Technologie reicht von der Möglichkeit per SMS über Ergebnisse abzustimmen bis hin zur Integration in Bezahlungssystemen (siehe Kapitel 2.4.1 Paybox).

Ein Nachteil von SMS ist, daß nicht gewährleistet werden kann, ob eine Nachricht angekommen ist oder nicht. In etwa vergleichbar mit Internetprotokollen, wie zum Beispiel UDP, ist SMS ein verbindungsloses Kommunikationsmittel. Das heißt es wird keine direkte Verbindung zwischen Sender und Empfänger, zur Übertragungüberwachung der Nachricht, aufgebaut.

4.2.2 Irda (Infrared Data Association)

Eine der ältesten Möglichkeiten, Daten nicht über ein Kabelnetz zu übertragen bietet die Infrarottechnologie. 1994 wurde von der Infrared Data Association der erste Standard für die Übertragung von Daten über einen Lichtleiter spezifiziert. Das Infrarotlicht arbeitet mit einer Wellenlänge von 850 bis 900 nm. Für die Übermittlung der Daten müssen sich Empfänger und Sender innerhalb einer Reichweite von einem Meter befinden. Ebenfalls muß auf die direkte Sichtverbindung geachtet werden, da die Breite des Sendekegels maximal 30 Grad beträgt. Die Anforderungen an Sichtbarkeit und Reichweite können je nach Einsatzgebiet von Vorteil oder Nachteil sein. Bei der Übertragung von geheimen Daten ist es von Vorteil, daß kein anderer, der sich außerhalb des 1-Meter Radius befindet, mitlauschen kann. Das Zurechtrücken der Notebooks für die Übertragung der Daten ist sicherlich ein mühseliger Nachteil.

War die Geschwindigkeit in den Anfängen der Infrarot-Technologie noch auf Übertragungsraten von maximal 115 kbit/s beschränkt, so erlauben aktuelle Spezifikationen eine Geschwindigkeit von 4 mbit/s beziehungweise 16 mbit/s. Die Bezeichnung für die zwei schnellen Spezifikationen lautet Fast InfraRed (FIR).

Die Datenübertragung per Infrarot-Schnittstelle bedarf einer Reihe von Protokollen:

- Serial Infrared Link Access Protocol (IrLAP):
  bietet die Grundlage für den Aufbau von Verbindungen und enthält ein 8 bit langes Adressfeld, Platz für 2045 Byte Nutzdaten und eine 16-Bit lange Prüfsumme.
- Infrared Link Management Protocol (IrLMP):
  sichert die Kontrolle über den Datenfluß.

- Link Management Information Access (LM-IAS):
  bietet die Grundlage zum Aufbau von Adhoc-Verbindungen zwischen zwei Geräten.

- Link Management Multiplexer (LM-MUX):
  regelt den Zugriff unterschiedlicher Applikationen auf die Infrarotschnittstelle.

4.2.3 Bluetooth

Durch ein von der Firma Ericsson gestartetes Projekt mit dem Titel ”Multi-Communicator Link” (zur Erforschung der kabellosen Kommunikation zwischen Endgeräten), entstand die Idee zur Realisierung der Bluetooth Technologie. Der Name Bluetooth wurde vom König von Dänemark, Harald Gormsen, beziehungsweise von dessen Beinamen, Blåtand, abgeleitet.

Die Idee diese kostengünstige Lösung für eine drahtlose Netzwerktechnik zu entwickeln, wurde ab 1998 von einer Vielzahl an Firmen wie zum Beispiel Intel, IBM, Nokia oder Toshiba unterstützt. Nach einer eher zaghaften Einführung von Bluetoothgeräten am Markt, kann mittlerweile auf eine Vielzahl von Entwicklungen, die diese Technologie unterstützen zurückgegriffen werden. Verwendung findet der Standard in Mobiltelefonen, Laptops, Headsets, Tastaturen und so weiter.

Für die Funkübertragung wird das lizenzfreie 2.4-Ghz Frequenzband verwendet. Grundlage für die Funktionsweise von Bluetooth ist das so genannte ”Frequency Hopping Spread Spectrum” oder zu-deutsch Frequenzsprungverfahren. Dabei werden 79 Kanäle innerhalb des Frequenzbandes zur Kommunikation verwendet. Ein Zufallsgenerator bestimmt die Folge der Frequenzen, auf die gewechselt wird. Damit unterschiedliche Geräte miteinander kommunizieren können, müssen sie die Reihenfolge und die Zeitspanne für die Nutzung einer Frequenz kennen. Die beteiligten Geräte müssen also den gleichen Startparameter erhalten, damit sie synchron bleiben können. Durch diesen Wechsel wird ein Abhören der Daten erschwert.

Organisiert wird eine Kommunikation von Bluetoothgeräten in einem sogenannten Piconetz. Dabei muß eines der kommunizierenden Geräte als ”Master” fungieren und die Koordination, zum Beispiel über die bereits erwähnte Vergabe der Frequenzfolge, übernehmen. Alle anderen Teilnehmer werden als ”Slave” bezeichnet und können nicht direkt miteinander kommunizieren, sondern müssen sich über den Master unterhalten.

Dieses Piconetz beschränkt die Teilnehmeranzahl auf einen Master und sieben Slaves. Es können jedoch weitere Geräte in einer Art Parkmodus an der Kommunikation teilnehmen. Diese werden auf Anfrage bezeihungsweise Anordnung des Masters, statt einem anderen Slave, in die Kommunikation aufgenommen. Die tatsächliche Anzahl der Slaves kann sich auf bis zu 255 Endgeräte erhöhen. Durch eine Erweiterung der Spezifikation können Piconetze zusammengeschlossen werden und es entsteht ein sogenanntes ”Scatternetz”, in dem die verschiedenen Piconetze untereinander kommunizieren können. Natürlich ergibt sich aus einer solchen Konstellation eine erhebliche Performanzeinbuße.

Der Aufbau einer Bluetoothverbindung gliedert sich in drei Stufen:

1. Inquiry:
   Dabei wird der bereitstehende Funkraum nach kommunikationsbereiten Geräten abgesucht. Wird eines gefunden, so werden Infromationen über die Zeittakte und die Zugehörigkeit zur Geräteklasse ausgetauscht. Soll nun eine Verbindung aufgebaut werden, so wird vom Sender der Inquiry die Paging Prozedur angestossen.

2. Paging:
   Darunter versteht man den eigentlichen Verbindungsaufbau. Es werden die Frequenzfolge, der Zeittakt, sowie die Master- und Slaverolle festgelegt. Der Slave erhält eine Active Member Address, mittels dieser er Datenpakete an den Master senden kann.

3. Pairing:
   Beim Pairing wird eine Zutrittsberechtigung zum Piconetz in Form eines 128 bit langen Schlüssels erstellt. Dieser wird ebenfalls, durch einen bei der Initialisierung einmalig erzeugten Schlüssel, verschlüsselt übersandt und auf beiden Geräten gespeichert. Für eine sicher Datenübertragung wird auf Basis des Zutrittsberechtigungsschlüssels eine Verkodierung initialisiert.

Die Adressierung zwischen den Bluetoothgeräten erfolgt, ähnlich den Ethernet Netzwerkkarten, über eine weltweit eindeutige 48 bit lange Gerätekennung.

Bei den Verbindungsgeschwindigkeiten kann unterschieden werden in

- asynchrone Datenübermittlung:
  Ist die Vollständigkeit der Übertragung von Wichtigkeit, so wird diese Methode verwendet. Zusätzlich kann unterschieden werden in

  1. asymetrische Verbindungen, mit Geschwindigkeiten von bis zu 723,2 kbit/s in eine, beziehungsweise 57,6 kbit/s in die andere Richtung.

2. symetrische Verbindungen, mit Geschwindigkeiten von bis zu 433,9 kbit/s in beide Richtungen.

- synchrone Datenübermittlung:
  Für zeitkritische Anwendung, wie zum Beispiel der Sprachübertragung, wird diese Art der Kommunikation verwendet. Hier ist die Reihenfolge der Daten wichtiger als die Vollständigkeit - lieber ein abgehacktes Gespräch als ein Gespräch mit vertauschten Stimmenlauten.

Je nach Einsatzgebiet wurde auch die Reichweite in drei Leistungsklassen geteilt:

1. maximale Sendeleistung von 100 Milliwatt und einer maximalen Reichweite von 100 Metern

2. maximale Sendeleistung von 2,5 Milliwatt und einer maximalen Reichweite von 20 Metern

3. maximale Sendeleistung von 1 Milliwatt und einer maximalen Reichweite von 10 Metern

4.2.4 Near Field Communication - NFC

Wie bereits der Name dieses Kommunikationsstandards verrät, soll der Datenaustausch auf ein örtliches Naheverhältnis beschränkt sein. Dieser Standard soll jedoch keine direkte Konkurrenz zu Bluetooth darstellen sondern sich auf eine Kommunikation zwischen Geräten beschränken, die nur wenige Zentimeter (0 - 20 cm) von einander entfernt liegen. Nicht nur in der Reichweite unterscheiden sich NFC und Bluetooth, sondern auch durch die Übertragungsgeschwindigkeit, die mit maximal 424 kBit/s deutlich unter der von Bluetooth liegt. Der Frequenzbereich der Near Field Communication beschränkt sich auf 13.56 Mhz.

Die Idee zur Begründung dieses Verfahrens liegt in der menschlichen Art zu kommunizieren. Wollen zwei Personen in ein Zwiegespräch eintreten, so suchen sie die Nähe ihres Unterhaltungspartners. Auch zwei Endgeräte sollen sich durch eine räumliche Annäherung unterhalten können, um zum Beispiel die Adressen auszutauschen.

Die Einsetzbarkeit dieses Verfahrens beschränkt sich nicht nur auf den simplen Austausch von Adressdaten, sondern kann auch zum Bespiel für den Aufbau einer Bluetooth Verbindung genutzt werden. Hierfür muß nicht, wie sonst üblich ein aufwändiges Pairing-Verfahren gestartet werden, sondern nur die Geräte eng aneinander gehalten werden und schon identifizieren sich diese und starten eine

Bluetooth Verbindung. Eine sonst übliche Funkraumsuche, Gerätewahl, Dienstwahl und Passwortübergabe ist daher nicht nötig. Der Einsatzbereich ist für alle ”Smart Objects” wie SmartCards oder Mobiltelefone ausgerichtet.

Die Technologie wurde bereits durch folgende Dokumente spezifiziert [NFC-Forum, 2004]:

1. ECMA-340 ”Near Field Communication - Interface Protocol (NFCIP-1)”
2. ISO/IEC 18092 (ISO/IEC JTC1 adopted ECMA-340 under their fast track procedure)
3. EMCA-352 ”Near Field Communication - Interface Protocol - 2 (NFCIP-2)”

Wichtige Unternehmen wie Phillips, Sony oder Nokia planen eine baldige Implementierung dieser Technologie und erhoffen sich dadurch innovative Lösungen im Bereich der mobilen Bezahlungssysteme.

Kapitel 5

Smart Cards

Unbewußt werden von uns täglich Smart Cards in Form von Bankomatkarten, Kundenkarten, Berechtigungskarten, etc. verwendet. Sie ermöglichen uns bestimmte Informationen sicher zu speichern.

Welche Eigenschaften weist eine Smart Card auf?

Eine Smart Card ist ein Ein-Chip Microcomputer der auf einer Plastikkarte angebracht ist. Der Chip hat in etwa die Größe von 25 mm^2. Es können damit Informationen verschlüsselt aufbewahrt werden; zum Beispiel elektronische Signaturen. Ein goßer Vorteil einer solchen Lösung ist, daß der Chip unabhängig arbeiten kann und die gespeicherten Daten die Karte nie verlassen müssen. Sie bieten die Möglichkeit Algorithmen abzuarbeiten und Passwörter zu kontrollieren.

Plastikkarten in Form von Scheck- und Kreditkarten gibt es schon seit einigen Jahrzehnten. Vor dem Zeitalter der Smart Card waren diese Plastikkarten nur mit einem Magnetstreifen ausgerüstet. Dieser Magnetstreifen ermöglichte zwar die Speicherung von Daten, konnte jedoch keine Funktionen wie Berechnungen und Passwortvergleiche ausführen. Ausserdem waren diese Informationen nicht davor geschützt, kopiert zu werden. Durch ein Patent im Jahre 1968 wurde es möglich die Daten vor dem Kopieren zu schützen und die Smart Card war geboren.

Den ersten Bekanntheitsgrad erhielten die Smart Cards durch die Einführung der elektronischen Geldbörse - in Österreich läuft dieses System unter dem Namen ”Quick”. Durch die Möglichkeit Informationen sicher zu speichern und auch Berechnungen durchzuführen eignet sich die Smart Card ideal als Zahlungsinstrument. Auch in den Mobiltelefonen arbeitet eine Smart Card in Form der SIM - Karte, die es ermöglicht den Mobiltelefonbenützer eindeutig zu identifizieren und auch persönliche Daten, wie etwa ein Telefonbuch zu speichern. Eine weite

Verbreitung findet diese Smart Card auch als Kundenkarte oder als Zutrittsberechtigungskarte.

5.1 Architektur der Smart Card

Der Chip (siehe Abb. 5.2) besteht aus einer CPU (Central Processing Unit) und Speicher. Durch die Normung der Smart Card in der ISO 7816 wird ein einheitlicher Industriestandard, sowie eine weltweite Lesbarkeit der Daten garantiert.

Bestandteile:

- CPU (Central Processing Unit): Je nach Preiskategorie kann der Prozessor ein 8-bit, 16-bit oder 32-bit Mikroprozessor sein. Zusätzlich besteht noch die Möglichkeit einen Kryptographie-Koprozessor zu integrieren. Dieser beschleunigt die Verschlüsselungsprozesse um ein Vielfaches.
- ROM (Read Only Memory): Der ROM Speicher ist jene Einheit, die bei der Produktion ”hardcodiert” wird und daher später nicht mehr verändert werden kann. Die Informationen des ROM werden daher dauerhaft auch ohne Stromversorgung gespeichert.
- EEPROM (Electrical Erasable Programmable Read Only Memory): Grundsätzlich übernimmt der EEPROM die gleiche Funktion wie der ROM, bietet jedoch die Möglichkeit die Informationen wieder zu überschreiben. Der EEPROM ist durch seine erweiterten Funktionen und die damit aufwendigere Produktion kostenintensiver. Die meisten Smart Cards sind daher mit der billigeren ROM - Lösung ausgestattet. Ein Beispiel für eine Speicherung auf EEPROM ist die Speicherung von Telefonnummern auf einer SIM - Karte eines Mobiltelefons.
- RAM (Random Access Memory): Dieser Speicher wird auch als Arbeitsspeicher bezeichnet. Er speichert die Daten nur solange die Stromversorgung aufrecht erhalten bleibt. Hauptaufgabe ist die Auslagerung von Informationen während eines CPU - Prozesses.

5.2 Sicherheit der Smart Card

Wie bereits eingangs erwähnt, bietet die Smart Card diverse Sicherheitsstrukturen. Die Daten werden verschlüsselt gespeichert und können nicht kopiert werden.

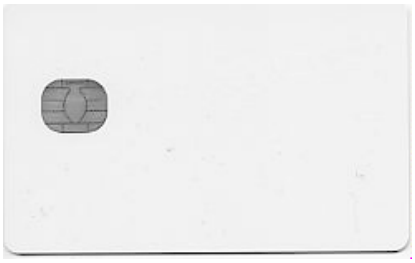

Abbildung 5.1: Handelsübliche Chipkarte

Abbildung 5.2: Smart Card Chip

Auf Ebene des CHIPs selbst erschwert die Verschmelzung zwischen CPU und Speicher, die Lesbarkeit der Daten die zwischen CPU und Speicher ausgetauscht werden. Zusätzlich werden noch Sicherheitsmaßnahmen integriert, die verhindern sollen, daß ein Auslesen der Daten durch physisches Ablösen einer Chip - Ebene möglich wird. Falls ein solcher Einbruch bemerkt wird, reagiert das System sofort mit Unterbrechen des aktuell ausgeführten Prozesses. Ebenfalls kann eine zu hohe Voltanzahl festgestellt und Gegenmaßnahmen getroffen werden.

5.2.1 Sicherheitsrisiken

Die Smart Card unterscheidet sich gegenüber herkömmlichen Computersystemen, wie zum Beispiel Personal Computer (PC) oder Personal Digital Assistant (PDA), durch die physische Trennung von Datenausgabe, Dateneingabe und deren Speicherung. Auf der Smart Card selbst werden eigentlich nur Daten verschlüsselt gespeichert und es wird eine Schnittstelle für spezielle Lese- und Schreibgeräte angeboten. Ein PC-System besteht aus einem Monitor über den Daten betrachtet werden, einer Tastatur als Eingabegerät und dem Rechenwerk samt Speicherlösung. Der Eigentümer eines PCs kann bestimmen welche Software installiert wird, welche Protokolle verwendet werden dürfen und hat dadurch eine Kontrolle über alle Vorgänge rund um dieses Gerät. Die Smart Card hingegen befindet sich in einem Vertrauensmodell. Der Eigentümer der Karte kann nicht bestimmen welches

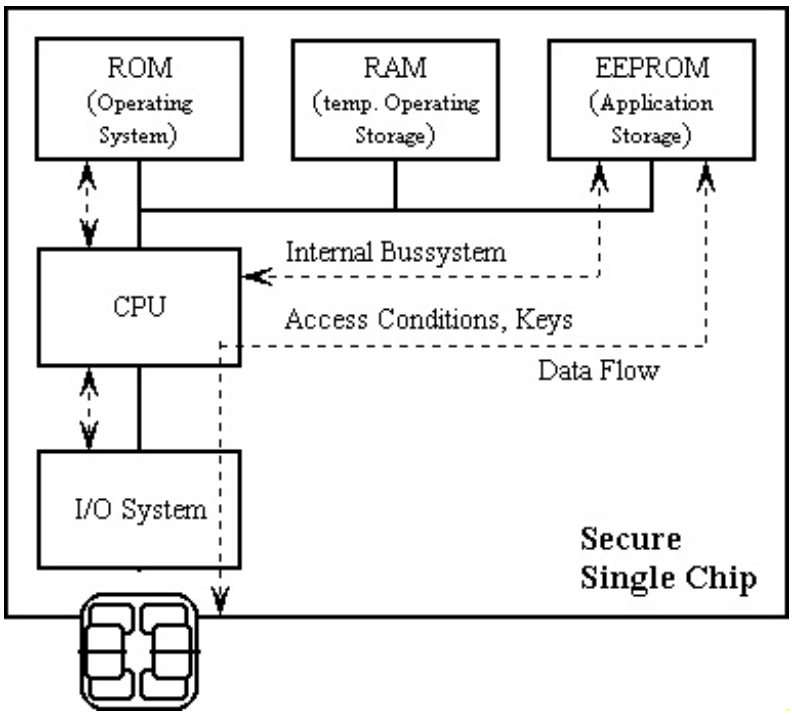

Abbildung 5.3: Aufbau eines Chips (Bildquelle: http://www.iscit.surfnet.nl/team/Barry/thesis/scards.html)

Betriebssystem auf der Karte installiert wird, er muß auf die Aussagen der ausgebenden Institution beziehungsweise des Herstellers vertrauen.

Übliche Beteiligungen am Vertrauensmodell der Smart Card am Beispiel einer Bankomatkarte ([Schneider, 1999]):

- Eigentümer der Karte:
  Der Eigentümer der Karte wäre in diesem Beispiel ein Konsument der die Möglichkeit hat in einem Geschäft mittels dieser Karte unter Verwendung einer Personal Identification Number (PIN) einzukaufen. Die Verantwortung des Konsumenten beschränkt sich auf Besitzstörungen wie Diebstahl oder auf einen möglichen Verlust. Der Konsument hat jedoch auch keine Information über die auf der Karte gespeicherten Daten und der verwendeten Software.

- Dateneigentümer:
  Die ausgebende Institution ist bei einer Bankomatkarte das Bankinstitut. Welche Daten auf der Karte gespeichert werden, um den Eigentümer als Kontoinhaber dieser Bank zu identifizieren, bleibt der ausgebenden Bank

vorbehalten. Die Bank ist daher auch der Eigentümer der gespeicherten Informationen.

- Hersteller:
  Der Produzent der Smartcard bestimmt die Architektur sowie das Betriebssystem der Karte. Es kann auch vorkommen, daß der Hersteller die Produktion der Karten an eine andere Firma auslagert und sich die Anzahl der beteiligten Parteien dadurch nochmals vergrössert.

- Softwareentwicklung:
  Der Ersteller des Betriebssytems legt die verwendeten Programmteile und Protokolle für die Kommunikation mit den Schnittstellen fest.

- Terminal:
  Die Verwendung der Bankomatkarte als Bezahlungsmittel in Geschäften wird über einen Terminal abgewickelt. Es werden daher die sicherheitskritischen Daten durch den Terminal eingelesen und weiterverarbeitet.

Die Aufzählung der beteiligten Parteien zeigt sehr gut, wie groß dieses Vertrauensmodell ist und wie groß die Gefahrenquelle sein kann, falls nur eine dieser Parteien das in sie gesetzte Vertrauen mißbraucht. Angriffe von Beteiligten wären, zum Beispiel durch den Karteninhaber, der den Terminaleigentümer zu betrügen versucht oder durch einen Kartenaussteller, der den Karteninhaber zu betrügen versucht, denkbar. Der Mißbrauch ist jedoch nicht nur auf die Gefahren der durch die Vielzahl der Vertrauensmodellmitglieder entsteht beschränkt, sondern entsteht auch durch Außenstehende. Beim Austausch der Daten zwischen Terminal und Karte werden wie bei der Kommunikation von anderen Computersystemen sogenannte Protokolle verwendet. Diese bieten einen möglichen Angriffspunkt für Außenstehende.

Jedes dieser Angriffsmodelle wäre denkbar und wurde zum Teil auch schon versucht.

5.3 Software

Grundsätzlich wird bei der Smart Card Software unterschieden in:

1. OFF-Card Software: Hierbei wird die Software, welche auf der Smart-Card ausgeführt werden soll, erst nach der Produktion der Smart Card erstellt und installiert. Beispiel hierfür ist die Java-Card. Es können daher Smart Cards individuell nach den Bedürfnissen erstellt und verändert werden.

2. ON-Card Software: Diese Software muß schon vor der Produktion der Smart Card fertiggestellt sein und wird direkt in den Chip integriert ohne die Möglichkeit der Veränderung. Bei der Implementierung des Betriebssystems, sowie der kryptographischen Fähigkeiten der Karte ist diese Art der Software unbedingt notwendig, da ansonsten die erwähnte Sicherheit der Karten nicht gewährleistet wäre. Zu vergleichen ist dies mit dem BIOS (Basic IN and OUT System) eines PCs (Personal Computers), welches ebenfalls in die Hardware integriert ist.

3. Dateisystem: Für die dauerhafte Speicherung von zusätzlichen Daten wird ein Dateisystem implementiert. Hiermit ist nicht die bereits erwähnte physische Speicherung von Daten auf dem EEPROM - Speicher gemeint, sondern die Möglichkeit für Software den EEPROM-Speicher anzusprechen (Beispiel: Speichern von Telefonnummern).

5.3.1 Betriebssystem

Die Architektur von Chipkarten hat bereits gezeigt, daß keine Benutzeroberflächen, sowie keinerlei Zugriffsmöglichkeiten auf externe Speichermedien benötigt werden. Die Betriebssysteme sind daher im Gegensatz zu den allgemein bekannten Betriebssystemen ganz auf die sichere Ausführung von Programmen und den geschützten Zugriff auf Daten konzipiert. Durch die Speicherung der Programmodule im ROM-Speicher ist die Progammierung sehr eingeschränkt, da nach der Herstellung des ROMs keinerlei Änderungen mehr vorgenommen werden können. Dies bedeutet, daß bei einem Programmierfehler der ganze Chip ausgetauscht werden muß. Sind solche Chipkarten bereits im Umlauf, so müssen teure Umtauschaktionen gestartet werden. Betriebssysteme müssen daher sehr sorgfältig programmiert werden, beziehungsweise längeren Testphasen unterzogen werden. Fehler wie sie zum Beispiel oft bei PC-Betriebssystemen vorkommen, die einem möglichen Angreifer Hintertüren zum System öffnen, müssen gänzlich ausgeschlossen werden.
Chipkartenbetriebssysteme haben nicht nur spezielle Anforderungen an die sichere und stabile Ausführung, sondern auch an sehr schnelle kryptografische Funktionen. Zur Beschleunigung des Programmcodes wird sehr oft die Programmiersprache Assembler oder eine hardwarenahe Hochsprache wie C verwendet und auf Multitaskingfunktionalitäten verzichtet.

Die Hautpaufgaben eines Chipkartenbetriebssystemes können wie folgt zusammengefasst werden:

- Datenübertragung von und zur Chipkarte

- Datenverwaltung
- sichere und schnelle Ausführung von Algorithmen

Bei den Betriebssystemen kann unterschieden werden in:

1. Chipkartenbetriebssyteme ohne nachladbaren Programmcode
2. Chipkartenbetriebssyteme mit nachladbaren Programmcode

5.3.2 Chipkartenbetriebssyteme ohne nachladbaren Programmcode

Diese Art von Chipkartenbetriebsystem ist die älteste und bietet keine Möglichkeit, einen selbsterstellten Programmcode nachträglich zu laden und auf der Chipkarte auszuführen. Für bestimmte Anwendungen werden diese Chipkarten sicherlich auch in Zukunft eine Rolle spielen, da sie durch die nichtvorhande Möglichkeit Aufrufe des Betriebssystems durch Dritte auszuführen einen Sicherheitsvorteil besitzen.

Jegliche Kommunikation des Betriebssystems mit der Umgebung läuft direkt über die seriellen I/O (Input/Output) Schnittstellen ab. Der I/O Manager prüft alle einkommenden/ausgehenden Kommandos und verfügt über Fehlererkennungs- und -korrekturmechanismen. Nach dem Empfang eines Kommandos prüft und entschlüsselt der Secure Messaging Manager dieses und gibt es an den Kommandointerpretor weiter, der es nach einer erfolgreichen Decodierung an den Logical Channel Manager weiterreicht. Durch ein Ermitteln des angewählten Kanals und Umschalten auf die entsprechenden Zustände wird im Gutfall der Zustandsautomat aufgerufen. Nach der Prüfung, ob das Kommando überhaupt mit den übergebenen Parametern im aktuellen Zustand ausgeführt werden darf, wird es entsprechend abgearbeitet. Tritt während einer dieser Schritte ein Fehlerzustand auf oder wird ein Antwortcode erzeugt, so übernimmt dies der Returncode Manager und gibt es über die I/O Schnittstelle zurück. Muß während der Programmausführung auf Dateien zugegriffen werden, so kann die Dateiverwaltung nach Prüfung der Zugriffsrechte über den Speichermanager auf die Daten im EEPROM zugreifen.

5.3.3 Chipkartenbetriebssyteme mit nachladbarem Programmcode

Neuere Entwicklungen der Chipkartenbetriebssysteme unterstützen sehr wohl das Nachladen, beziehungsweise Ausführen von Programmen auf der Chipkarte. Er-

ste solche Betriebssystem wurden 1997/98 entwickelt. Ein Hauptaugenmerk dabei bilden Java unterstützende Betriebssysteme, die durch eine durchdachte API (application programming interface) Zugriffe auf die wichtigsten Funktionen des Betriebssystems ermöglichen.

Ein Programmcode kann auf zwei verschiedene Arten nachgeladen werden:

1. als compilierter Code in der Maschinensprache des Zielprozessors. Dies bietet zwar den Vorteil, daß die Programme sehr schnell abarbeitbar sind, jedoch eine Wiederverwendung auf andere Chipkarten ausschliessen.

2. als interpretierbaren Code in einer Sprache wie Java. Dieser Code wird durch einen Interpretor auf der Chipkarte ausgeführt. Dabei ist jedoch besonderes Augenmerk darauf zu richten, daß die Interpretation möglichst schnell abläuft und daß der Interpretor möglichst wenig Speicher verbraucht. Die derzeit meistverbreitestten Lösungen sind die Java Card sowie der C-Interpretor MEL (Multos executeable language). Der eindeutige Vorteil dieser Variante ist die Wiederverwendbarkeit von Programmcode sowie die leichte Programmierbarkeit in weitverbreiteten Programmiersprachen wie Java.

