Diplomarbeit

Immanuel Scholz

8. November 2004

Immanuel Scholz
Matrikel: 2658156
geb. am: 16.10.1978

Diplomarbeit

Verschlüsselte SMS auf Basis des vorgeschlagenen Standards SEMS

Zielstellung:
Der Bedarf an Datenkommunikation über mobile Endgeräte wächst rasant an. Dabei wächst zugleich auch der Bedarf an Sicherheit. Bisher übliche Kurznachrichten werden meist als SMS versandt, dessen Übertragungsprotokoll keinen ausreichenden Schutz gewährleisten kann. Der vorgeschlagene Secure Message Standard (SEMS) soll auf seine Anwendbarkeit überprüft und gegebenenfalls verbessert werden. Es ist eine Lösung zur Verschlüsselung und Sicherung der Integrität von Kurzmitteilungen (SMS) unter Java mit der Schnittstelle MIDP zu implementieren.

Schwerpunkte:

- Analyse der Anforderungen an sichere Kommunikation von Kurznachrichten über mobile Endgeräte,
- Analyse des vorgeschlagenen Standards SEMS auf Übereinstimmung mit diesen Anforderungen,
- Erweiterung/Anpassung von SEMS und
- Beispielimplementation in Java.

Betreuer:	Dr.-Ing. Andreas Westfeld
Ausgehändigt am:	13. April 2004
Einzureichen am:	13. Oktober 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Die Kommunikation mit einem mobilen Endgerät, einem Handy, ist in der heutigen Gesellschaft nicht mehr wegzudenken. Ob Schüler, Student, Erwachsener oder Rentner, die Zahl der benutzten Handys steigt stetig an. Vier von fünf Deutschen besitzen bereits eins, viele mehrere. Der Absatzmarkt blüht, es werden täglich mehr Handys verkauft und mehr Anrufe getätigt.

In Kreisen wie beispielsweise für Geschäftsleute, Reporter oder andere hoch mobile Angestellte ist das Handy längst zu einer notwendigen Bedingung geworden. Sichere Kommunikation ist dabei erforderlich. Das Risiko der Wirtschaftsspionage ist für die drahtlose Nachrichtenübertragung besonders hoch, da hier die technischen Rahmenbedingungen für ein Abhören sehr einfach herzustellen sind.

Der Schock war nicht groß, als 1998 die GSM-Verschlüsselung gebrochen wurde. Einige Meldungen in Fachzeitschriften, in Mailinglisten und im Usenet – die Probleme gingen unter, erreichten längst nicht die Medienwirksamkeit der „aufladbaren Telefonkarten", welche etwa zeitgleich die Zeitungen füllten. Doch mit der Zeit wurden Telefonzellen und -karten immer unwichtiger, das Mobiltelefon immer bedeutender. UMTS ist längst noch nicht so verbreitet, dass es das führende System für Handys werden kann. So entsteht eine empfindliche Lücke in der Sicherheitssoftware für den Mobilfunkbereich.

Diese Arbeit beschäftigt sich mit einem vorgeschlagenen Standard zur Absicherung des Kurznachrichtenverkehrs, mit dem Secure Message Standard. Kapitel 1 zeigt grundlegende Mechanismen der mobilen Kommunikation über das GSM-Netz, stellt verschiedene Szenarien des wirtschaftlichen Nutzens der SMS vor und definiert Anforderungen zu deren Erfüllung. Kapitel 1 untersucht auch bestehende Standards und Produkte danach, ob diese Anforderungen zutreffen. Kapitel 2 behandelt den „Secure Message Standard". Die Struktur und der Aufbau von SEMS wird beschrieben, es folgt eine Kryptoanalyse des Standards. Schließlich werden Änderungen an SEMS präsentiert, die notwendig waren bzw. verworfen wurden. Kapitel 3 stellt den praktischen Teil vor: Die im Rahmen dieser Arbeit entwickelte SEMS Java Implementation. Schließlich bietet Kapitel 4 eine Zusammenfassung und einen Ausblick auf weitere Arbeitsmöglichkeiten.

Kapitel 1

Voruntersuchung

Als erstes soll ein kurzer Überblick über den SMS- und GSM-Standard und deren Sicherheit gegeben werden. Anschließend wird der Frage nachgegangen, ob SMS heute und in Zukunft relevant für privaten und kommerziellen Gebrauch ist und es werden Kriterien für eine Nutzung herausgestellt. Es folgt eine Untersuchung bestehender Standards und Produkte auf Übereinstimmung mit diesen Kriterien.

1.1 SMS-Protokoll

SMS wurde erst durch das „European Telecommunications Standard Institute“ [ETSI], später dann durch das „3rd Generation Partnership Project“ [3GPP] definiert. Aktueller Stand ist das Dokument 3GPP TS 23.040 in der Version 6.3.0.[1]

Vereinfacht unterscheidet der Standard zwischen zwei Arten der Sender und Empfänger. Das Service Center (SC) sind Rechner auf der Kommunikationsstrecke der SMS, während mit Mobile Stations (MS) alle Art von mobilen Endgeräten bezeichnet werden. Die Übertragung der SMS von einem Handy zu einem anderen beinhaltet also stets folgende Schritte:

- MS-SC - Übertragung der Nachricht vom Handy in das Netz per Funk. Diese Kommunikation wird auch SM MO genannt (Short Message Mobile Originated). Sie ist verschlüsselt nach einem GSM-internen Standard.

- SC-SC - Übertragung innerhalb des Telekommunikationsnetzes; es werden meist dieselben Leitungen wie für Festnetzkommunikation benutzt. Diese Übertragung ist unverschlüsselt.[2]

[1] Dieses Dokument beinhaltet ebenso den aktuellen Standard für EMS und MMS.

[2] Einige Länder verschlüsseln mittlerweile die Übertragung zwischen den SC ebenfalls. Deutschland gehört allerdings nicht zu diesen.

- SC-MS - Übertragung an das Zielhandy, was auch als SM MT bezeichnet wird (Short Message Mobile Terminated). Dieser Teil der Übertragung ist wieder verschlüsselt.

Der letzte Schritt kann dabei auch verzögert stattfinden, etwa weil das Zielhandy nicht erreichbar ist. Abbildung 1.1 verdeutlicht den Aufbau. Die Mo-

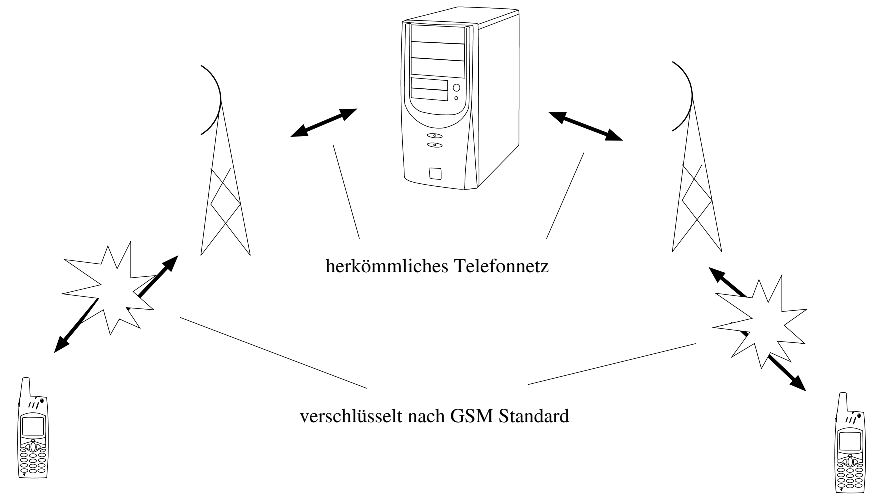

Abbildung 1.1: Übersicht über die Übertragung im GSM-Netz

bilfunkteilnehmer telefonieren über Funk mit dem ersten Service Center, welches die Daten über Festnetzstrecken zum letzten SC weiterleitet. Der letzte SC versucht das Zielhandy zu erreichen und überträgt die SMS.

Durch die unverschlüsselte Übertragung über das Festnetz ist die erreichbare Sicherheit durch GSM maximal so hoch wie die des herkömmlichen Telefonnetzes. Ein Angreifer der ein Service Center kontrolliert, kann alle Datenkommunikation über diesen mitlesen.

1.2 GSM-Schwächen

Kurznachrichten werden zusammen mit der Sprachkommunikation über das GSM Netz übertragen.[3] Ein Designziel bei der Entwicklung von mobiler Kommunikation war, die Sicherheit nicht schwächer als das bisher bestehende und akzeptierte Telefonfestnetz zu machen. Das bedeutet konkret: Ein Angreifer soll nicht in der Lage sein die Gespräche abzuhören, solange er sowohl keine SC kontrolliert als auch keinen Zugang zum physischen Übertragungskanal besitzt. Da die Übertragungsstrecke von dem letzten SC zum Handy über Funkstrecke geschieht, ist ein Zugriff auf diesen Kanal praktisch für jeden möglich. Aus diesem Grund wird diese Übertragungsstrecke kryptographisch abgesichert. Die Verschlüsselung geschieht nach dem GSM/A5.1

[3]Bei UMTS-Handys wird das UMTS-Netzwerk benutzt.

(stark) oder GSM/A5.2 (schwach) Verschlüsselungsstandard. Die Authentisierung erfolgt nach GSM/A3, die Schlüsselerzeugung nach GSM/A8. Die Algorithmen sind von der GSM-Kommission nicht offengelegt.

An GSM/A5.1 gab es bereits 1994 die ersten Zweifel in der Sicherheit, 1999 veröffentlichte Birykov und Shamir schließlich einen erfolgreichen Angriff. GSM/A8 und A3 wurden 1998 gebrochen, der selten verwendete GSM/A5.2 ist per Definition brechbar (GSM/A5.2 benutzt einen verkürztzten Schlüssel. Die Idee hinter diesem Algorithmus war, schwachen Angreifern mit ungenügend Rechnekapazität das Brechen zu vereiteln, aber Bedarfsträgern mit entsprechenden Möglichkeiten die Entschlüsselung zu ermöglichen). Darüber hinaus verwenden viele Telekommunikationsanbieter nur eine reduzierte Schlüssellänge von 54 Bit, statt der vorgeschriebenen 64 Bit, gerüchtweise ebenfalls um Bedarfsträgern das Mithören zu ermöglichen. Schließlich zweifeln auch viele Kryptographen generell die Sicherheit von 64-Bit-Schlüsseln an.

[GSMHack] gibt eine Zusammenfassung einiger Quellen und Publikationen zu Schwächen in der GSM Verschlüsselung.

Für Kurznachrichten kommt noch hinzu, dass beispielsweise die automatische Extraktion einer Kreditkartennummer aus einer Textnachricht trivial ist im Vergleich zur Suche einer gesprochenen Zahl in einem Audiostrom. Das Angreiferpotential ist hier also ungleich höher.

1.3 Szenarien der SMS-Nutzung

In wirtschaftlichen Anwendungen der Kommunikation wird üblicherweise zwischen drei verschiedenen Szenarien unterschieden. Diese werden kurz vorgestellt.

C2C

„Customer to customer“, kurz C2C ist die Kommunikation der Kunden untereinander. Steht diese im Vordergrund eines Produktes, verdient der Vertreiber an einer einmaligen Bereitstellungsgebühr oder an laufenden Gebühren für das Erhalten und Bereitstellen des Services. Ein Beispiel ist die normale, private Urlaubs-SMS mit einem Handy. Die Zahl der Kunden und die Zahl der Verbindungen kann hier extrem hoch sein.

B2C

Kommuniziert der Kunde vorrangig mit dem Betrieb, so wird dies „Buisiness to Customer“ oder B2C genannt. Die Abrechnung erfolgt hier meist direkt für in Anspruch genommene Leistungen beim Betreiber. Beispiele sind der Abruf von Nachrichtenmeldungen oder Online Banking. Wie beim C2C soll das Produkt möglichst viele Kunden erreichen. Die Zahl der Verbindungen ist zwar geringer, aber dennoch bedeutend.

B2B

„Buisiness to Buisiness“ wird die Kommunikation zwischen Betrieben untereinander genannt. Ein Beispiel für B2B-Anwendungen ist eine innerbetriebliche Lagerverwaltung. Die Zahl der Kunden und Verbindungen ist bei B2B niedrig, dafür sind die Sicherheits- und Stabilitätsansprüche an die Software hoch.

1.4 Wirtschaftliche Nutzung

Laut einer Studie der französischen Regulierungsbehörde für Telekommunikation „Autorité de Régulation des Télécommunications“ [ART] versenden deutsche Mobilfunknutzer rund 40 SMS pro Monat. Die Studie bezieht sich auf das erste Quartal 2003. Nach einer Erhebung des „Verbandes der Anbieter von Telekommunikations- und Mehrwertdiensten“ [VATM] wurden im Jahr 2003 rund 25,5 Milliarden SMS verschickt. Diese Zahl wurde auch von der Initiative D21 bestätigt. Nach deren Studie besitzen außerdem vier von fünf Bundesbürgern ein Handy, wobei lediglich jeder zweite Haushalt (53 Prozent) einen Internetzugang vorzuweisen hat [ID21].

Bei den versendeten SMS handelt es sich jedoch zum überwiegenden Teil um Privatkommunikation. Das Sicherheitsbedürfnis ist hier nur gering, eine fehlende verschlüsselte Kommunikation wird nicht als störend empfunden. Dies ändert sich, sobald die Kommunikation kommerzielle Interessen betrifft. So zeigt eine Studie der Firma Fittkau & Maaß GmbH aus dem Jahre 2001 (Abbildung 1.2) über die größten Vorbehalte gegenüber der kommerziellen Nutzung des Internets deutlich, dass den Nutzern hier Sicherheit besonders wichtig ist. Es ist davon auszugehen, dass diese Ergebnisse auch auf die kommerzielle Nutzung von Kurznachrichten anzuwenden ist.

Eine andere Statistik der Saxonia Systems AG besagt, dass die momentane Nutzung von Kurznachrichten zwar vorrangig im Kunde-zu-Kunde-Bereich angesiedelt ist (C2C), aber eine starke Marktentwicklung im Bereich Betrieb zu Kunde (B2C) stattfinden wird (Abbildung 1.3). Wie Abschnitt 1.2 zeigt, ist eine unberechtigte Nutzung oder ein Ausspähen vertraulicher Kommunikationsinhalte über SMS technisch nicht mehr auszuschließen. Dies führt zu Missbrauchsfällen und rechtlichen Konsequenzen für kommerzielle Dienstanbieter, ähnlich den EC-Kartenfällen der Vergangenheit.

Diese Studien verdeutlichen, dass ein immenser wirtschaftlicher Bedarf an Kommunikation kurzer, schneller und direkter Nachrichten besteht. Kurznachrichten weisen gegenüber dem Telefonat verschiedene Vorteile auf. Sie lassen sich sehr gut archivieren, automatisch erzeugen und auswerten. Der Bedarf an Bandbreite im Kanal ist sehr gering und die Kommunikation läuft

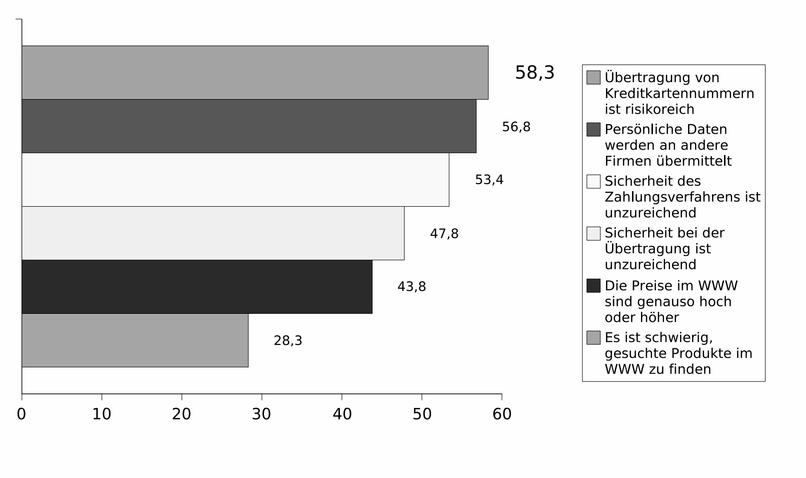

Abbildung 1.2: Sorgen bei kommerzieller Internetbenutzung

asynchron ab. Alle diese Vorteile gelten auch für die Zukunft. Dadurch wird wohl der Bedarf des Versendens von Kurznachrichten auch in Zukunft hoch sein.

Zwar werden SMS heute fast ausschließlich im privaten Bereich eingesetzt, jedoch ändert sich die Nutzung in naher Zukunft. Mit der weiteren Akzeptanz und Verbreitung von Handys wird die SMS auch in den Buisinessbereich übergreifen. Dort ist das Sicherheitsbedürfnis ungleich höher, was den Einsatz zusätzlicher Sicherheitssoftware attraktiv macht.

1.5 Kriterien für Verschlüsselungssoftware von Kurznachrichten

Die letzten Abschnitte stellten heraus, dass die bestehenden kryptographischen Absicherungen von SMS unzureichend für den kommerziellen Gebrauch sind. Außerdem wurden einige Statistiken über Bedürfnisse und Entwicklungen in Bezug auf Kurznachrichten vorgestellt. Nun werden Anforderungen herausgearbeitet, die an Verschlüsselungssoftware zu stellen sind.

1.5.1 Sicherheit

Das zugrunde liegende Verfahren sollte selbstverständlich sicher gegenüber allen bekannten Angriffen sein. Die Benutzung von länger untersuchten kryptographischen Algorithmen ist von Vorteil.

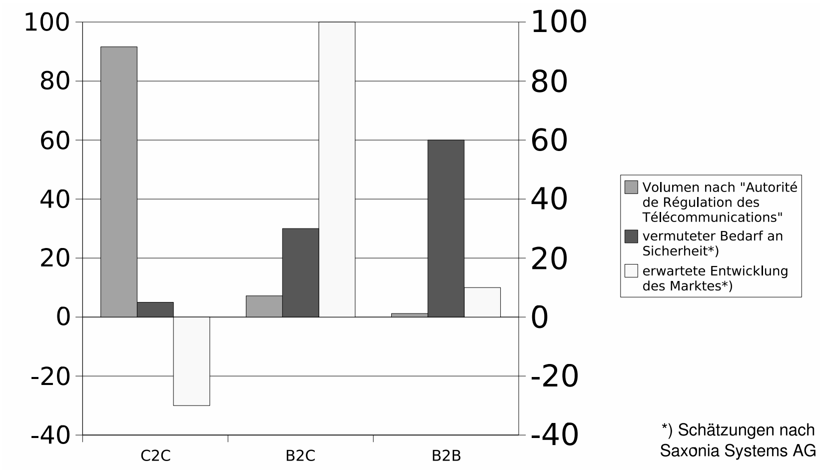

Abbildung 1.3: Marktverteilung und Sicherheitsbedürfnis von SMS

1.5.2 Öffentlich

Die Verschlüsselungsprotokolle von GSM haben wieder gezeigt, dass die Geheimhaltung von kryptographischen Protokollen oder Verfahren kein oder ein zu geringes Hindernis für die Analyse von Protokollen ist. Ein Sicherheitsvorteil aufgrund von Geheimhaltung wird von Kryptographen generell angezweifelt.

1.5.3 Frei verfügbar

Die Software sollte nicht nur für einige Hersteller verfügbar sein. Diese Forderung ist besonders für den C2C Bereich wichtig, da hier soviel Kunden wie nur möglich erreicht werden sollen. Ist das Verfahren zudem unpatentiert, kann Konkurrenz in der Implementation entstehen, was auch zu besserer Software führt.

1.5.4 Asymmetrie

Asymmetrische Authentikationssysteme sind von großer Bedeutung für die kommerzielle Nutzung des Mobiltelefones. Die Herkunft der damit geleisteten digitalen Signaturen ist, im Gegensatz zu Authentikationskodes der symmetrischen Kryptographie, vor Dritten beweisbar. Soll sichere Kommunikation mit vielen verschiedenen Kommunikationspartnern erfolgen, so sind asymmetrische Schlüssel von Vorteil. So braucht jeder Kommunikationspartner nur einmal einen Schlüssel generieren, den er für alle Kommunikationen benutzt.

1.5.5 Geschwindigkeit

Die Geschwindigkeit der Berechnung ist nur von untergeordneter Bedeutung, da die Kurznachrichten asynchron versendet werden. Ist der Algorithmus jedoch zu langsam, könnten Anwender versehentlich das Handy ausschalten, bevor die SMS verschickt werden konnte. Eine Bearbeitungsdauer von mehreren Sekunden ist hier vollkommen akzeptabel. In Abschnitt 3.6 werden beispielhaft Grenzen für die Geschwindigkeit getroffen.

1.5.6 Hardwareanforderungen

Das Programm muss auf heute verbreiteten Handys laufen. Häufig sind die Entwicklungstools für Mobiltelefone und kleine PDA's wie Sharp Zaurus oder Sony Clie identisch. Etliche Programme wurden erst für diese Plattformen entwickelt und nun notdürftig an die leistungsschwächeren Handys angepasst. Die Mindestanforderungen sollten nicht über einem Speicherverbrauch von etwa 512 KB RAM und einem persistenten Speicher von 2 MB liegen.

1.6 Standards und Produkte

Der folgende Abschnitt untersucht einige ausgewählte Produkte und Standards näher. Abschnitt 1.6.8 ist eine Zusammenfassung in Bezug auf die herausgestellten Kriterien.

1.6.1 Fortress SMS

Ein kommerzielles Produkt der Firma Silicon Village ist „Fortress SMS" ([FortSMS]). Die Verschlüsselung basiert hier rein symmetrisch auf AES mit einem nutzergewählten Passwort als Schlüssel. Eine Schlüsselspeicherung findet nicht statt. Mit Fortress SMS-M werden die Nachrichten auch verschlüsselt auf dem Handy abgelegt. Die Anwendung benutzt Herstellerspezifische API's und ist beispielsweise für Ericsson P800/P900 oder Nokia 7650 implementiert.

1.6.2 SecureMessenger

SecureMessanger [SecMes] ist eine Open Source Projekt zur symmetrischen Ende-zu-Ende-Verschlüsselung. Es besteht aus einer API für Java-fähige Handys. Drei verschiedene Implementationen der API existieren – für Siemens SL45i, Siemens M50/S55 und Nokia Series 60. Der Schlüssel wird, ähnlich wie bei Fortress SMS, nicht gespeichert und direkt aus dem Passwort errechnet. Als Verschlüsselungsverfahren kommt 3DES mit zwei DES-Schlüsseln zum Einsatz. Eine Authentikation findet nicht statt.

1.6.3 Secure SMS messaging using Quasigroups

Auch das Protokoll der Arbeit von Marko Hassinen und Smile Markovski aus Finnland arbeitet mit reiner symmetrischer Verschlüsslung. Dabei stützt sich die Verschlüsselung auf einen relativ neuen mathematischen Kryptographie-Ansatz. Zusammen mit der Arbeit implementierten die Autoren das Protokoll auf Java-fähigen Handys mithilfe Nokia- und Siemensspezifischer SMS-Schnittstellen. Ein Übergang zu einer MIDP-Schnittstelle ist bereits vorgesehen und sollte problemlos möglich sein.

1.6.4 PointSec Enterprise File Encryption Suite

PointSec hat in Zusammenarbeit mit Nokia eine komplette Sicherheitslösung für Nokia Series 60 und 80 Mobiltelefone entwickelt. Das Hauptaugenmerk der Suite zielt auf die Verschlüsselung der im Handy gespeicherten Informationen. Die Authentisierung des Nutzers erfolgt über ein Passwort.[4] Die Suite enthält auch eine Komponente zur SMS-Verschlüsselung. Auf Anfrage teilte Pointsec mit, dass die Verschlüsselung der SMS symmetrisch mithilfe AES und vorher ausgetauschten Schlüsseln geschieht.

1.6.5 Kannel

Das OpenSource Programm „Kannel“ ([Kannel]) ist ein WAP- und SMS Gateway. Mit Hilfe eines Addon ist es möglich, die SMS über WAP direkt auf dem SC zu lesen. Diese Verbindung kann nun noch mittels WTLS verschlüsselt werden. Durch geschickte Vergabe von Zertifikaten an Wapserver und Handys kann so eine asymmetrische Übertragung von SMS erreicht werden. Allerdings stellen alle existierenden WTLS-Implementationen hohe Anforderungen an die Leistungsfähigkeit der Handys. BouncyCastle, eine freie Implementation, braucht etwa 20 Sekunden allein zum beidseitig zertifizierten Verbindungsaufbau mit einem WAP-Server. Da außerdem das Handy zum Verbindungsaufbau GPRS-fähig sein muss, stellt sich natürlich die Frage, ob es sinnvoller ist zur Übertragung der Kurznachricht gleich ein anderes Protokoll als SMS zu verwenden.

1.6.6 Secure Message Standard

Der Secure Message Standard, ist ein vorgeschlagener Standard der Universität Mannheim, Der Firma „NAH6“ und der freien Universität Amsterdam. Er arbeitet asynchron mit RSA und zwei verschiedenen Schlüssellängen. SEMS ist auf die minimale Anzahl zu versendender SMS optimiert. Nach Angaben der Entwickler existieren bisher keine vollständigen Implementationen des Standards.

[4] PointSec benutzt ein als „PicturePin“ bezeichnetes Verfahren, bei dem Symbole statt Buchstaben die PIN bilden.

1.6.7 Naive Ansätze

Durch die lange Abwesenheit eines Standards für die Ende zu Ende Verschlüsselung von Kurznachrichten entstanden auch viele naive Ansätze. Diese kommen häufig gänzlich ohne Software und Schlüssel aus. Die Bürgerinitiative für Sicherheit im Versand von Kurznachrichten [BSVK] empfiehlt beispielsweise die Verwendung von verschiedenen Cäsar-Chiffren oder die pauschale Negierung jeder Aussage als „semantische Verschlüsslung“. Als anderes Beispiel wird unter [T9Enc] durch Ausnutzen der verschiedenen Interpretationen von Ziffernfolgen des T9-Eingabemodus in verschiedenen Sprachen eine Verschleierung des Textes erreicht.

1.6.8 Zusammenfassung

Eine kurze Zusammenfassung in Abbildung 1.4 zeigt tabellarisch, wie sich die verschiedenen untersuchten Standards und Produkte in Bezug auf die geforderten Kriterien verhalten.

Produkt	sicher	öffentlich	frei verfügbar	asymmetrisch	Geschwindigkeit	Hardware
Fortress SMS	ja, AES	nein	nein	nein	schnell	schnell
SecureMessenger	ja, 3DES	ja	ja, GPL	nein	sehr schnell	ok
Quasigroups	unerforscht	ja	keine Lizenz	nein	sehr schnell	ok
PointSec	ja, AES	nein	nein	nein	ausreichend	ok
Kannel	ja, WTLS	ja	ja	ja	langsam	nur neuere
SEMS	ja, RSA	ja	ja, GPL	ja	ausreichend	ok
Naive	nein, T9	ja	ja	nein	k.A.	ok

Abbildung 1.4: Zusammenfassung der Standards und Produkte

Aus der Tabelle erfüllt lediglich SEMS alle geforderten Kriterien. Die „freie Verfügbarkeit“ ergibt sich erst durch diese Arbeit. Die in Kapitel 3 vorgestellte Implementation ist unter der General Public Licence lizensiert.

Kapitel 2

Secure Message Standard

Der „Secure Message Standard" (SEMS) ist ein vorgeschlagener Standard zum Verschlüsseln und Authentisieren von Kurznachrichten über GSM-Netze. Er basiert direkt auf dem RSA-Verfahren, arbeitet also rein asymmetrisch.[1]

2.1 Aufbau

SEMS unterscheidet grundlegend zwei verschiedene Sicherheitsstufen. Die mittlere Sicherheit bietet mit einer Schlüssellänge von 1120 Bit guten Schutz für heutige Angreifer. Zudem benötigt man hier – eine genügend kurze Nachricht vorausgesetzt – nur eine SMS. Demgegenüber ist die hohe Sicherheitsstufe mit 2208 Bit Schlüssellänge selbst für die längere Zeit ausreichend gegen einen Brute-Force-Angriff[2].

Unabhängig von der Schlüssellänge können Nachrichten signiert, verschlüsselt oder signiert und verschlüsselt übertragen werden. Signaturen können dabei entweder direkt in der SMS enthalten sein oder als eigene SMS getrennt versendet werden.[3] Die Textkodierung ist ebenfalls durch das Protokoll festgelegt und reicht von einer 5-Bit Darstellung (Telex-Stil) bis 8-Bit unkodiert. Die Nutzdaten können anstelle eines Textes auch ein Schlüssel mit bis zu 32 angehängten Signaturen oder ein Schlüsselwiderruf sein. Außerdem sieht SEMS noch die Möglichkeit vor, eine „No Operation"-Nachricht zu versenden, welche vom Empfänger ignoriert wird um Analysen der Verkehrsdaten zu erschweren.

[1] Mit einer Ausnahme: Für einen bestimmten Operationsmodus wird ein symmetrisches Verfahren benötigt.

[2] Das Durchprobieren sämtlicher möglicher Schlüssel.

[3] Nicht zu verwechseln mit der Möglichkeit aus dem EMS-Protokoll mehrere geteilte SMS zu versenden.

2.2 Operationsmodi

Die im vorangegangenen Abschnitt vorgestellten Operationsmodi sollen nun näher betrachtet werden. Dabei werden zuerst die verschiedenen Grundtypen von SEMS kurz vorgestellt und anschließend jeder Modus einzeln auf Bedeutung und Probleme untersucht.

2.2.1 Unverschlüsselt

SEMS bietet die Möglichkeit an, Daten ohne Verschlüsselung zu versenden. Dadurch können Kurznachrichten auch an Personen übertragen werden, von denen man keinen Schlüssel besitzt. Auch zur Übertragung der Schlüssel selber bieten sich die unverschlüsselten Modi an.

2.2.2 Mittlere Sicherheit

Die mittlere Sicherheitsstufe bietet mit einer Schlüssellänge von 1120 Bit für das RSA Protokoll gerade noch ausreichenden Schutz gegen Angriffe, die eine Faktorisierung des Schlüssels bedingen. Nach Schätzungen in [ShTr03] ([DS81] für eine gute Zusammenfassung) wird zur Faktorisierung eines 1024 Bit RSA-Schlüssels mit einem Hardwareaufwand von ca. 10 Millionen Euro weniger als ein Jahr benötigt. Dies war Anfang 2003. Die mittlere Sicherheitsstufe ermöglicht den Versand verschlüsselter Nachrichten in nur einer SMS.