Die Java Card

Ausgangspunkt für die Erfindung der Java Card war eine im Jahr 1996 von Europay veröffentlichte Studie, die eine Open Terminal Architektur (OTA) beschrieb. Hierbei sollte es ermöglicht werden, durch eine einheitliche Softwarearchitektur eine hardwareunabhängige Terminalprogrammierung zu gewährleisten. Grund für diese Idee war es, eine möglichst billige Programmierbarkeit der Kreditkarten-Paymentterminals zu schaffen.

Schon im Herbst 1996 stellte die Firma Schlumberger eine Chipkarte vor, die das Abarbeiten von Javaprogrammen ermöglichte. 1997 kam es dann zur ersten Java Card Konferenz, beziehungsweise zur Gründung des Java Card Forums (JFC), welches die Spezifikationen für die Implementierung der Java Virtual Machine (JVM) übernahm. Die Teilnehmer der ersten Java Card Konferenz waren die großen Chipkartenhersteller, sowie die Firma Sun. Zur Zeit ist die Java Card Spezifikation in Version 2.2 aktuell.

Zusätzlich zur leichten Programmierung durch Java bietet die Java Card den großen Abnehmern von Chipkarten eine einheitliche Umgebung für die Programmausführung, da diese Unternehmen oft bei verschiedenen Herstellern einkaufen und bei

einer Programmierung in einer anderen Sprache die Programme jeweils neu portieren, kompilieren und testen müßten.

Die Java Card beinhaltet in ihrer derzeitigen Spezifikation kein Dateisystem auf welches beliebig zugegriffen werden kann. Nur innerhalb eines Java Applets kann ein Dateisystem aufgebaut und verwaltet werden. Die Java Virtuel Machine wird bei der Kartenfertigung aktiviert und am Ende des Kartenlebenszykluses deaktiviert.

Das Applet wird durch ein SELECT Kommando mit einer eindeutigen Applet ID (AID) ausgewählt und abgearbeitet. Um eine Datenredundanz zu vermeiden können Applets untereinander auf die jeweiligen Dateisysteme der anderen Applets zugreifen. Die einzigen zusätzlichen appletunabhängigen Kommandos dienen dem Laden von Applets in die Chipkarte. Sie werden im EEPROM gespeichert.

5.4 Anwendungsmöglichkeiten

Die Anwendungsmöglichkeiten von Smart Cards sind sehr vielfältig. Hier einige Beispiele:

- elektronische Geldbörse
- digitale Scheckkarten
- Telefonwertkarten
- SIM-Karte
- elektronische Mitgliedsausweise
- Zeiterfassungssystem
- elektronischer Krankenschein
- Loyality - Kundenkarte

5.5 Funktionsweise der elektronischen Geldbörse

[Selhorst, 2002]

Wie bereits erwähnt ist eine sehr verbreitete Anwendung von Chipkarten der Einsatz als elektronische Geldbörse. In Österreich ist die Firma Europay Austria

Zahlungsverkehrssysteme GmbH, zugehörig zur Firma Europay, für das System ”QUICK” verantwortlich. Leider konnte durch eine Anfrage keinerlei Information über die dabei verwendete Technologie in Erfahrung gebracht werden. Die Firma Europay Austria Zahlungsverkehrssysteme GmbH begründete diese Absage mit den Worten ”Fragen dieser Art können wir aus Sicherheitsgründen leider nicht beantworten.”

Meiner Annahme nach ähnelt dieses System den sonstigen in Europa verwendeten Systemen und ich beziehe mich daher auf Informationen des deutschen Systems. Dieses System wird auch in anderen europäischen Ländern wie zum Beispiel Luxemburg und Frankreich verwendet. Die Vermutung, daß das österreichische System dem seiner Nachbarn ähnelt, liegt nahe.

Aufgebaut ist das System auf zwei verschiedene Chipkartensysteme:

1. die Händlerkarte:
   dient dem Händler als Ausweis gegenüber dem Kunden und wird in den POS-Terminal entweder als Chipkarte oder in Form von Software installiert.

2. die Kundenkarte:
   ermöglicht es dem Kunden die Geldkarte mit Geld zu beladen und am POS-Terminal zu entladen.

   Abhängig von der ausgebenden Stelle der Kundenkarte, wird unterschieden in

   - kontogebundene Karten:
     werden von Banken ausgegeben und sind an Girokonten gebunden.

   - und kontoungebundene Karten, auch White Card genannt:
     werden von anderen Institutionen ausgegeben und sind unabhängig von Girokonten.

Der Hauptunterschied zwischen diesen beiden Kartenformen liegt im Aufladen der Geldkarte. Bei kontogebundenen Karten erfolgt der Aufladevorgang durch ein Lastschriftverfahren auf dem jeweiligen Girokonto des Karteninhabers. Bei kontoungebundenen Karten wird der Betrag direkt an dem Aufladeterminal bar einbezahlt.

Eine Mischform dieser beiden Kartentypen ist sehr häufig, das heißt mit einer kontogebundenen Karte kann sowohl ein Aufladevorgang mittels Girokontoabbuchung als auch durch Bareinzahlung durchgeführt werden. Umgekehrt ist dies natürlich nicht möglich, da sonst keine Verbindung zu einem Girokonto besteht.

5.5.1 Chipladevorgang

Der Ladevorgang beginnt grundsätzlich bei einem Ladeterminal in den die Karte eingeführt wird und über eine Onlineverbindung gegen die kartenausstellende Instanz (Ladeinstanz) authentifiziert wird. Wird nun der Betrag direkt bar eingezahlt so wird dieser der Karte gutgeschrieben. Jede Karte ist eindeutig einer sogennanten Karten-Evidenzstelle zugeordnet. Dort wird ein ”Schattenkonto” für jede Karte geführt. Diesem wird der eingezahlte Betrag gutgeschrieben.
Bei einer kontogebundenen Karte kann der Benutzer nach Eingabe eines PIN-Codes den gewünschten Betrag der Karte gutschreiben. Der Unterschied zur ungebundenen Karte liegt in der Verrechnung des Betrages zwischen dem Girokonto und dem ebenfalls vorhandenen ”Schattenkonto”. Im ersten Fall wird also direkt der Barbetrag an das ”Schattenkonto” eingezahlt, im zweiten Fall geschieht dies durch eine Verbuchung zwischen der Bank und der Karten-Evidenzstelle.
Vorteil der kontogebundenen Karte ist der Ladevorgang ohne Barmittel und die meist vorhandene Möglichkeit beide Ladeterminals zu verwenden.

Bei Verlust einer Karte kann der bereits verbuchte Betrag nicht zurückgefordert werden, da bei einem Zahlungsvorgang nicht gegenüber dem ”Schattenkonto” geprüft wird. Einzige Möglichkeit ist das jeweilige Ablaufdatum einer Chipkarte. Ist eine Karte abgelaufen und der Betrag wurde nicht verwendet kann dieser von der Karten-Evidenzstelle refundiert werden.

5.5.2 Zahlungsvorgang

Möchte der Karteninhaber mit einer Chipkarte bezahlen so muß diese in einen Terminal eingeführt werden und der abzubuchende Betrag bestätigt werden. Dieser Betrag wird anschließend der Händlerkarte gutgeschrieben. Der Händler verfügt ebenfalls über ein sogenanntes ”Schattenkonto” bei der Händler-Evidenzstelle. Der Händler reicht seine Karte bei der Kunden-Evidenzstelle ein und der Betrag wird vom ”Schattenkonto” des Kunden auf das ”Schattenkonto” des Händlers übertragen. Über dieses Konto kann der Händler anschließend verfügen.

Der größte Vorteil für den Händler gegenüber einer Bankomatkartenzahlung des Kunden ist die nicht notwendige Onlineverbindung zu einem Kreditinstitut. Die Kundenkarte wird also beim Zahlungsvorgang nicht online verifiziert. Der Händler spart sich daher die hohen Kosten einer Onlineverbindung.

5.5.3 Sind elektronsiche Geldbörsen anonym?

Die oftmals gepriesene Anonymität bei Bezahlvorgängen mittels elektronischen Geldbörsen kann nur darin liegen, daß die gespeicherten Informationen nicht weiterverwendet werden. Bei jedem Zahlungsvorgang werden sowohl beim Händler die Daten der Kundenkarte, als auch beim Kunden die Daten der Händlerkarte sowie die Daten des Händlerterminals gespeichert.

Die Speicherung der Daten dient der Sicherheit, da durch diese ein möglicher Mißbrauch leicht aufgedeckt werden kann. Der Kunde hat ebenfalls einen Nutzen durch die aufgezeichneten Daten, da er die zuletzt getätigten Ausgaben beobachten kann.
Eine Anonymität wie es sie bei der Bezahlung mittles Bargeld gibt kann durch ein solches System aus Sicherheitsgründen nicht gewährleistet werden. Einzig der Umstand, daß die Daten der Zahlungsvorgänge nach einer Verrechnung aufgrund gesetzlicher Vorschriften gelöscht werden müssen, birgt eine gewisse Anonymität. Falsch ist es jedoch den Kunden im Irrglauben zu belassen, daß er genau wie mit Bargeld anonym zahlen kann!

5.5.4 Aufbau des Authentifizierungsprozesses laut Visa Spezifikation (Offline Data Authentication)

Die Kreditkartenfirma Visa hat eine Spezifikation für die Funktionsweise von elektronischen Geldbörsen veröffentlicht, aufbauend auf der ”Europay Mastercard and Visa Spezification” (EMV) [EMV, 2000]. Die EMV-Spezifikation gilt als Grundlage für das Archtekturdesign von Smart Cards.

Wie bereits im Kaptitel 2.2 ”Gliederung von elektronischen Zahlungsformen” erklärt wurde, kann bei Offline-Transaktionen keine unmittelbare Prüfung der Identitäten durch die zahlungsmittelausstellende Institution erfolgen. Es ist daher nötig, die eindeutige Identifikation eines Zahlungsmittels schon im voraus festzulegen. Um dies zu erreichen verwendet man asymetrischen Verschlüsselungsverfahren, digitale Signaturen sowie eine Public Key Infrastructure (siehe Kapitel 3.5 Public Key Infrastructure).

Bei der Kommunikation zwischen dem Terminal am POS und der Smart Card gibt es zwei unterschiedliche Verfahren:

1. SDA - Static Data Authentication:
   Die statische Methode prüft ob die auf der Smart Card gespeicherten Daten seit ihrer Erstellung durch eine authentisierte Institution verändert wurden.

2. DDA - Dynamic Data Authentication:
   Bei einer dynamischen Prüfung, werden die Daten ebenfalls auf ihre Unveränderthеit untersucht. Zusätzlich kann mittels dieses Verfahrens noch überpüft werden, ob diese Karte durch eine verbotene Kopie der Orginalkarte erzeugt wurde.

Die folgenden Ausführungen beziehen sich nur auf die SDA.

Für die Erstellung von privaten, öffentlichen Schlüsseln und Signaturen wird das RSA-Verfahren verwendet (siehe Kaptitel 3.3 Verschlüsselung RSA).

Die gesamte Infrastruktur wird in drei Bereiche aufgeteilt:

1. Visa Certificate Authority (CA)
   Die Aufgaben der Visa CA:
   - Erstellen der Visa RSA Schlüsselpaare
   - Erstellen des Zertifikates für die ausgebende Institution von Smart Cards
   - Verwaltung der Zertifikate und Bereitstellen einer Public Key Infrastructure
2. Issuer - ausgebende Institution von Smart Cards (zum Beispiel Bank)
   Aufgaben:
   - Erstellen der Issuer RSA Schlüsselpaare
   - Weitergabe der öffentlichen Schlüssel an die Visa CA zur Erstellung einer Signatur
3. Smart Card
   Aufgaben sind die Speicherung:
   - des öffentlichen Schlüssels der ausgebenden Institution
   - des von der CA für die ausgebende Institution ausgestellten Zertifikates

Folgende Grafik soll die Vorgänge rund um die Erstellung dieser Sicherheitsstruktur erläutern (siehe Abb. 5.4).

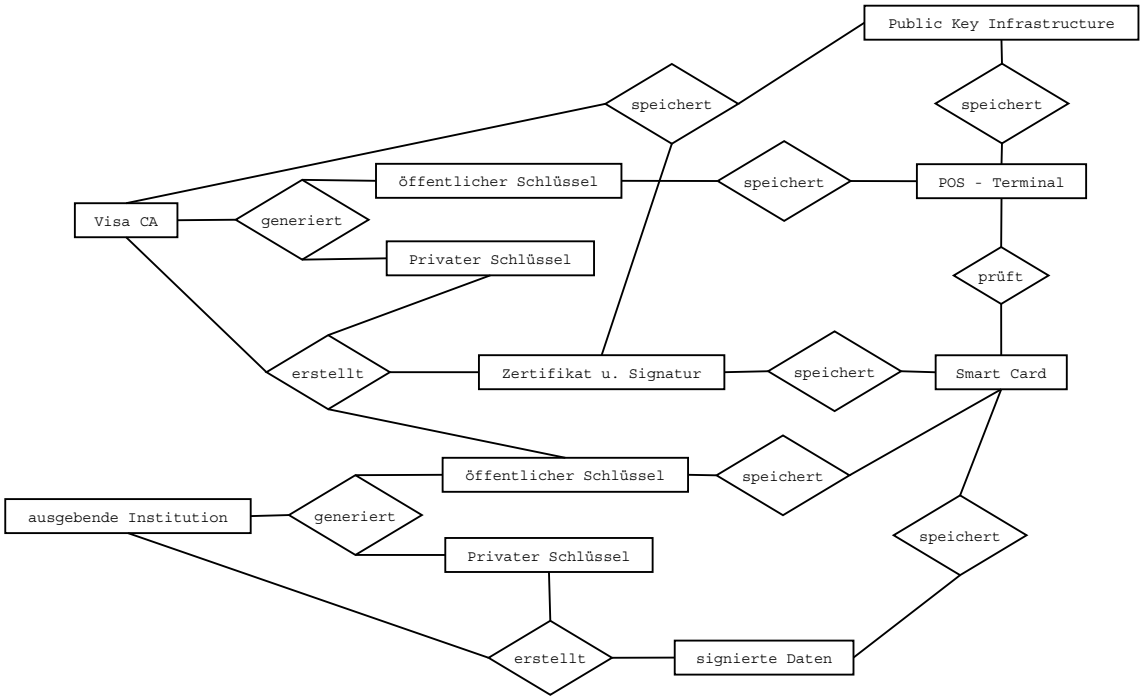

Abbildung 5.4: Static Data Authentication

Die Grafik, als auch die folgende Beschreibung bezieht sich nur auf die Grundfunktionalitäten und geht nicht auf die Spezialitäten der Visa Zertifikate, die erzeugten Hashwerte und so weiter ein.
Bevor eine Static Data Authentication überhaupt durchgeführt werden kann, müssen sowohl Visa als auch die ausgebende Institution ein RSA - Schlüsselpaar erzeugen. Die ausgebende Insitution sendet den erstellten öffentlichen Schlüssel an Visa. Visa generiert mit Hilfe seines privaten Schlüssels ein Zertifikat, welches in der Visa Public Key Infrastructure gespeichert wird. Die öffentlichen Schlüssel von Visa werden in den POS-Terminals gespeichert. Auf der Smart Card werden das erstellte Zertifikat, sowie die Authentifizierungsdaten verschlüsselt mit dem privaten Schlüssel der ausgebenden Institution gespeichert. Bei der Authentifizierung erhält der POS-Terminal das auf der Smart Card gespeichert Zertifikat. Der Terminal kann dieses durch einen Blick in die PKI verifizieren und mit Hilfe des öffentlichen Schlüssels von Visa kann der öffentliche Schlüssel der ausstellenden Institution ermittelt werden. Jetzt kann der Terminal die von der Smart Card erhaltenen Authentifizierungsdaten mit Hilfe des öffentlichen Schlüssels prüfen. Es wird, wie im Kapitel Datenintegrität 3.1 erklärt, der Hashwert der Authentifizerungsdaten in Klartext mit dem verschlüsselt übermittelten Hashwert verglichen. Bei Übereinstimmung ist eine eindeutige Identifizierung der Karte gegeben.

5.5.5 Umsetzung der elektronischen Geldbörse im Prototypen

Für die Implementierung einer mobilen Zahlungslösung wäre es ideal die Verschlüsselung und Speicherung sensibler Daten einem Smart Card Chip zu überlassen. In jedem Mobiltelefon ist ein solcher Chip in Form einer SIM Karte vorhanden. Es sollte daher theoretisch möglich sein, einen Chipkartenprozess, der eine Verschlüsselung oder Authentifizierung ausführt, durch eine API anzustossen.
Damit könnte das Mobiltelefon auch als eine elektronische Geldbörse verwendet werden und es könnte zusätzlich auf bereits bestehende Strukturen aufgebaut werden. Unter bestehende Strukturen ist das bereits ausgereifte System der elektronischen Geldbörse sowie die aufgebaute Infrastruktur mit den einzelnen Terminals zu verstehen.
Dem Mobiltelefon würde also die Rolle des Kommunikationsmediums zwischen Chipkarte und Terminal zukommen. Als Kommunikationsweg wäre zum Beispiel die Infrarotschnittstelle oder auch die Bluetoothtechnologie denkbar.

Für eine Umsetzung dieser Idee innerhalb meines Prototypen müßte es möglich sein durch eine Java 2 ME API auf eine API der SIM Karte zuzugreifen. Das Kapitel Programmiersprachen wird noch genauer auf die Möglichkeiten von Java 2 ME und das für den Prototypen verwendete MIDP eingehen. Schon vorweg kann gesagt werden, daß es keine Unterstützung für das Anstossen eines SIM Karten Prozesses unter Java 2 ME gibt, und es wird daher nicht möglich sein eine sichere und auf bestehenden Strukturen aufbauende Lösung für eine elektronische Geldbörse zu erstellen.
Dies heißt natürlich nicht, daß die Implementierung des Prototypen fehlschlägt, sondern nur, daß sich die Implementierung nicht an bestehenden Strukturen ausrichten kann.

Kapitel 6

Programmiersprachen

In diesem Kapitel wird auf Programmiersprachen eingegangen, die sich für die Entwicklung eines Bezahlungssystems für mobile Endgeräte eigenen. Hauptaugenmerk wird auf die Sprache Java 2 Micro Edition gelegt, da schon zu Beginn der Diplomarbeit diese als Grundlage für die Implementierung des Prototypen festgelegt wurde. Die angeführten Argumente für den Einsatz von Java 2 Micro Edition sollen die bereits getroffene Entscheidung untermauern. Zusätzlich werden im Kapitel 6.1.4 Alternativen zu Java 2 Micro Edition behandelt.

6.1 Java 2 Micro Edition

Java 2 Micro Edition (J2ME oder Java 2 ME) ist eine Entwicklungs- und Laufzeitumgebung, entworfen für die Ausführung von Javaprogrammen auf Endgeräten mit geringen Speicher- und Rechenkapazitäten. Java Programme, die für Personal Computer (PC) geschrieben wurden, können aufgrund ihrer Anforderungen an die Hardware nicht auch auf einem Mobiltelefon ausgeführt werden.

Java ist eine objektorientierte Programmiersprache, die es dem Programmierer erlaubt ein Programm zu erstellen und zu kompilieren, um es später auf einer Vielzahl von verschiedenen Plattformen zu verwenden. Die erstellte Software ist daher unabhängig vom Betriebssystem. Durch diese Möglichkeit, auch ”write once run everywhere” genannt, hat es Java geschafft, innerhalb von nur wenigen Jahren eine Vielzahl an Programmierern zu überzeugen und den Markt zu erobern.

Die starke Durchdringung der Bevölkerung mit mobilen Endgeräten und deren technische Weiterentwicklung stellte die Anforderung an Java auch diese zu unterstützen. Technische Unterschiede, wie zum Beispiel die Größe des Display oder der geringe Speicherplatz verhindern natürlich die Verwendung von Javaprogram-

men, die für einen Desktop - PC geschrieben wurden. Daher wurde J2ME, eine abgespeckte Version der Java-Runtime, spezifiziert. J2ME ist eine speziell auf die Bedürfnisse von mobilen Endgeräten ausgerichtete Programmiersprache. Jeder Java Programmierer kann, unter Beachtung einiger Details, J2ME Software erstellen. Auch muß erwähnt werden, daß ein J2ME-Programm auf jedem mobilen Endgerät, das diesen Standard unterstützt, funktioniert.

Die Vorteile der Programmiersprache Java:

- Sicher:
  Java war von Anfang an für die Verwendung in Netzwerkarchitekturen konzepiert und daher wurde besonders auf die Sicherheit Wert gelegt. Gerade in Netzwerken lauert eine ständige Gefahr von Attacken anderer Systeme.

- Zuverlässig:
  Programmierer müssen sich in Java nicht um Speicherplatzbelegungen und -bereinigungen kümmern. Dies wird ihnen durch die Java Runtime Umgebung verlässlich abgenommen. Anders als zum Beispiel in C++ kann der Softwareentwickler nicht auf die Freigabe von Speicherbereichen vergessen.

- Objektorientiert:
  Als eine objektorientierte Sprache ermöglicht Java die leichte Wiederverwendbarkeit von Programmteilen zu einem späteren Zeitpunkt. Diese Form der Programmierung erleichtert die Abbildung der realen in die virtuelle Welt einer Software. Software kann dadurch auch für Außenstehende leichter verständlich gemacht werden; zum Beispiel, daß ein Konto aus den Attributen Kontonummer und Kontobenutzername besteht. Die Zuverlässigkeit zusammen mit der Objektorientierung ermöglicht eine schnellere und einfachere Softwareentwicklung. Vor allem für Entwickler, die bereits Erfahrung mit anderen objektorientierten Programmiersprachen haben, ist ein Umstieg sehr schnell realisierbar.

- Frei erhältlich:
  Für die Verbreitung von Java war es wichtig, daß die grundlegende Spezifikation frei erhältlich ist, und somit eine Implementierung auf allen Plattformen ermöglicht wird. Es gibt natürlich auch eine komerzielle Sparte von Java, die alle zusätzlichen Bereiche, wie Entwicklungsumgebungen, spezielle Application Provided Interfaces (API) und soweiter, betrifft.

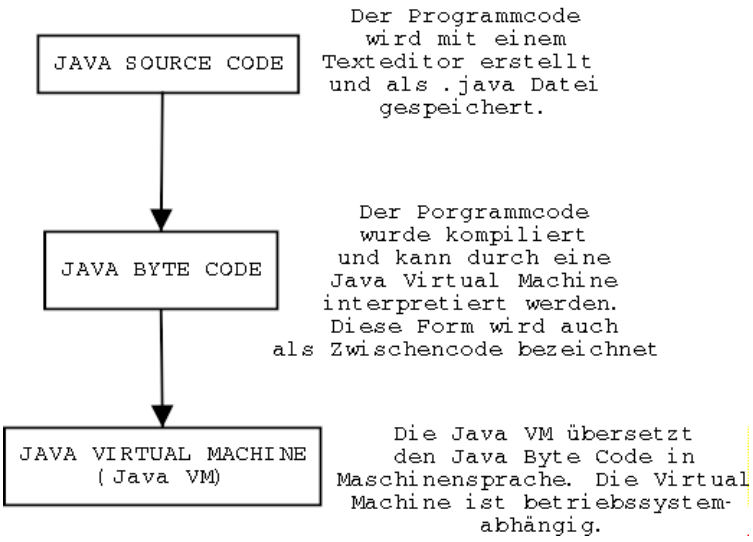

Abbildung 6.1: Java Runtime

6.1.1 Welche Endgeräte können J2ME unterstützen

Die Produktpalette für die Java 2 Micro Edition Spezifikation erstellt wurde, beginnt bei Smart Cards und endet erst bei der Java Spezifikation für Personal Computer. Die Abbildung 6.2 zeigt die Spannweite der J2ME Spezifikation und die dabei zum Einsatz kommenden Konfigurationen CLDC und CDC (siehe Kapitel 6.1.3). Die Produktpalette umfasst somit:

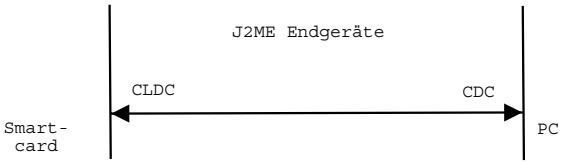

Abbildung 6.2: Breite der Endgeräte

1. drahtlose Endgeräte:
   kommunizieren durch eine Funkverbindung mit ihrer Umgebung.

2. mobile Endgeräte:
   können, müssen aber nicht drahtlose Endgeräte sein. Ein Personal Digital Assistant (PDA) ist zwar ein mobiles Endgerät, kommuniziert allerdings nicht unbedingt mit anderen Geräten über eine drahtlose Funkverbindung. Ausnahmen sind natürlich möglich. Mobile Endgeräte bieten die

Möglichkeit sie überall hin mitzunehmen und zu verwenden. Drahtlose Geräte sind jedoch oft an ihre Umgebung gebunden; zum Beispiel eine Funkmaus ist an die Reichweite des Funkempfängers und damit an die Umgebung des Computers gebunden - die Mitnahme einer Maus und Verwendung außerhalb der Reichweite des Empfängers macht diese nutzlos.
Das Beispiel eines mobilen und drahtlosen Endgerätes wäre ein Mobiltelefon, welches einen MP3 Player integriert hat. Es kann sowohl über die Funkverbindung zum Telefonieren, als auch zum Abspielen von Musik außerhalb des Mobilfunknetzes verwendet werden.

Java 2 Micro Edition beschränkt sich jedoch nicht nur auf drahtlose und mobile Endgeräte sondern unterstützt auch die Programmierung einer Vielzahl anderer Geräte. Nachfolgende Aufzählung soll dies verdeutlichen:

- Fernsehgeräte
- Navigationssystem
- Auto - Entertainmentsysteme
- Smart Cards
- Pager
- Personal Digital Assistants
- Mobiltelefone
- POS-Systeme
- Bildtelefone

6.1.2 Die J2ME Architektur

Ziele der J2ME Architektur können wie folgt charakterisiert werden
[White and Hemphill, 2002]:

- Produktvielfalt:
  Die vorangegangene Aufzählung der J2ME Produktpalette zeigt sehr deutlich, daß eine große Anzahl an verschiedensten Endgeräten durch J2ME unterstützt werden. Jedes dieser Geräte hat spezielle Anforderungen, wie zum Beispiel Datenspeicherung, Netzwerkverkehr, Stromsparfunktionen, Sicherheit, verfügbarer Speicher, Eingabegeräte und so weiter. Bezogen auf die Speicherkapazitäten reicht die Produktpalette von cirka 160 Kilobyte bei

Mobiltelefonen bis zu TV-Geräten mit einer Speicherkapazität ähnlich einem Personal Computer. Einige wichtige Java Klassen müssen daher auf jedem dieser Geräte verfügbar sein; dazu gehören zum Beispiel Klassen der Packages java.lang, java.util und java.io. Diese wichtigsten Klassen sind jedoch abgespeckte Versionen gegenüber Java 2 Standard Edition und Java 2 Enterprise Edition.

- Optimierbarkeit:
  Die Architektur soll für jede auch noch so ”kleine” Umgebung optimierbar sein.

- Personalisierbarkeit:
  Abstimmung auf die persönlichen Bedürfnisse bei Geräten, die nur von einer Person genutzt werden.

- Netzwerkanbindung:
  Viele der Endgeräte haben eine Netzwerkanbindung. Diese Anbindungen können verschiedenster Art sein: zum Beispiel mit kleiner und großer Bandbreite, oder drahtlos und so weiter.

- Einbindung von Applikationen:
  Nicht jedes dieser Geräte verfügt über die gleichen Möglichkeiten neue Programme zu installieren. Zum Beispiel macht es für ein Mobiltelefon Sinn, eine neue Applikation direkt über eine Internetverbindung zu installieren. Bei einem Personal Digital Assistant jedoch, ist es sinnvoller diese Applikationen über einen PC-Anschluß zu überspielen.