2.2.3 Hohe Sicherheit

Soll die Übertragung auch gegen professionelle Angreifer mit größerem Etat für Hardware sicher sein, so ist die Benutzung der hohen Sicherheitsstufe erforderlich. Die Schlüssellänge beträgt hier 2208 Bit, was als sicher gilt. Das hat allerdings zur Folge, dass jede Nachricht stets mindestens zwei Kurznachrichten zum Versand benötigt.

2.2.4 Unsigniert

Kurznachrichten können unsigniert versendet werden. Allerdings ist das unsignierte Verschlüsseln von SMS generell mit Bedacht zu verwenden, da der Absender solcher Nachrichten nicht sicher zuordenbar ist.[4]

2.2.5 Integrierte Signatur

Wird eine Nachricht im Klartext mit einer integrierten Signatur versendet, so richtet sich die Anzahl der benötigten Nachrichten nach der eigenen Si-

[4] Der Absender einer SMS lässt sich mit nur wenig Aufwand fälschen.

cherheitsstufe. Auch hier ist im günstigsten Fall nur eine SMS nötig. Die Signatur der Nachricht kann sofort bei Erhalt der SMS geprüft werden.

Integrierte Signatur bedeutet hier jedoch unter anderem, dass die SMS mit dem eigenen privaten Schlüssel verschlüsselt wird. Dadurch ist das Lesen der eigentlichen Nachricht nur möglich, wenn der Empfänger den öffentlichen Schlüssel des Senders besitzt.

2.2.6 Ausgelagerte Signatur

Eine Signatur kann auch in eine eigene SMS ausgelagert werden. Dies bietet den Vorteil, dass der Empfänger nicht zwingend über den Testschlüssel verfügen muss um die Nachricht zu lesen. Außerdem ist es möglich, Signaturen für bereits verschickte SMS sozusagen nachzureichen, falls der Empfänger sein Misstrauen äußert. Der Nachteil der ausgelagerten Signatur ist die zusätzlich benötigte SMS.

Eine etwaige Verschlüsselung bezieht sich hier immer nur auf die ausgelagerte Signatur, nicht auf die vorangegangene zu signierende SMS. Da der Hashwert für die Signatur vor der Verschlüsselungsoperation gebildet wird, ist der Versandmodus jener SMS irrelevant. Außerdem wird bei der Verschlüsselung der Signatur ein symmetrisches Verfahren mit festem Schlüssel benutzt, so dass hier nicht zwischen mittlerer und hoher Sicherheitsstufe unterschieden wird.

Die Tabelle 2.1 stellt die verschiedenen Operationsmodi und deren benötigte Zahl an SMS noch einmal dar.

	unverschlüsselt	verschlüsselt, mittlere Sicherheit	verschlüsselt, hohe Sicherheit
unsigniert	1 SMS	1 SMS	2 SMS
integrierte Signatur mittlerer Sicherheit	1 SMS	1 SMS	2 SMS
integrierte Signatur hoher Sicherheit	2 SMS	—	2 SMS
ausgelagerte Signatur mittlerer Sicherheit	2 SMS	2 SMS	2 bzw. 3 SMS
ausgelagerte Signatur hoher Sicherheit	3 SMS	3 SMS	3 bzw. 4 SMS

Abbildung 2.1: Notwendige Zahl der SMS

Die Zahlen beziehen sich dabei auf die mindestens notwendige Anzahl. Für die ausgelagerte Signatur hängt diese Zahl der SMS von der Verschlüsselungsstufe der vorangegangenen SMS ab (die erste Zahl für mittlerer Sicherheit, die zweite für hohe Sicherheit).

2.2.7 Einige Operationsmodi im Detail

Es werden nun einige der zwölf „erlaubten“ Operationsmodi kurz vorgestellt. Nicht erlaubt ist dabei die Kombination Verschlüsselung mittlerer Sicherheitsstufe mit integrierter Signatur hoher Sicherheitsstufe.[5]

Abbildung 2.2 zeigt eine Legende für alle Abbildungen dieses Abschnittes. Die Zahlen in den Klammern der Erläuterung sind die Kapitel innerhalb von SEMS.

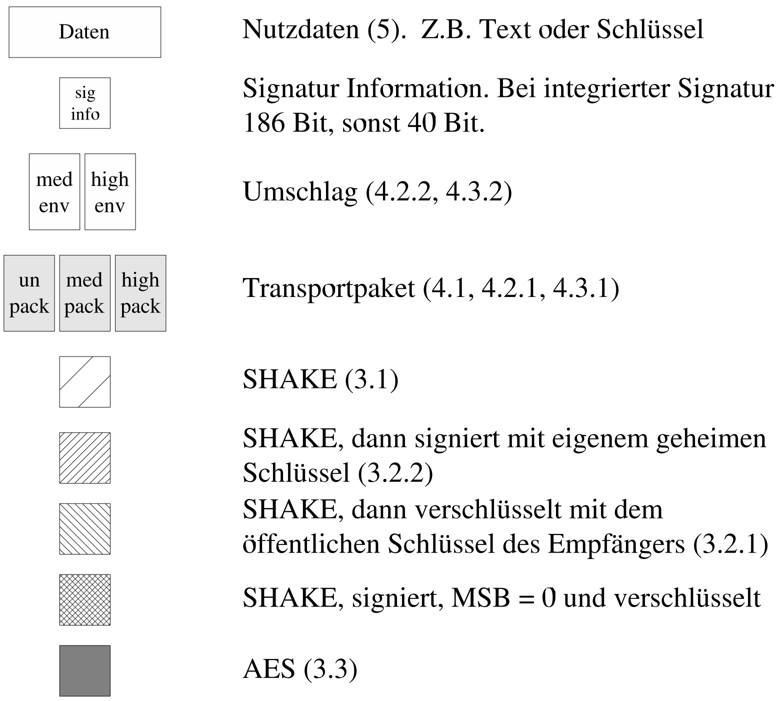

Abbildung 2.2: Legende der Abbildungen zu SEMS

Unverschlüsselt, unsigniert

Abbildung 2.3 zeigt das entstehende Datenmodell für den Versand von unverschlüsselten und unsignierten SMS in SEMS. Der Datenbereich im Inneren entspricht dabei einem Text oder einem Schlüssel[6]. Dieser Vorgang ist im Standard unter 2.2.1 beschrieben.

[5]Diese Kombination hat nur Nachteile gegenüber der ausgelagerten Signaturvariante gleicher Sicherheitsstufen.

[6]Beim Versand eines Schlüssels sind zwei SMS notwendig, wobei jede SMS ein „encapsulated payload object“ (Punkt 5.6) enthält.

Unverschlüsselte und unsignierte SMS entsprechen am ehesten den her-

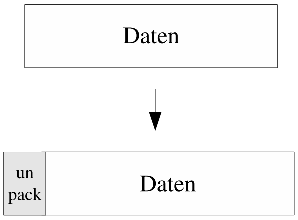

Abbildung 2.3: Unverschlüsselt, unsigniert

kömmlichen Kurznachrichten, lediglich Protokollvorteile wie die verschiedenen Kodierungen oder das Aufteilen von Nachrichten über mehrere SMS können genutzt werden.[7]

Momentan ist es von einem Java Midlet mit MIDP 2.0 aus nicht möglich, auf das Adressbuch des Handys zuzugreifen. Daher muss für standardkonforme Implementationen ein eigenes Adressbuch geführt werden. Soll einfach eine herkömmliche Nachricht an einen Empfänger in diesem Adressbuch versandt werden und hat der Empfänger ebenfalls eine SEMS-Implementierung lauffähig, so bietet sich dieser Modus an.

Unverschlüsselt, integrierte Signatur mittlerer Sicherheit

Der in Abbildung 2.4 abgebildete Modus stellt den Standardweg dar, eine Kurznachricht signiert und mit wenig Overhead zu versenden.

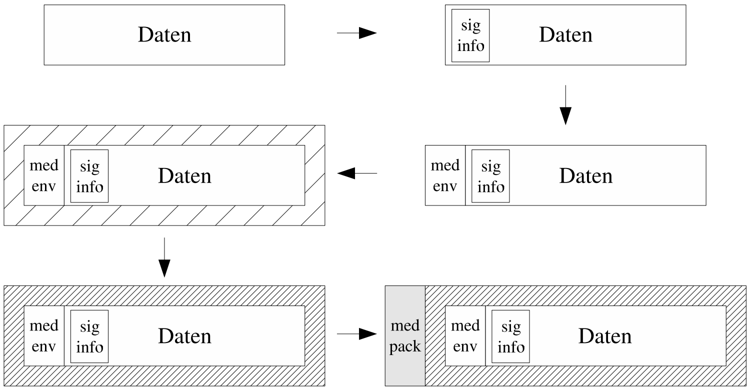

Abbildung 2.4: Unverschlüsselt, integrierte Signatur mittlerer Sicherheit

Es ist allerdings zu beachten, dass der Empfänger zum Lesen der Nachricht den Testschlüssel des Senders benötigt. Ein pauschales Signieren aller ausgehenden SMS, wie es bei Emails zum Teil empfohlen wird, ist hier also nicht angebracht.

[7] Wobei praktisch jedes heutige Handy bereits EMS unterstützt, welches ebenfalls Nachrichten aufteilen kann.

Verschlüsselung und ausgelagerte Signatur mittlerer Sicherheit

Im Operationsmodus aus Abbildung 2.5 wird erst ein Hashwert der vorangegangenen Nachricht gebildet, welcher dann signiert und mit AES verschlüsselt wird.

Der Schlüssel und Initialisierungsvektor (IV) des symmetrischen Verfah-

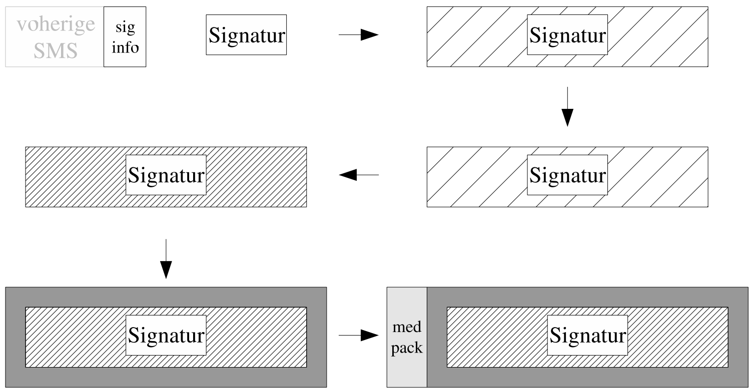

Abbildung 2.5: Verschlüsselt, ausgelagerte Signatur mittlerer Sicherheit

rens sind dabei wiederum Hashwerte die aus der ursprünglichen Nachricht gebildet werden. Hätte man hier zur Verschlüsselung wieder RSA benutzt, so bräuchte man für die hohe Sicherheitsstufe[8] zwei SMS als Signatur, was wohl ein Grund für die Wahl eines symmetrischen Verfahrens war.

Außerdem ist auf der Abbildung zu erkennen, dass bei ausgelagerten Signaturen viel Platz innerhalb des envelope ungenutzt bleibt. Die eigentlichen Nutzdaten, die in den Signierprozess eingehen, sind nur eine 160 Bit große SHA-1 Ausgabe.

Verschlüsselung hoher Sicherheitsstufe, integrierte Signatur hoher Sicherheitsstufe

Abbildung 2.6 verdeutlicht die Teilung der verschlüsselten Informationen auf mehrere Pakete einer Nachricht hoher Sicherheitsstufe. Auch dieser Operationsmodus benötigt, wie der Modus des letzten Abschnittes, zum Versand mindestens zwei SMS. Hier sind die Nutzdaten jedoch mit einem Schlüssel hoher Sicherheitsstufe signiert. Es gelten die Einschränkungen für integrierte Signaturen – der Empfänger benötigt den öffentlichen Schlüssel des Senders und es können keine bereits versendete SMS nachträglich signiert werden (ohne den Inhalt erneut zu übertragen). Wird hier die Nachricht so groß, dass die Daten auf mehrere SMS aufgeteilt werden müssen, so werden vier (bzw. sechs, acht usw.) SMS benötigt, da ein einzelnes Teilpaket stets zwei

[8]Wenn der Empfänger einen Schlüssel mit hoher Sicherheitsstufe besitzt.

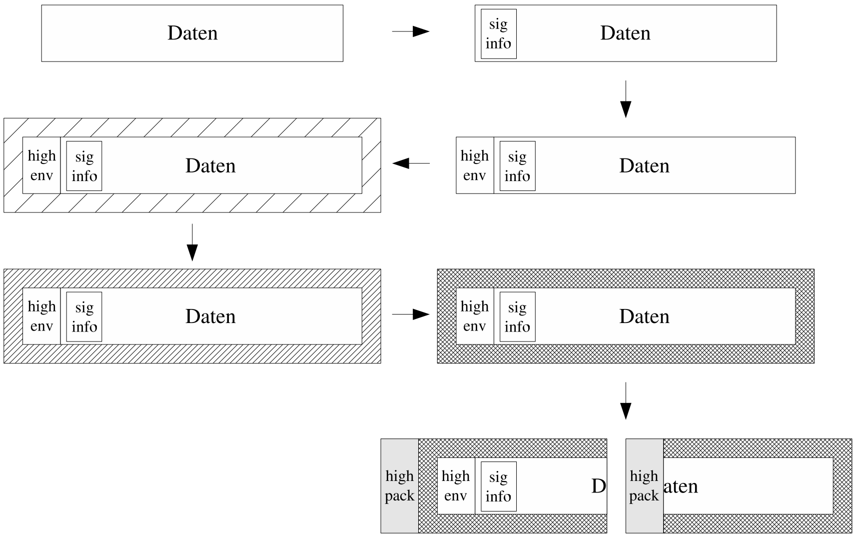

Abbildung 2.6: Verschlüsselt und integriert signiert hoher Sicherheitsstufe

SMS ausfüllt. Die ausgelagerte Signatur ist in jedem Fall immer nur eine (bei mittlerer Sicherheitsstufe) bzw. zwei SMS (bei hoher Sicherheitsstufe) lang.

Zusammenfassung

Abbildungen 2.7 bis 2.9 stellen Übersichten aller erlaubten Operationsmodi in SEMS dar. Abbildung 2.7 zeigt die Struktur der zu versendenden Daten für unverschlüsselte Übertragung, Abbildung 2.8 für Verschlüsselte mittlerer Sicherheitsstufe und Abbildung 2.9 für hohe Sicherheitsstufe.