- Plattformkonformität:
  Es sollen möglichst alle Endgeräte einer Kategorie (zum Beispiel Mobiltelefone) unterstützt werden und zusätzlich auf die Vorteile der einzelnen Geräte eingegangen werden.

- Flexibilität:
  Auf sich verändernde Marktverhältnisse kann durch leichte Erweiterbarkeit rasch reagiert werden.

- Unabhängigkeit:
  Auch Drittanbieter (gegenüber dem Hersteller) können Programme für Geräte erstellen und vertreiben. Die Abhängigkeit vom Geräteerzeuger ist daher nicht mehr gegeben.

- Portierbarkeit:
  Programme können durch kleine Veränderungen sehr schnell an neue Umgebungen angepasst werden.

6.1.3 J2ME Konfigurationen und Profile

Innerhalb der Java 2 Micro Edition Spezifikation gibt es zwei architektonische Konzepte auf Basis von:

1. Konfigurationen:
   sollen die Portierbarkeit zwischen verschiedenen Geräten erleichtern. Es werden zum Beispiel die unterstützten Hauptklassen, wie String, Stream, Thread,... , definiert.

2. Profile:
   konzentrieren sich auf gerätespezifische Merkmale, wie zum Beispiel die Unterstützung von Infrarotschnittstellen oder Spiele-APIs.

Hauptunterschied zwischen Konfigurationen und Profile ist, daß Profile voneinander unabhängig sein können, Konfigurationen allerdings nicht. Jede Konfiguration muß im Definitionsbereich ihrer Mutterkonfiguration stehen. Dies soll die bereits erwähnte Grundidee von Java garantieren ”write once run everywhere”, und damit die Hersteller zur Einhaltung dieser Standards zwingen.
Jedes Programm, daß für eine mit weniger Funktionalität spezifizierte Konfiguration geschrieben wurde, muß auch auf allen umfangreicher spezifizierten Konfigurationen funktionieren.

Bereits in der Abbildung 6.2 wurden die zwei bisher definierten Konfigurationen gezeigt:

- CLDC (Connected Limited Device Configuration):
  definiert die Anforderungen für Endgeräte mit geringen Speicher- und Rechenleistungen. Der Speicherbereich ist auf 160 kB bis 512 kB begrenzt. Nicht implementierte Funktionalitäten sind zum Beispiel die Unterstützung von Java Native Interfaces oder Ojekte vom Typ Float.

- CDC (Connected Device Configuration:)
  richtet sich an Geräte mit einer höheren Speicher- und Rechenleistung.

Profile wiederum basieren auf den definierten Konfigurationen und spezifizieren welche APIs für die Unterstützung der jeweilig benötigten Funktionalitäten implentiert werden müssen - zum Beispiel muß ein Profil für die Programmierung von Spielen eine Möglichkeit zur Darstellung von Grafiken haben.

Beispielprofile sind:

- Mobile Information Device Profile (MIDP):
  eine ”Virtuell Java Machine”, die dieses Profil implementiert, bietet viele

Funktionalitäten, die für die Entwicklung von Mobiltelefonapplikationen wichtig sind. Beispiel hierfür ist die Möglichkeit, eine HTTP Verbindung zu öffnen oder die Darstellung von Eingabefeldern. Dieses Profil basiert auf der CLDC Konfiguration.

- PDA Profile (PDAP)
  ist abgestimmt auf die besonderen Bedürfnisse von Personal Digital Assistants, wie zum Beispiel die Nutzung des relativ großen Displays; basiert auf der CLDC Konfiguration.

- RMI Profile
  ein auf der CDC Konfiguration basierendes Profil, das die Implementierung von ”Remote Method Invocation” definiert.

- Gaming Profile
  unterstützt die Darstellung von 2D und 3D Grafiken und basiert sowohl auf der CDC, als auch auf der CLDC Konfiguration. Zahlreiche Hersteller von Mobiltelefonen haben dieses Profil in ihrer Java Virtuell Machine implementiert.

Die Spezifikation der Konfigurationen und Profile wird durch die Java Process Community geleitet ([(JCP), 2004]).

Mobiltelefone haben prinzipiell das Mobile Information Device Profile implementiert. Die derzeit meist verwendete Spezifikation ist die Version 1.x des MIDP. Diese enthält jedoch keinerlei für den Prototypen unbedingt notwendige Unterstützung zur Verschlüsselung von Daten. Um diese Funktinalitäten trotzdem implementieren zu können, muß auf zusätzliche Klassenbibliotheken zurückgegriffen werden (siehe Kapitel 6.1.5).

Das für den Prototypen zum Einsatz kommende MID Porfil in der Version 1.03 verfügt über folgende Standard API:

- java.io:
  Deutlich, im Vergleich zur Java 2 Standard Edition, abgespeckte Version des io packages. Es fehlen zum Beispiel die Filehandling Klassen.

- java.lang:
  Auch hier fehlen wichtige Klassen, wie zum Beispiel Float.

- java.util:
  Ebenfalls wurde auf die Implementierung von substanziellen Klassen, wie zum Beispiel List verzichtet.

- javax.microedition.io:
  Zusatzklassen für den Zugriff auf Netzwerkfunktionalitäten.

- javax.microedition.lcdui:
  Zusatzklassen für die Ausgabe am Display.

- javax.microedition.midlet:
  Dieses Package beinhaltet die Klasse MIDlet, welche für die Ausführung und Zustandsbeschreibung eines Java 2 ME Programms zuständig ist. Eine genauere Beschreibung hierfür folgt im zweiten Teil der Diplomarbeit.

- javax.microedition.rms:
  Im rms Package befindet sich die Bibliothek zur Speicherung von Daten. Jedes Java 2 ME Programm hat seinen eigenen Speicher und es kann auch nur auf diesen zugreifen.

6.1.4 Alternativen zu Java 2 ME

Wie schon bei der Einführung in das Kapitel Java 2 Micro Edition erwähnt, bietet Java den Vorteil, daß der einmal erstellte Programmcode auf einer Vielzahl anderer Systemumgebungen ausgeführt werden kann.
Für die Programmierung von Endgeräten mit den erwähnten speziellen Anforderungen gibt es natürlich auch die Möglichkeit andere Programmiersprachen einzusetzen. Der erstellte Programmcode muß jedoch für eine Ausführung auf jedem System neu kompiliert werden. Dies verhindert eine rasche Verbreitung von Software auf einer Vielzahl unterschiedlicher Geräte. Ebenfalls bieten andere Programmiersprachen nicht immer die Vorteile von Java, wie Sicherheit, Zuverlässigkeit und Objektorientierung.
Ein sehr gut verständliches Bespiel für die genannten Vorteile sind für Mobiltelefone programmierte J2ME - Spiele: durch eine große Gemeinschaft an Softwareentwicklern besteht bereits eine Vielzahl an wiederverwendbaren, objektorientierten Code, für die leichte und schnelle Umsetzung von Ideen. Auch die Anzahl der Mobiltelefone die eine J2ME - Runtime mitliefern steigt stetig. Möchte nun ein engagierter Entwickler sein Spiel entgeltlich oder frei verbreiten, so ist dies durch das Anbieten im Internet sehr leicht möglich. Der Konsument dieses Spieles kann wiederum auf die sichere und beschränkte Ausführumgebung der Runtime vertrauen und sicher sein, daß keine gefährlichen Informationen über sein Mobiltelefon verbreitet werden. Es verschwinden somit die Grenzen zwischen den einzelnen Herstellerfirmen wie Nokia, Siemens, Ericson,...
Würde der Entwickler statt dessen auf eine Implementierung in C++ setzen, so hätte er darauf zu achten, daß der Code auch auf einer möglichst großen Anzahl

von Endgeräten funktioniert. Weiters könnte er nur bedingt auf wiederverwendbaren Programmcode zurückgreifen und müßte den Programmcode zusätzlich für jedes einzelne Gerät neu kompilieren. Das heißt nicht nur für jede Firma, sondern auch für jede einzelne Type einer Firma, soweit die einzelnen Geräte nicht auf dem gleichen Betriebssystem basieren.

Auch Java 2 Micro Edition kann nicht als die berühmte ”eierlegende Wollmilchsau” bezeichnet werden, da es gerade im Bereich der Performance oft an ihre Grenzen stößt und es daher manchmal notwendig ist, Software in einer ”schnelleren” Programmiersprache zu implementieren.

Innerhalb der Java-Programmiersprache gibt es auch für die erwähnten Endgeräte Alternativen zu J2ME . Es handelt sich dabei meist um spezielle umfangreichere Implementierungen, als die der J2ME Spezifikation: [White and Hemphill, 2002]

- Chai VM von Hewlett-Packard
- VisualAge Micro Edition von IBM
- Waba von Wabasoft

Außerhalb der Javawelt gibt es noch folgende Alternativen:

- WAP/WML:
  beschränkt sich auf die Darstellung von Inhalten auf Display, ähnlich der Kombination von HTTP und HTML. Für die Anzeige von dynamischen Inhalten wird WMLScript verwendet, welches an die Programmiersprache Javascirpt angelehnt ist. Eine Speicherung von Daten ist jedoch nicht möglich.
- C/C++
  bietet vor allem Performanzvorteile gegenüber Java. Nachteil ist die kompliziertere Programmierung, sowie Lauffähigkeit auf anderen Systemen.
- .net Programmiersprachen
  beschränkt sich auf die Betriebssysteme von Microsoft, zum Beispiel: Windows CE, PocketPC oder PhoneEdition

6.1.5 Klassenbibliotheken

Die Java 2 ME Spezifikation bietet zwar für die Programmierung von Anwedungen eine Vielzahl an Klassenbibliotheken, es wurde jedoch durch die Rücksichtnahme auf die beschränkte Speicherkapatzität der Endgeräte auf einige Standardimplementierungen, die in der Java 2 Standard Edition enthalten sind, verzichtet.

Es ist daher notwendig Klassenbibliotheken von anderen Organisationen zu verwenden. Für die Implementierung des Prototypen habe ich mich dazu enschlossen ausschließlich Open Source Zusatzbibliotheken zu verwenden.

Nano XML Lite

Eine dieser Open Source Bilbliotheken ist Nano XML. Nano XML ermöglicht eine leichte und zuverlässlige Verarbeitung von XML (Xtensible Markup Language) Daten. Je nach Bedarf kann aus unterschiedlichen Paketen ausgewählt werden:

1. Nano XML Java:
   Eine W3C (World Wide Web Consortium) konforme Implementierung eines SAX (Simple API for XML) XML-Parsers mit zusätzlichen Möglichkeiten zur Validierung der XML Struktur durch Prüfung der DTD (Document Type Definition) und einer Klasse zur Erstellung von XML Dateien. Die Binary-Version dieses Paketes hat eine Größe von ca. 33 Kilobyte.

2. Nano XML Lite:
   Eine abgespeckte Version von Nano XML Java, die nur den SAX XML-Parser beinhaltet und dadurch auf eine Größe von ca. 6 Kilobyte verringert wurde.

Für die Implementierung des Prototypen wird voraussichtlich die kleinere Version Nano XML Lite verwendet, da sie sich durch die geringe Größe ideal für die Implementierung innerhalb einer mobilen Endgeräteumgebung eignet.

Folgendes Beispiel soll die leichte Einsetzbarkeit der Nano XML API zeigen: Es wird ein XML Dokument geparst und anschließend die Daten zusammen mit dem XML-Tag Namen ausgegeben. Dieses Programm ist allerdings nicht für Java 2 ME geschrieben, sondern für die Ausführung auf einem PC:

```
import nanoxml.XMLElement;
import nanoxml.XMLParseException;
import java.util.Enumeration;

/**
 * Eine Beispielanwendung unter Nutzung der Nano XML API
 * author Klaus Brosche
 */

                                                                    10
public class TestNanoXml
{
```

```
/**
 * Definition eines XML Dokuments. Dieses kÃ¶nnte auch als
 * Datei direkt von einem Reader verwendet werden.
 */
private static String exampleXmlDoc =
      "<?xml version=\"1.0\"?>"
      +"<cmsdkConfigData>"
        +"<hostName>kbrosche.olymp.ontec.at</hostName>"
        +"<dbPort>1521</dbPort>"
        +"<dbServiceName>iasdb.olymp.ontec.at</dbServiceName>"
        +"<dbSchema>IFSSYS</dbSchema>"
        +"<dbSchemaPw>VrobWGy+NA4h7CFPME8JfQ==</dbSchemaPw>"
        +"<systemUserPw>psfcrVl94ZCtM35O8dutew==</systemUserPw>"
      +"</cmsdkConfigData>";



public TestNanoXml()
{
}

public static void main(String[] args)
{
   TestNanoXml testNanoXml = new TestNanoXml();
   testNanoXml.doParse();
}

/**
 * Methode zur Ausfuehrung des Parsevorganges und Ausgabe
 * am Bildschirm
 */

public void doParse()
{
   try
   {
      //Konstruktor eines Nano XML Elements
      XMLElement xml = new XMLElement();

      //parsen des uebergebenen XML Dokuments
      xml.parseString(exampleXmlDoc);

      //Schleife ueber die einzelnen Elemente des XML Dokuments
      for (Enumeration e = xml.enumerateChildren();       e.hasMoreElements() ;)
      {
        //jedes Element wird zu einem Objekt der Klasse nanoxml.XMLElement
        XMLElement child = (XMLElement) e.nextElement();
        //Ausgabe der einelnen Elemente
        System.out.println(child.getName() + " hat den Wert: "
```

```
                                + child.getContent());
        }
      }
      //Fehlerbehandlung
      catch (XMLParseException xmlExcep)
      {
        System.out.println(xmlExcep.getMessage() + " in Zeile "
                           + xmlExcep.getLineNr());
      }
    }
  }
}
```

Bouncy Castle

[BouncyCastle, 2002]
Bereits im Kapitel 5.5.5 wurde auf die fehlende Möglichkeit, sensible Daten unter Java 2 ME zu verschlüsseln, hingewiesen. Da es aus Sicherheitsgründen auch keine Möglichkeit gibt direkt auf die crypthographischen Fähigkeiten der SIM Karte eines Mobiltelefons zuzugreifen, müssen diese durch eine Drittanbieter-API ersetzt werden.

Eine Java 2 ME kompatible Open Source Bibliothek zur Verschlüsselung von Daten bietet Bouncy Castle (BC). Nachteil der BC-API ist die äußerst dürftige Dokumentation. Es mußte daher relativ viel Zeit in die Programmierung erster Prototypen zum Testen von verschiedenen Algorithmen investiert werden. Der Entschluß, trotz der mangelden Dokumentation die BC-API zu verwenden, liegt eindeutig in ihrer freien Verfügbarkeit, sowie in ihren umfangreichen Möglichkeiten. Darunter versteht man die zahlreichen Algorithmen, die durch BC unterstützt werden.

Wie im Kapitel 3 (Verschlüsselung) erklärt wurde, handelt es sich bei crypthographischen Funktionen um sehr komplexe Konstrukte, daher hat die Implementierung einer solchen API die Größe von ca 250 Kilobyte. Für ein mobiles Endgerät kann diese Größe zu sehr langen Programmwartezeiten oder sogar zu einer Nichtausführbarkeit des Programmcodes führen. Es stellt sich daher wieder die Frage, ob die Verschlüsselungsfunktionen überhaupt durch eine Verwendung von Java 2 ME gelöst werden sollen, oder ob nicht doch eher eine Verlagerung dieser Funktionen in die Hardware, genauer in die Smart Card, sinnvoller wäre. Probleme bei der Verwendung einer Softwareimplementierung von Verschlüsselungsfunktion entstehen nicht nur durch den enormen Anstieg der Programmgröße und ihren Folgen, sondern auch durch Sicherheitsrisiken. Mit Sicherheitsrisiken ist gemeint, daß die unter dem Kapitel 3 erwähnten Private Keys nicht sicher auf dem mobilen Endgerät gespeichert werden können.

Die einzige Möglichkeit diese Daten sicher zu speicher ist es, sie ebenfalls zu verkodieren.
Ein mögliches Szenario könnte eine symetrische Verschlüsselung der asymetrischen Schlüsselpaare unter Einsatz eines PIN Codes sein. Der PIN Code könnte bei jedem Zugriff auf die Schlüsselpaare abgefragt werden. Aus dieser Abfrage kann ein symetrischer Schlüssel erstellt werden. Dies würde allerdings wiederum zu einem Performanzverlust führen.

Warum können die Verschlüsselungsfunktionalitäten also nicht doch einfach auf eine Chipkarte verlagert werden?

1. keine J2ME Unterstützung:
   In der Spezifikation von Java 2 Micro Edition ist bisher keine Implementierung zur Kommunikation zwischen Chipkarten und der J2ME vorhanden. Die Java Community Process - Institution, verantwortlich für die J2ME Spezifikation, hat im Oktober 2003 eine Paper [Process, 2003] veröffentlicht, daß eine solche Implementierung enthält. Wann jedoch eine konkrete Umsetzung dieser Konzepte folgt, ist fraglich.

2. Sicherheitsrisiko:
   Java 2 ME Programme laufen, ähnlich den Java Applets, in einer sehr beschränkten Laufzeitumgebung. Diese Beschränkung wird auch als Sandkastenprinzip bezeichnet. Unter J2ME heißen diese ausführbaren Programme Midlets. Sie können per Internet auf ein mobiles Endgerät geladen und ausgeführt werden. Der Benutzer hat nur beschränkt auf sein Wissen, welches Programm er geladen hat, Einfluß auf das ausgeführte Programm. Wäre es nun möglich unter J2ME auf sensible Daten, wie zum Beispiel Private und Öffentliche Schlüssel zuzugreifen, bestände die Gefahr, daß der Benutzer sich unabsichtlich ein Programm aus dem Internet lädt, daß diese Daten mißbraucht. Ein unbeschränkter Zugriff auf die Verschlüsselungsfunktionalitäten der Chipkarten ist daher gar nicht wünschenswert. Genau wie bei Java Applets, verfügen auch Midlets ab der Version 2.0 des MIDP
   [Process, 2002] über die Möglichkeit, diese als signed Midlets zur Verfügung zu stellen. Dafür wird ein X.509 Zertifikat (siehe Kapitel 3.5.1) auf dem Endgerät gespeichert. Wird nun ein Midlet installiert, so wird geprüft ob es signiert ist oder nicht. Bei signierten Applikationen wird das Zertifikat auf seine Richtigkeit geprüft. Nach positiver Verifizierung kann das Programm auf private Daten des Nutzers, wie zum Beispiel SMS Nachrichten, zugreifen. Bei Ablehnung oder nicht Vorhandensein eines Zertifikates können nur die Standard APIs vom Programm verwendet werden. Welche APIs nur von signierten Anwendungen verwendet werden dürfen wird im sogenannten

”Policy File” festgelegt.

Hier würde auch die Nutzung einer Chipkarten- Verschlüsselungsfunktionalität anschließen und nur nach erfolgter positiver Verifizierung diese API für die Applikation freigeben.

Teil II

Praktische Umsetzung

Kapitel 7

Programmierung unter Java 2 Micro Edition

Jedes Programm das unter J2ME lauffähig sein soll, muß als eine erweiterte Klasse von javax.microedition.midlet.MIDlet implementiert werden. Dieses Konzept entspricht in etwa dem, eines Java Applets oder eines Java Servlets. Die Applikation selbst ist daher eine Subklasse der abstrakten Klasse MIDlet. Um ein funktionsfähiges Programm zu schreiben, müssen Methoden zum Starten, Stoppen und Pausieren der Applikation implementiert werden.
Ein Beispielprogramm, daß den Text ”Hello World!” am Display eines Mobiltelefons ausgibt:

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet
{
  private TextBox textbox;

  public HelloWorld()
  {
    textbox = new TextBox("", "Hello World!", 20, 0);                    10
  }

  public void startApp()
  {
    Display.getDisplay(this).setCurrent(textbox);
  }

  public void pauseApp()
  {
  }                                                                      20
```