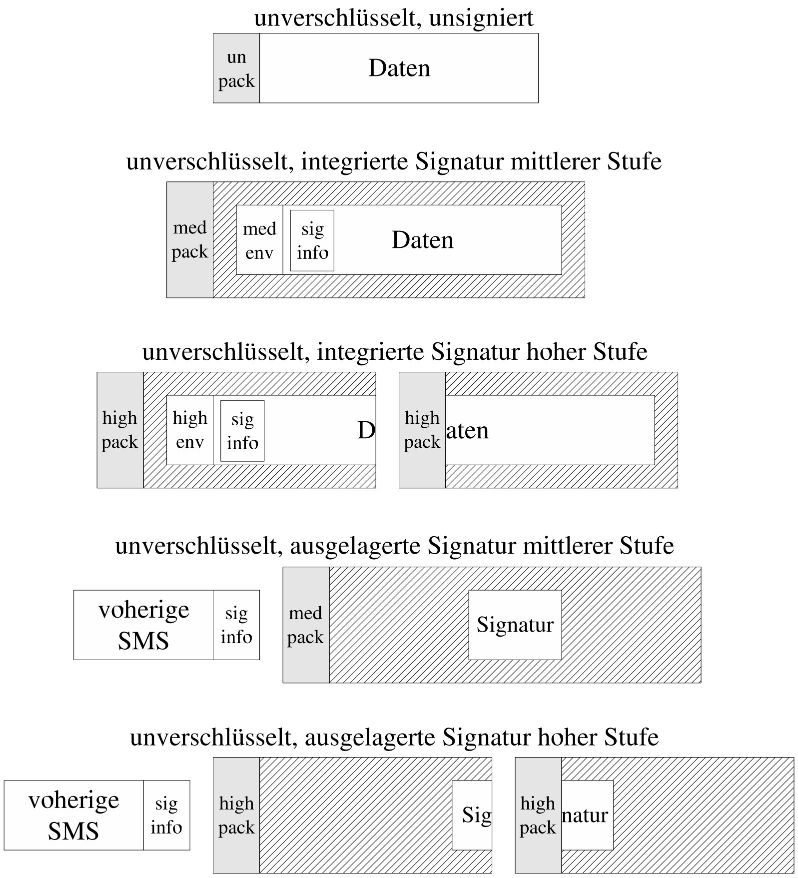

Abbildung 2.7: Operationsmodi für unverschlüsselte Übertragung

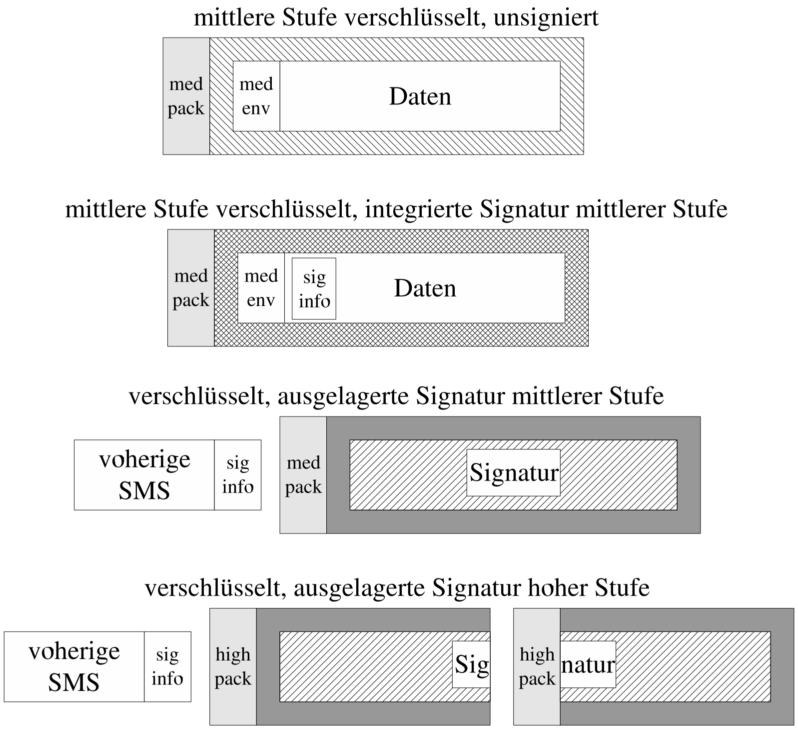

Abbildung 2.8: Operationsmodi für verschlüsselte Übertragung mittlerer Sicherheitsstufe

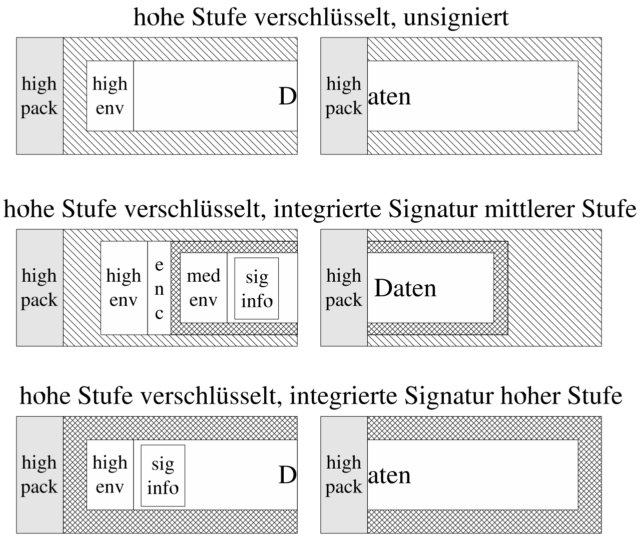

Abbildung 2.9: Operationsmodi für verschlüsselte Übertragung hoher Sicherheitsstufe

2.3 Kryptoanalyse

Es folgt nun eine einfache und kurze Analyse der kryptographischen Sicherheit von SEMS. Der vorgeschlagene Standard soll dazu unter folgenden Gesichtspunkten betrachtet werden:

- Sicherheit der benutzten Komponenten und deren Wechselwirkung
- Sicherheit der Schlüssel, Schlüsselmanagement
- Sonstige Probleme

2.3.1 Sicherheit der benutzten Komponenten und deren Wechselwirkung

SEMS setzt verschiedene kryptographische Verfahren und Algorithmen ein. Der zu benutzende Algorithmus mit seinem Arbeitsmodus und seiner Schlüssellänge sind vom Standard fest vorgeschrieben.

RSA

Im Mittelpunkt der asymmetrischen Kryptographie steht RSA. Es arbeitet auf Basis des Faktorisierungsproblems und gilt gegen passive Angreifer als sicher. Bis heute ist kein erfolgreicher Angriff auf RSA im allgemeinen Fall bekannt. SEMS benutzt jedoch einen besonders performanten Modus von RSA, bei dem der öffentliche Schlüssel auf den Wert 3 festgesetzt ist. Hier existieren zahlreiche Probleme und Schwächen, denen sich auch SEMS stellen muss. Da dies wohl den offensichtlichsten und umfangreichsten Aspekt der Kryptoanalyse darstellt, wird das Problem „RSA direkt“ in Abschnitt 2.4 gesondert diskutiert.

SHA-1

Zur Bildung aller Hashwerte wird SHA-1 eingesetzt. Die Ausgabe des Verfahrens beträgt 160 Bit und ist somit ausreichend sicher gegen Durchprobieren aller Möglichkeiten. Bei Hashverfahren benötigt man im Schnitt nur $2^{länge/2}$ Versuche um eine Kollision zu finden. Aber auch 2^{80} Versuche können heutzutage nicht iteriert werden. Außerdem enthält die Eingabe zu SHA immer Redundanz, was die Suche nach Kollisionen erschwert.[9] Es ist bis heute kein erfolgreicher Angriff auf die Kollisionsresistenz von SHA-1 bekannt.

Allerdings wird nicht nur die Kollisionsresistenz als Sicherheitsmerkmal benötigt. Der Schlüssel und IV für AES, welches zur Verschlüsselung von

[9] Die Suche wird schwerer, weil nicht alle Eingaben zulässig sind. Dies behindert Verfahren, die mathematisch versuchen den Hashalgorithmus umzukehren, nicht aber die Suche mittels Durchprobieren.

ausgelagerten Signaturen eingesetzt wird, gewinnt man durch Konkatenation und Addition modulo zwei aus zwei SHA-1 Operationen. Die Sicherheit dieser AES-Operation könnte daher durch SHA-1 abgeschwächt werden. Eine hinreichende Forderung an das Hashverfahren wäre hier eine Ausgabe, die in vertretbarem Aufwand ohne Kenntnis der Eingabe nicht von einer Pseudozufallszahl unterscheidbar ist.[10] Man nimmt allgemein an, dass diese Forderung unterstützt wird, wobei die Bedenken hier größer sind als bei der Kollisionsresistenz von SHA-1.

AES

Das bereits erwähnte AES wird zur Absicherung ausgelagerter Signaturen eingesetzt. Dabei wird AES im CFB Modus mit einer Schlüssellänge von 256 Bit und einem IV von 128 Bit verwendet. Der Standard lautet hier wie folgt (|| ist die Konkatenation):

$$K_1 = SHA(0x00 \mid\mid envelope)$$

$$K_2 = SHA(0x0000 \mid\mid envelope)$$

$$Schl\ddot{u}ssel = K_1 \mid\mid K_1\ XOR\ K_2$$

$$IV = K_2\{letzten\ 128\ Bit\}$$

Da der AES-Schlüssel nur 256 Bit lang ist, aber $K_1 \mid\mid K_1\ XOR\ K_2$ 320 Bit lang, werden vermutlich nur die ersten 256 Bit für den Schlüssel verwendet.[11]

Bei dieser Definition gilt also für AES, dass es nicht nur bei einem rein zufällig gewählten Schlüssel und IV sicher sein muss, sondern auch unter folgenden Einschränkungen:

- Die ersten 64 Bit des IV (und damit der Verschlüsselungsoperation) sind gleich der letzten 64 Bit des Schlüssels.
- Die ersten 160 Bit des Schlüssels sind eine SHA-1 Ausgabe deren Eingabe mit 0x00 (8 Nullbits) beginnt
- Die letzten 96 Bit des Schlüssels sind gleich der ersten 96 Bit XOR einer SHA-1 Ausgabe. Die Eingabe zu SHA-1 ist die vorne um nochmal 8 Nullbits erweiterte Eingabe der ersteren.
- Die anderen 64 Bit des IV sind der Rest der zweiten SHA-1 Ausgabe.

Geht man nun von der Tatsache aus, dass die Ausgabe von SHA-1 nicht von einer Pseudozufallszahl unterscheidbar ist, egal wie die Eingabe zu SHA-1 aussieht (siehe Analyse zu SHA-1 oben), so bleibt nur die erste Forderung bestehen. Es sind derzeit keine Studien von AES unter diesen Bedingungen bekannt.

[10] Wobei die Eingabe keine Zufallszahl sein muss.

[11] In Abschnitt 2.5.4 werden diese und andere Ungereimtheiten diskutiert.

SHAKE

Ein eigens für SEMS entwickeltes Verfahren zur Verteilung der Entropie ist SHAKE. Der invertierbare Algorithmus hat die Funktion, alle Bits der Eingabe mathematisch komplex über die gesamte Eingabe zu verstreuen. Das Grundprinzip besteht dabei darin, SHA-1 Ausgaben modulo 2 auf die Eingabe zu addieren. Die Entropie der Ausgabe von SHAKE ist mindestens so zufällig verteilt wie die der Ausgabe von SHA-1.[12]

2.3.2 Sicherheit der Schlüssel, Schlüsselmanagement

Nicht nur die Sicherheit der verwendeten Verfahren und Algorithmen ist wichtig für ein kryptographisches System, sondern auch, falls es denn damit arbeitet, das Management der Schlüssel. Während die meisten untersuchten Kryptosysteme für Kurznachrichten überhaupt nicht oder nur sehr spärlich auf das Schlüsselmanagement eingehen, glänzt SEMS hier mit eigenen Datenstrukturen für Schlüssel, Schlüsselwiderruf und einer Option, den Schlüssel eines Kommunikationspartners anzufordern. Schlüssel können signiert werden und jeder Schlüssel kann bis zu 32 Signaturen enthalten.

In keinem Teil des vorgeschlagenen Standards wird verlangt, dass der geheime Schlüssel in irgendeiner Form preisgegeben werden oder transferiert werden muss. Der Schlüssel ist an eine Telefonnummer, einen Namen und ein Verfallsdatum gekoppelt welches auch „unendlich“ betragen kann.

2.3.3 Sonstige Probleme

Einige kleinere Unzulänglichkeiten existieren, die vermutlich auf Fehler bei der Erstellung des Dokumentes für den Standard zurückzuführen sind. Das Dokument [SEMS] enthält selber leider keine Versionsnummer, es beschreibt den Standard in der Version 1.0.

Einige der hier aufgeführten Punkte werden noch einmal in 2.5.4 aufgelistet und genauer spezifiziert. Hier geschieht die Untersuchung auf sicherheitsrelevante Aspekte.

- In der Beschreibung der Operationsmodi von SEMS wird SHAKE nur für unverschlüsselte Transportobjekte erwähnt. Dies ist so nicht sicher. SHAKE muss immer auf den Klartext angewendet werden, bevor auf diesen eine RSA-Operation angewandt wird.

- An einigen Stellen werden Referenzen auf falsche Strukturen genannt, vermutlich aufgrund von Kopier- und Einfügeoperationen. So soll z.B.

[12] Dies gilt nur unter der Voraussetzung, dass die Ausgabe von SHA-1 statistisch unabhängig von der Eingabe ist. Wenn die Annahme der Ununterscheidbarkeit mit Pseudozufallszahlen von oben gilt, ist dies auch gegeben.

in Abschnitt 2.4.3 des SEMS-Standards eine Signatur hoher Sicherheitsstufe in einen Umschlag mittlerer Stufe eingebettet werden. Dies ist unsicher und wahrscheinlich auch nicht im Sinne des Erstellers.

- Die Länge des Payload in einem High security crypto envelope (Abschnitt 4.3.2) ist mit 11 Bit zu kurz um alle Nutzdaten von 2084 Bit zu spezifizieren. Ein Umschlag, der mehr als 2048 Bit Nutzdaten enthält, kann am Empfänger nicht korrekt dargestellt werden. Der nicht nutzbare Bereich wird dennoch übertragen und mitsigniert, was man beispielsweise als „unsichtbares Kleingedrucktes“ in einer Signatur missbrauchen kann.[13]

- SEMS enthält keine Spezifikation der Fehlermeldungen, so wie es beispielsweise bei PKCS der Fall ist. Bei einer Implementation muss man darauf achten, dass beispielsweise keine korrupte Ausgaben der RSA-Operationen zur Anzeige gebracht werden.[14] Andernfalls ist RSA und damit auch SEMS nicht gegenüber aktiven Angreifern sicher.

2.4 RSA direkt

Der Kerngedanke des Protokolls ist die direkte Benutzung von RSA als Verschlüsselung. Üblicherweise wird RSA nach dem Standard PKCS#1 und auch nur zum Verschlüsseln eines symmetrischen Sitzungsschlüssels benutzt, etwa bei PGP oder TLS/SSL. Mit diesem symmetrischen Schlüssel werden dann die Nutzdaten verschlüsselt. Die Verwendung von PKCS#1 hat gegenüber einem neuen Protokoll mit der direkten Benutzung von RSA mehrere Vorteile.

- Bereits die Tatsache ist ein Vorteil, dass PKCS#1 ein etablierter Standard ist.
- Die direkte Benutzung von RSA ist gegen viele aktive Angriffe unsicher.
- Die Performance des symmetrischen Teil der hybriden Verschlüsselung ist höher.
- Mit hybriden Verfahren können Daten einfacher für mehrere Teilnehmer lesbar verschlüsselt werden. Dazu werden die Daten einmal mit einem symmetrischen Schlüssel verschlüsselt, welcher dann mit den jeweiligen asymmetrischen Schlüsseln geschützt wird.

Im Folgenden werden diese Punkte genauer für das Szenario des Versendens von Kurznachrichten untersucht.