```
    public void destroyApp(boolean unconditional)
    {
    }
}
```

Funktionen der einzelnen Methoden des Hello World Programms:

- HelloWorld:
  In diesem Beispiel erstellt der Konstruktor ”HelloWorld()” ein Objekt der Klasse TextBox. Dieses enthält den Text ”Hello World!”.

- startApp:
  Der Start einer J2ME Applikation wird durch die Methode startApp initialisiert. In unserem Beispiel wird dem Objekt Display das durch den Konstruktor erstellte Objekt der Klasse TextBox zugewiesen. Auf dem Display des Mobiltelefons wird nun ”Hello World!” ausgegeben.

- pauseApp:
  Eine Applikation kann dem Benuzter die Möglichkeit bieten, diese zu pausieren. Hierzu müßte ausprogrammiert werden, was in der Pause geschehen soll. Sinn macht dies zum Beispiel bei einem Spiel. In unserem Fall wird diese Methode nicht benötigt.

- destroyApp:
  Beim Beenden des Programmes wird diese Methode aufgerufen. Es kann ein boolean Wert übergeben werden, der Einfluß auf das Programmende hat. Ist der Wert ”True” so hat die Applikation keine Möglichkeit das Ende zu verzögern. Wird ein ”False” übergeben so kann das Beenden der Applikation verhindert werden und zum Beispiel eine neue Meldung am Display ausgegeben werden. Sinn der Methode ist es die Applikation ”sauber” beenden zu können und die bestehenden Systemresourcen freizugeben (zum Beispiel alle bestehenden Netzwerkverbindungen zu trennen).

Damit das erstellte Programm auf einem Mobiltelefon ausgeführt werden kann muß es zuerst kompiliert und anschließend ”preferified” werden. Der Kompiliervorgang übersetzt das Programm in einen Zwischencode der bei der Ausführung durch die Java Virtual Machine interpretiert wird. Aus Sicherheitsgründen, damit eine Java Klasse nur auf die erlaubten Speicherbereiche zugreift, muß das Programm noch den ”Preverifying” Prozess durchlaufen. Ist dieser erfolgreich so ist die Applikation grundsätzlich lauffähig.
Das in einem Texteditor, mit dem Namen HelloWorld.java, gespeicherte Programm wird mit folgendem Befehl kompiliert:

$ javac -bootclasspath $MIDPClasses HelloWorld.java

Der Klassenpfad für die Variable $MIDPClasses muß zuvor mit dem Befehl

$ export MIDPClasses=/midp.jar

gesetzt werden und zeigt auf die J2ME Library. Das Ergbnis ist die Datei HelloWorld.class.

Das Preverifiying wird mit dem Befehl

$ preferivy -classpath $MIDPClasses -d preverified HelloWorld

ausgeführt. Die Option ”-d” teilt dem preferivy Programm mit, daß das Ergebnis des Prozesses in den Dateiordner ”preverified” geschrieben werden soll.

Die erstellte, kompilierte und überprüfte Applikation muß nun auf das mobile Endgerät ”deployed” werden. Dazu wird noch zusätzlich eine Beschreibungsdatei benötigt. Eine solche Datei beinhaltet diverse Informationen rund um die Applikation und wird unter der Dateiendung ”*.jad” gespeichert.

Beispielinhalt der Datei HelloWorld.jad:

MIDlet-1: HelloWorld, , HelloWorld
MIDlet-Jar-Size: 906
MIDlet-Jar-URL: http://localhost/HelloWorld.jar
MIDlet-Name: HelloWorld
MIDlet-Vendor: Klaus Brosche
MIDlet-Version: 1.0

Gemeinsam mit dieser Beschreibungsdatei wird die kompilierte Klasse in eine jar Datei zusammengefasst und auf dem mobilen Endgerät gespeichert. Der Speichervorgang kann entweder direkt über eine PC-Verbindung oder durch ein Laden aus dem Internet erfolgen. Anschließend kann das Programm ausgeführt werden. Der Befehl zum erzeugen einer jar Datei lautet:

$ jar -cf HelloWorld.jar HelloWorld.class HelloWorld.jad

Abbildung 7.1: Screenshot des HelloWorld Programmes

Kapitel 8

Analyse

Durch die Klärung der Anforderungen an eine mobile Bezahlungslösung unter J2ME sowie die technischen Möglichkeiten zur Umsetzung eines solchen Vorhabens im ersten Teil meiner Diplomarbeit, kann in diesem Kapitel die Funktionsweise des Prototypen analysiert werden.

8.1 Kommunikationsfluss

Das Kapitel 4.2 hat die für die Implementierung relevanten und von mobilen Endgeräten teilweise unterstützten Kommunikationsstandards bereits genauer beschrieben. Aufbauend auf die beschriebenen Übertragungsmitteln, zeigt die Abbildung 8.1 wie der Kommunikationsfluss des Prototypen aussehen soll.

Der hier gezeigte Kommunikationsfluss beschäftigt sich mit der Abwicklung einer Zahlungstransaktion in der realen Welt. Das heißt ein Kunde kauft in einem Lebensmittelgeschäft ein und möchte an der Kassa mit seinem mobilen Endgerät bezahlen. Zwischen dem mobilen Endgerät und dem Point of Sale besteht eine räumliche Nähe. Daher kann zwischen diesen beiden Systemen durch eine Funkverbindung, wie Bluetooth oder NFC, oder durch eine optische Verbindung, wie Irda, kommuniziert werden. Die Kommunikationsarten NFC und Irda bieten den Vorteil, daß sie nur eine sehr beschränkte Reichweite haben und dadurch eine gewisse ”Abhörsicherheit” bieten. Da es für NFC bisher keine Implementierung in ein mobiles Endgerät gibt, konzentriert sich der Prototyp auf die Kommunikation per Irda. Leider ist auch hier die Anzahl der Geräte die eine Irda-API für J2ME anbieten sehr beschränkt.

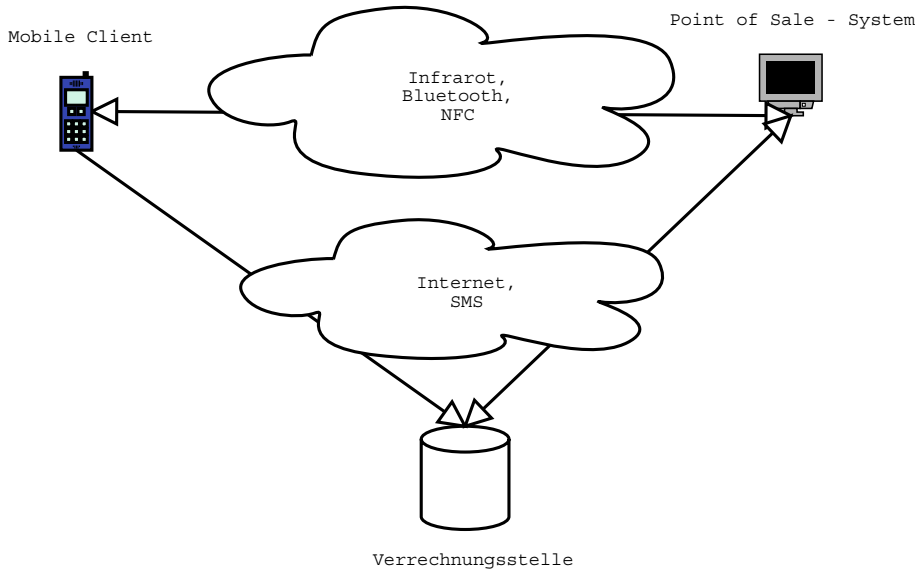

Abbildung 8.1: Kommunikationsfluss

Soll nun eine Zahlung getätigt werden, sendet das POS System einen Kassabeleg per Irda an das mobile Endgerät.

Wird der übermittelte Kassabeleg durch den Benutzer des mobilen Endgerätes für gültig befunden, so muß der durch den Benutzer signierte Beleg an die Verrechnungsstelle zur Gültigkeitsprüfung gesandt werden. Hierfür kann die Kommunikation entweder per SMS direkt an die Verrechnungsstelle, oder per Irda an das POS-System und von dort weiter per Internet an die Verrechnungsstelle erfolgen. Die Implementierung wird nur die Kommunikation über das Internet umfassen.

Nach einer Prüfung wird die Transaktion durch die Verrechnungsstelle gegenüber dem POS akzeptiert oder abgelehnt. Dies kann ebenfalls per SMS oder per Internet erfolgen.

Alle Nachrichten werden stets im XML-Format ausgetauscht.

Findet eine Zahlungstransaktion nicht in der ”realen Welt”, sondern in der ”virtuellen Welt” statt, das heißt ein Einkauf im Internet, so muß die Kommunikation zwischen dem POS und dem mobilen Endgerät per SMS erfolgen. Ebenfalls könnte eine Internetverbindung zwischen dem POS und dem mobilen Endgerät aufgebaut werden, allerdings hat dies zum Nachteil, daß der User hierfür den Rechnernamen des POS Systems eingeben müsste. Eine SMS hingegen kann von

einer J2ME-Applikation abgearbeitet werden.
Die Realisierung der Abwicklung von Zahlungstransaktionen in der ”virtuellen Welt” wird nicht in den Umfang des Prototypen aufgenommen.

8.2 Anforderungen an die Applikation

Nachdem im Kapitel 8.1 geklärt wurde, wie die Kommunikation zwischen den beteiligten Systemen (mobiles Endgerät, POS System, Verrechnungsstelle) stattfinden soll, beschäftigt sich dieses Kapitel mit der Beschreibung der Funktionalitäten des Prototypen.

Die Abbildung 8.2 zeigt die drei Parteien, die bei der Abrechnung einer Zahlungstransaktion interagieren müssen.

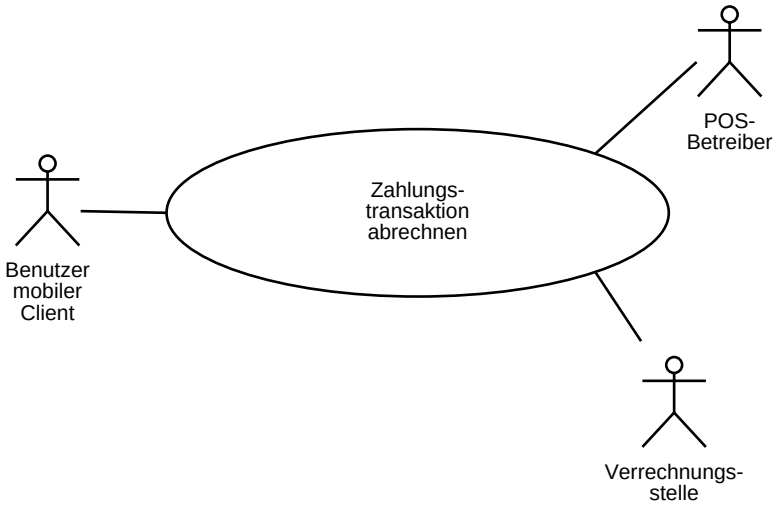

Abbildung 8.2: Abrechnung einer Zahlungstransaktion

Jedes System muß hierfür unterschiedliche Funktionalitäten aufweisen:

1. POS System:

   - Signieren des Kassabeleges
     Um eine eindeutige Authentifizierung des POS Systems gegenüber dem mobilen Endgerät gewährleisten zu können, wird der Kassabeleg digital signiert (siehe Kapitel 3.4).
   - Versenden und Empfangen von XML-Nachricht
     Für die Kommunikation zwischen den Systemen müssen sowohl Daten per Internet als auch per Irda übertragen werden.

2. Verrechnungsstelle:

   - Erstellen von Schlüsselpaaren für die Erzeugung der digitalen Signaturen
     Zum Signieren der Nachrichten müssen das POS System und das mobile Endgerät über private und öffentliche Schlüssel verfügen.
   - Speicherung der erstellten Schlüsselpaare
     Um die Signatur eines mobilen Clients überprüfen zu können, müssen die zuvor erstellten Schlüsselpaare gespeichert werden.
   - Versenden und Empfangen von XML-Nachricht
   - Überprüfung von signierten Nachrichten
     Die vom mobilen Client erstellte digitale Signatur für einen Kassabeleg wird von der Verrechnungsstelle verifiziert. Das Ergebnis wird an das POS System kommuniziert.
   - Abrechnung der Zahlungen
     Alle getätigten Transaktionen müssen auf den jeweiligen Konten (Konto des POS Eigentümers, Konto des mobilen Client) verbucht werden.

3. mobiles Endgerät:

   - Verifizieren des signierten Kassabeleges
     Der Client muß sicher gehen, daß der übermittelte Kassabeleg auch wirklich von einem authorisierten POS System stammt. Hierfür wird der öffentliche Schlüssel des POS Systems am Client gespeichert.
   - Signieren des akzeptierten Kassabeleges
     Wird ein Kassabeleg für richtig befunden, unterschreibt ihn der mobile Client mit seiner digitalen Signatur. Durch diese Unterschrift wird die Zahlung bestätigt.
   - Versenden und Empfangen von XML-Nachricht
   - Speicherung von Daten
     Am Client müssen verschiedene Daten wie Schlüsselpaare und getätigte Transaktionen gespeichert werden.
   - Darstellung der Benutzermasken
     Der Benutzer muß über grafische Oberflächen die Aktivitäten steuern können. Zusätzlich kann er auch Abfragen über bereits getätigte Transaktionen machen.

Die Aufgaben der Applikation am mobilen Endgerät werden zusammenfassend in der Abbildung 8.3 dargestellt. Prinzipell muß der mobile Client nur die

Bezahlung abwickeln und die getätigten Transaktionen für den Benutzer am Display darstellen. Die vorhergehende Auflistung zeigt jedoch, daß hinter der Abwicklung einer Bezahlung mehrere Funktionen stehen.

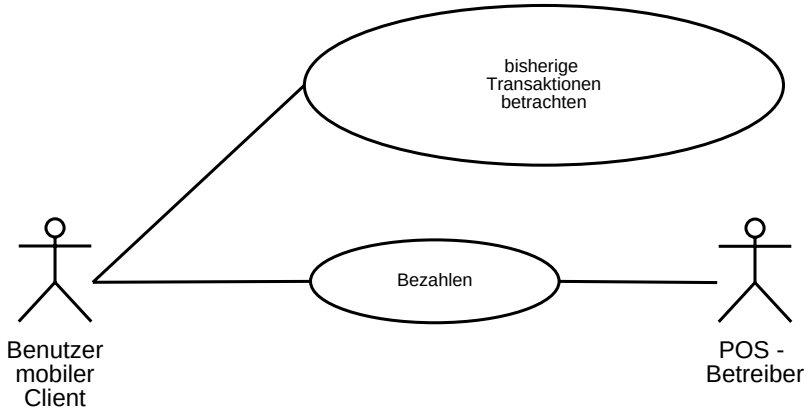

Abbildung 8.3: Aktionen des Benutzers

Kapitel 9

Design

Im vorangegangenen Kapitel Analyse wurden die Funktionalitäten sowie die Kommunikationswege der einzelnen Systeme grob skizziert. Die nachfolgenden Diagramme bieten eine detailierte Beschreibung der einzelnen Komponenten.

9.1 Klassendiagramme

Jedes der drei Systeme (mobiles Endgerät, POS System, Verrechnungsstelle) wird in einer objektorientierten Programmiersprache implementiert. Die Applikation für das mobile Endgerät wird in Java 2 Micro Edition programmiert, die Applikation für das POS System sowie für die Verrechnungsstelle wird in Java 2 erstellt. Die Unterschiede zwischen Java 2 und J2ME wurden bereits im Kapitel 6.1 erarbeitet.

Für jede Applikation wurde ein eigenes Klasssendiagramm erstellt:

1. Klassendiagramm mobiles Endgerät (siehe Abbildung 9.1):
   Um eine bessere Übersicht gewährleisten zu können, wurde die Applikation in zwei ”Packages” geteilt. Im Package mps.helper befinden sich Klassen die eine Kommunikation mit Dritt-APIs anbieten. Die strichlierten Linien zu den Packages bouncycastle und nanoxml sollen dies verdeutlichen. Das Package mps.actor enthält sozusagen das Herz der Applikation, also die Klassen, mit Hilfe derer die Applikation gesteuert wird. Details zu den angeführten Klassen und Packages:

   - Package nanoxml:
     Diese bereits im Kapitel 6.1.5 beschriebene API ermöglicht das Verarbeiten (Parsen) von XML-Dokumenten.

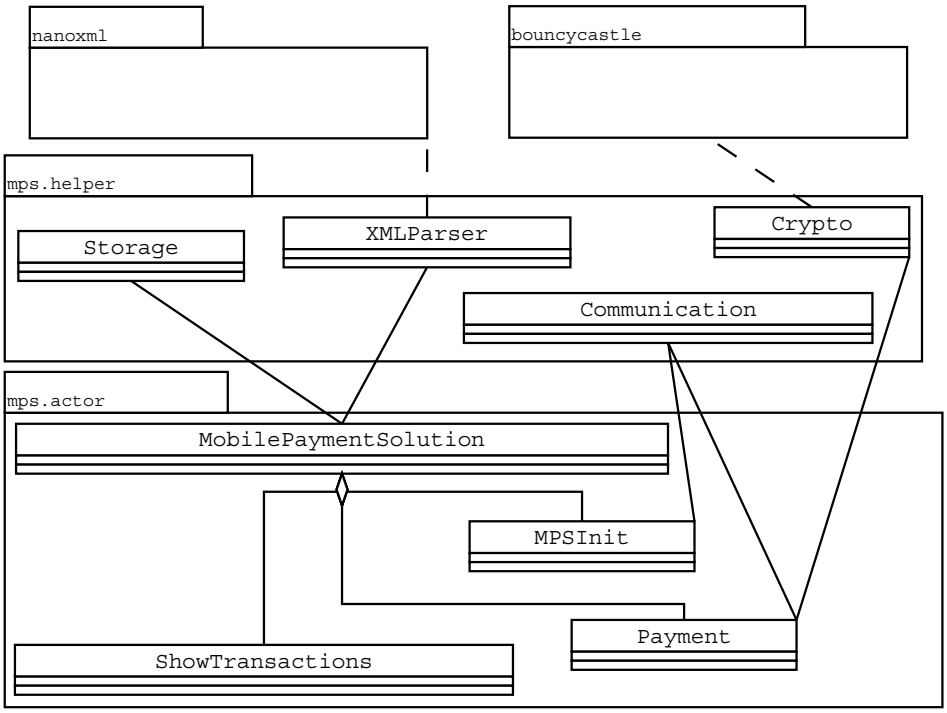

Abbildung 9.1: Klassendiagramm mobiles Endgerät

- Package bouncycastle:
  Alle Verschlüsselungsfunktionalitäten werden mit Hilfe der bereits im Kapitel 6.1.5 beschriebenen API, des Open Source Packages BouncyCastle, verarbeitet.
- Klasse mps.helper.Storage:
  Diese Klasse kümmert sich um die Speicherung der Daten und kommuniziert mit dem J2ME Package javax.microedition.rms.
- Klasse mps.helper.XMLParser:
  Die genaue Vergehensweise beim Parsen der XML-Dokumente wird in dieser Klasse implementiert.
- Klasse mps.helper.Communication:
  Diese Klasse bietet die Funktionalitäten zur Kommunikation per Irda. Da es keine Unterstützung für Irda unter J2ME gibt muß die Klasse eine vom Gerätehersteller angebotenen API ansteuern (in unserem Beispiel com.siemens.mp.io.Connection).
- Klasse mps.helper.Crypto:
  Alle Methoden zur Verschlüsselung und zum Signieren von Dokumenten werden mit Hilfe dieser Klasse implementiert.

- mps.actor.MobilePaymentSolution:
  Gestartet wird die Applikation durch einen Aufruf dieser Klasse. Über eine Menüführung können anschließend die verschiedenen Funktionalitäten angesteuert werden. Diese Klasse ist also der zentrale Dreh- und Angelpunkt für den Benutzer.
- mps.actor.MPSInit:
  Die gesamte Applikation muß vor der ersten Bezahlung initialisiert werden. Hierfür müssen die Schlüsselpaare für das Ersellen der bzw. für das Kontrollieren der Signaturen von der Applikation der Verrechnungsstelle geholt werden. Diese Klasse kümmert sich um die Initialisierung.
- mps.actor.ShowTransactions:
  Um dem Benutzer die bereits getätigten Zahlungen anzuzeigen wird diese Klasse aufgerufen.
- mps.actor.Payment:
  Ein Zahlungsvorgang wird durch die Payment Klasse angestossen. Das heißt diese Klasse kontrolliert die Kommunikation durch mps. helper.Communication, verarbeitet die empfangenen XML-Dokumente mit Hilfe von mps. helper.XMLParser und prüft bzw. erstellt Signaturen durch die Klasse mps.helper.Crypto.

Vorteil der Funktionsaufteilung in verschiedene Klassen ist die leichte Austauschbarkeit, sowie die bereits erwähnte gute Gesamtübersicht. Zum Beispiel kann die Klasse mps.helper.Communication, die eine Kommunikation per Irda ermöglicht, leicht gegen eine ander Klasse mps.helper.Communication, die eine Kommunikation per NFC ermöglicht, ausgetauscht werden.

2. Klassendiagramm Point of Sale System (POS) (siehe Abbildung 9.2):

   Um die Funktionsfähigkeit der mobilen Bezahlungslösung demonstrieren zu können muß auch eine Applikation, die den POS emuliert, implementiert werden. Das gesamte Programm wird als Klassse pos.PointOfSaleSystem umgesetzt. Für die Kommunikation per Irda wird die Open Source API von RxTx [Jarvi, 2004] verwendet. Vorteil dieser API gegenüber der von Sun Microsystems vorhandenen API, ist die Verfügbarkeit für verschiedene Betriebssystem wie Linux, Mac OS, Windows, etc.. Für die Einbindung der Verschlüsselungsfunktionalitäten wird auch in dieser Applikation die Open Source API von Bouncy Castle verwendet. Die Verarbeitung der übertragenen XML-Dokumente erfolgt durch die Nanoxml API.

3. Klassendiagramm Verrechnungsstelle (Clearing House) (siehe Abbildung 9.3):

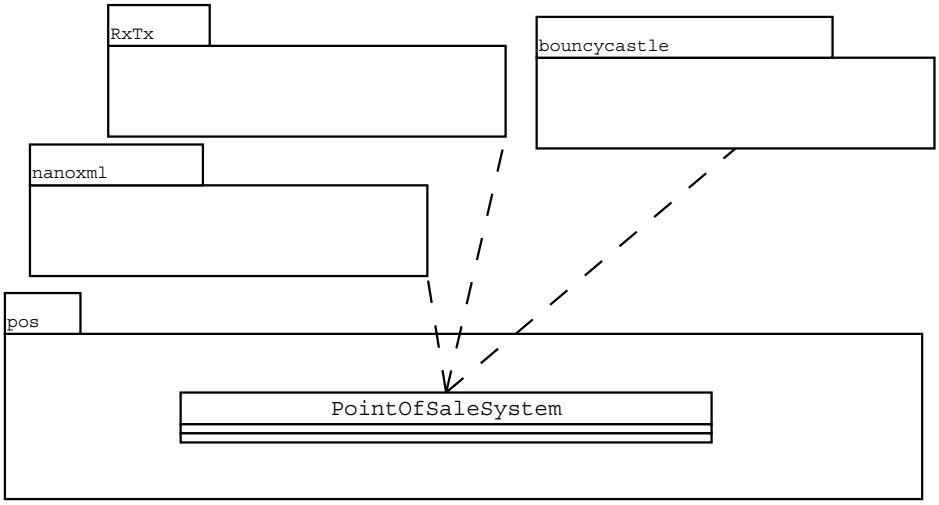

Abbildung 9.2: Klassendiagramm Point of Sale System (POS)

Ähnlich der Applikation für den Point of Sale wird dieses Programm von der Klasse clearing.PaymentTransaction gesteuert. Das Erstellen der Schlüsselpaare und damit verbundene Freischalten eines neuen mobilen Client übernimmt die Klasse clearing.CreateNewAccount. Die Kommunikation, Verschlüsselung und Verarbeitung von XML-Dokumenten erfolgt, genau wie bei der Applikation des POS, durch die Hilfe der APIs RxTx, Bouncy Castle und Nanoxml.

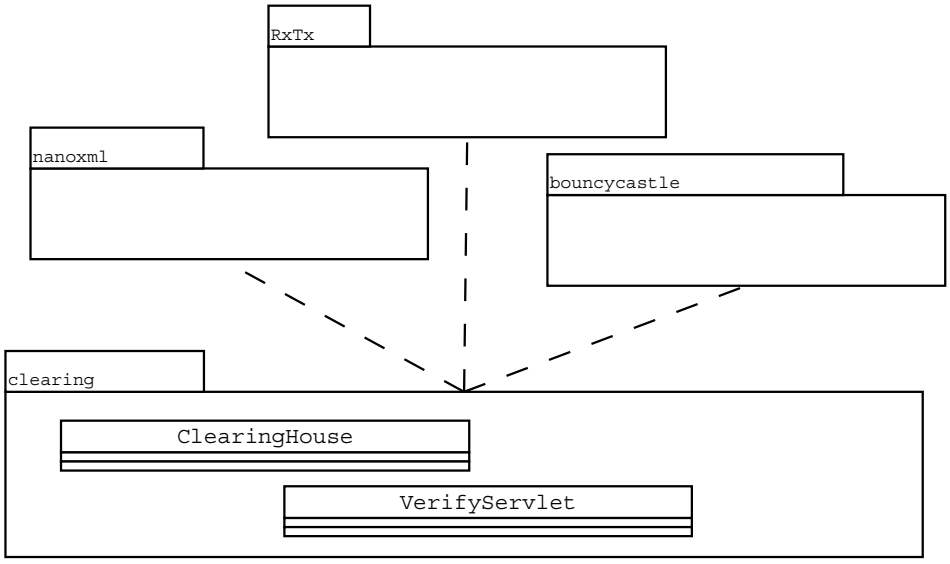

Abbildung 9.3: Klassendiagramm Verrechnungsstelle (Clearing House)

9.2 Sequenzdiagramme

Die nachfolgenden Sequenzdiagramme beschreiben die Programmabläufe des Prototypen. Details wie das Auslesen von XML-Dokumenten oder die Aktivierung der Infrarotschnittstelle wurden nicht in das Design aufgenommen. In Abbildung 9.4 wird der Initialisierungsprozess der mobilen Bezahlungslösung dargestellt. Hierfür fordert der mobile Client (zum Beispiel ein Mobiltelefon) ein Schlüsselpaar für die eindeutige Authentifizierung von der Verrechnungsstelle an. Die Verrechnungsstelle generiert dieses Schlüsselpaar für den Benutzer und sendet die Daten an den mobilen Client zurück. Die Daten werden am mobilen Client für die spätere Verwendung gespeichert. Zusätzlich speichert das mobile Endgerät den öffentlichen Schlüssel der POS-Systeme, um die digitale Signatur dieser Systeme prüfen zu können.

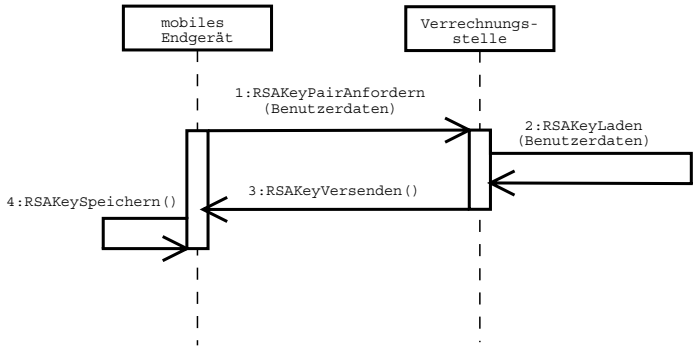

Abbildung 9.4: Initialisierung der mobilen Bezahlungslösung

Die Abbildung 9.5 zeigt den Ablauf eines Zahlungsvorganges. Der Zahlungsbeleg wird vom POS-System mit dessen privaten Schlüssel signiert und über die Irda Schnittstelle an den mobilen Client gesendet. Dieser prüft mit Hilfe des öffentlichen POS-Schlüssels, ob die Signatur richtig ist. Bei positiver Prüfung kann der Benutzer den Zahlungsbeleg bestätigen. Dazu wird der Beleg durch den mobilen Client abermals mit einer digitalen Signatur versehen und an das POS-System über die Irda Schnittstelle zurückgesandt.

Das POS-System muß wiederum die Richtigkeit der Unterschrift prüfen (siehe Abbildung 9.6). Dazu werden die vom mobilen Endgerät erhaltenen Daten an die Verrechnungsstelle, per Netzwerk, versandt. Die Verrechnungsstelle kontrolliert die Signatur des mobilen Client und sendet das Ergebnis an das POS-System zurück. Bei einer positiven Rückmeldung speichert das POS-System die erfolgreiche Transaktion. Eine negative Rückmeldung hat eine Ablehnung der Zahlungstransaktion zur Folge. Die erfolgreichen Transaktionen werden auf den Konten des POS-Betreibers und des Besitzers des mobilen Endgerätes bei der Verrech-

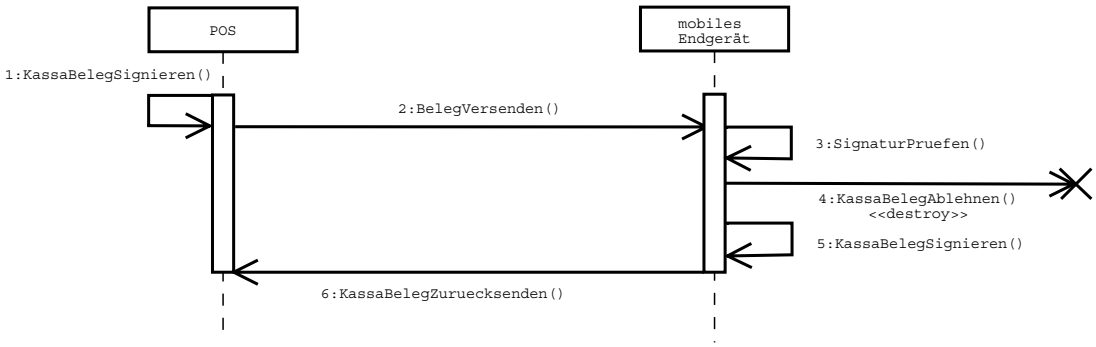

Abbildung 9.5: Bezahlungsvorgang

nungsstelle verbucht (wird nicht implementiert).

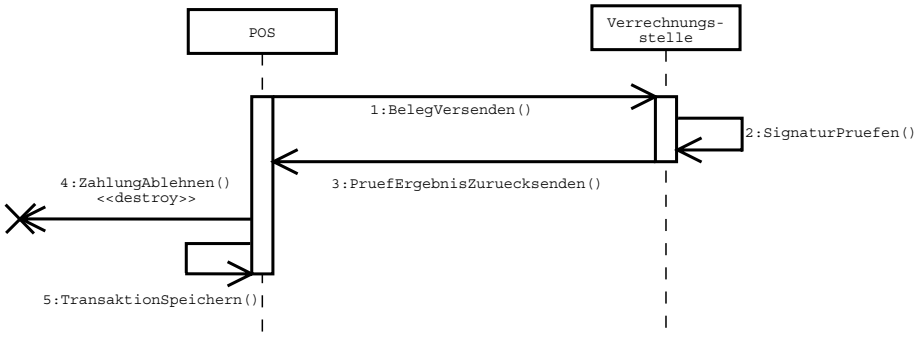

Abbildung 9.6: Signatur - Prüfungsvorgang

9.3 XML-Dokumente für den Datenaustausch

Bereits im Kapitel 8 ”Analyse” wurde festgelegt daß alle Daten zwischen den einzelnen Systemen im XML-Format ausgetauscht werden. Im Nachfolgenden werden die, für die gesamte Applikation notwendigen, XML-Dokumente spezifiziert. Alle Dokumente sind mit dem XML-Standard des ”World Wide Web Consortium” [Consortium, 2004] konform.

Das erste XML-Dokument speichert die Daten zur Übermittlung der einzelnen RSA-Schlüsselparameter zwischen der Verrechnungsstelle und dem mobilen Client (siehe Abb. 9.4).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE keyParams SYSTEM "keyparameters.dtd">
```

```
<keyParams>
        <mpsKeyParams>
                <mpsId></mpsId>
                <mod></mod>
                <pubExponent></pubExponent>
                <privExp></privExp>
                <p></p>
                <q></q>
                <dp></dp>
                <dq></dq>
                <qInv></qInv>
        </mpsKeyParams>
        <posKeyParams>
                <posmod></posmod>
                <pospubExponent></pospubExponent>
        </posKeyParams>
</keyParams>
```

Beschreibung:

Innerhalb des mpsKeyParams - ”Tag” werden die Daten über den privaten Schlüssel des mobilen Client sowie dessen eindeutige Kennung (mpsId) gespeichert. Der ”Tag” posKeyParameters beinhaltet die Parameter für den öffentlichen Schlüssel des Point of Sale Systems. Um die Spezifikation für das XML-Dokument zu vervollständigen, folgt nun die nötige ”Document Type Definition (DTD)”. Die DTD ist die Grundlage eines XML-Dokuments und enthält alle Elementbeschreibungen, die im XML-Dokument verwendet werden dürfen. Die ”Document Type Definition” wurde mit dem ”Tag” <!DOCTYPE mpsDocument SYSTEM ”invoice.dtd” > eingebunden.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT keyParams(mpsKeyParams, posKeyParams) >
<!ELEMENT mpsKeyParams (mpsId, mod, pubExponent, privExp, p, q, dp, dq, qInv) >
<!ELEMENT mpsId (#PCDATA)>
<!ELEMENT mod (#PCDATA)>
<!ELEMENT pubExponent (#PCDATA)>
<!ELEMENT privExp (#PCDATA)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT q (#PCDATA)>
<!ELEMENT dp (#PCDATA)>
<!ELEMENT dq (#PCDATA)>
<!ELEMENT qInv (#PCDATA)>
<!ELEMENT posKeyParams (posmod, pospubExponent) >
<!ELEMENT posmod (#PCDATA)>
<!ELEMENT pospubExponent (#PCDATA)>
```

Die leicht verständliche Notation für eine DTD beschreibt alle verwendbaren XML-Elemente sowie deren Verschachtelungen. Die Bezeichnung #PCDATA gibt an, daß an dieser Stelle alle möglichen Datentypen, wie zum Beispiel Boolean- oder Numerische-Werte, stehen dürfen.

Das zweite XML-Dokument ist die Grundlage für eine Zahlungstransaktion. Hier werden alle Daten über die Rechnung und die geleisteten digitalen Unterschriften gespeichert.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mpsDocument SYSTEM "invoice.dtd">
<mpsDocument>
        <invoice>
                <posId></posId>
                <currency></currency>
                <amount></amount>
                <invoiceDate></invoiceDate>
                <invoiceNumber></invoiceNumber>
        </invoice>
        <pos>
                <posSignature></posSignature>
        </pos>
        <mps>
                <mpsId></mpsId>
                <mpsSignature></mpsSignature>
        </mps>
        <clearingHouse>
                <signatureVerified></signatureVerified>
        </clearingHouse>
</mpsDocument>
```

Beschreibung:

Die für einen gültigen Kassabeleg notwendigen Daten, wie Rechnungsnummer, Aussteller (posID), etc., werden im invoice ”Tag” zusammengefasst. Das Point of Sale System bestätigt die Richtigkeit dieser Daten durch das Setzen einer digitalen Unterschrift innerhalb des pos - invoiceSignature ”Tags”. Wird der an den mobilen Client übermittelte Beleg durch den Benutzer ebenfalls für richtig befunden, so setzt die Applikation im mpsId ”Tag” die eindeutige Benutzerkennung, sowie die digitale Unterschrift im invoiceSignature ”Tag”, ein. Um die Transaktion abschließen zu können, muß der Rechnungsbeleg mit dem geleisteten digitalen Unterschriften noch von der Verrechnungsstelle für gültig befunden werden. Die Gültigkeitsprüfung wird in den clearingHouse - signatureVerified Bereich eingetragen. Der ausgefüllte Wert kann entweder ”true” oder ”false” sein. Nachfolgend

die ”Document Type Definition” des XML-Dokuments.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT mpsDocument (invoice, pos, mps, clearingHouse) >
<!ELEMENT invoice (posId, currency, amount, invoiceDate, invoiceNumber)>
<!ELEMENT posId (#PCDATA)>
<!ELEMENT currency (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT invoiceDate (#PCDATA)>
<!ELEMENT invoiceNumber (#PCDATA)>
<!ELEMENT pos (posSignature)>
<!ELEMENT posSignature (#PCDATA)>
<!ELEMENT mps (mpsId, mpsSignature)>
<!ELEMENT mpsId (#PCDATA)>
<!ELEMENT mpsSignature (#PCDATA)>
<!ELEMENT clearingHouse (signatureVerified)>
<!ELEMENT signatureVerified (true|false)>
```

Unterschied zur vorhergehenden DTD ist das Element ”signatureVerified”, daß die erlaubten Werte auf ”true” oder ”false” durch die Definition ”(true—false)” eingrenzt.

Kapitel 10

Prototyping

Nachdem in der Designphase der Aufbau und Ablauf des Prototypen spezifiziert wurde, beschreibt dieses Kapitel den programmtechnischen Implementierungsprozess, die verwendeten Entwicklungstools und die im Zuge der Umsetzung aufgetretenen Probleme.

10.1 Entwicklungsumgebung

Bei der Auswahl eines mobilen Endgerätes lag das Hauptaugenmerk auf der Unterstützung der Irda-Schnittstelle unter J2ME. Leider ist diese Auswahl sehr beschränkt. Das Siemens SL45i (siehe Abb. 4.1) ist eines der wenigen Mobiltelefone das eine solche Unterstützung anbietet. Die Infrarotansteuerung erfolgt über eine Siemens eigene J2ME API. Die Klasse hierfür heißt com.siemens.mp.io.Connection.

Siemens bietet weiters eine Testumgebung mit Emulatoren für verschiedene Mobiltelefonmodelle (Siemens Mobility Toolkit (SMTK) for Java(tm) enabled mobile phones). Dieser SMTK wird im Prototypen nur bedingt verwendet, da es bei Tests immer wieder zu Programmabstürtzen kam. Vom SMTK wird die Klassenbibliothek, sowie der ”Preverifier” verwendet. Der SMTK bietet auch keinerlei Möglichkeiten zur Simulation einer Infrarotschnittstelle. Zusätzlicher Nachteil dieser Testumgebung ist die Plattformgebundenheit an Windows.

Als grafische Entwicklungsumgebung wird der Oracle JDeveloper in der Version 9.03 verwendet. Der JDeveloper bietet neben diversen Debuggingfunktionalitäten auch einen Wizard zum Erstellen von J2ME Midlets.

Für das Testen der Verschlüsselungsfunktionalitäten wird der Java 2 Micro Editi-

on Wireless Toolkit von Sun verwendet. Dieser bietet eine stabile Plattform zum Ausführen von J2ME Applikationen. Genau wie der SMTK bietet auch dieser Toolkit keine Möglichkeit zur Simulation von Infrarotschnittstellen.

Ablauf des Entwicklungsprozesses:

1. Programmierung mit dem JDeveloper
2. Mit dem Java 2 Micro Edition Wireless Toolkit von Sun werden testbare Programmteile in der Testumgebung kompiliert, preverified und ausgeführt
3. Programmteile, die properitäre Siemens Klassen benötigen, müssen ”händisch” kompiliert, preverified und auf das Mobiltelefon kopiert werden. Der Test wird direkt am Mobiltelefon ausgeführt.

10.2 Probleme während der Implementierung

Die in der Designphase geplante Implementierung der Datenübertragung per Irda unter Verwendung der RxTx API konnte nicht umgesetzt werden. Grund hierfür ist die fehlende Dokumentation von Seiten des Mobiltelefonherstellers über das verwendete Übertragungsprotokoll. Um trotzdem den Austausch der XML-Dokumente per Infrarotschnittstelle zu ermöglichen, wird das von Siemens angebotene Programm zum Datenaustausch per Irda verwendet. Dieses Programm ist nur unter Windows verfügbar. Eine Plattformunabhängigkeit ist daher nicht gegeben.

In der Siemens Dokumentation zur Ansteuerung der Irda-Schnittstelle am Mobiltelefon unter J2ME finden sich keinerlei Angaben zum Aufbau einer Datei, die unter Java 2 Micro Edition ausgetauscht werden soll. Das Senden von Dateien war von Anfang an kein Problem, jedoch konnten keine Daten vom Mobiltelefon empfangen werden. Durch eine Analyse der vom Mobiltelefon aus gesendeten Dateien konnte ich ermitteln, daß in den ersten vier Bytes einer Datei, Informationen über die Datenlänge gespeichert sind. Die Berechnung hierfür erwies sich als etwas ungewöhnlich: Zuerst wird der Hexwert der Datenlänge ermittelt. Dieser Wert wird in zweistellige Felder geteilt und in umgekehrter Reihenfolge nach Berechnung des entsprechenden Integerwertes als Bytewert in die Datei geschrieben.

Ein Beispiel:
Die Stringlänge eines Textes beträgt 306 Zeichen.
Der berechnete Hexwert davon ist 132.
In das erste Byte der Datei muß daher der Integerwert der Hexzahl 32 geschrieben werden und in das zweite Byte der Integerwert der Hexzahl 1.

Dies bedeutet für das erste Byte 50 und für das zweite Byte 1.
Die restlichen zwei Bytes werden mit Nullen gefüllt.

Programmtechnisch umgesetzt wurde diese Berechnung in der Methode calculateByteValues der Klasse pos.PointOfSaleSystem.

Für die Übermittlung dieser Datei an das J2ME Programm muß die Dateiendung .jd benannt werden.

Ob diese Probleme bei anderen Mobiltelefonen, die eine Irda-Schnittstelle anbieten, ebenfalls auftreten wurde nicht nachgeprüft.

Die API nanoxml, welche zum Parsen von XML Dokumenten vorgesehen war, konnte nicht im Applikationsteil des mobilen Client verwendet werden. Grund hierfür ist, daß in der API die Klasse java.io.StringReader verwendet wird, diese aber nicht unter J2ME vorhanden ist. Prinzipiell ist zwar nanoxml für den Einsatz unter J2ME vorgesehen, jedoch wird die benötigte Klasse java.io.StringReader erst ab der CDC Konfiguration unterstützt.

Die Java Virtuel Machine des für die Implementierung verwendeten Siemens SL45i weist einen Fehler auf. Dieser wurde beim Testen zur Erstellung von digitalen Signaturen entdeckt. Ein Programm, das eine Signatur erstellt und anschließend selbst verifiziert, wurde in der Testumgebung des Sun WTK erfolgreich ausgeführt. Das gleiche Ergebnis konnte beim Ausführen auf einem Siemens S55 festgestellt werden. Läuft die Applikation jedoch auf dem Siemens SL45i, so wird eine fehlerhafte Signatur erstellt und auch die Verifizierung kann nicht durchgeführt werden. Den Beweis hierfür bietet die Applikation im Anhang A.

Die Abbildungen 10.1, 10.2 und 10.3 zeigen den erfolgreichen Ablauf des Programmes in der Testumgebung des Sun WTK

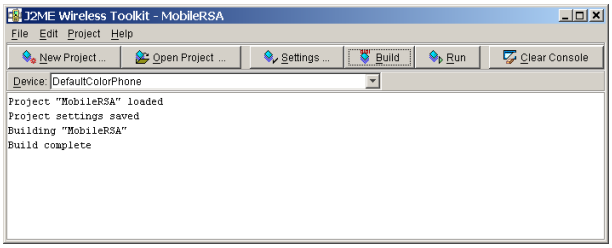

Abbildung 10.1: Kompilieren und Preverifying der Applikation

Abbildung 10.2: Starten der Applikation

Abbildung 10.3: Ausgabe der erfolgreichen Durchführung

10.3 Ablaufbeschreibung und Screenshots des Prototypen

Die folgenden Screenshots zeigen den gesamten Implementierungsumfang. Um das mobile Bezahlungssystem initialisieren zu können, erstellt die Verrechnungsstelle ein RSA Schlüsselpaar. Diese Schlüessel werden gemeinsam mit den öffentlichen Schlüssel des POS-Systems in einem XML Dokument zusammengefasst. Die Identität des Kunden wird durch die ”MPS-ID” festgelegt. Das erstellte XML Dokument wird am Filesystem der Verrechnungsstelle gespeichert und kann anschließend über die Infrarotschnittstelle versandt werden. Die Abbildungen 10.4, 10.5 und 10.6 zeigen diese Schritte.

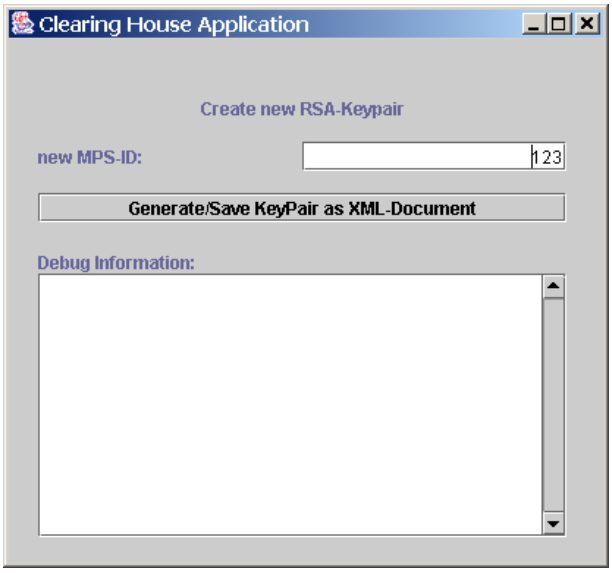

Abbildung 10.4: Eingabe der neuen MPS-ID

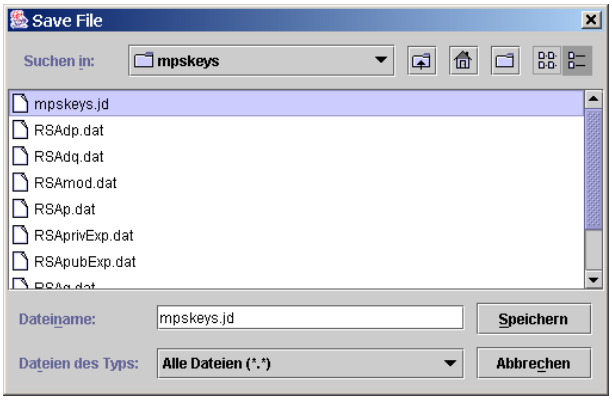

Abbildung 10.5: Speichern des XML Dokuments

Abbildung 10.6: Ausgabe des erstellten XML Dokuments in einer Textbox

Das erstellte XML Dokument wird nun per Irda an das Mobiltelefon geschickt. Der Infrarotschnittstellen-Listener des Mobiltelefons empfängt die Daten und verarbeitet, beziehungsweise speichert Teile des XML Dokuments. Die Abbildungen 10.7, 10.8 und 10.9 zeigen diesen Ablauf.

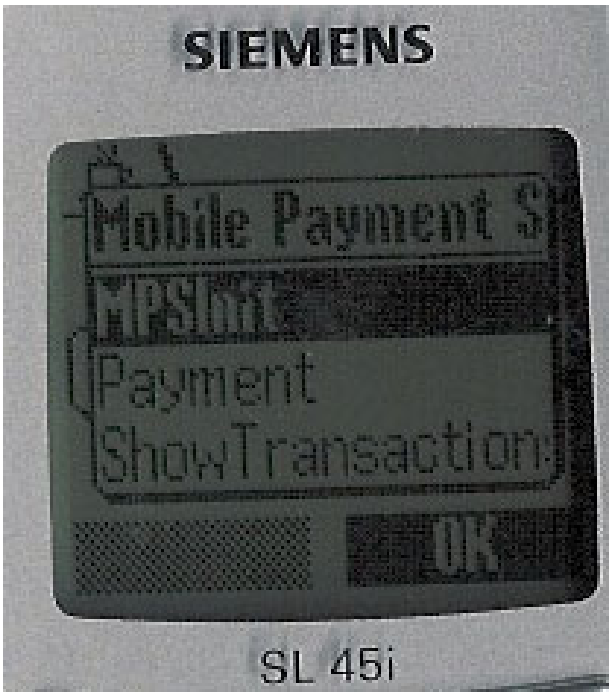

Abbildung 10.7: Starten der Applikation am Mobiltelefon

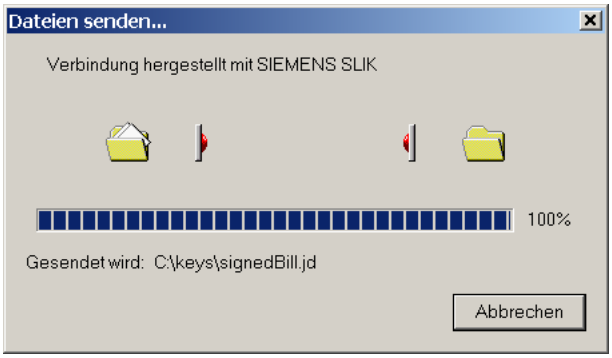

Abbildung 10.8: Versenden des XML Documents

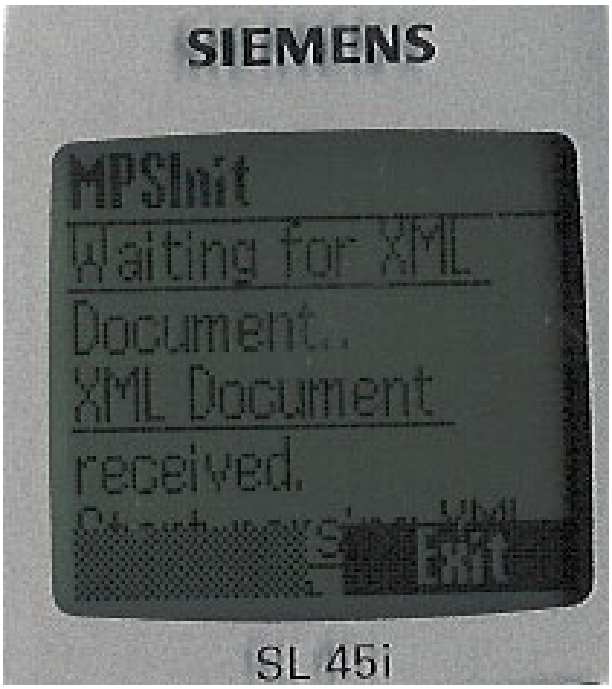

Abbildung 10.9: Verarbeiten des XML Documents

Für die Abwicklung von Zahlungen kommt das PointOfSale Programm zum Einsatz. Es müssen zuerst die Rechnungsdaten in die grafische Oberfläche eingegeben werden (siehe Abbildung 10.10).

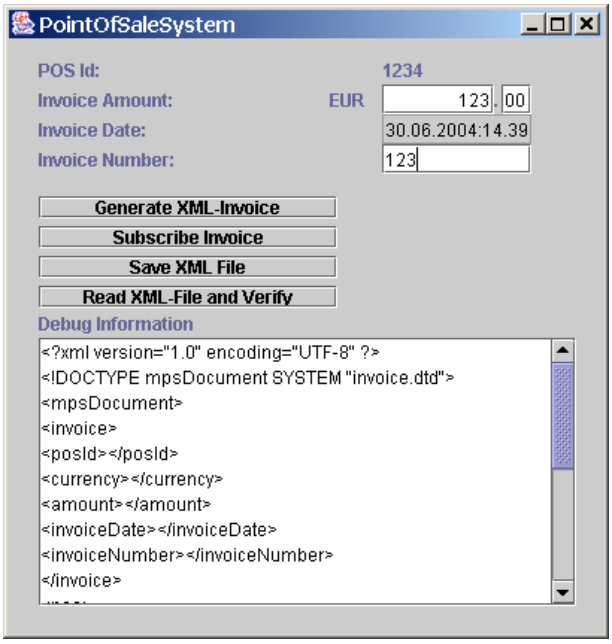

Abbildung 10.10: Eingabe der Rechnungsdaten

Die Abbildung 10.11 zeigt das aus den Rechnungsdaten erstellte Dokument.

Das erstellte XML Dokument wird nun digital signiert. Dazu wird der Inhalt des invoice ”tags” inklusive des ”tag”-Elements an die Signaturfunktion übergeben. Die Signatur wird ebenfalls in das XML Dokument eingebunden (posSignature) und am Filesystem gespeichert (siehe Abbildungen 10.12 und 10.13).

Das XML Dokument wird an den mobilen Client versendet.

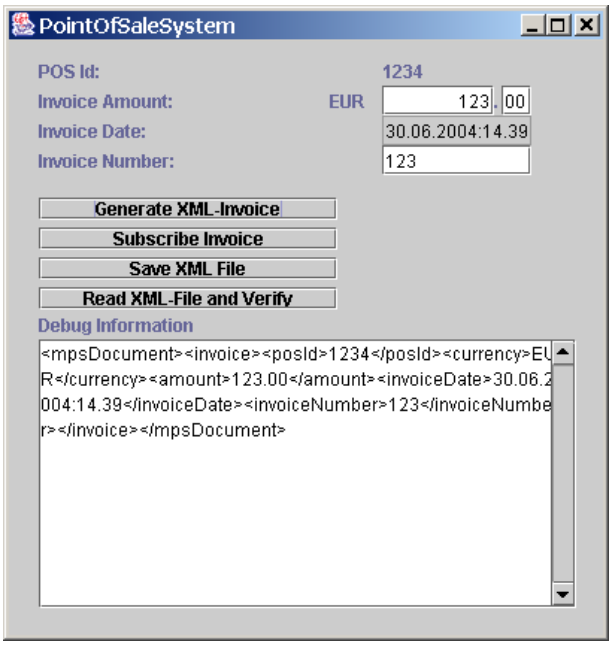

Abbildung 10.11: XML Dokument mit Rechnungsdaten

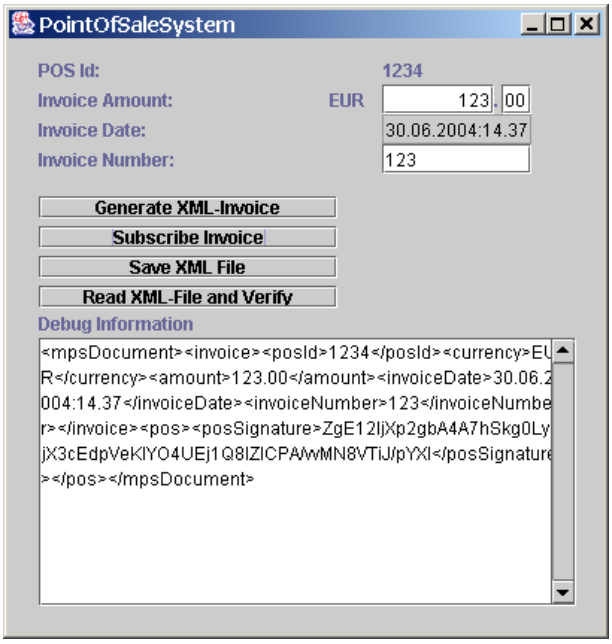

Abbildung 10.12: Ausgabe des signierten XML Dokuments

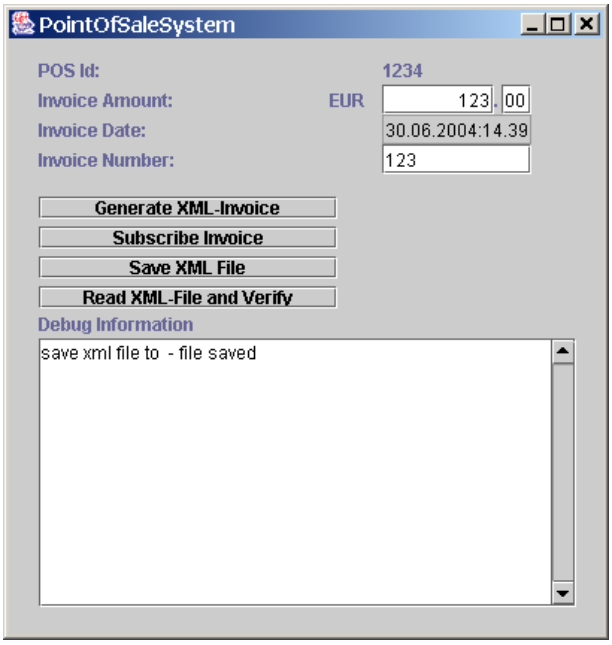

Abbildung 10.13: Speicherung am Filesystem

Das erhaltene XML Dokument wird geparst und die Unterschrift mit dem öffentlichen Schlüssel des POS-Systems verifiziert. Dies kann leider mit dem Siemens SL45i nicht richtig durchgeführt werden, da wie bereits unter Kapitel 10.2 beschrieben ein Fehler in der Virtuel Machine des Mobiltelefons besteht.

Der Benutzer bekommt nach der Verifizierung einen zusätzlichen Aktionsbutton am Mobiltelefon angezeigt. Durch ein Drücken auf diesen Button wird die Rechnung vom Benutzer digital signiert, an das POS-System zurückgesandt und die Transaktion gespeichert (siehe Abbildung 10.14).

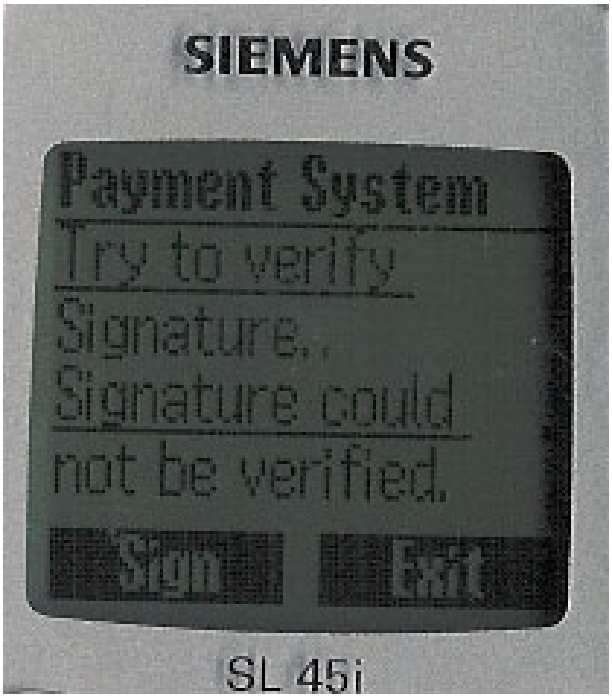

Abbildung 10.14: Payment Applikation

Das vom Mobiltelefon zurückgesandte XML Dokuemnt wird vom Filesystem geladen, geparst und über eine HTTP Verbindung an ein Servlet der Verrechnungsstelle zur Verifizierung geschickt. Nachdem die Verrechnungsstelle die Signatur des mobilen Client mit Hilfe des öffentlichen Schlüssels verifiziert hat, antwortet die Verrechnungsstelle entweder mit einem positiven OK oder bei Ablehnung der Signatur mit einem DENIED. Das POS System gibt diese Meldung in der grafischen Oberfläche aus. Die Abbildungen 10.15, 10.16 und 10.17 zeigen diese Schritte.

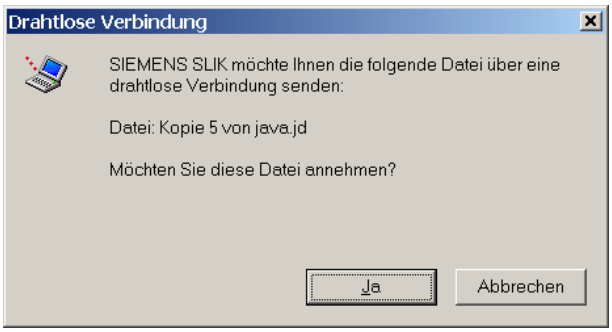

Abbildung 10.15: Laden des vom Mobiltelefon übertragenen XML Dokuments

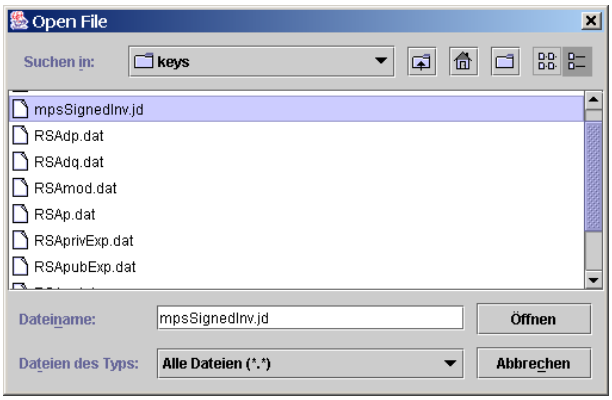

Abbildung 10.16: Laden des vom Mobiltelefon übertragenen XML Dokuments

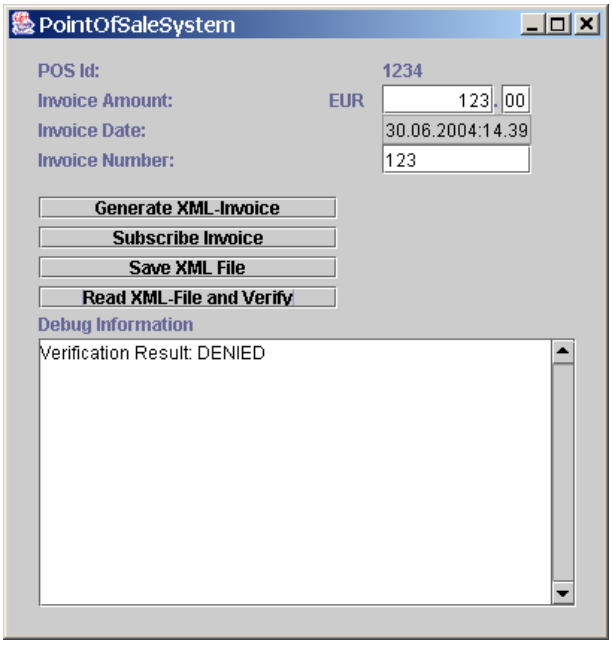

Abbildung 10.17: Verifizierungskommunikation mit der Verrechnungsstelle

Kapitel 11

Resümee

Die zu Beginn der Diplomarbeit herrschende Euphorie über die Möglichkeiten eine vollfunktionsfähige mobile Bezahlungslösung unter J2ME zu implementieren, wurde durch fehlende Standards zur Schnittstellenkommunikation unter J2ME sehr bald getrügt.

Das Ergebnis der Diplomarbeit ist eine auf properitären APIs von Siemens basierende mobile Bezahlungslösung.

Die entstandene funktionsfähige Applikation konnte aufgrund des im Kapitel 10.2 beschriebenen Fehlers leider nie vollständig durchlaufen werden. Es kann aber nach unzähligen Einzeltests garantiert werden, daß das komplete Programm lauffähig ist.

Die in den verschiedenen Kapiteln beschriebenen Voraussetzungen für die Marktreife einer solchen Applikation sind im Moment leider noch nicht gegeben. Prinzipiell bestehen zwar sehr viele Ansätze, Schnittstellen zu allen benötigten Technologien für ein mobiles Bezahlungssystem bereitzustellen, deren Umsetzung setzt jedoch eine breite Unterstützung der Hersteller voraus. Der Einfluß der Mobilfunkunternehmen auf die Hersteller von mobilen Endgeräten, die aus ökonomischen Gründen eine Datenübertragung per Mobilfunknetz für eine solche Applikation bevorzugen werden, fördert die rasche Integration neuer Kommunikationstechnologien (wie zum Beispiel NFC, siehe Kapitel 4.2.4) nicht.

Voraussetzungen für die Marktreife der hier beschriebenen mobilen Bezahlungslösung sind:

- Einsatz von standardisierten J2ME APIs:
  Es ist nicht möglich eine Standardapplikation für mobile Clients zu ent-

wickeln, wenn jeder Hersteller andere APIs zur drahtlosen Kommunikation mit anderen Endgeräten anbietet.

- J2ME Schnittstelle zur Smart Card:
  Diese Schnittstelle ermöglicht die performante Abarbeitung von cryptografischen Anfragen.

- Breite Unterstützung einer drahtlosen Kommunikationsschnittstelle unter J2ME:
  Zur Zeit sind nur sehr wenig mobile Endgeräte am Markt, die die Kommunikation per Infrarot oder Bluetooth unter J2ME unterstützen.

- Entwicklungsumgebungen, die wirklich alle Funktionalitäten der mobilen Endgeräte simulieren können:
  Wie ich im Kapitel 10.1 beschrieben habe, erlaubt es keine der Entwicklungsumgebungen eine Infrarotschnittstelle zu simulieren. Diese fehlende Eigenschaft verlängert den Entwicklungsprozess merklich, da für jeden Test das Programm kompiliert, preverified, kopiert und am mobilen Endgerät gestartet werden muß.

Wären diese Forderungen alle erfüllt, so würde der Entwicklung eines marktreifen Systems nichts mehr im Wege stehen.

Kapitel 12

Source Code

Um einen besseren Überblick vom Source Code zu erhalten, wurde dieser in drei Kapitel (je nach Applikation) aufgeteilt.

12.1 Java Package clearing