[13] Durch die Reduzierung der Nutzdaten in der Implementation dieser Arbeit verschwindet das Problem, siehe auch 2.5.1

[14] Korrupt bedeutet z.B., wenn die Zufallszahl im verschlüsselten Paket nach der Entschlüsselung nicht der im Header entspricht.

2.4.1 Bestehende Standards

Das schwerwiegendste Argument gegen SEMS ist, dass es ein neuer Standard ist. Andere Verfahren sind seit Jahren implementiert, etabliert und getestet. Es existieren einfache Bibliotheken, auch für Java-Handys, die OpenPGP oder TLS/SSL-Funktionalität zur Verfügung stellen (z.B. [BC]). Eine Implementation eines neuen Standards macht die darauf aufbauenden Anwendungen inkompatibel zu allen anderen bereits bestehenden Standards.

SEMS kann in diesem Gebiet keine Vorzüge aufweisen. Es ist praktisch unbekannt und es existieren keine vollständigen Implementationen.

Allerdings sind alle bisherigen Standards nicht optimal, wenn es um die Anzahl zu versendender SMS geht. Bei Verfahren, die auf PKCS#1 beruhen, ist die kleinste Nachricht immer noch mindestens etwa 1500 Bit lang. Geht man nun von der Annahme aus, dass das Abrechnungsschema für SMS auch in naher Zukunft immer noch auf Nachrichtenbasis stattfindet, so ist ein Standard notwendig, der diese Zahl der versendeten Nachrichten minimiert.

2.4.2 Angriffe auf RSA

Da SEMS RSA direkt benutzt, muss es sich mit allen bekannten Angriffen auf dieses Verfahren auseinandersetzen und diesen begegnen (für eine gute Zusammenfassung einiger dieser Angriffe siehe [RSAAttack] oder [20YearRSA]). Darüber hinaus existieren noch eine Reihe anderer Angriffe, die sich nicht direkt auf den Algorithmus beziehen, wie Zeitmessungen der Entschlüsselungsoperation oder das Benutzen von Quantencomputern. Solche „esoterisch" genannten Angriffe werden in dieser Arbeit nicht weiter betrachtet.

SEMS schreibt einen öffentlichen Exponenten $e = 3$ zwingend vor. Verschiedene erfolgreiche Angriffe auf RSA sind für kleine Exponenten bekannt. Tabelle 2.10 listet einige solche Angriffe und Sicherungsmöglichkeiten dagegen auf.

SEMS fügt 80 Bit zufälliges Padding an den zu verschlüsselnden Klartext an. Zudem wird auf die Eingabe [SHAKE] angewandt, was die Zufälligkeit über die ganze Nachricht verteilt. Dadurch werden einige der oben genannten Angriffe vereitelt. Dennoch ist SEMS unsicher gegen Angriffe bei teilweise bekanntem, privatem Exponent.

Der Angriff von Boneh, Durfee und Frankel, später von Blömer präzisiert, ist erfolgreich für Exponenten in RSA deren Länge kleiner als $\sqrt{l}$ wobei l die Länge des Modulus ist. Bei einem maximalen Modulus der Länge 2208 Bit wäre dies also für Exponenten unter ca. 2^{47}.

Die Schwäche gegen diese Art von Angriffen lässt sich einfach beheben, indem ein anderer fester, aber größerer Exponent als $e = 3$ gewählt wird. Eine Wahl eines Exponenten, der gegen diese Angriffe als sicher gilt, wird unter Abschnitt 2.5.2 getroffen.

Angriff	Bedingung	Sicherung in SEMS
[Has88]	Linear abhängige Klartexte mit verschiedenen Moduli	Nicht linear abhängiges Padding
[CFPR96]	Linear abhängiger Klartexte mit gleichem Modulus	Nicht linear abhängiges Padding
[Cop97]	Einseitiges Padding nur für MSB oder LSB	Padding mit Hash „verteilen“
[Cop97]	Klartext sehr klein	Padding um Klartext zu vergrößern
[BDF98], [Blö03]	Teile des privaten Exponenten bekannt	kein Schutz

Abbildung 2.10: Angriffe auf RSA mit kleinem öffentlichen Exponenten

2.4.3 Performance von RSA

Die Performance ist für das Ausführen von symmetrischen Verfahren um den Faktor 1000 besser. Dies nutzt man bei PKCS#1 aus, um nur einen kleinen symmetrischen Schlüssel zu verschlüsseln, mit welchem dann die meist wesentlich längere Nachricht symmetrisch kodiert wird. Im Falle von Kurznachrichten ist dies jedoch unnötig, da hier die Nachrichten selbst meist viel kürzer als ein symmetrischer Schlüssel sind. Außerdem spielt selbst bei einem Einsatz auf langsameren Endgeräten die Performance eine stark untergeordnete Rolle, die Übertragung der SMS selbst beansprucht oft weit mehr Zeit.

2.4.4 Versand an mehrere Teilnehmer

Auch der Vorteil der hybriden Systeme Daten so zu verschlüsseln, dass mehrere Teilnehmer entschlüsseln können, ist hier nicht notwendig. Bei SMS wird jede ausgehende SMS einzeln abgerechnet, ein Versand an mehrere Teilnehmer gleichzeitig ist nicht möglich bzw. führt dazu, dass an jeden Teilnehmer einzeln eine Nachricht versendet wird.

2.5 Änderungen an SEMS

Um die Implementation in Java zu ermöglichen, war eine Änderung am Standard zwingend notwendig. Der „User Data Header“, der zur Adressierung von Ports in Java-Midlets notwendig ist, verkürzt den zur Verfügung stehenden Nachrichtenraum. Außerdem kann eine bessere Sicherheit bei alternativer Wahl des öffentlichen Exponenten erzielt werden. Schließlich werden noch einige kleine Ungereimtheiten im Text des Standards genauer definiert. Um den Unterschied der Versionen des Standards festzusetzen, wird die Versionsnummer des Standards eigenmächtig auf 1 (binär 001) erhöht, in der Hoffnung, nicht mit anderen Verbesserungen an SEMS zu kollidieren.[15]

2.5.1 User Data Header

Der User Data Header (UDH) ist eine Erweiterung an SMS, die ab Version 6.3 des SMS Standards [TS23.040] enthalten ist. Der UDH ist eine Struktur variabler Größe, der Metainformationen über die Kurznachricht enthält wie zum Beispiel die Größe oder den Stil der Schriftart des Textes, aber auch den Typ der Nachricht und vor allem den sogenannten Quell- und Zielport. Diese Technik wurde dem TCP/IP-Stack entlehnt. Eine Zahl gibt den Quellport, eine weitere den Zielport der Kurznachricht an. Damit wird es ermöglicht, Nachrichten an bestimmte auf dem Zielhandy laufende Midlets zu versenden.

Allerdings wird der User Data Header im Datenbereich der Nutzdaten platziert. Dadurch steht für Anwendungen, die dieses Feature nutzen, eine geringere Nachrichtenlänge zur Verfügung.

Die Wireless Messaging API [WMA] schreibt vor, dass eine an einen Zielport versendete 8-Bit SMS mindestens 133 Bytes Daten enthalten kann.[16] Eine Applikation, die auf unterschiedlichen WMA-Plattformen laufen soll, muss dies als obere Grenze für die Nutzdaten beachten. SEMS nutzt stets die komplette Nachrichtenlänge aus. Verschiedene Möglichkeiten existieren um dieses Problem zu lösen.

Problem ignorieren – EMS-Splitting

Wird das Problem ignoriert, so tritt nach dem WMA-Standard das Aufteilen der Nachricht in mehrere SMS in Kraft, wodurch für jede versendete SMS innerhalb von SEMS stets eine zusätzliche SMS benötigt wird. Zwar muss man so den Standard nicht ändern, jedoch widerspricht diese Lösung stark den Grundprinzipien des SEMS-Designs, so wenig SMS wie möglich zum Versand zu benötigen. Bei einer solchen Implementation werden andere Produkte und Standards schnell zu einer weit besseren Wahl.

Kein UDH

Der UDH ist eine optionale Komponente. Ohne ihn kann eine SMS nicht direkt an ein Midlet adressiert werden. Allerdings existieren verschiedene MIDP-Erweiterungen der Hersteller, die es ermöglichen, entweder unadressierte SMS als Grundeinstellung an bestimmte Midlets zu senden (Siemens) oder Midlets im Nachhinein mit bestimmten SMS zu verknüpfen (Nokia). Der klare Vorteil dieser Methode ist, dass der SEMS-Standard nicht angepasst

[15] Entwickler Rüdiger Weis verneinte die Existenz anderer Versionen von SEMS.

[16] Eine normale 8-Bit SMS enthält 140 Byte Daten.

werden muss. Der Nachteil, man verliert die MIDP-Standardkonformität, wiegt jedoch weit mehr auf.

Nutzdaten verkürzen

Es wäre möglich, die Nutzdaten, die SEMS zur Verfügung stellt, um die Größe des UDH zu verkürzen, so dass dieser vor das eigentliche SEMS-Paket gestellt werden kann. Die Änderung bedeutet, dass Nutzer der Implementation 7 Bytes Nachricht pro SMS weniger versenden können, dies entspricht etwa 8-10 Zeichen bei mittlerer Sicherheitsstufe und 16-20 Zeichen bei hoher. Bei verschlüsselten und intern signierten Nachrichten mittlerer Sicherheitsstufe (der Modus mit dem größten Overhead pro SMS) reduziert dies die effektive Nachrichtenlänge von 795 Bits auf 739 Bits. Das entspricht etwa 7 Prozent. Es muss dafür auch die Länge des Schlüssels auf 1056 bzw. 2096 Bits gekürzt werden, da durch eine RSA-Operation auf dem security envelope mit einem vollen Schlüssel die Originallänge wieder hergestellt wäre.

Header verkürzen

Man kann versuchen, unnötige Felder im Header der SEMS-Datenobjekte zu kürzen um so Platz für den UDH zu schaffen. Die CRC32 – Checksumme ist nicht zwingend notwendig, die Konsistenzprüfungen kann aufgrund anderer Redundanz der Datenstrukturen genügend genau stattfinden. Dadurch „gewinnt“ man 4 freie Bytes. Die restlichen Byte in der mittleren Sicherheitsstufe könnten durch Reduzierung der Zufallszahl auf 56 Bit gewonnen werden. Für die hohe Stufe ist dies nicht ohne weiteres möglich, hier müsste man die komplette Zufallszahl streichen.

Zusätzlich muss noch die Schlüssellänge um 56 bzw. 112 Bit verkürzt werden.

Mischform - Header und Nutzdaten verkürzen

In der vorliegenden Arbeit wird der Platz für den UDH durch Wegstreichen von Informationen aus Nutzdaten und Header gewonnen. Als Erstes wird die CRC-Prüfsumme aus allen Strukturen gestrichen. Für ein unverschlüsseltes oder verschlüsseltes Paket mittlerer Sicherheitsstufe müssen noch die Nutzdaten auf 1045 bzw. 965 Bit verringert werden.

Die Schlüssel verkürzen sich ebenfalls auf 1056 bzw. 2096 Bits. Dafür müssen überall im Fließtext des Standards die Angaben der Schlüssellängen angepasst werden. Folgende Stellen müssen ebenfalls geändert werden:

- 4.1 Unencrypted – Das Feld CRC wegstreichen und den Payload von 1069 auf 1045 Bit.
- 4.2.1 Medium security packet format – Die Länge des „Medium security crypto envelope“ von 1112 auf 1056 Bit.

- 4.2.2 Medium security crypto envelope – Das Feld CRC wegstreichen und den Payload von 989 auf 965 Bit.
- 4.3.1.1 und 4.3.1.2 High security packet – Den „Security crypto envelope“ von 1104 auf 1048 Bit.
- 4.3.2 High security crypto envelope – Das Feld CRC wegstreichen und den Payload von 2084 auf 2004 Bit.[17]
- 5.3 Key – das Feld „Public Key“ von „1112 or 2208“ auf „1056 or 2096“ ändern.

Durch die Verkürzung der Schlüssellänge ergeben sich ein Vor- und ein Nachteil in der Implementation. Ein Vorteil entsteht dadurch, dass beide Schlüssellängen nun durch 16 und deren Faktoren damit durch 8 teilbar sind, was bedeutet, dass beide Primfaktoren p und q des Modulus für RSA auf Bytegrenzen ausgerichtet sind. Sie sind in ganzen Bytes speicherbar, was die Verarbeitung erleichtert. Allerdings ist die Größe eines High Security Envelope mit 2004 Bits nun nicht mehr durch 8 teilbar, was die Kryptographischen Operationen auf solchen Datenstrukturen erschwert. Dies ist ein Grund, warum die hohe Sicherheitsstufe nicht Bestandteil der Beispielimplementation ist, siehe auch Abschnitt 3.5.[18]

2.5.2 Öffentlicher Exponent

SEMS definiert den öffentlichen Exponenten der RSA-Operation kategorisch auf $e = 3$. Abschnitt 2.4.2 befasst sich mit den sicherheitstechnischen Implikationen dieser Festlegung. In dieser Arbeit wird statt dessen der Exponent $e = 0x4000000800001$ vorgeschlagen. Dies ist die kleinste Primzahl mit den wenigsten Einsen, die größer als 2^{50} ist. Mit diesem Exponent ist RSA sicher gegen den Angriff von Blömer.[19]

Die Wahl des Exponenten $e = 3$ folgte auf Nachfragen bei den Entwicklern lediglich aus Gründen der Performance. Messungen über die Geschwindigkeit der RSA-Operation werden unter Abschnitt 3.6.3 behandelt. Zusammenfassend kann gesagt werden, dass die Geschwindigkeit einer Verschlüsselung oder eines Signaturtests nicht unter dem in dieser Arbeit vorgeschlagenen Exponenten leidet.

SEMS sieht die Wahl von alternativen Exponenten in der Datenstruktur für Schlüssel bereits vor ([SEMS], Abschnitt 5.3). Eine Änderung ist also nur in der konstanten Belegung des Feldes „Key Type“ notwendig. Hier wurde der Exponent $0x4000000800001$ auf den Wert 2, also 010 binär, definiert.

[17] Dadurch fällt auch der sonst nicht nutzbare Bereich zwischen Bit 2048 und Bit 2083 weg, siehe Abschnitt 2.5.4.

[18] Ein Einfluss auf die Performance war durch die „krumme“ Bitgröße nicht feststellbar, siehe Abschnitt 3.6 für mehr.

[19] Das Programm „FindFastPrime.java“, welches zum Finden der Primzahl führte befindet sich auf der beiliegenden CD.

2.5.3 Durchprobieren passender Schlüssel

Abschnitt 2.1.1 des vorgeschlagenen Standards nennt als eine Bedingung für den erfolgreichen Empfang einer SMS mit integrierter Signatur, dass die Telefonnummer des Senders während der Übermittlung erhalten bleibt und passend zu einem gespeicherten Telefonbucheintrag beim Empfänger vorliegt. Der Autor dieser Arbeit teilt diese Meinung nicht. In der Beispielimplementation werden bei Empfang einer integriert-signierten SMS stets alle im Adressbuch gespeicherten öffentlichen Schlüssel nacheinander durchprobiert, beginnend mit denen, die am wahrscheinlichsten zutreffen (etwa weil deren Telefonnummer mit der des Senders übereinstimmt). Somit können auch „anonyme“[20] SMS korrekt empfangen werden. Für eine Diskussion der Performance beim Empfangen von SMS siehe auch Abschnitt 3.6.