```
package clearing;

/*Bouncy Castle Klassen*/
import java.security.*;
import java.math.*;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.generators.*;
import org.bouncycastle.crypto.signers.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.digests.*;                                    10
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.util.encoders.*;
import java.io.*;

public class BCUtil
{
   private static RSAKeyParameters pubKey;
   private static RSAPrivateCrtKeyParameters privKey;
   public static int KEY_STRENGTH = 384;
   public static int KEY_CERTAINTY = 4;                                      20
   public static String FILE_DIRECTORY = "C:\\mpskeys\\";

   /*Schluesselparameter*/
   private static BigInteger pubExp = new BigInteger("10001", 16);
   private static BigInteger mod;
   private static BigInteger privExp;
```

```
private BigInteger pubExponent;
private BigInteger dp;
private BigInteger dq;
private BigInteger p;
private BigInteger q;
private BigInteger qInv;
private FileOutputStream out;

public void BCUtil()
{
}

/*Erstellung neuer Schluesselpaare.*/
public static synchronized void generateKeyPair() throws Exception
{
  /*erstellt eine sichere Randomzahl (=Zufallszahl)*/
  SecureRandom secRandom = new SecureRandom();
  RSAKeyGenerationParameters RSAKeyGenPara = new RSAKeyGenerationParameters
                                (pubExp, secRandom, KEY_STRENGTH, KEY_CERTAINTY);
  RSAKeyPairGenerator RSAKeyPairGen = new RSAKeyPairGenerator();
  RSAKeyPairGen.init(RSAKeyGenPara);
  AsymmetricCipherKeyPair keyPair = RSAKeyPairGen.generateKeyPair();

  privKey = (RSAPrivateCrtKeyParameters) keyPair.getPrivate();
  pubKey = (RSAKeyParameters) keyPair.getPublic();
}

/*Speichert alle Schluesselparameter.*/
public void saveKeyPairToFile() throws Exception
{
  mod = privKey.getModulus();
  saveToFile("RSAmod.dat", mod);

  privExp = privKey.getExponent();
  saveToFile("RSAprivExp.dat", privExp);

  pubExponent = privKey.getPublicExponent();
  saveToFile("RSApubExp.dat", pubExponent);

  dp = privKey.getDP();
  saveToFile("RSAdp.dat", dp);

  dq = privKey.getDQ();
  saveToFile("RSAdq.dat", dq);

  p = privKey.getP();
  saveToFile("RSAp.dat", p);

  q = privKey.getQ();
```

```
    saveToFile("RSAq.dat", q);

    qInv = privKey.getQInv();
    saveToFile("RSAqInv.dat", qInv);
                                                                    80
}

/*Durchfuehrung der Speicherung.*/
private void saveToFile(String fileName, BigInteger fileContent)
throws Exception
{

    String result = new String(Base64.encode(fileContent.toByteArray()));
    out = new FileOutputStream(FILE_DIRECTORY + fileName);
    out.write(result.getBytes());                                   90
    out.flush();
    out.close();
}

/*Laden der Schluesselparameter.*/
public void loadKeyPair() throws Exception
{
    mod = loadFile("RSAmod.dat");
    privExp = loadFile("RSAprivExp.dat");
    pubExponent  = loadFile("RSApubExp.dat");                       100
    p = loadFile("RSAp.dat");
    q = loadFile("RSAq.dat");
    dp = loadFile("RSAdp.dat");
    dq = loadFile("RSAdq.dat");
    qInv = loadFile("RSAqInv.dat");

    /*Initialisieren des privaten und des oeffentlichen Schluessels.*/
    privKey = new RSAPrivateCrtKeyParameters(mod, pubExponent,
                    privExp, p, q, dp, dq, qInv);
    pubKey = new RSAKeyParameters(false, mod, pubExponent);         110
}

/*Durchfuehrung des Ladevorgangs.*/
private BigInteger loadFile(String fileName) throws Exception
{
    File inputFile = new File(FILE_DIRECTORY + fileName);
    InputStream in = new FileInputStream(inputFile);

    byte[] inBuf = new byte[(int)inputFile.length()];
                                                                    120
    int offset = 0;
    int numRead = 0;
    while (offset < inBuf.length
            && (numRead=in.read(inBuf, offset, inBuf.length-offset)) >= 0) {
```

```
        offset += numRead;
    }

    if (offset < inBuf.length) {
        throw new IOException("Could not completely read file "+fileName);
    }
    in.close();

    BigInteger returnValue = new BigInteger(Base64.decode(inBuf));

    return returnValue;
}

/*Eine Signatur fr eine Nachricht erstellen,
 * der Rckgabewert ist ein Base64 verkodierter String.*/
static public String getSignatureAsBase64 (String message) throws Exception
{
    /*Hashwert mit der Grï¿½ von 160 Bit*/
    SHA1Digest digEng = new SHA1Digest();
    RSAEngine rsaEng = new RSAEngine();

    /*initailisieren der Signatur*/
    PSSSigner signer = new PSSSigner(rsaEng, digEng, 16);
    signer.init(true, privKey);

    /*die zu signierende Nachricht einfgen*/
    signer.update(message.getBytes(), 0, (message.getBytes().length));

    /*das Signaturresultat als ByteArray*/
    byte [] sig = signer.generateSignature();

    return new String(Base64.encode(sig));
}

/*Eine signierte Nachricht verifizieren, die Signatur wird als
 * Base64 verkodierter String uebergeben.*/
static public boolean verifySignature (String aSig, String aText)
throws Exception
{
    byte[] aMessage = aText.getBytes();
    byte[] aSignature = Base64.decode(aSig);

    SHA1Digest    sha1digest = new SHA1Digest();

    RSAEngine rsaengine = new RSAEngine();

    PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

    psssigner.init(false, pubKey);
```

```
    psssigner.update(aMessage, 0, aMessage.length);

    return psssigner.verifySignature(aSignature);
  }

  public String getmod()                                                    180
  {
    mod = privKey.getModulus();
    return new String(Base64.encode(mod.toByteArray()));
  }

  public String getprivExp()
  {
    privExp = privKey.getExponent();
    return new String(Base64.encode(privExp.toByteArray()));
  }                                                                         190

  public String getpubExponet()
  {
    pubExponent = privKey.getPublicExponent();
    return new String(Base64.encode(pubExponent.toByteArray()));
  }

  public String getdp()
  {
    dp = privKey.getDP();                                                   200
    return new String(Base64.encode(dp.toByteArray()));
  }

  public String getdq()
  {
    dq = privKey.getDQ();
    return new String(Base64.encode(dq.toByteArray()));
  }

  public String getp()                                                      210
  {
    p = privKey.getP();
    return new String(Base64.encode(p.toByteArray()));
  }

  public String getq()
  {
    q = privKey.getQ();
    return new String(Base64.encode(q.toByteArray()));
  }                                                                         220

  public String getqInv()
```

```
  {
    qInv = privKey.getQInv();
    return new String(Base64.encode(qInv.toByteArray()));
  }
}
```

```
package clearing;

import javax.swing.JFrame;
import java.awt.Dimension;
import javax.swing.JLabel;
import java.awt.Rectangle;
import javax.swing.JEditorPane;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFileChooser;

import nanoxml.*;
import java.util.*;
import java.io.*;

public class ClearingHouse extends JFrame
{
  private JLabel jLabel1 = new JLabel();
  private JLabel jLabel2 = new JLabel();
  private JButton jButton1 = new JButton();
  private JLabel jLabel3 = new JLabel();
  private JTextArea jTextArea1 = new JTextArea();
  private JScrollPane jScrollPane;
  private JTextField jTextField1 = new JTextField();

  private XMLElement xmlElement = new XMLElement();

  /*Directory Pfad fuer die Keyparameter.*/
  public static String FILE_DIRECTORY = "C:\\mpskeys\\";

  public ClearingHouse()
  {
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
```

```
      e.printStackTrace();
    }

}

public static void main(String[] args)
{
    ClearingHouse clear = new ClearingHouse();                          50
    clear.setVisible(true);
    clear.repaint();
}

/*Grafische Oberflaeche wird gestartet.*/
private void jbInit() throws Exception
{
    this.getContentPane().setLayout(null);
    this.setSize(new Dimension(413, 385));
    this.setTitle("Clearing House Application");                        60

    jLabel1.setText("Create new RSA-Keypair");
    jLabel1.setHorizontalAlignment(jLabel1.CENTER);
    jLabel1.setBounds(new Rectangle(20, 35, 360, 25));

    jLabel2.setText("new MPS-ID:");
    jLabel2.setBounds(new Rectangle(20, 70, 155, 20));

    jButton1.setText("Generate/Save KeyPair as XML-Document");
    jButton1.setBounds(new Rectangle(20, 105, 360, 20));                70


    jLabel3.setText("Debug Information:");
    jLabel3.setBounds(new Rectangle(20, 140, 190, 25));

    jTextArea1.setText("");
    jTextArea1.setEditable(false);
    jTextArea1.setLineWrap(true);

    jScrollPane = new JScrollPane(jTextArea1);                          80
    jButton1.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent e)
        {
          jButton1_actionPerformed(e);
        }
      });
    jScrollPane.setVerticalScrollBarPolicy(
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    jScrollPane.setPreferredSize(new Dimension(70, 70));                90
    jScrollPane.setBounds(new Rectangle(20, 160, 360, 180));
```

```
    jTextField1.setText("123");
    jTextField1.setBounds(new Rectangle(200, 70, 180, 20));
    jTextField1.setHorizontalAlignment(JTextField.RIGHT);


    this.getContentPane().add(jTextField1, null);
    this.getContentPane().add(jLabel3, null);
    this.getContentPane().add(jButton1, null);                          100
    this.getContentPane().add(jLabel2, null);
    this.getContentPane().add(jLabel1, null);
    this.getContentPane().add(jScrollPane, null);
}

private void jButton1_actionPerformed(ActionEvent e)
{
    try
    {
      checkValues();                                                    110
      /*Ein neues RSA Schluesselpaar wird erstellt.*/
      BCUtil bcutil = new BCUtil();
      bcutil.generateKeyPair();
      bcutil.saveKeyPairToFile();

      /*Das RSA Schluesselpaar wird in einem XML Dokument
        * gespeichert.*/
      jTextArea1.setText("Generate XML-Invoice");
      xmlElement = new XMLElement();
                                                                        120
      xmlElement.setName("keyParams");
      xmlElement.addChild(new XMLElement());


      for (Enumeration enum = xmlElement.enumerateChildren();
            enum.hasMoreElements() ;)
      {
        XMLElement child = (XMLElement) enum.nextElement();
        child.setName("mpsKeyParams");
                                                                        130
        /*Zusaetzliche Childelemente werden hinzugefuegt*/
        setNewXMLChild(child, "mpsId", jTextField1.getText());
        setNewXMLChild(child, "mod", bcutil.getmod());
        setNewXMLChild(child, "pubExponent", bcutil.getpubExponet());
        setNewXMLChild(child, "privExp", bcutil.getprivExp());
        setNewXMLChild(child, "p", bcutil.getp());
        setNewXMLChild(child, "q", bcutil.getq());
        setNewXMLChild(child, "dp", bcutil.getdp());
        setNewXMLChild(child, "dq", bcutil.getdq());
        setNewXMLChild(child, "qInv", bcutil.getqInv());                140
```

```
}

xmlElement = setNewXMLChild(xmlElement, "posKeyParams");

/*Neuinitialisierung der Klasse BCUtil zum Laden der
 * Parameter des POS public keys.*/
bcutil = new BCUtil();
bcutil.FILE_DIRECTORY = "C:\\keys\\";
bcutil.loadKeyPair();
                                                                    150
for (Enumeration enum = xmlElement.enumerateChildren();
        enum.hasMoreElements() ;)
{
  XMLElement child = (XMLElement) enum.nextElement();

  if (child.getName().equalsIgnoreCase("posKeyParams"))
  {
    setNewXMLChild(child, "posMod", bcutil.getmod());
    setNewXMLChild(child, "posPubExponent", bcutil.getpubExponet());
  }                                                                 160
}

jTextArea1.setText("created XML Document: " + xmlElement.toString());

/*Dateimanager zum speichern oeffnen.*/
JFileChooser fc = new JFileChooser();
fc.setDialogTitle("Save File");

/*Auswahl zeigt nur Dateien keine Ordner*/
fc.setFileSelectionMode( JFileChooser.FILES_ONLY);                  170

/*Starte in dieser Directory.*/
fc.setCurrentDirectory(new File(FILE_DIRECTORY+"."));

/*Oeffne das Fenser.*/
int result = fc.showSaveDialog(this);

if( result == JFileChooser.CANCEL_OPTION)
{
  jTextArea1.setText("No File selected!");                          180
}
else if( result == JFileChooser.APPROVE_OPTION)
{
  /*Die ausgewaehlte Datei wird zum lesen uebergeben.*/
  File file = fc.getSelectedFile();

  FileOutputStream out = new FileOutputStream(file);

  /*Die Laenge der XML - Daten werden als Hex-Wert berechnet.*/
```

```
        Integer hexLength = new Integer                                    190
              (Integer.toHexString(xmlElement.toString().length()));
        System.out.println(hexLength);

        /*Um Dateien fuer den J2ME Client lesbar zu machen ist
        * es notwendig die Datenlaenge in die ersten vier Bytes
        * der uebertragenen Datei zu schreiben.*/
        Hashtable twoBytePair = calculateByteValues(hexLength);

        for (int x=0; x < twoBytePair.size(); x++)
        {                                                                  200
          out.write(((Integer)
            twoBytePair.get(Integer.toString(x))).intValue());
        }

        /*Werden nicht alle Bytes fuer die Dateilaenge verwendet, so muessen
          * sie mit 0 befuellt werden.*/
        for (int x=4-twoBytePair.size(); x > 0; x--)
        {
          out.write(0);
        }                                                                  210

        /*Die Nutzdaten werden in die Datei geschrieben*/
        out.write(xmlElement.toString().getBytes());
        out.flush();
        out.close();

        jTextArea1.append(" - File saved!");

      }
                                                                           220
    }
    catch (Exception excep)
    {
      jTextArea1.setText(excep.getMessage());
    }
}

private void jButton2_actionPerformed(ActionEvent e)
{
                                                                           230
}

/*Prueft ob in dem Textfeld eine MPS ID eingetragen wurde.*/
private void checkValues() throws Exception
{
  if (jTextField1.getText().equalsIgnoreCase(""))
  {
      throw new RuntimeException("Values not filled!");
```

```
    }
}                                                                    240

/*Erzeugt ein neues XML Child ohne Inhalt - ruft setNewXMLChild
 * mit einem leeren Zeichenstring auf.*/
private XMLElement setNewXMLChild(XMLElement aElement, String name)
{
    aElement = setNewXMLChild(aElement, name, "");
    return aElement;
}

/*Erzeugt ein neues XML Child mit Inhalt.*/                          250
private XMLElement setNewXMLChild(XMLElement aElement, String name,
                                  String value)
{
    aElement.addChild(new XMLElement());

    for (Enumeration enumChild = aElement.enumerateChildren();
         enumChild.hasMoreElements() ;)
    {
        XMLElement childChild = (XMLElement) enumChild.nextElement();
        if (childChild.getName() == null)                            260
        {
            childChild.setName(name);
            childChild.setContent(value);
        }
    }
    return aElement;
}

private Hashtable calculateByteValues(Integer hexValue)
{                                                                    270
    /*Die Datenlaenge muss in einem bestimmten Format am Dateianfang
     * stehen. Der Hexwert der Datenlaenge wird in zwei-Zeichen lange
     * TeilStrings geteilt und anschlieszend wird der Integerwert
     * des Hexwertes berechnet.*/

    int hexValueLength = hexValue.toString().length();
    Hashtable twoBytePair = new Hashtable();

    int i = 0;
                                                                     280
    Integer mathValue;
    Integer secValue;

    while(hexValueLength > 0)
    {

        if (hexValueLength > 1)
```

```
        {
          mathValue = new Integer(hexValue.toString().substring(hexValueLength-2,
                                  hexValueLength-1));
          secValue = new Integer(hexValue.toString().substring(hexValueLength-1,
                                  hexValueLength));

          twoBytePair.put(Integer.toString(i), new Integer(mathValue.intValue()*
                                                           16+secValue.intValue()));
        }
        else
        {
          twoBytePair.put(Integer.toString(i), new Integer
            (hexValue.toString().substring(hexValueLength-1, hexValueLength)));
        }
        i++;
        hexValueLength = hexValueLength-2;
      }

      return twoBytePair;
    }
}
```

```
package clearing;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.PrintWriter;
import java.io.IOException;
import java.net.*;

import org.bouncycastle.util.encoders.Base64;
import java.util.Enumeration;

import nanoxml.*;

/*Dieses Servlet ueberprueft die in einem XML Dokument uebergebene
 * digitale Signatur und gibt entweder ein OK oder eine DENIED zurueck.*/
public class VerfiyServlet extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html; charset=windows-1252";
  private String xmlDocument;
  private BCUtil bcutil = new BCUtil();
  private XMLElement xmlElement = new XMLElement();

  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
  }
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
{                                                                          30
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  xmlDocument = request.getParameter("xmldoc");
  String aSignature = "";
  String aMessage = "";

  try
  {
    /*Parsen des XML Dokuments.*/
    if (xmlDocument.length() > 0)                                          40
    {
      xmlElement.parseString(xmlDocument);
      System.out.println("XmlElement:" + xmlElement.toString());

      for (Enumeration enum = xmlElement.enumerateChildren();
        enum.hasMoreElements() ;)
      {
        XMLElement child = (XMLElement) enum.nextElement();
        if (child.getName().equalsIgnoreCase("invoice"))
        {                                                                  50
          aMessage = child.toString();
        }
        else if (child.getName().equalsIgnoreCase("mps"))
        {
          for (Enumeration childEnum = child.enumerateChildren();
          childEnum.hasMoreElements() ;)
          {
            XMLElement childChild = (XMLElement) childEnum.nextElement();
            if (childChild.getName().equalsIgnoreCase("mpsSignature"))
            {                                                              60
              aSignature = childChild.getContent();
            }
          }
        }
      }

      System.out.println(aSignature);
      System.out.println(aMessage);

      /*Ueberpruefen der Signatur.*/                                       70
      bcutil.loadKeyPair();
      if    (bcutil.verifySignature(aSignature, aMessage))
      {
        out.println("OK");
      }
```

```
            else
            {
               out.println("DENIED");
            }
         }
         else
         {
            out.println("DENIED");
         }
      }
      catch (Exception e)
      {
         System.out.println(e.getMessage());
         out.println("DENIED");
      }

      out.close();
   }
}
```

12.2 Java Package pos

```
package pos;

/*Bouncy Castle Klassen*/
import java.security.*;
import java.math.*;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.generators.*;
import org.bouncycastle.crypto.signers.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.util.encoders.*;
import java.io.*;

public class BCUtil
{
   private static RSAKeyParameters pubKey;
   private static RSAPrivateCrtKeyParameters privKey;
   public static int KEY_STRENGTH = 384;
   public static int KEY_CERTAINTY = 4;
   public static String FILE_DIRECTORY = "C:\\keys\\";

   /*Schluesselparameter*/
   private static BigInteger pubExp = new BigInteger("10001", 16);
   private BigInteger mod;
```

```
private BigInteger privExp;
private BigInteger pubExponent;
private BigInteger dp;
private BigInteger dq;
private BigInteger p;                                                    30
private BigInteger q;
private BigInteger qInv;
private FileOutputStream out;

public void BCUtil()
{
}

/*Erstellung neuer Schluesselpaare.*/
public static synchronized void generateKeyPair() throws Exception       40
{
  /*erstellt eine sichere Randomzahl (=Zufallszahl)*/
  SecureRandom secRandom = new SecureRandom();
  RSAKeyGenerationParameters RSAKeyGenPara = new RSAKeyGenerationParameters
                                (pubExp, secRandom, KEY_STRENGTH, KEY_CERTAINTY);
  RSAKeyPairGenerator RSAKeyPairGen = new RSAKeyPairGenerator();
  RSAKeyPairGen.init(RSAKeyGenPara);
  AsymmetricCipherKeyPair keyPair = RSAKeyPairGen.generateKeyPair();

  privKey = (RSAPrivateCrtKeyParameters) keyPair.getPrivate();           50
  pubKey = (RSAKeyParameters) keyPair.getPublic();
}

/*Speichert alle Schluesselparameter.*/
public void saveKeyPairToFile() throws Exception
{
  mod = privKey.getModulus();
  saveToFile("RSAmod.dat", mod);

  privExp = privKey.getExponent();                                       60
  saveToFile("RSAprivExp.dat", privExp);

  pubExponent = privKey.getPublicExponent();
  saveToFile("RSApubExp.dat", pubExponent);

  dp = privKey.getDP();
  saveToFile("RSAdp.dat", dp);

  dq = privKey.getDQ();
  saveToFile("RSAdq.dat", dq);                                           70

  p = privKey.getP();
  saveToFile("RSAp.dat", p);
```

```
    q = privKey.getQ();
    saveToFile("RSAq.dat", q);

    qInv = privKey.getQInv();
    saveToFile("RSAqInv.dat", qInv);
                                                                    80
}

/*Durchfuehrung der Speicherung.*/
private void saveToFile(String fileName, BigInteger fileContent)
throws Exception
{

    String result = new String(Base64.encode(fileContent.toByteArray()));
    out = new FileOutputStream(FILE_DIRECTORY + fileName);
    out.write(result.getBytes());                                   90
    out.flush();
    out.close();
}

/*Laden der Schluesselparameter.*/
public void loadKeyPair() throws Exception
{
    mod = loadFile("RSAmod.dat");
    privExp = loadFile("RSAprivExp.dat");
    pubExponent  = loadFile("RSApubExp.dat");                       100
    p = loadFile("RSAp.dat");
    q = loadFile("RSAq.dat");
    dp = loadFile("RSAdp.dat");
    dq = loadFile("RSAdq.dat");
    qInv = loadFile("RSAqInv.dat");

    /*Initialisieren des privaten und des oeffentlichen Schluessels.*/
    privKey = new RSAPrivateCrtKeyParameters(mod, pubExponent,
                    privExp, p, q, dp, dq, qInv);
    pubKey = new RSAKeyParameters(false, mod, pubExponent);         110
}

/*Durchfuehrung des Ladevorgangs.*/
private BigInteger loadFile(String fileName) throws Exception
{
    File inputFile = new File(FILE_DIRECTORY + fileName);
    InputStream in = new FileInputStream(inputFile);

    byte[] inBuf = new byte[(int)inputFile.length()];
                                                                    120
    int offset = 0;
    int numRead = 0;
    while (offset < inBuf.length
```

```
        && (numRead=in.read(inBuf, offset, inBuf.length-offset)) >= 0) {
      offset += numRead;
  }

  if (offset < inBuf.length) {
      throw new IOException("Could not completely read file "+fileName);
  }
  in.close();

  BigInteger returnValue = new BigInteger(Base64.decode(inBuf));

  return returnValue;
}

/*Eine Signatur fr eine Nachricht erstellen,
 * der Rckgabewert ist ein Base64 verkodierter String.*/
static public String getSignatureAsBase64 (String message) throws Exception
{
  /*Hashwert mit der Grï¿½ von 160 Bit*/
  SHA1Digest digEng = new SHA1Digest();
  RSAEngine rsaEng = new RSAEngine();

  /*initailisieren der Signatur*/
  PSSSigner signer = new PSSSigner(rsaEng, digEng, 16);
  signer.init(true, privKey);

  /*die zu signierende Nachricht einfgen*/
  signer.update(message.getBytes(), 0, (message.getBytes().length));

  /*das Signaturresultat als ByteArray*/
  byte [] sig = signer.generateSignature();

  return new String(Base64.encode(sig));
}

/*Eine signierte Nachricht verifizieren, die Signatur wird als
 * Base64 verkodierter String uebergeben.*/
static public boolean verifySignature (String aSig, String aText)
throws Exception
{
  byte[] aMessage = aText.getBytes();
  byte[] aSignature = Base64.decode(aSig);

  SHA1Digest  sha1digest = new SHA1Digest();

  RSAEngine rsaengine = new RSAEngine();

  PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);
```

```
      psssigner.init(false, pubKey);

      psssigner.update(aMessage, 0, aMessage.length);

      return psssigner.verifySignature(aSignature);
   }
}
```

```
package pos;

import javax.swing.JFrame;
import java.awt.Dimension;
import javax.swing.JLabel;
import java.awt.Rectangle;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;                                          10
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JScrollBar;
import javax.swing.JFileChooser;
import javax.swing.JComboBox;

/*XML Klassen*/
import nanoxml.*;

import java.util.*;                                                  20
import java.text.SimpleDateFormat;
import java.io.*;
import java.net.*;

public class PointOfSaleSystem extends JFrame
{
   private BCUtil bcUtil =     new BCUtil();
   private IrdaUtil irdaUtil = new IrdaUtil();

   private JLabel jLabel1 = new JLabel();                            30
   private JLabel jLabel2 = new JLabel();
   private JLabel jLabel3 = new JLabel();
   private JLabel jLabel4 = new JLabel();
   private JLabel jLabel5 = new JLabel();
   private JTextField jTextField2 = new JTextField();
   private JTextField jTextField3 = new JTextField();
   private JTextField jTextField4 = new JTextField();
   private JButton jButton1 = new JButton();
   private JButton jButton2 = new JButton();
   private JButton jButton3 = new JButton();                         40
```

```
private JTextArea jTextArea1 = new JTextArea();

/*Directory Pfad fuer die Keyparameter.*/
public static String FILE_DIRECTORY = "C:\\keys\\";

/*Beispiel fuer ein leeres XML-Dokument eines Kassabeleges.*/
private String emptyXmlDocument = ""
                + "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n"
                + "<!DOCTYPE mpsDocument SYSTEM \"invoice.dtd\">\n"
                + "<mpsDocument>\n"                                    50
                +  "<invoice>\n"
                +    "<posId></posId>\n"
                +    "<currency></currency>\n"
                +    "<amount></amount>\n"
                +    "<invoiceDate></invoiceDate>\n"
                +    "<invoiceNumber></invoiceNumber>\n"
                +  "</invoice>\n"
                +  "<pos>\n"
                +    "<posSignature></posSignature>\n"
                +  "</pos>\n"                                          60
                +  "<mps>\n"
                +    "<mpsId></mpsId>\n"
                +    "<mpsSignature></mpsSignature>\n"
                +  "</mps>\n"
                +  "<clearingHouse>\n"
                +    "<signatureVerified></signatureVerified>\n"
                +  "</clearingHouse>\n"
                + "</mpsDocument>";

private String xmlDocument = "";                                       70
private XMLElement xmlElement = new XMLElement();

/*Datumsformat fuer den Kassabeleg.*/
private SimpleDateFormat dateFormater = new SimpleDateFormat
        ("dd.MM.yyyy:HH.mm");
private JScrollPane jScrollPane;
private JLabel jLabel6 = new JLabel();
private JLabel jLabel7 = new JLabel();
private JTextField jTextField1 = new JTextField();
private JLabel jLabel8 = new JLabel();                                 80
private JLabel jLabel9 = new JLabel();
private JComboBox jComboBox1 = new JComboBox();
private JButton jButton4 = new JButton();

public static void main(String[] args)
{
  PointOfSaleSystem pos = new PointOfSaleSystem();
  pos.setVisible(true);
  pos.repaint();
```