2.5.4 Ungereimtheiten

Einige unklare Definitionen und Ungereimtheiten des vorgeschlagenen Standards wurden während der Kryptoanalyse und Implementation bemerkt. Zur Abgabe dieser Arbeit lagen keine Klärungen durch SEMS-Entwickler vor. Folgende Vermutung werden aufgestellt:

- Länge eines high security envelope (4.3.2) – Die maximale Länge des payload eines high security envelope ist auf 2084 festgesetzt. Allerdings stehen zur Angabe nur 11 Bit zur Verfügung, was einer maximalen Größe von 2048 Bit entspricht. Dieses Problem wird schon durch die Verkürzung der Nutzdaten aufgrund des User Data Header behoben, siehe 2.5.1.

- Fehlendes SHAKE – Bei der Beschreibung der Operationsmodi fehlt vermutlich an einigen Stellen die SHAKE-Operation. Eine Punkt mit dem Text „perform SHAKE“ wird als ein neuer 3. Punkt an folgenden Stellen angenommen: 2.3.1, 2.3.2[21], 2.4.1, 2.4.2[21], 2.4.3[21]

- Falsche Sicherheitsstufe (2.4.3) – In der Detailbeschreibung des Operationsmodus „high security included signature“ sollen die Daten in ein „medium security envelope“ eingefügt werden. Hier ist sicher ein „high security envelope“ gemeint.

- Jahr des timestamp (5.5) – Der Datentyp „timestamp“ ist definiert als „minutes since Jan 1st“. Das Jahr bleibt undefiniert. Dies ist vermutlich

[20] Die eigene Telefonnummer zu invalidieren stellt selbstverständlich keine Anonymität her, besonders nicht, wenn die SMS wie hier mit dem eigenen Schlüssel signiert wurde.

[21] Diese Stellen können diskutiert werden. Hier wird die Ausgabe einer secret-key-Operation mit nicht-SHAKE-Eingabe weiter verschlüsselt. Daher ist SHAKE nicht zwingend erforderlich. Aufgrund der Aussagen Rüdiger Weis: „SHAKE wird auf jede Klartexteingabe an RSA angewandt“ sollte hier dennoch eine SHAKE-Operation vorangestellt werden.

nach Unix-Tradition 1970, was ein maximalen Zeitstempel von Mitte 2097 bedeutet – mehr als ausreichend.

- Ausrichtung der SHA-1 Eingabe – Die Eingabe zu einer SHA-1 Operation muss auf Bytegrenzen ausgerichtet sein. Hierzu werden Eingaben üblicherweise mit Nullen aufgefüllt. Bei Operationen auf großen Zahlen wie bei RSA geschieht dies häufig am Anfang, für Blockoperationen wie SHA-1 oder symmetrische Chiffre meist am Ende. Für SeJI werden notwendige Nullen am Ende aufgefüllt.

- AES-CBF (3.3) – Für AES ist der Modus „CBF“ vorgeschrieben. Hier ist wohl statt dessen „CFB“ (cipher feedback) gemeint.

- AES-320 (3.3.1) – Der Schlüssel von AES-256 ist nach der Definition 320 Bit lang. Vermutlich sollen nur die ersten 256 Bit benutzt werden und der Rest abgeschnitten.

- Ein unnötiges X (3.3.1) – Bei der Definition der Schlüsselerzeugung für AES ist eine Variable $X = K_1 \mid\mid K_2\{first\ 96\ bits\}$ definiert, die nie benutzt wird. Die Variable wird ignoriert.

- Packet ID (4.3.1) – Die Belegung des Feldes „Packet ID“ ist undefiniert. Damit ist vermutlich ähnlich TCP/IP eine eindeutige ID über einen bestimmten Zeitraum gemeint um die beiden high security packets einander zuzuordnen. In SeJI wird dieser Wert für jedes SMS-Paar inkrementell erhöht.

2.6 Verworfene Änderungen

Während der Analyse und Einarbeitung wurden mehrere Ideen und Verbesserungsvorschläge wieder verworfen. Diese sollen hier kurz vorgestellt werden. Einige Ideen entstanden durch den Versuch den Standard unkomplizierter und überschaubarer zu gestalten, andere waren Erwägungen die Geschwindigkeit oder Akzeptanz zu erhöhen. Die Ablehnung dieser Ideen entstand zum Teil aus Diskussionen mit Rüdiger Weis, einem der Erfinder von SEMS.

2.6.1 SEMS nur für den Austausch symmetrischer Schlüssel

Eine radikale Änderung des Standards wäre, dass der bisherige asymmetrische Mechanismus lediglich dazu dient, symmetrische Benutzerschlüssel auszutauschen. Diese gelten dann für einen vorher festgelegten Zeitraum und werden automatisch erneuert. Die Vorteile, die dadurch erzielt werden sollen, sind bessere Geschwindigkeit und längere nutzbare Nachrichten.

Geschwindigkeit ist jedoch nicht von großer Bedeutung. Da die SMS asynchron übertragen wird, kommt es bei der Kodierung der Nachricht auf Sekunden nicht an (siehe dazu auch 1.5.5). Die angeblich größere nutzbare

Nachrichtenlänge ist nur illusorisch, da eine RSA-Operation die Eingabedaten nicht verlängert, also ebenfalls längentreu ist.[22]

Demgegenüber stehen gravierende Sicherheitsbedenken. Das Schlüsselmanagement wird anfälliger, da nun geheime Schlüssel transportiert werden. Der größte Nachteil liegt jedoch in den Signaturen. Durch symmetrische Authentikationscodes kann man Nachrichten lediglich gegenüber den beteiligten Partnern absichern. Eine Beweiskraft vor Dritten, wie es etwa bei kommerziellen Diensten wünschenswert wäre, ist nicht möglich.[23]

2.6.2 Elliptische Kurven

Ein Gedanke, der beim Design von Sicherheitsapplikationen auf Geräten mit begrenzten Ressourcen besonders attraktiv erscheint, ist Kryptographie auf Basis von elliptischen Kurven. Verfahren auf dieser Basis arbeiten schneller als Verfahren wie RSA, die auf Potenzen beruhen. Der Speicherverbrauch für Schlüsselspeicherung sinkt auch rapide – 160 Bit Schlüssel in der elliptische Kurvenkryptographie entsprechen vermutlich 1024 Bit RSA Schlüsseln, was die Anzahl der benötigten SMS für den Schlüsselaustausch auf 1 reduziert.

Die Formulierung „vermutlich“ drückt bereits ein Problem der elliptischen Kurven aus. Das zugrunde liegende mathematische Modell ist erst seit etwa 25 Jahren im Mittelpunkt der Mathematik (durch die Benutzung in der Kryptographie). Demgegenüber versuchen Mathematiker schon seit über 100 Jahren, Zahlen möglichst effizient zu faktorisieren. Fortschritte in der Algorithmentechnik stellen hier also ein Sicherheitsrisiko dar.[24] Und da SMS ohnehin asynchron versendet werden, ist die Rechendauer der Operationen nicht von besonderer Bedeutung. Auch die Kapazität des persistenten Speichers der Handys ist mittlerweile mehr als ausreichend.

Der größte Nachteil der elliptischen Kurven liegt jedoch in der Beschränkung auf Verfahren, die auf dem diskreten Logarithmus beruhen. Üblicherweise wird zur Verschlüsselung ElGamal eingesetzt, während zur Authentikation DSA zum Einsatz kommt. ElGamal ist nicht längentreu. Die verschlüsselte Nachricht ist doppelt so lang wie das Original. Das beschränkt die nutzbare Länge der SMS erheblich. Setzt man statt ElGamal das kryptographisch stärkere Verfahren von Cramer und Shoup ([CS98]) ein, so wird das Problem noch gravierender denn CS98 vervierfacht seinen Schlüsseltext. Längentreue und gut untersuchte Chiffren, auf die elliptische Kurven anwendbar sind, sind dem Autor nicht bekannt.

[22] Abgesehen von dem ersten Bit, das immer auf 0 gesetzt wird.

[23] Außer man bedient sich einer dritten, vertrauenswürdigen Instanz als Mittler. Es fallen dann Kosten für zusätzliche SMS von und zu dieser Instanz an.

[24] Es ist zu bemerken, das es nicht unbedingt notwendig ist, Zahlen zu faktorisieren oder diskrete Logarithmen in Körpern über elliptischen Kurven zu berechnen um RSA bzw. ECC-ElGamal zu brechen. Ein Beweis für die Äquivalenz der Probleme steht noch aus.

2.6.3 Weglassen der hohen Sicherheitsstufe

Der Standard benötigt in der momentanen Form viele Operationsmodi nur, um die reibungslose Kommunikation zwischen Schlüsseln mittlerer und hoher Sicherheitsstufe zu gewährleisten. Eine starke Vereinfachung kann man erzielen, indem eine der beiden Stufen komplett entfernt wird. Da die hohe Sicherheitsstufe immer mehrere SMS zum Versand benötigt, sollte diese aus dem Standard entfernt werden.

Neueste Bestrebungen in der Faktorisierung ([ShTr03], [DS81]) lassen jedoch Zweifel aufkommen, ob 1120 Bit lange genug ausreichend sein werden. Da der Standard sich auf eine feste Bitzahl beschränkt, muss diese so gewählt sein, dass auch in Zukunft keine Sicherheitsbedenken die Akzeptanz beeinträchtigen.

2.6.4 1024 und 2048 Bit Schlüssel zulassen

1120 und 2208 Bit sind ungewöhnliche Schlüssellängen. Auch die in SeJI benutzte Schlüssellänge von 1056 oder 2096 Bit sind nicht üblich. Es ist unwahrscheinlich, dass Anwender bereits geheime Schlüssel dieser Längen besitzen. Erlaubte man beispielsweise den Import von typischen Schlüsseln mit der Länge 1024 und 2048, so könnten Anwender bestehende Infrastrukturen und Schlüssel weiterverwenden.

So zumindest war die Idee. Allerdings benutzt SEMS einen festen Exponenten des öffentlichen Schlüssels, während andere Standards wie PKCS dies nicht vorschreiben oder sogar verbieten.[25] Ein Import von Schlüsseln wäre also nur für diejenigen möglich, die ebenfalls diesen Exponenten einsetzen. Den Exponenten Teil des öffentlichen Schlüssels machen, so wie in anderen Standards, würde die Zahl der benötigten SMS pro Schlüsselaustausch erhöhen.

Außerdem ist die Vorgehensweise kritisch, geheime Schlüssel über Rechnergrenzen hinweg zu transportieren. Besonders bei der Übertragung auf ein mobiles Gerät muss dies sorgfältig geschehen, damit der Schlüssel nicht während des Transports abgehört werden kann. Sicherer ist es, wenn geheime Schlüssel das Gerät, auf dem sie erzeugt werden, niemals verlassen – wenn erst gar keine Schnittstelle zum Transport dieser Schlüssel existiert.

Zudem reduzieren kürzere Schlüssel die maximal nutzbare Nachrichtenlänge, wodurch Platz in der SMS ungenutzt bleibt.

2.6.5 AES durch RSA ersetzen

Die AES-Operation wird nur an einer Stelle des Standards benötigt, beim Verschlüsseln von ausgelagerten Signaturen. Wie Abschnitt 2.3.1 zeigt, existieren einige Vorbehalte gegenüber der Schlüsselgenerierung. Außerdem wür-

[25] Siehe Abschnitt 2.4 für eine Diskussion über Probleme mit $e = 3$.

de dies den Standard vereinfachen, da nun als einzige kryptographische Grundlage RSA eingesetzt wird. Die Sicherheit wird durch den Einsatz von AES nicht größer, denn wenn ein Angreifer nur RSA brechen kann, reicht dies bereits, um vorangegangene Nachricht entschlüsseln zu können, aus der sich dann der Klartext der ausgelagerten Signatur berechnen ließe.

Ein Gedanke war nun, die AES- durch eine RSA-Operation mit dem öffentlichen Schlüssel des Empfängers zu ersetzen. Die Idee wurde aus zwei Gründen verworfen. Erstens benötigt man eine zusätzliche SMS, wenn der Empfänger einen Schlüssel hoher Sicherheitsstufe besitzt. Und zweitens könnte man die ausgelagerte Signatur nicht verschlüsseln, wenn man keinen öffentlichen Schlüssel des Empfängers besitzt. Der zweite Grund mag eventuell seltsam erscheinen, da aus der Kenntnis der vorangegangenen Nachricht die ausgelagerte Signatur berechnet werden kann. Es scheint keinen Sinn zu haben, die eigentliche Nachricht unverschlüsselt, die ausgelagerte Signatur aber verschlüsselt zu versenden. Dazu muss man jedoch bedenken, dass der Versand der ausgelagerten Signatur zeitlich durchaus weit nach der eigentlichen Nachricht erfolgen kann[26] und der ursprüngliche Schlüssel bereits abgelaufen oder zurückgezogen sein kann.

[26] Es ist möglich, nachträglich bereits versendete Nachrichten zu signieren.

Kapitel 3

SEMS Java Implementation

Im praktischen Teil wird die im Rahmen dieser Arbeit erarbeitete SEMS Java Implementation (SeJI) vorgestellt. Als erstes werden dazu die Anforderungen an die Software analysiert. Es folgt das Design der Softwarekomponenten. Anschließend wird die Implementation beispielhaft präsentiert. Es folgt eine kurze Präsentation von Testklassen für alle Komponenten. Abschließend werden einige Messungen der Performance von SeJI präsentiert.

SeJI wurde mit Hilfe des sogenannten „einfachen Wasserfallmodells“ implementiert, das die Aufgabe in die Teilgebiete Analyse, Design, Implementation und Test teilt, welche zeitlich voneinander getrennt bearbeitet werden. Rückwirkende Korrekturen, beispielsweise der Klassenmodelle aufgrund von implementationsspezifischen Notwendigkeiten, wurden nur begrenzt durchgeführt. Grund für diese Entscheidung war der relativ knappe Zeitrahmen, der für die Bearbeitung der Implementation zur Verfügung stand. Viele der Änderungen, die zu Beginn der Aufgabenstellung vom Autor geplant waren, später aber wieder verworfen wurden, hätten die Komplexität der Implementation erheblich reduziert (z.B. Weglassen der hohen Sicherheitsstufe oder symmetrische Verschlüsslung).

3.1 Analyse

Da SeJI den Secure Message Standard implementieren soll, lohnt es, zunächst dessen Struktur zu analysieren. SEMS kann man in drei Grundkomponenten zerlegen, welche durch drei Komponenten von SeJI zuzüglich einer “Sonstiges“-Komponente abgebildet werden:

- Senden von SMS – Nachrichten verschiedener Typen können gesendet werden. Die Nachricht wird dabei entsprechend gewählter Einstellungen kodiert.

- Empfang von SMS – Es können empfangene Nachrichten dekodiert werden. Da die Kodierung aufgrund von fehlenden oder ungültigen

Schlüsseln bzw. SMS fehlschlagen kann, sollte eine Möglichkeit bestehen, den Dekodierversuch zu wiederholen.

- Schlüsselmanagement – Schlüssel können in SEMS generiert, signiert, widerrufen und versendet werden. Schlüssel sind stets Personen und Telefonnummern zugeordnet. Eine Adressverwaltung mit integriertem Schlüsselmanagement bietet sich hier an.
- Sonstiges – Verschiedene Funktionalität wie das Einstellen von Sicherheitsoptionen ist notwendig.