```
}

public PointOfSaleSystem()
{
  try
  {
    jbInit();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }

}

/*Grafische Oberflaeche wird gestartet.*/
private void jbInit() throws Exception
{
  this.getContentPane().setLayout(null);
  this.setSize(new Dimension(406, 426));
  this.setTitle("PointOfSaleSystem");

  jLabel1.setText("POS Id: ");
  jLabel1.setBounds(new Rectangle(20, 10, 195, 20));

  jLabel2.setText("Invoice Amount: ");
  jLabel2.setBounds(new Rectangle(20, 30, 165, 20));
  jLabel3.setText("Invoice Date: ");
  jLabel3.setBounds(new Rectangle(20, 50, 190, 20));
  jLabel4.setText("Invoice Number: ");
  jLabel4.setBounds(new Rectangle(20, 70, 195, 20));

  jLabel5.setBounds(new Rectangle(250, 10, 100, 20));
  jLabel5.setText("1234");

  jTextField2.setText("");
  jTextField2.setBounds(new Rectangle(250, 30, 75, 20));
  jTextField2.setHorizontalAlignment(JTextField.RIGHT);

  jTextField3.setText(dateFormater.format(new Date()));
  jTextField3.setEditable(false);
  jTextField3.setBounds(new Rectangle(250, 50, 100, 20));

  jTextField4.setText("");
  jTextField4.setBounds(new Rectangle(250, 70, 100, 20));

  jButton1.setText("Generate XML-Invoice");
  jButton1.setBounds(new Rectangle(20, 105, 200, 15));
  jButton1.addActionListener(new ActionListener()
```

```
  {
    public void actionPerformed(ActionEvent e)
    {
      jButton1_actionPerformed(e);
    }
  });

jButton2.setText("Subscribe Invoice");
jButton2.setBounds(new Rectangle(20, 125, 200, 15));
jButton2.addActionListener(new ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jButton2_actionPerformed(e);
    }
  });

jButton3.setText("Save XML File");
jButton3.setBounds(new Rectangle(20, 145, 200, 15));

jTextArea1.setText(emptyXmlDocument);
jTextArea1.setEditable(false);
jTextArea1.setLineWrap(true);

jScrollPane = new JScrollPane(jTextArea1);
jButton3.addActionListener(new ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      jButton3_actionPerformed(e);
    }
  });
  jScrollPane.setVerticalScrollBarPolicy(
      JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
  jScrollPane.setPreferredSize(new Dimension(70, 70));
jScrollPane.setBounds(new Rectangle(20, 200, 360, 180));

jLabel6.setText("EUR");
jLabel6.setBounds(new Rectangle(215, 30, 30, 20));

jLabel7.setText(".");
jLabel7.setBounds(new Rectangle(325, 30, 5, 20));

jTextField1.setText("");
jTextField1.setBounds(new Rectangle(330, 30, 20, 20));

jLabel8.setText("Debug Information");
jLabel8.setBounds(new Rectangle(20, 180, 225, 20));
```

```
  jButton4.setText("Read XML-File and Verify");
  jButton4.setBounds(new Rectangle(20, 165, 200, 15));
  jButton4.addActionListener(new ActionListener()                    190
    {
      public void actionPerformed(ActionEvent e)
      {
        jButton4_actionPerformed(e);
      }
    });

  this.getContentPane().add(jButton4, null);
  this.getContentPane().add(jLabel8, null);
  this.getContentPane().add(jTextField1, null);                      200
  this.getContentPane().add(jLabel7, null);
  this.getContentPane().add(jLabel6, null);
  this.getContentPane().add(jScrollPane, null);
  this.getContentPane().add(jButton3, null);
  this.getContentPane().add(jButton2, null);

  this.getContentPane().add(jButton1, null);
  this.getContentPane().add(jTextField4, null);
  this.getContentPane().add(jTextField3, null);
  this.getContentPane().add(jTextField2, null);                      210
  this.getContentPane().add(jLabel5, null);
  this.getContentPane().add(jLabel4, null);
  this.getContentPane().add(jLabel3, null);
  this.getContentPane().add(jLabel2, null);
  this.getContentPane().add(jLabel1, null);
}

/*Erzeugt mit den eingegebenen Daten ein XML Dokument.*/
private void jButton1_actionPerformed(ActionEvent e)
{                                                                    220
  try
  {
    checkValues();
    jTextArea1.setText("Generate XML-Invoice");

    /*Der Kassabeleg wird als XML-Document erstellt.*/
    xmlElement = new XMLElement();

    xmlElement.setName("mpsDocument");
    xmlElement.addChild(new XMLElement());                           230


    for (Enumeration enum = xmlElement.enumerateChildren();
          enum.hasMoreElements() ;)
    {
      XMLElement child = (XMLElement) enum.nextElement();
```

```
        child.setName("invoice");

        /*Es werden die Child Elemente gessetzt.*/
        setNewXMLChild(child, "posId", jLabel5.getText());
        setNewXMLChild(child, "currency", jLabel6.getText());
        setNewXMLChild(child, "amount", jTextField2.getText()+
                               jLabel7.getText()+jTextField1.getText());
        setNewXMLChild(child, "invoiceDate", jTextField3.getText());
        setNewXMLChild(child, "invoiceNumber", jTextField4.getText());
      }

      jTextArea1.setText(xmlElement.toString());

    }
    catch (Exception excep)
    {
      jTextArea1.setText(excep.getMessage());
    }

}

/*Signiert den gesamten Inhalt des invoice Tags inklusive des Tags
 * selbst.*/
private void jButton2_actionPerformed(ActionEvent e)
{
    try
    {
      checkValues();
      jTextArea1.setText("Subscribe Invoice");
      jButton1_actionPerformed(e);

      /*Der invoice Teil des XML Documents wird mit dem
       * privaten Schluessel des POS-Systems signiert und
       * anschlieszend die Signatur in das XML-Document
       * eingebunden.*/
      xmlElement = setNewXMLChild(xmlElement, "pos");

      String invoiceSig = "";

      /*Laden des privaten POS-Schluessels.*/
      bcUtil.FILE_DIRECTORY = "C:\\keys\\";
      bcUtil.loadKeyPair();

      /*Digitale Signature fuer die Rechnung erzeugen und in das
       * XML Dokument einsetzen.*/
      for (Enumeration enum = xmlElement.enumerateChildren();
           enum.hasMoreElements() ;)
      {
        XMLElement child = (XMLElement) enum.nextElement();
```

```
        if (child.getName().equalsIgnoreCase("invoice"))
        {
          invoiceSig = bcUtil.getSignatureAsBase64(child.toString());
          /*Prufung der erstellten Signatur.*/
          /*if (bcUtil.verifySignature(invoiceSig, child.toString()))
          {
            System.out.println("Invoice could be verified!");
          }
          else
          {
            System.out.println("Invoice could NOT be verified!");
          }*/
        }
      }

      /*Signatur einfuegen.*/
      for (Enumeration enum2 = xmlElement.enumerateChildren();
            enum2.hasMoreElements() ;)
      {
        XMLElement child = (XMLElement) enum2.nextElement();

        if (child.getName().equalsIgnoreCase("pos"))
        {
          setNewXMLChild(child, "posSignature", invoiceSig);
        }
      }

      jTextArea1.setText(xmlElement.toString());
    }
    catch (Exception excep)
    {
      jTextArea1.setText(excep.getMessage());
    }
  }

  private void jButton3_actionPerformed(ActionEvent e)
  {
    /*Zuerst muessen die Aktionen des zweiten Buttons ausgefuehrt
     * werden.*/
    jButton2_actionPerformed(e);

    try
    {
      /*Speicherung des erstellten und signierten XMLDocuments zur
       * anschlieszenden Versendung per Infrarotschnittstelle.*/

      jTextArea1.setText("save xml file to ");
      FileOutputStream out = new FileOutputStream
                                (FILE_DIRECTORY + "signedBill.jd");
```

```
      /*Die Laenge der XML - Daten werden als Hex-Wert berechnet.*/
      Integer hexLength = new Integer
            (Integer.toHexString(xmlElement.toString().length()));
      System.out.println(hexLength);

      /*Um Dateien fuer den J2ME Client lesbar zu machen ist
      * es notwendig die Datenlaenge in die ersten vier Bytes
      * der uebertragenen Datei zu schreiben.*/
      Hashtable twoBytePair = calculateByteValues(hexLength);

      for (int x=0; x < twoBytePair.size(); x++)
      {
        out.write(((Integer) twoBytePair.get(Integer.toString(x))).intValue());
      }

      /*Werden nicht alle Bytes fuer die Dateilaenge verwendet, so muessen
        * sie mit 0 befuellt werden.*/
      for (int x=4-twoBytePair.size(); x > 0; x--)
      {
        out.write(0);
      }

      /*Die Nutzdaten werden in die Datei geschrieben*/
      out.write(xmlElement.toString().getBytes());
      out.flush();
      out.close();

      jTextArea1.append(" - file saved ");

    }
    catch(Exception excep)
    {
      jTextArea1.setText(excep.getMessage());
    }
}

private void jButton4_actionPerformed(ActionEvent e)
{
  /*Das vom mobilen Client signierte Dokument wird von
    * der Verrechnungsstelle verifiziert und eine Bestaetigung
    * zurueckgesandt*/

  /*Das zu verifizierende File auswaehlen.*/
  JFileChooser fc = new JFileChooser();
  fc.setDialogTitle("Open File");

  /*Auswahl zeigt nur Dateien keine Ordner*/
  fc.setFileSelectionMode( JFileChooser.FILES_ONLY);
```

```
/*Starte in dieser Directory.*/
fc.setCurrentDirectory(new File(FILE_DIRECTORY+"."));

/*Oeffne das Fenser.*/
int result = fc.showOpenDialog(this);
                                                                    390
if( result == JFileChooser.CANCEL_OPTION)
{
  jTextArea1.setText("No File selected!");
}
else if( result == JFileChooser.APPROVE_OPTION)
{
  /*Die ausgewaehlte Datei wird zum lesen uebergeben.*/
  File file = fc.getSelectedFile();

  try                                                               400
  {
    String fileString = readFile(file);

    if( fileString != null)
    {
      jTextArea1.setText(fileString);
      /*Starte Verifierung des XML-Documents.*/

      try
      {                                                             410
        /*Sende das XML Dokument an ein Servlet zur Verifizierung.*/
        URL u = new URL("http://localhost:8988/j2me-ClearingHouse-context-"
          +"root/servlet/VerfiyServlet?xmldoc="
          +URLEncoder.encode(fileString.toString()));
        InputStream in = u.openStream();
        InputStreamReader isr = new InputStreamReader(in);
        BufferedReader br = new BufferedReader(isr);
        String theLine;

        /*Ausgabe der Verifizierung.*/                            420
        jTextArea1.setText("Verification Result: ");

        while ((theLine = br.readLine()) != null)
        {
          jTextArea1.append(theLine);
        }
      }
      catch (Exception excep)
      {
        jTextArea1.setText(excep.getMessage());                     430
      }
```

```
        }
        else
        {
          jTextArea1.setText("File is empty!");
        }
      }
      catch (Exception excep)
      {                                                                    440
        jTextArea1.setText(excep.getMessage());
      }
    }
}

/*Erzeugt ein neues XML Child ohne Inhalt - ruft setNewXMLChild
 * mit einem leeren Zeichenstring auf.*/
private XMLElement setNewXMLChild(XMLElement aElement, String name)
{
  aElement = setNewXMLChild(aElement, name, "");                       450
  return aElement;
}

/*Erzeugt ein neues XML Child mit Inhalt.*/
private XMLElement setNewXMLChild(XMLElement aElement, String name,
                                  String value)
{
  aElement.addChild(new XMLElement());

  for (Enumeration enumChild = aElement.enumerateChildren();           460
       enumChild.hasMoreElements() ;)
  {
    XMLElement childChild = (XMLElement) enumChild.nextElement();
    if (childChild.getName() == null)
    {
      childChild.setName(name);
      childChild.setContent(value);
    }
  }
  return aElement;                                                     470
}

/*Prueft ob die Textfelder ausgefuellt wurden.*/
private void checkValues() throws Exception
{
  if (jTextField2.getText().equalsIgnoreCase("") |
      jTextField4.getText().equalsIgnoreCase(""))
  {
     throw new RuntimeException("Values not filled!");
  }                                                                    480
  else
```

```
    {
      if (jTextField1.getText().equalsIgnoreCase(""))
      {
        jTextField1.setText("00");
      }
    }
}

private Hashtable calculateByteValues(Integer hexValue)                    490
{
  /*Die Datenlaenge muss in einem bestimmten Format am Dateianfang
    * stehen. Der Hexwert der Datenlaenge wird in zwei-Zeichen lange
    * TeilStrings geteilt und anschlieszend wird der Integerwert
    * des Hexwertes berechnet.*/

  int hexValueLength = hexValue.toString().length();
  Hashtable twoBytePair = new Hashtable();

  int i = 0;                                                               500

  Integer mathValue;
  Integer secValue;

  while(hexValueLength > 0)
  {
    if (hexValueLength > 1)
    {
      mathValue = new Integer(hexValue.toString().substring(hexValueLength-2,
                                  hexValueLength-1));                      510
      secValue = new Integer(hexValue.toString().substring(hexValueLength-1,
                                  hexValueLength));

      twoBytePair.put(Integer.toString(i), new Integer(mathValue.intValue()*
                                                       16+secValue.intValue()));
    }
    else
    {
      twoBytePair.put(Integer.toString(i), new Integer
        (hexValue.toString().substring(hexValueLength-1, hexValueLength)));    520
    }
    i++;
    hexValueLength = hexValueLength-2;
  }

  return twoBytePair;
}

/*Liest das vom mobilen Client signierte Document wieder ein.*/
public String readFile(File file) throws Exception                         530
```

```
  {
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();

    FileReader fReader = new FileReader(file);

    int i;
    int x = 0;

    /*Schreibe den Inhalt der Datei in einen ByteArrayOutputStream.*/
    while ((i = fReader.read()) != -1)                                    540
    {
      /*Die ersten drei Bytes beinhalten Angaben zur Laenge der
        *Datei und sollen nicht in den ByteArrayOutputStream geschrieben
        * werden. */
      if (x > 3)
      {
        outStream.write(i);
      }
      x++;
    }                                                                     550

    return new String(outStream.toByteArray());
  }
}
```

12.3 Java Package mps

```
MIDlet-1: MPSInit, , mps.actor.MPSInit
MIDlet-2: Payment, , mps.actor.Payment
MIDlet-3: ShowTransactions, , mps.actor.ShowTransactions
MIDlet-Description: Mobile Payment Solution
MIDlet-Jar-Size: 48
MIDlet-Jar-URL: MPS.jar
MIDlet-Name: Mobile Payment Solution
MIDlet-Vendor: Klaus Brosche
MIDlet-Version: 0.1
```

```
package mps.actor;

import com.siemens.mp.io.*;
import java.lang.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import java.util.Hashtable;
```

```
                                                                    10
import mps.helper.*;

/*Um das Mobile Payment System zu aktivieren werden die
 * Parameter fuer den privaten Schluessel, sowie die Parameter
 * des oeffentlichen POS-Schluessels benoetigt. Diese Klasse
 * kuemmert sich um den notwendigen Datenverkehr und um die
 * Speicherung dieser Daten.*/
public class MPSInit extends MIDlet
  implements CommandListener
{                                                                   20
  private Command exitCommand = new Command("Exit", Command.SCREEN, 1);
  private Display display;
  private Form frm;

  public MPSInit()
  {
    display = Display.getDisplay(this);
  }

  public void startApp()                                            30
  {
    String receivedXMLDoc = "";
    frm = new Form("MPSInit");
    StringItem si = new StringItem("Waiting for XML Document..", "");
    frm.append(si);
    frm.addCommand(exitCommand);
    frm.setCommandListener(this);
    display.setCurrent(frm);

    try                                                             40
    {
      boolean notReceived = true;

      /*Initialisieren der Infrarotschnittstelle.*/
      Communication aIrdaListener = new Communication(frm, display);

      /*Warteschleife bis Daten per Irda empfangen werden.*/
      while(notReceived)
      {
        try                                                         50
        {
          receivedXMLDoc = aIrdaListener.getData();

          /*Wenn die erhaltenen Daten den String keyParams enthalten
           * wird die Schleife beendet.*/
          if (receivedXMLDoc.length() > 0 &&
            receivedXMLDoc.indexOf("keyParams") > 0)
          {
```

```
                notReceived = false;
              }
            }
            catch(Exception e)
            {
            }
        }
        handleXMLDocument(receivedXMLDoc);
    }
    catch (Exception e)
    {
        displayAlert(e.getMessage());
    }
}

public void pauseApp()
{
}

public void destroyApp(boolean unconditional)
{
}

public void commandAction(Command c, Displayable s)
{
    if (c == exitCommand)
    {
        destroyApp(false);
        notifyDestroyed();
    }
}

/*Anzeige von Fehlermeldungen.*/
public void displayAlert (String mess)
{
    Alert a = new Alert("Error: ", mess, null, AlertType.ERROR);
    a.setTimeout(Alert.FOREVER);
    display.setCurrent(a);
}

/*Parst und speichert die uebermittelten Daten.*/
private void handleXMLDocument(String receivedXMLDoc)
{
    XMLParser xmlParser = new XMLParser(receivedXMLDoc);
    try
    {

        StringItem si = new StringItem("Start parsing XML Document..", "");
        frm.append(si);
```

```
        display.setCurrent(frm);

        /*Das per Irda Schnittstelle erhaltene XML Document wird                    110
          * geparst und anschlieszend gespeichert.*/
        Storage storage = new Storage();
        Hashtable hashtable = new Hashtable();

        hashtable.put("mpsId", new String(xmlParser.getContent("mpsId")));
        hashtable.put("mod", new String(xmlParser.getContent("mod")));
        hashtable.put("pubExponent", new String(
                                        xmlParser.getContent("pubExponent")));
        hashtable.put("privExp", new String(xmlParser.getContent("privExp")));
        hashtable.put("p", new String(xmlParser.getContent("p")));                120
        hashtable.put("q", new String(xmlParser.getContent("q")));
        hashtable.put("dp", new String(xmlParser.getContent("dp")));
        hashtable.put("dq", new String(xmlParser.getContent("dq")));
        hashtable.put("qInv", new String(xmlParser.getContent("qInv")));

        si = new StringItem("MPS params stored. .", "");
        frm.append(si);
        display.setCurrent(frm);

        storage = new Storage();                                                   130
        storage.openStore(storage.mpsStoreName);
        storage.saveMPSParams(hashtable);

        hashtable = new Hashtable();

        hashtable.put("posMod", new String(xmlParser.getContent("posMod")));
        hashtable.put("posPubExponent",
                    new String(xmlParser.getContent("posPubExponent")));

        storage.openStore(storage.posStoreName);                                   140
        storage.savePOSParams(hashtable);

        si = new StringItem("All Params stored you can exit. .", "");
        frm.append(si);
        display.setCurrent(frm);
      }
      catch (Exception e)
      {
        displayAlert(e.getMessage());
      }                                                                            150
    }
}
```

```
package mps.actor;
```

```
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import java.util.Hashtable;

import org.bouncycastle.util.encoders.Base64;
                                                                    10
import mps.helper.*;

/*Diese Klasse behandelt die eigentliche Zahlungstransaktion.*/
public class Payment extends MIDlet
  implements CommandListener
{
  private Command exitCommand = new Command("Exit", Command.SCREEN, 1);
  private Command signCommand = new Command("Sign", Command.SCREEN, 2);
  private Display display;
  private XMLParser xmlParser;                                      20
  private Crypto crypto;
  private Storage storage;
  private Communication aIrdaListener;
  private Form frm;
  private StringItem si;

  public Payment()
  {
    display = Display.getDisplay(this);
  }                                                                 30

  public void startApp()
  {
    String receivedXMLDoc = "";

    frm = new Form("Payment System");
    si = new StringItem("Waiting for new Bill...", "");
    frm.append(si);
    frm.addCommand(exitCommand);
    frm.setCommandListener(this);                                   40
    display.setCurrent(frm);

    try
    {
      aIrdaListener = new Communication(frm, display);

      boolean notReceived = true;

      /*Warteschleife bis Daten per Irda empfangen werden.*/
      while(notReceived)                                            50
      {
```

```
  try
  {
    receivedXMLDoc = aIrdaListener.getData();

    /*Wenn die erhaltenen Daten den String mpsDocument enthalten
     * wird die Schleife beendet.*/
    if (receivedXMLDoc.length() > 0 &&
      receivedXMLDoc.indexOf("mpsDocument") > 0)
    {                                                              60
      notReceived = false;
    }
  }
  catch(Exception e)
  {
  }
}

si = new StringItem("Start parsing XML Document..", "");
frm.append(si);                                                    70
display.setCurrent(frm);

/*Das per Irda Schnittstelle erhaltene XML Document
 * wird geparst. Die vom POS - System erstellte
 * Unterschrift fuer den Invoice Teil des Documents
 * wird uberprueft. Bei positiver Ueberpruefung signiert
 * der Benutzer ebenfalls den      Invoice Teil des Documents
 * und sendet ihn zurueck an das POS - System. Eine
 * bestaetigte Rechung wird zur spaeteren Anzeige gespeichert.*/
xmlParser = new XMLParser(receivedXMLDoc);                         80
crypto = new Crypto();

storage = new Storage();

StringItem si = new StringItem("Loading POS keys..", "");
frm.append(si);
display.setCurrent(frm);

storage.openStore(storage.posStoreName);
                                                                   90
/*Die Schluesselparameter werden geladen.*/
Hashtable hashtable = storage.readPOSParams();

si = new StringItem("Try to verify Signature..", "");
frm.append(si);
display.setCurrent(frm);

/*Die uebermittelte Signatur fuer die Daten der Rechnung
 * wird geprueft.*/
byte [] aSignature = Base64.decode(xmlParser.getContent("posSignature"));   100
```

```
      /*Anzeige ueber die Verifizierung und Setzen eines neuen
        * Buttons fuer den Benutzer. Leider kann das SL45i die Signatur
        * weder pruefen noch erstellen, daher wird in jedem Fall ein
        * Button zum Signieren gesetzt.*/
      if(crypto.verifySignedMessage(hashtable, xmlParser.getElement("invoice"),
                                    aSignature))
      {
        si = new StringItem("Signature verified.", "");
        frm.append(si);                                                    110
        frm.addCommand(signCommand);
        frm.setCommandListener(this);
        display.setCurrent(frm);
      }
      else
      {
        si = new StringItem("Signature could not be verified.", "");
        frm.append(si);
        frm.addCommand(signCommand);
        frm.setCommandListener(this);                                      120
        display.setCurrent(frm);
      }
    }
    catch (Exception e)
    {
      displayAlert(e.getMessage());
    }
}

public void pauseApp()                                                     130
{
}

public void destroyApp(boolean unconditional)
{
}

public void commandAction(Command c, Displayable s)
{
  if (c == exitCommand)                                                    140
  {
    destroyApp(false);
    notifyDestroyed();
  }
  else if (c == signCommand)
  {
    signBill();
  }
}
```

```
                                                                        150
/*Signieren der Rechnung und zuruecksenden an das POS - System.*/
public void signBill()
{
  try
  {
    si = new StringItem("Loading MPS keys. . .", "");
    frm.append(si);
    display.setCurrent(frm);

    storage.openStore(storage.mpsStoreName);                            160

    /*Laden der privaten Schluesselparameter.*/
    Hashtable hashtable = storage.readMPSParams();

    si = new StringItem("Signing Bill. . .", "");
    frm.append(si);
    display.setCurrent(frm);

    String aText =    xmlParser.getElement("invoice");
                                                                        170
    /*Erstellen der Signatur.*/
    byte [] aSignature = crypto.signMessagePrivKey(hashtable, aText);

    String signature = new String(Base64.encode(aSignature));

    /*Einfuegen der erstellten Signatur in das XML Dokument.*/
    xmlParser.setNewElement("mpsDocument", "mps", "");
    String mpsId = (String)hashtable.get("mpsId");
    xmlParser.setNewElement("mpsDocument", "mpsId", mpsId);
    String signedXMLDoc = xmlParser.setNewElement("mps",                180
                                  "mpsSignature", signature);

    /*Pruefung der selbst erstellten Signatur.*/
    /*si = new StringItem("Try to verify own signature...", "");
    frm.append(si);
    display.setCurrent(frm);

    if(crypto.verifyOwnSignedMessage(hashtable,
      aText, aSignature))
    {                                                                   190
      si = new StringItem("could verify.", "");
      frm.append(si);
      display.setCurrent(frm);
    }
    else
    {
      si = new StringItem("could not verify", "");
      frm.append(si);
```

```
        display.setCurrent(frm);
      }*/                                                                    200

      si = new StringItem("Sending Bill...", "");
      frm.append(si);
      display.setCurrent(frm);

      /*Versenden des XML Dokuments.*/
      aIrdaListener.send(signedXMLDoc);

      si = new StringItem("Storing Transaction...", "");
      frm.append(si);                                                        210
      display.setCurrent(frm);

      /*Speicherung der Transaktion.*/
      storage.saveTransaction(" Datum: " + xmlParser.getContent("invoiceDate") +
                              " Betrag: " + xmlParser.getContent("amount"));

      si = new StringItem("Transaction stored - go to exit", "");
      frm.append(si);
      display.setCurrent(frm);
    }                                                                        220
    catch (Exception e)
    {
      displayAlert(e.getMessage());
    }
  }

  /*Anzeige von Fehlermeldungen.*/
  public void displayAlert (String mess)
  {
    Alert a = new Alert("Error: ", mess, null, AlertType.ERROR);             230
    a.setTimeout(Alert.FOREVER);
    display.setCurrent(a);
  }
}
```

```
package mps.actor;

import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import java.util.Hashtable;

import mps.helper.*;
                                                                             10
/*Diese Klasse ermoeglicht es eine Liste der bereits
```

```
 * bezahlten Rechnungen anzusehen.*/
public class ShowTransactions      extends MIDlet
  implements CommandListener
{
  private Command exitCommand = new Command("Exit", Command.SCREEN, 1);
  private Display display;
  private XMLParser xmlParser;
  private Storage storage;
                                                                          20
  public ShowTransactions()
  {
    display = Display.getDisplay(this);
  }

  public void startApp()
  {
    Form frm = new Form("Transactions:");
    frm.addCommand(exitCommand);
    frm.setCommandListener(this);                                         30
    display.setCurrent(frm);

    Storage storage = new Storage();

    StringItem si;
    String transactions = "";
    try
    {
      /*Laden der Transaktionen.*/
      transactions = storage.loadTransactions();                          40
    }
    catch (Exception e)
    {
      displayAlert(e.getMessage());
    }

    si = new StringItem(transactions, "");
    frm.append(si);
    display.setCurrent(frm);
                                                                          50
  }

  public void pauseApp()
  {
  }

  public void destroyApp(boolean unconditional)
  {
  }
                                                                          60
```

```
  public void commandAction(Command c, Displayable s)
  {
    if (c == exitCommand)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }

  /*Anzeige von Fehlermeldungen.*/                                    70
  public void displayAlert (String mess)
  {
    Alert a = new Alert("Error: ", mess, null, AlertType.ERROR);
    a.setTimeout(Alert.FOREVER);
    display.setCurrent(a);
  }
}
```

```
package mps.helper;

import com.siemens.mp.io.*;
import java.lang.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

import mps.actor.MPSInit;
                                                                    10
/*Diese Klasse repraesentiert die Kommunikationsschnittstelle zum
 * Point of Sale System. Die Implementierung umfasst nur die Irda
 * Schnittstelle. Die Irda Anbindung erfolgt ueber die properitaere
 * Klasse von Siemens und ist daher nur auf einem Siemens
 * Mobiltelefon verfuegbar.*/
public class Communication implements com.siemens.mp.io.ConnectionListener
{
  private com.siemens.mp.io.Connection connection;
  private Form afrm;
  private Display display;                                          20
  private String xmlDocument;

  public Communication(Form afrm, Display display)
  {
    this.afrm = afrm;
    this.display = display;

    /*Oeffnet die Irda-Schnittstelle.*/
    connection = new com.siemens.mp.io.Connection("IRDA:");
                                                                    30
```