Die weitere Analyse der Anforderungen und Funktionen des Programms soll anhand von verschiedenen Anwendungsfällen geschehen.

3.1.1 Schlüsselgenerierung

Der Nutzer kann eigene, private Schlüssel generieren. Die Schlüsselerzeugung findet vollständig auf dem Mobiltelefon statt und darf lange dauern, da sie nur einmalig geschieht. Die Zufallszahlenquelle ist die der unterliegenden Java-Platform. Der Nutzer darf beliebig viele eigene Schlüssel besitzen. Als Parameter sind auszuwählen:

- Sicherheitsstufe – Entweder mittlere Stufe, was einem 1056-Bit Schlüssel entspricht, oder hohe Stufe, was 2096-Bit entspricht.
- Name – Der Nutzer kann jedem Schlüssel einen beliebigen, maximal 64 Zeichen langen Namen geben. Die Kodierung erfolgt in 7-Bit-ASCII.
- Ablaufdatum – Das Ablaufdatum des Schlüssels kann entweder „unendlich“ oder jeder Tag von 01.08.2001 bis 04.05.2180 sein.[1]
- Telefonnummer – Auch die Telefonnummer ist frei wählbar, muss jedoch internationaler Norm entsprechen (insbesondere maximal 15 Stellen enthalten). Natürlich sind nur Nummern sinnvoll, unter denen das Handy auch erreichbar ist.

Sind alle Parameter eingestellt, so beginnt die Erzeugung des privaten Schlüssels. Während der Schlüsselerzeugung ist die Benutzung des Midlets gesperrt.

3.1.2 Signieren eines öffentlichen Schlüssels

Öffentliche Schlüssel können signiert werden, auch eigene Schlüssel. Dazu wird erst der zu signierende Schlüssel und anschließend, falls der Nutzer mehrere private Schlüssel besitzt, der Signierschlüssel ausgewählt. Die Schlüsselsignatur wird gebildet und an den öffentlichen Schlüssel angehangen. Ist der zu signierende Schlüssel widerrufen, erscheint eine Warnung.

[1] Die Einschränkung des Datums ergibt sich aus der Definition entsprechender Felder im Standard.

3.1.3 Widerrufen eines geheimen Schlüssels

Eigene noch gültige Schlüssel können widerrufen werden (key revocation). Der Widerruf ist entweder mit einer ausgelagerten oder integrierten Signatur versehen. Zu Schlüsseln kann maximal ein Widerruf gespeichert werden (trifft ein Neuer ein, wird der Alte ersetzt). Angehangene Widerrufe können versendet und gelöscht werden.

3.1.4 Das Adressbuch

Der Nutzer kann alle Einträge in seinem Adressbuch betrachten. Öffentliche Schlüssel können komplett und mit Fingerprint abgerufen werden. Zu geheimen Schlüsseln wird nur der Fingerprint angezeigt. Die Einträge haben folgende Werte:

- Name – Der Name der Person, dem dieser Eintrag zugeordnet ist. Dieser muss nicht mit dem Namen aus angehangenen Schlüsseln übereinstimmen. Jeder Name darf nur einmal im Adressbuch vorkommen.
- Telefonnummer – Die Telefonnummer, an die alle SMS zu dieser Person gesendet werden sollen. Die Nummer muss nicht mit der in den angehangenen Schlüsseln übereinstimmen.
- Schlüssel – Es können beliebig viele Schlüssel angehangen sein. Jeder Schlüssel kann beliebig viele Signaturen und einen Widerruf besitzen.

Einträge können erzeugt, geändert und gelöscht werden.

3.1.5 Versand einer SMS

Der Nutzer kann Nachrichten versenden. Dazu stehen ihm verschieden Optionen zur Verfügung:

- Typ der SMS – Es können Textnachrichten, öffentliche Schlüssel, Schlüsselwiderrufe und ausgelagerte Signaturen versendet werden. Der Nutzer trifft diese Wahl bereits im Vorfeld, indem er die Versand-Funktion aus verschiedenen Menüpunkten heraus anwählt. Die zu versendenden Daten können vor dem Versand betrachtet werden. Ist der Typ ein Schlüssel, stehen weitere Optionen zur Verfügung:
  - Angehängte Signaturen – Es können bis zu 32 Signaturen beliebig ausgewählt werden.
  - „send me yours“ – Der Nutzer kann wählen, ob der Empfänger mit seinem eigenen Schlüssel antworten soll.

  Es können eigene und fremde öffentliche Schlüssel versendet werden. Ist der Schlüssel widerrufen, erscheint eine Warnung.

- Verschlüsselt – Wird eine SMS verschlüsselt versendet, muss der Empfänger mindestens einen gültigen[2] öffentlichen Schlüssel besitzen. Bei mehreren möglichen Schlüsseln wählt der Nutzer einen aus.
- Integrierte Signatur – Die SMS kann eine integrierte Signatur enthalten. Dazu muss der Nutzer mindestens einen gültigen[3] geheimen Schlüssel besitzen. Besitzt er mehrere, wählt er vor dem Versand einen aus.
- Ausgelagerte Signatur – Es kann eine ausgelagerte Signatur zu dieser SMS erzeugt werden. Diese Signatur wird unmittelbar im Anschluss an die SMS versendet. Ist der Typ der SMS kein Schlüssel, kann entweder eine ausgelagerte Signatur oder eine integrierte Signatur als Option gewählt werden.
- Kodierung von Textnachrichten – Falls Textnachrichten versendet werden sollen, wählt der Nutzer eine von SEMS unterstützte Kodierung für die Nachricht. Enthält die Nachricht für die gewählte Kodierung ungültige Zeichen, wird dies vor dem Versand mit einer Fehlermeldung angezeigt. Der Versand von 8-Bit Binärnachrichten ist nicht möglich. Bleibt der Nachrichtentext leer und ist die Option „verschlüsselt“ angewählt, so wird eine Dummie-Nachricht („No Operation“) versendet.

Einmal versendete SMS werden in einer Liste gespeichert. Einträge aus dieser Liste können gelöscht werden. Zu Einträgen der Liste können Signaturen versendet werden, außer der Eintrag ist selber eine ausgelagerte Signatur.

3.1.6 Empfang einer SMS

Empfangene SMS werden, wenn sie verschlüsselt oder signiert sind zunächst in einer separaten Liste gespeichert. Danach werden verschlüsselte SMS entschlüsselt und vorhandene Signaturen geprüft. Schlägt die Entschlüsselung fehl oder ist eine integrierte Signatur fehlerhaft, wird das Ergebnis verworfen und die SMS verbleibt verschlüsselt bzw. signiert. Fehlerhafte ausgelagerte Signaturen werden an der entschlüsselten SMS vermerkt. Die Entschlüsselte SMS wird nun nochmals in der Eingangsliste gespeichert. Der Nutzer wird über den Empfang und den Erfolg der Entschlüsselung und die Gültigkeit von Signaturen informiert. Der Nutzer kann jederzeit aus der Liste der verschlüsselten oder signierten SMS die Entschlüsselung bzw. den Test der Signatur erneut versuchen. Für Einträge aus der Liste der entschlüsselten SMS (Eingangsliste) stehen folgende Optionen zur Verfügung:

- Alle Einträge können betrachtet und gelöscht werden.

[2] Die Verschlüsselung mit widerrufenen Schlüsseln ist nicht möglich.
[3] Auch das Leisten einer Signatur ist mit einem widerrufenen Schlüssel nicht möglich.

- Schlüssel können in das Adressbuch übernommen werden. Dazu kann ein bestehender Eintrag ausgewählt oder ein Neuer erzeugt werden.
- Schlüsselwiderrufe können an den zugehörigen Schlüssel angehangen werden (der passende Schlüssel wird automatisch ausgewählt).
- Ausgelagerte Signaturen können erneut überprüft werden (wenn die zugehörige Nachricht ebenfalls vorhanden ist. Die Nachricht wird automatisch ausgewählt).

3.2 Design

Die im letzten Abschnitt vorgestellten Funktionen sollen nun auf Datenstrukturen und Oberflächenmasken abgebildet werden.

3.2.1 Komponenten von SeJI

Zur besseren Handhabung wird das Programm in verschiedene Komponenten aufgeteilt. Diese werden nun vorgestellt.

SEMS Datenklassen

SEMS definiert verschiedene Strukturen um die Nachrichten zu verwalten. Diese werden direkt in Java Klassen umgesetzt. Jede dieser Datenstrukturen ist dabei fähig, sich aus einer Bitfolge zu konstruieren und in eine Bitfolge zu serialisieren. Auf den Strukturen 4.2.2 und 4.3.2 („envelopes") können zudem kryptographische Operationen durchgeführt werden.

Verwaltung der Adressen und SMS-Listen

Da in MIDP keine Schnittstelle zum Handy-internen Adressbuch existiert, muss eine Adressverwaltung selbst implementiert werden. Die Verwaltung des Adressbuches und der Listen für eingehende, gesendete und verschlüsselte Nachrichten erfordert weitere Datenklassen. Hierzu zählen auch Strukturen für die gespeicherten Schlüssel.

Kryptographische Operationen

Die kryptographischen Operationen SHA-1, SHAKE, AES und RSA sind in einer Klasse vereint, welche auf Bitfolgen arbeitet.

Toolklassen

Toolklassen sind allgemeine Werkzeuge zur Hilfe bei allen anderen Strukturen. Darunter fallen beispielsweise Klassen, die Bitfolgen zerlegen und zusammensetzen können. Diese Klassen werden dann als Stellvertreter benutzt, je nachdem ob eine Bitfolge konstruiert oder ausgewertet werden soll.

Oberflächenklassen

Die Menüführung benötigt ebenfalls Klassen, welche in einer eigenen Komponente untergeordnet sind.

3.2.2 Klassenmodell

Der nächste Schritt im Design der Applikation besteht in der Erstellung eines Klassenmodells für die verschiedenen Komponenten. Die Datenklassen von SEMS lassen sich recht einfach auf ein objektorientiertes Modell abbilden, da sie im Standard bereits nach einer Vererbungshierarchie angeordnet sind. Abbildung 3.1 zeigt ein UML-Diagramm der SEMS-Datenklassen. Ein `Data`-Objekt ist in ein `Envelope` eingebettet, welches wiederum in ei-

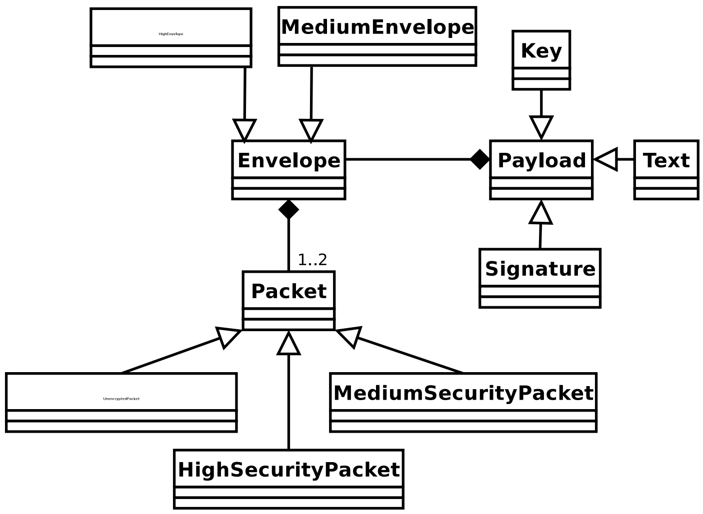

Abbildung 3.1: Klassenmodell für die Datenklassen von SEMS

nem `Packet` enthalten ist. Dabei gibt es eine Besonderheit in der Klasse `HighSecurityEnvelope`: Sie besteht aus zwei `HighSecurityPackets`, im Standard beschrieben durch 4.3.1.1 und 4.3.1.2. Die Klassen lassen sich alle durch eine Bitfolge erzeugen, wobei die umschließenden Objekte dazu die eingebetteten Konstruktoren aufrufen. Umgekehrt sind auch alle Klassen wieder in eine Bitfolge umwandelbar.

Die Klassen zur Adressverwaltung sind ebenfalls Datenklassen. Abbildung 3.2 zeigt eine Übersicht über diese Komponente. `Key` ist hier dieselbe Klasse wie aus den SEMS-Datenklassen. Dadurch ist die Adressverwaltung jedoch nicht in der Lage, private Schlüssel zu speichern. Deren Verwaltung geschieht gesondert. Durch diese Abgrenzung sensitiver Daten könnten

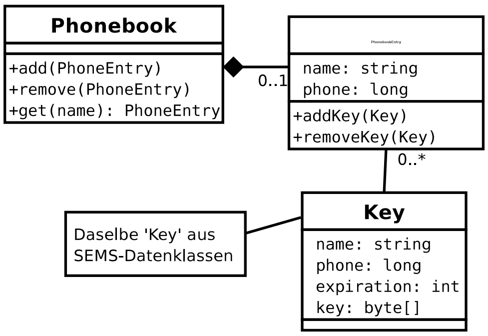

Abbildung 3.2: Klassenmodell für die Datenklassen der Adressverwaltung

beispielsweise die geheimen Schlüssel auf einem speziell dafür vorgesehenen Speicher platziert werden.

SEMS kommt mit wenigen kryptographischen Funktionen aus. Der Zugriff auf diese wurden in einer einzigen Klasse implementiert, welche die Schnittstelle zu den Implementationen der Bibliothek „Bouncy Castle“ bilden soll (ohne Abbildung). Die Klasse `Crypto` repräsentiert eine Bitfolge, welche sowohl mit RSA als auch mit AES ver- und entschlüsselt werden kann. Sie kann mit SHAKE kodiert und dekodiert werden und von ihr kann eine SHA-1 Prüfsumme errechnet werden.

Schließlich sind die Oberflächenklassen auf die verschiedenen Grundkomponenten (siehe Abschnitt 3.1) verteilt. Abbildung 3.3 zeigt ein Klassenmodell der Oberflächenklassen. Für das Versenden von Nachrichten, die An-

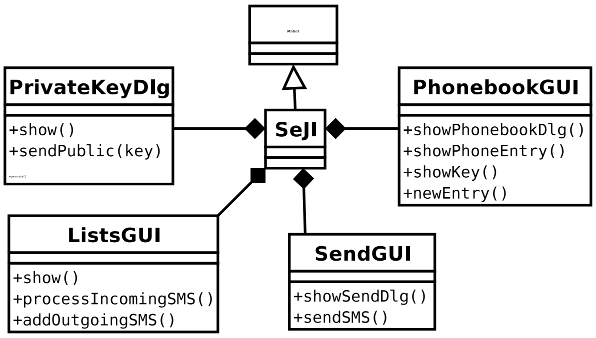

Abbildung 3.3: Klassenmodell für die Oberflächenklassen

zeige eingegangener Nachrichten und das Adressbuch existiert jeweils eine Oberflächenklasse, die wiederum mehrere Dialoge enthält. Somit wurde ein Kompromiss zwischen Übersichtlichkeit: pro Dialog eine Klasse, und Code-Effizienz: alles in eine Klasse, getroffen. `SeJI` ist der Applikationseinstieg und erbt somit von `Midlet`. Diese Klasse stellt die Verbindung der Oberflächen- zu den Daten- und Logikklassen dar.