```
    /*Setzt sich selbst als Listener auf den Irda-Port.*/
    connection.setListener(this);
  }

  /*Ermoeglicht das senden von Nachrichten per Irda.*/
  public void send(String mess)
  {
    try
    {
      connection.send(mess.getBytes());                                    40
    }
    catch ( Exception e )
    {
      displayAlert(e.getMessage());
    }
  }

  /*Diese Methode empfaengt die uebermittelten Daten.*/
  public void receiveData(byte[] data)
  {                                                                        50
    xmlDocument = new String(data);

    if (xmlDocument.length() > 0)
    {
      StringItem si = new StringItem("XML Document received.", "");
      afrm.append(si);
      display.setCurrent(afrm);
    }

  }                                                                        60

  /*Anzeige von Fehlermeldungen.*/
  public void displayAlert (String mess)
  {
    Alert a = new Alert("Error: ", mess, null, AlertType.ERROR);
    a.setTimeout(Alert.FOREVER);
    display.setCurrent(a);
  }

  /*Liefert den Inhalt der empfangenen Daten zurueck.*/                    70
  public String getData()
  {
    return xmlDocument;
  }
}
```

```
package mps.helper;
```

```
/*Bouncy Castle Klassen*/
import java.math.BigInteger;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.generators.*;
import org.bouncycastle.crypto.signers.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.RSAEngine;                    10
import org.bouncycastle.util.encoders.Base64;

import java.util.Hashtable;

/*Diese Klasse bearbeitet alle cryptografischen
 * Anforderungen des MPS mit Hilfe der Bouncy Castle API.*/
public class Crypto
{
   private static RSAKeyParameters pubKey;
   private static RSAPrivateCrtKeyParameters privKey;                20

   private BigInteger mod;
   private BigInteger privExp;
   private BigInteger pubExponent;
   private BigInteger dp;
   private BigInteger dq;
   private BigInteger p;
   private BigInteger q;
   private BigInteger qInv;
                                                                     30
   public Crypto()
   {
   }


   /*Liest aus einem Hashtable die Parameter zur Aktivierung des
    * privaten MPS-Schluessels und erzeugt die digitale Signatur
    * fuer die uebergebene Nachricht.*/
   public byte[] signMessagePrivKey(Hashtable hashtable, String aText) throws Exception
   {                                                                 40
      mod = new BigInteger(Base64.decode((String)hashtable.get("mod")));
      privExp = new BigInteger(Base64.decode((String)hashtable.get("privExp")));
      pubExponent = new BigInteger(Base64.decode((String)
                                          hashtable.get("pubExponent")));
      dp = new BigInteger(Base64.decode((String)hashtable.get("dp")));
      dq = new BigInteger(Base64.decode((String)hashtable.get("dq")));
      p = new BigInteger(Base64.decode((String)hashtable.get("p")));
      q = new BigInteger(Base64.decode((String)hashtable.get("q")));
      qInv = new BigInteger(Base64.decode((String)hashtable.get("qInv")));
                                                                     50
      /*Erstellen des privaten Schluessels.*/
```

```
    privKey = new RSAPrivateCrtKeyParameters(mod, pubExponent,
                    privExp, p, q, dp, dq, qInv);

    /*Signieren der uebergebenen Nachricht.*/
    byte[] aSignature = signMessage(aText.getBytes());

    return aSignature;
  }

  /*Initialisiert den eigenen oeffentlichen Schluessel mit den uebergebenen
   * Daten des Hashtables und verifiziert die ebenfalls ubergebene
   * Signatur. Diese Methode dient diversen Tests zur Pruefung der
   * erstellten Signatur.*/
  public boolean verifyOwnSignedMessage(Hashtable hashtable, String aText,
                                        byte[] aSignature) throws Exception
  {
    BigInteger posMod = new BigInteger(Base64.decode((String)
                                          hashtable.get("mod")));
    BigInteger posPubExponent = new BigInteger(Base64.decode((String)
                                          hashtable.get("pubExponent")));

    pubKey = new RSAKeyParameters(false, posMod, posPubExponent);

    byte[] aMessage = aText.getBytes();

    SHA1Digest    sha1digest = new SHA1Digest();

    RSAEngine rsaengine = new RSAEngine();

    PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

    psssigner.init(false, pubKey);

    psssigner.update(aMessage, 0, aMessage.length);

    return psssigner.verifySignature(aSignature);
  }

  /*Liest aus einem Hashtable die Parameter zur Aktivierung des
   * oeffentlichen POS-Schluessels und verifiziert die uebergebene
   * Signatur.*/
  public boolean verifySignedMessage(Hashtable hashtable, String aText,
                                     byte[] aSignature) throws Exception
  {
    BigInteger posMod = new BigInteger(Base64.decode((String)
                                          hashtable.get("posMod")));
    BigInteger posPubExponent = new BigInteger(Base64.decode((String)
                                          hashtable.get("posPubExponent")));
```

```
      pubKey = new RSAKeyParameters(false, posMod, posPubExponent);

      byte[] aMessage = aText.getBytes();

      SHA1Digest sha1digest = new SHA1Digest();

      RSAEngine rsaengine = new RSAEngine();

      PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

      psssigner.init(false, pubKey);


      psssigner.update(aMessage, 0, aMessage.length);

      return psssigner.verifySignature(aSignature);
    }

    /*Erzeugt eine RSA Signatur.*/
    private byte[] signMessage(byte[] aMessage)
    throws Exception
    {
      if (privKey == null)
        throw new Exception ("private key not set");

      SHA1Digest sha1digest = new SHA1Digest();

      RSAEngine rsaengine = new RSAEngine();

      PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

      psssigner.init(true, privKey);

      psssigner.update(aMessage, 0, aMessage.length);

      byte [] signrsult = psssigner.generateSignature();

      return signrsult;
    }
}
```

```
package mps.helper;

import javax.microedition.rms.*;
import java.util.Hashtable;
import java.util.Enumeration;
```

```
import com.siemens.mp.io.File;

/*Alle Daten werden durch diese Klasse am mobilen System
 * gespeichert.*/
public class Storage
{
  public static String mpsStoreName = "mpsstore";
  public static String posStoreName = "posstore";
  public static String transStoreName = "transstore";

  private static int mpsId = 1;
  private static int mod = 2;
  private static int pubExponent = 3;
  private static int privExp = 4;
  private static int p = 5;
  private static int q = 6;
  private static int dp = 7;
  private static int dq = 8;
  private static int qInv = 9;
  private static int posMod = 1;
  private static int posPubExponent = 2;

  private RecordStore store;

  public Storage()
  {
  }

  public void openStore(String storeName) throws Exception
  {
    store = RecordStore.openRecordStore(storeName, true);
  }

  /*Liest die Parameter fuer den privaten Schluessel des
   * mobilen Client ein.*/
  public Hashtable readMPSParams() throws Exception
  {
    Hashtable hashtable = new Hashtable();
    hashtable.put("mpsId", new String(store.getRecord(mpsId)));
    hashtable.put("mod", new String(store.getRecord(mod)));
    hashtable.put("pubExponent", new String(store.getRecord(pubExponent)));
    hashtable.put("privExp", new String(store.getRecord(privExp)));
    hashtable.put("p", new String(store.getRecord(p)));
    hashtable.put("q", new String(store.getRecord(q)));
    hashtable.put("dp", new String(store.getRecord(dp)));
    hashtable.put("dq", new String(store.getRecord(dq)));
    hashtable.put("qInv", new String(store.getRecord(qInv)));

    store.closeRecordStore();
```

```
    return hashtable;
}

 /*Speicher die bei der Systeminitialisierung uebergebenen Schluessel
   * Parameter.*/
public void saveMPSParams(Hashtable hashtable) throws Exception
{

    byte[] outBuf = ((String)(hashtable.get("mpsId"))).getBytes();
    int modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("mod"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("pubExponent"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("privExp"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("p"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("q"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("dp"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("dq"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("qInv"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    store.closeRecordStore();
}

/*Liest die Parameter fuer den oeffentlichen Schluessel des
  * POS-Systems*/
public Hashtable readPOSParams() throws Exception
{
    Hashtable hashtable = new Hashtable();
    hashtable.put("posMod", new String(store.getRecord(posMod)));
    hashtable.put("posPubExponent", new String(
                                        store.getRecord(posPubExponent)));

    store.closeRecordStore();
```

```
    return hashtable;
  }

  /*Speicher die bei der Systeminitialisierung uebergebenen Schluessel
    * Parameter des POS-Systems.*/
  public void savePOSParams(Hashtable hashtable) throws Exception
  {

    byte[] outBuf = ((String)(hashtable.get("posMod"))).getBytes();
    int modId = store.addRecord(outBuf, 0, outBuf.length );

    outBuf = ((String)(hashtable.get("posPubExponent"))).getBytes();
    modId = store.addRecord(outBuf, 0, outBuf.length );

    store.closeRecordStore();
  }

  /*Speichert Daten ueber die durchgefuehrten Zahlungstransaktionen.*/
  public void saveTransaction(String xmlInvoice) throws Exception
  {
    File file = new File();
    int fileDescriptor = file.open(this.transStoreName);
    file.seek(fileDescriptor, file.length(fileDescriptor));
    file.write(fileDescriptor, xmlInvoice.getBytes(), 0,
                    xmlInvoice.getBytes().length);
    file.close(fileDescriptor);
  }

  /*Laedt Daten ueber die durchgefuehrten Zahlungstransaktionen.*/
  public String loadTransactions() throws Exception
  {
    File file = new File();
    int fileDescriptor = file.open(this.transStoreName);
    byte[] buf = new byte[file.length(fileDescriptor)];

    file.read(fileDescriptor, buf, 0, file.length(fileDescriptor));
    file.close(fileDescriptor);
    return new String(buf);
  }
}
```

```
package mps.helper;

import java.util.Enumeration;

/*Diese Klasse ermoeglicht, auf einfache Weise, das Parsen
 * von XML-Dokument, sowie das Einfuegen von neuen Elementen
```

```
 * in ein Dokument. Leider konnte nicht, wie vorgesehen die
 * nanoxml API verwendet werden, da diese API die Klasse
 * java.io.StringReader benoetigt, welche unter J2ME nicht
 * unterstuetzt wird.*/
public class XMLParser
{
  private static String xmlDocument;

  public XMLParser(String xmlDocument)
  {
    this.xmlDocument = xmlDocument;
  }

  /*Liefert den Inhalt des uebergebenen XML Tags.*/
  public String getContent(String elementName)
  {
    String startElement = "<" + elementName + ">";
    int start = xmlDocument.indexOf(startElement);
    int end = xmlDocument.indexOf("</" + elementName + ">");

    return xmlDocument.substring(start + startElement.length(), end);
  }

  /*Liefert den Inhalt des uebergebenen XML Tags inklusive des
   * XML Tags.*/
  public String getElement(String elementName)
  {
    String startElement = "<" + elementName + ">";
    int start = xmlDocument.indexOf(startElement);
    int end = xmlDocument.indexOf("</" + elementName + ">");

    return xmlDocument.substring(start, end+startElement.length()+1);
  }

  /*Setzt ein neues Element unter einem bestimmten Element.*/
  public String setNewElement(String treeElement, String newElementName,
                                                    String newValue)
  {
    String startElement = "<" + treeElement + ">";
    int start = xmlDocument.indexOf(startElement);
    String firstPart = xmlDocument.substring(0, start + startElement.length());
    String secondPart = xmlDocument.substring(start + startElement.length());
    xmlDocument = firstPart + "<" + newElementName + ">" + newValue + "</" +
                  newElementName + ">" + secondPart;
    return xmlDocument;
  }
}
```

Literaturverzeichnis

[BouncyCastle, 2002] BouncyCastle. Bouncy castle, 2002. http://www.bouncycastle.org.

[Brothers, 2000] Lehman Brothers, 2000. http://www.lehman.com.

[Cervera, 2002] Anders Cervera. Analysis of j2me for developing mobile payment systems, 2002.

[Consortium, 2004] World Wide Web Consortium. Xml, 2004. http://www.w3c.org/XML.

[derstandard.at, 2004] derstandard.at. tele.ring mischt heimischen mobilfunkmarkt auf, 2004. http://www.standard.at.

[eCoin Incorporated, 2004] eCoin Incorporated. ecoin, 2004. http://www.ecoin.net.

[EMV, 2000] EMVCo (Europy Mastercard and Visa). *EMV Integrated Cricuit Card Specification for Payment Systems*, 4.0 edition, 12 2000.

[Janssen, 2004] Wilhelm Janssen. Internet-lexikon, 2004. http://www.at-mix.de.

[Jarvi, 2004] Keane Jarvi. Rxtx, 2004. http://www.rxtx.org.

[(JCP), 2004] Java Community Process (JCP). Java community process (jcp), 2004. http://jcp.org.

[Ltd, 1999] Durlacher Research Ltd. Mobile commerce report, 1999. http://www.durlacher.com.

[NFC-Forum, 2004] NFC-Forum. Near field communication, 2004. http://www.nfc-forum.org.

[of Standards and (NIST), 1997] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes) development effort, 1997. http://csrc.nist.gov/CryptoToolkit/aes.

[Process, 2002] Java Community Process. Mobile information device profile, v2.0, 2002. http://jcp.org.

[Process, 2003] Java Community Process. Security and trust services for j2me, 10 2003. http://jcp.org.

[Schneider, 1999] Bruce Schneider. Counterpane systems, 1999. http://www.counterpane.com/smar-card-threats.html.

[Schneider, 2002] M Schneider. Rc5-64 project nach 1757 tagen endlich vollendet! *WCM Wiener Computer Markt*, 178, 10 2002.

[Selhorst, 2002] Marcel Selhorst. Die geldkarte, 2002. http://www.crypto.ruhr-uni-bochum.de/Seminare/BeitraegeITS/ Geldkarte_Ausarbeitung.pdf.

[WEBAGENCY, 2002] WEBAGENCY. Definition von mobile commerce, 2002. http://www.webagency.de.

[White and Hemphill, 2002] James P. White and David A. Hemphill. *Java2 Micro Edition*. Manning Publications Co, Greenwhich, CT, 2002.

[WI, 2002] Abteilung WI. Zahlungssysteme im internet. Abteilung fr Wirtschaftsinformatik Wirtschaftsuniversitï¿½ Wien, 2002. http://wwwi.wu-wien.ac.at/Studium/Abschnitt_2/LVA_ss02/04-Zahlungssysteme.pdf.

[Wirtschaftskammer, 2002] Österreich Wirtschaftskammer. Der telekommunikationsmarkt in zahlen. 05 2002.

[www.heise.de, 2002] www.heise.de. Wie man sim-karten faelscht, 2002. http://www.heise.de/newsticker/meldung/27232.

[www.paybox.at, 2004] www.paybox.at. Paybox, 2004. http://www.paybox.at.

Abbildungsverzeichnis

Anhang A

RSA - Signatur Test

Für die Verifizierung der cryptografischen Fähigkeiten von Mobiltelefonen unter J2ME wurde nachfolgendes Beispiel programmiert. Dieses Java Programm konnte sowohl am Sun SMTK, als auch auf einem Siemens S55 Mobiltelefon erfolgreich ausgeführt werden. Wird dieses Programm auf einem Siemens SL45i gestartet, kommt es zwar zu keinerlei Laufzeitfehlern, die erstellte Signatur kann jedoch nicht verifiziert werden. Dies beweist, daß ein Fehler in der Siemens Java Virtuel Machine des SL45i besteht. Auf einem Siemens S55 Mobiltelefon konnte die Signatur sowohl vom Mobiltelefon selbst, als auch von einem Java Programm am Personel Computer verifiziert werden.

```
MIDlet-Version: 1.0.3
MIDlet-Vendor: Klaus Brosche
MIDlet-Description: RSA Test
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: TestRSA, , TestRSA
MicroEdition-Profile: MIDP-1.0
MIDlet-Name: TestRSA
MIDlet-Jar-Size: 11
MIDlet-Jar-URL: TestRSA.jar
```

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;

import org.bouncycastle.util.encoders.Base64;

import java.util.Hashtable;

/*Diese Klasse beweist, dass das Signieren auf anderen Mobiltelefonen,
 * sowie auf dem Emulator funktioniert.*/
public class TestRSA
```

```
    extends MIDlet
  implements CommandListener
{
  private Command exitCommand = new Command("Exit", Command.SCREEN, 1);
  private Display display;

  public TestRSA()
  {
    display = Display.getDisplay(this);                                    20
  }

  public void startApp()
  {
    doYourJob();
  }

  private void doYourJob()
  {
    Form frm = new Form("Crypto Test");                                    30

    frm.addCommand(exitCommand);
    frm.setCommandListener(this);

    display.setCurrent(frm);
    StringItem si;

    try
    {
      si = new StringItem("Create Signature!", "");                        40
      frm.append(si);
      display.setCurrent(frm);

      Crypto crypto = new Crypto(frm, display);

      /*Zu signierende Nachricht.*/
      String text = "hello my name is . . .";

      byte [] aSignature = crypto.signMessagePrivKey(text);
                                                                           50
      String stringSignature = new String(Base64.encode(aSignature));

      si = new StringItem("Verify Signature!", "");
      frm.append(si);
      display.setCurrent(frm);

      if (crypto.verifySignedMessage(text, Base64.decode(stringSignature)))
      {
        si = new StringItem("Signature verified!", "");
        frm.append(si);                                                    60
```

```
            display.setCurrent(frm);
          }
          else
          {
            si = new StringItem("Signature NOT verified!", "");
            frm.append(si);
            display.setCurrent(frm);
          }

        }                                                                    70
        catch (Exception e)
        {
        }
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean unconditional)                            80
    {
    }

    public void commandAction(Command c,
                              Displayable s)
    {
      if (c == exitCommand)
      {
        destroyApp(false);
        notifyDestroyed();                                                   90
      }
      else
        System.out.println("Command not managed");
    }
}
```

```
/*Bouncy Castle Klassen*/
import java.math.BigInteger;
import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.generators.*;
import org.bouncycastle.crypto.signers.*;
import org.bouncycastle.crypto.params.*;
import org.bouncycastle.crypto.digests.*;
import org.bouncycastle.crypto.engines.RSAEngine;
import java.security.*;                                                      10

import org.bouncycastle.util.encoders.Base64;
```

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;

import java.util.Hashtable;

/*Diese Klasse bearbeitet alle cryptografischen                                    20
 * Anforderungen des MPS mit Hilfe der Bouncy Castle API.*/
public class Crypto
{
  private Form frm;
  private Display display;

  public Crypto(Form frm, Display display)
  {
    this.frm = frm;
    this.display = display;                                                        30
  }

  private static RSAKeyParameters pubKey;
  private static RSAPrivateCrtKeyParameters privKey;

  private static BigInteger pubExp;
  private BigInteger mod;
  private BigInteger privExp;
  private BigInteger pubExponent;
  private BigInteger dp;                                                           40
  private BigInteger dq;
  private BigInteger p;
  private BigInteger q;
  private BigInteger qInv;

  public byte[] signMessagePrivKey(String aText) throws Exception
  {
    mod = new BigInteger(Base64.decode("AI3TjXQYlw7BlIvNE5XSaSq+LjPkiyXrDXADC"
                        +"sI5wQxUR6Fb6zowX5mwywt19vMW4Q=="));
    privExp = new BigInteger(Base64.decode("LIlWTjK6nOOlJhGMKOweIlH+aEVIfOv"     50
                        +"Gn0gM5gTj+hMijB8A77VNIzVB4pIfp6bB"));
    pubExponent = new BigInteger(Base64.decode("AQAB"));
    dp = new BigInteger(Base64.decode("AI4bMxOdhk2IzpNX9neKUxie3BWNtkMxcQ=="));
    dq = new BigInteger(Base64.decode("f/6GkxWoUsh8V9pbesnKl6+qcpVXcuzN"));
    p = new BigInteger(Base64.decode("AN93l4TbOtDklHr9rUrrmR3qEBihc5wYNQ=="));
    q = new BigInteger(Base64.decode("AKJ5SV5JQNV+xLkuj8BR5pIJo9BMcxjRfQ=="));
    qInv = new BigInteger(Base64.decode("UUxjWPKjjGV6pTn3+gmnwvVGU4pSrlrq"));

    privKey = new RSAPrivateCrtKeyParameters(mod, pubExponent,
                    privExp, p, q, dp, dq, qInv);                                  60
```

```
    byte[] aSignature = signMessage(aText.getBytes());

    return aSignature;
  }

  public boolean verifySignedMessage(String aText, byte[] aSignature)
  throws Exception
  {
    BigInteger mod = new BigInteger(Base64.decode("AI3TjXQYlw7BlIvNE5XSaSq+Lj"
                                +"PkiyXrDXADCsI5wQxUR6Fb6zowX5mwywt19vMW4Q=="));
    BigInteger pubExponent = new BigInteger(Base64.decode("AQAB"));

    pubKey = new RSAKeyParameters(false, mod, pubExponent);

    byte[] aMessage = aText.getBytes();

    SHA1Digest    sha1digest = new SHA1Digest();

    RSAEngine rsaengine = new RSAEngine();

    PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

    psssigner.init(false, pubKey);

    psssigner.update(aMessage, 0, aMessage.length);

    return psssigner.verifySignature(aSignature);

  }

  private byte[] signMessage(byte[] aMessage)
  throws Exception
  {
            SHA1Digest sha1digest = new SHA1Digest();

            RSAEngine rsaengine = new RSAEngine();

            PSSSigner psssigner = new PSSSigner(rsaengine, sha1digest, 16);

            psssigner.init(true, privKey);

            psssigner.update(aMessage, 0, aMessage.length);

            byte [] signrsult = psssigner.generateSignature();

      return signrsult;
  }

}
```

Anhang B

Beispiel zur symetrischen Verschlüsselung

```
package kbrosche.security;

/*this is an example implentation of the Rijndael algorithm
 *based on the Java API of Bouncy Castle
 *Author: Klaus Brosche*/

import org.bouncycastle.crypto.BufferedBlockCipher;
import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.engines.RijndaelEngine;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Base64;

public class BCRijndael
{
  private BufferedBlockCipher buffCipher;
  private KeyParameter keyParam;
  //ByteArray for the Outputvalue
  private byte[] out;
  //Size of out
  private int arraySize;

  public BCRijndael(String key, boolean encryption)
  {
    //create Key
    keyParam = new KeyParameter(key.getBytes());
    //initialize the BufferedBlockCipher
    buffCipher = new PaddedBufferedBlockCipher(new RijndaelEngine());
    //initialize the BufferedBlockCipher for encryption
```

```
    //with true or for decryption with false
    buffCipher.init(encryption, keyParam);
  }

  //do the Rijndael Encryption and return a Base64 encrypted Value
  public String encrypt(byte[] data)
  {
    //get the Output Array Size
    arraySize = buffCipher.getOutputSize(data.length);
    //initialize the Output Array                                    40
    out = new byte[arraySize];

    //process a single byte, producing an output block if neccessary
    //and return the number of output bytes copied to out
    int outOff = buffCipher.processBytes(data, 0, data.length, out, 0);

    //process the last block in the buffer
    try
    {
      buffCipher.doFinal(out, outOff);                               50
    }
    catch (CryptoException e)
    {
      System.out.println("Encrypt failed: " + e.toString());
    }

    //return a base64 encrypted value
    return new String(Base64.encode(out));
  }
                                                                     60
  //do the Rijndael Decryption and return the Result as a clear String
  public String decrypt(byte[] data)
  {
    //get the Output Array Size
    arraySize = buffCipher.getOutputSize(data.length);
    //initialize the Output Array
    out = new byte[arraySize];

    //process a single byte, producing an output block if neccessary
    //and return the number of output bytes copied to out            70
    int outOff = buffCipher.processBytes(data, 0, data.length, out, 0);

    //process the last block in the buffer
    try
    {
      buffCipher.doFinal(out, outOff);
    }
    catch (CryptoException e)
    {
```

```
      System.out.println("Decryption failed: " + e.toString());
    }

    //Arraylength of real Output
    int arrayLength = 0;

    //get the Arraylength of real Output where out integer value != 0
    while (arrayLength < out.length && (
        new Integer(out[arrayLength])).intValue() != 0)
    {
      arrayLength++;
    }

    //initialize a Byte Array with arrayLength
    byte[] result = new byte[arrayLength];

    //assign the value of out to result
    for (int i = 0; i != result.length; i++)
    {
      result[i] = out[i];
    }

    //return the clear text result
    return new String(result);
  }


  public static void main(String[] args)
  {
    //initialize this Class with the key and true for encryption
    //use a key with 16 bytes to do a 128 bit encryption
    BCRijndael rijndael = new BCRijndael("testtesttesttest", true);

    //get the encrypted value as a base64 encrypted string
    String result = rijndael.encrypt("hello world".getBytes());
    System.out.println(result);

    //initialize this Class for decryption
    rijndael = new BCRijndael("testtesttesttest", false);

    //get the clear text
    result = rijndael.decrypt(Base64.decode(result));
    System.out.println(result);
  }
}
```

```
package kbrosche.security;
```

```
/*this is an example implentation of the RC6 algorithm
  *based on the Java API of Bouncy Castle
  *Author Klaus Brosche*/

import org.bouncycastle.crypto.BufferedBlockCipher;
import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.engines.RC6Engine;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.params.KeyParameter;
import org.bouncycastle.util.encoders.Base64;

public class BCRC6
{
   private BufferedBlockCipher buffCipher;
   private KeyParameter keyParam;
   //ByteArray for the Outputvalue
   private byte[] out;
   //Size of out
   private int arraySize;


   public BCRC6(String key, boolean encryption)
   {
      //create Key
      keyParam = new KeyParameter(key.getBytes());
      //initialize the BufferedBlockCipher
      buffCipher = new PaddedBufferedBlockCipher(new RC6Engine());
      //initialize the BufferedBlockCipher for encryption with
      //true or for decryption with false
      buffCipher.init(encryption, keyParam);
   }

   //do the RC6 Encryption and return a Base64 encrypted Value
   public String encrypt(byte[] data)
   {
      //get the Output Array Size
      arraySize = buffCipher.getOutputSize(data.length);
      //initialize the Output Array
      out = new byte[arraySize];

      //process a single byte, producing an output block if neccessary
      //and return the number of output bytes copied to out
      int outOff = buffCipher.processBytes(data, 0, data.length, out, 0);

      //process the last block in the buffer
      try
      {
         buffCipher.doFinal(out, outOff);
```

```
    }
    catch (CryptoException e)
    {
      System.out.println("Encrypt failed: " + e.toString());
    }

    //return a base64 encrypted value
    return new String(Base64.encode(out));
  }

  //do the RC6 Decryption and return the Result as a clear String
  public String decrypt(byte[] data)
  {
    //get the Output Array Size
    arraySize = buffCipher.getOutputSize(data.length);
    //initialize the Output Array
    out = new byte[arraySize];

    //process a single byte, producing an output block if neccessary
    //and return the number of output bytes copied to out
    int outOff = buffCipher.processBytes(data, 0, data.length, out, 0);

    //process the last block in the buffer
    try
    {
      buffCipher.doFinal(out, outOff);
    }
    catch (CryptoException e)
    {
      System.out.println("Decryption failed: " + e.toString());
    }

    //Arraylength of real Output
    int arrayLength = 0;

    //get the Arraylength of real Output where out integer value != 0
    while (arrayLength < out.length && (
        new Integer(out[arrayLength])).intValue() != 0)
    {
      arrayLength++;
    }

    //initialize a Byte Array with arrayLength
    byte[] result = new byte[arrayLength];

    //assign the value of out to result
    for (int i = 0; i != result.length; i++)
    {
      result[i] = out[i];
```

```
    }

    //return the clear text result
    return new String(result);
  }


  public static void main(String[] args)
  {
    //initialize this Class with the key and true for encryption
    //use a key with 16 bytes to do a 128 bit encryption
    BCRC6 rc6 = new BCRC6("testtesttesttest", true);

    //get the encrypted value as a base64 encrypted string
    String result = rc6.encrypt("hello world".getBytes());
    System.out.println(result);

    //initialize this Class for decryption
    rc6 = new BCRC6("testtesttesttest", false);

    //get the clear text
    result = rc6.decrypt(Base64.decode(result));
    System.out.println(result);
  }
}
```