3.2.3 Grafische Nutzerschnittstelle

SeJI ist eine Implementation für Java-fähige Handys. Dadurch ist die grafische Oberfläche auf die Elemente beschränkt, die Java Midlets zur Verfügung stehen. Diese sind:

- Textbeschriftungen und Bilder zur Darstellung von Informationen. Texte können als Laufschrift animiert werden.
- Auswahllisten geben die Möglichkeit, aus mehreren Alternativen auszuwählen
- Menüpunkte sind ausführbare Aktionen
- Eingabefelder ermöglichen die Eingabe von Telefonnummern, Zahlen, Datum und Uhrzeiten oder beliebigen Texten.

Eine Besonderheit der GUI-Programmierung von Midlets ist, dass die genaue Darstellungsweise nicht vom Programm festgelegt werden kann. So könnten Menüpunkte als nummerierte Liste auf dem Display erscheinen, die über die Betätigung der entsprechenden Nummer ausgewählt werden. Falls das Handy Kurzwahltasten unterstützt könnten die Menüpunkte auch diesen zugeordnet werden. Das Design der Oberfläche kann dadurch nur abstrakt durch Aufzählung der Bedienelemente geschehen. Die Darstellung erfolgt skizzenhaft.

Die Sprache der Oberfläche ist in Englisch verfasst. Sollen SMS in Listen angezeigt werden, so wird der Typ der Nachricht als Buchstabe voran gehängt, sofern er ermittelbar[4] ist. „T“ steht für Textnachricht, „K“ für Schlüssel, „S“ für eine Signatur und „R“ für ein Schlüsselwiderruf. Zusätzlich ist die Telefonnummer des Absenders bzw. Empfängers im internationalen Format aufgeführt.

Hauptmenü

Das Hauptmenü ist der Einstiegspunkt der Applikation. Hier sind Verzweigungen zu allen Hauptfunktionen von SeJI vorhanden.

- Send – SMS können hier versendet werden.
- Lists – Es können eingehende und bereits versendete SMS betrachtet und für verschlüsselte oder signierte SMS erneut das Entschlüsseln bzw. Testen probiert werden.
- Addresses – Hier wird das Adressbuch verwaltet.

[4] Verschlüsselte SMS, für die kein geheimer Schlüssel, oder SMS mit integrierten Signaturen, für die kein Testschlüssel zur Verfügung stehen, können nicht ausgewertet werden.

Abbildung 3.4: Skizze zum „Hauptmenü“

- Options – Voreinstellungen zum SMS Versand werden getroffen.
- Exit – Beendet SeJI.

Send

Das Menü „Send“ dient dem Versand von SMS. Hierhin gelangt man entweder über:

- das Hauptmenü – Es werden Textnachrichten versendet.
- die Outbox – Es werden ausgelagerte Signaturen versendet.
- die Adressverwaltung – Es können Schlüssel und Schlüsselwiderrufe versendet werden.

Lists

Es können verschieden Listen betrachtet werden. Über Menüpunkte wird die entsprechende Liste ausgewählt.

- Inbox – Die Nachrichten-Eingangsliste der entschlüsselten und getesteten Nachrichten.
- Problems – Eine Liste der verschlüsselten oder ungetesteten Nachrichten bei denen die Entschlüsselung bzw. der Signaturtest fehlschlug.
- Outbox – Eine Liste der bereits versendeten Nachrichten.

Inbox

Der Nachrichteneingang listet alle empfangenen und entschlüsselten Nachrichten auf. Nachrichten können über Menüpunkte gelöscht und betrachtet werden. Schlüssel können zudem importiert werden.

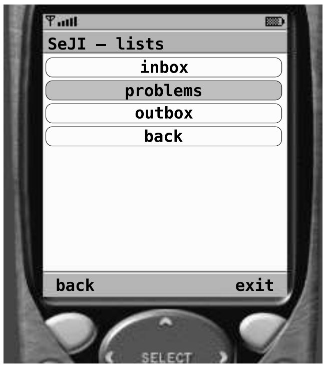

Abbildung 3.5: Skizze zum Menü „Lists“

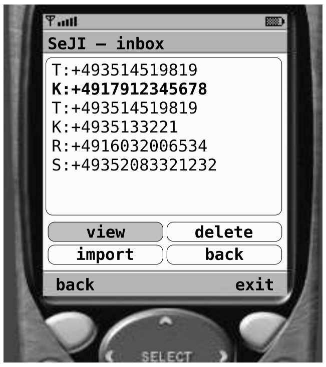

Abbildung 3.6: Skizze zur Liste „Inbox“

Problems

Alle Nachrichten die empfangen wurden und bei denen es kryptographische Probleme gab (weil benötigte Schlüssel nicht vorhanden waren oder Operationen fehlschlugen) werden hier aufgelistet. Über Menüpunkte können Einträge gelöscht oder der Entschlüsselungs- bzw. Testvorgang wiederholt werden.

Outbox

Alle versendeten SMS werden hier aufgelistet. Einträge in dieser Liste können gelöscht, betrachtet oder signiert werden.

SMS anzeigen

Zu einer betrachteten SMS wird, im Falle einer Textnachricht, der Nachrichtentext und die Kodierung, in jedem Fall aber eine Information, ob die SMS verschlüsselt war, und ggf. die Signatur mit ihrer Gültigkeit, angezeigt. Handelt es sich um einen Schlüssel, ist dies eine Liste aus Signaturen.

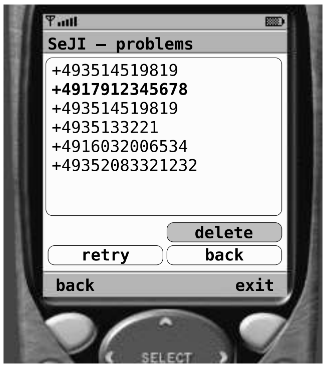

Abbildung 3.7: Skizze zur Liste „Problems“

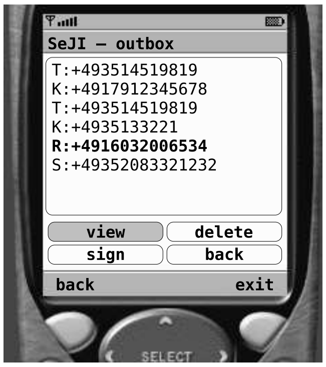

Abbildung 3.8: Skizze zur Liste „Outbox“

Adresses

Die Adressliste beinhaltet alle Telefonbucheinträge, die innerhalb von SeJI aufgenommen wurden. Über Menüpunkte kann ein neuer Eintrag erstellt order ein ausgewählter Eintrag geändert oder gelöscht werden.

Telefonbucheintrag anzeigen

Es werden Name, Telefonnummer und die Anzahl der angehangenen Schlüssel angezeigt. Über einen Menüpunkt können die Schlüssel betrachtet werden. Der erste Buchstabe der Liste gibt die Sicherheitsstufe des Schlüssels an. „h“ steht für hohe, „m“ für mittlere Sicherheitsstufe. Ist der Buchstabe in Großbuchstaben, handelt es sich um einen geheimen, sonst um einen öffentlichen Schlüssel. Schlüssel können angezeigt und gelöscht werden. Über einen Menüpunkt können zudem neue Schlüssel generiert werden.

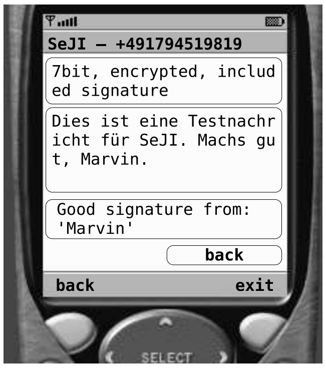

Abbildung 3.9: Skizze zur Funktion „SMS anzeigen“

Abbildung 3.10: Skizze zur Liste „Addresses“

Schlüssel anzeigen

Von einem Schlüssel wird der zugeordnete Name, die Telefonnummer, das Ablaufdatum und der Fingerprint angezeigt. Über einen Menüpunkt kann die Liste der Signaturen angewählt werden.

Zusammenfassung

Die Oberflächendialoge werden in der Grafik aus Abbildung 3.13 noch einmal zusammengefasst. Die durchweg doppelten Pfeilverbindungen zeigen, dass in jedem Bildschirm eine „back“ Menüoption vorhanden ist, die in den jeweils Letzten zurückführt.

3.3 Implementation

SeJI wurde nur unvollständig implementiert. Abschnitt 3.5 zählt Funktionen von SEMS auf, die in SeJI nicht enthalten sind. Weitere Änderungen am Standard sind unter 2.5 zu finden. Die Applikation dient zu Demonstrations-

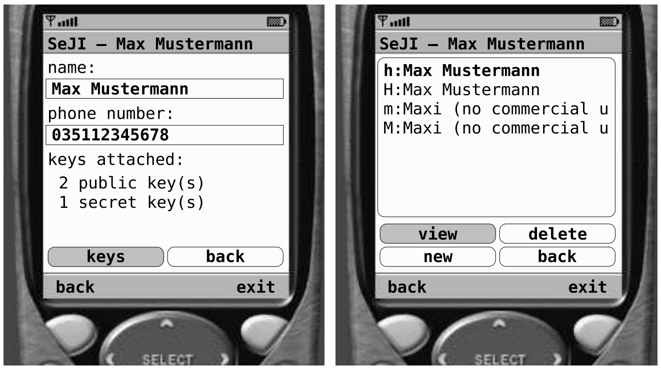

Abbildung 3.11: Skizze zur Funktion „Telefonbuch anzeigen“

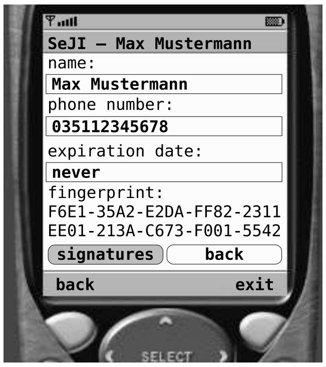

Abbildung 3.12: Skizze zur Funktion „Schlüssel anzeigen“

zwecken kryptographischer Anwendungen auf Java-Handys, als Beweis der Implementierbarkeit von SEMS und als Messplattform für Performance von Handys, mobilen Java ME Implementationen und Bibliotheken. Im Folgenden werden Teile der Implementation vorgestellt, die diese Ziele erreichen.

3.3.1 Wahl der Komponenten

Die Grundkomponenten wurden mit einer Bibliothek von Sun auf der Java ME WTK 2.1 Spezifikation erstellt. Daraus wird das MIDP2.0 (Mobile Independent Device Profile Version 2.0) und WMA1.1 (Wireless Message API Version 1.1) benötigt. SeJI benötigt demzufolge ein Handy, welches beide Bibliotheken zur Verfügung stellt.

Für kryptographische Operationen wurden Klassen der freien Bibliothek „Bouncy Castle“ in SeJI integriert. Unter dem Packet `seji.crypto` sind die Klassen `BigInteger`, `AESLightEngine`[5], `Digest`, `GeneralDigest`, `SHA1Digest` und `SecureRandom` importiert. Zu `BigInteger` wurde eine Funk-

[5]Von der aktuellen Implementation unbenutzt.

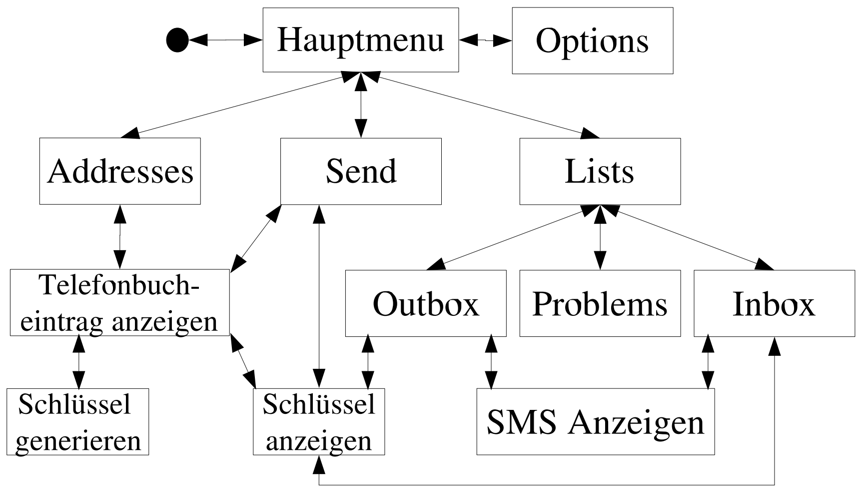

Abbildung 3.13: Übersicht aller Bildschirme in SeJI

tion `toUnsignedByteArray(int)` hinzugefügt, welche die Zahl ohne Vorzeichen als ein `byte`-Array gegebener Größe interpretiert. Es werden hier keine weiteren Bibliotheken nötig. Da die Paketnamen geändert wurden, kollidiert SeJI auch nicht mit einer eventuell bestehenden Installation von Bouncy Castle.

Die Testklassen benutzen Teile der Java 1.4.2 SE, dafür enthalten diese keine Referenz zur Java ME. Für Performancemessungen sind die Testklassen ungeeignet, mehr dazu in Abschnitt 3.4.

3.3.2 Ausgewählte Quelltexte

Einige Quelltextabschnitte werden nun vorgestellt um typische Struktur der Implementation zu verdeutlichen.

Abbildung 3.14 zeigt einen Ausschnitt des Hauptprogramms von SeJI aus der gleichnamigen Klasse. Die Hauptinstanz definiert hier einige statische Variablen, die allen anderen Oberflächenkomponenten zur Verfügung stehen wie beispielsweise der Bildschirm `mainDisplay` oder die Zufallszahlenquelle. Außerdem enthält die Hauptklasse einige Dialoge wie das Hauptmenü oder den Optionsbildschirm. Im Konstruktor werden dann diese Variablen gesetzt, die Dialoge initialisiert und mit Menüpunkten aufgefüllt. Die Funktion `commandAction`, deklariert im Interface `CommandListener` wird bei Anwahl eines Menüpunktes durch den Benutzer angesprungen. `SeJI` ist selbst ein Lauscher für alle eigenen Menüpunkte, die Unterscheidung welcher Menüpunkt angewählt wurde trifft der abgebildete `if-then-else`-Block. Dieses Prinzip der Ereignisbehandlung findet sich in allen Oberflächenklassen unter `seji.gui` wieder.

Öffentlichen Variablen sind auch in fast allen Datenklassen vorhanden.

```
public class SeJI extends MIDlet implements CommandListener {
        public static Options options = new Options();

        public static Display mainDisplay;
        public static SeJI seji;
        public static SecureRandom rnd = new SecureRandom();
        public static PhoneBook phoneBook;

        public static Form main =
                        new Form("Sems Java Implementation");

        private Form optionsDlg = new Form("Options");
        private ChoiceGroup optionEncrypt =
                        new ChoiceGroup("Encryption",ChoiceGroup.MULTIPLE);

...
        /**
         * Construct a new SeJI-Midlet containing main gui interface
         */
        public SeJI() {
                seji = this;
                mainDisplay = Display.getDisplay(this);
                main.addCommand(new Command(
                                "send","Sending SMS", Command.SCREEN,1));
                main.addCommand(new Command(
                                "lists","SMS lists", Command.SCREEN,1));
                main.addCommand(new Command(
                                "addresses","Phonebook manager", Command.SCREEN,1));
...
       }

        public void commandAction(Command c, Displayable display) {
                if (display == main) {
                        if (c.getLabel().equals("send"))
                                sendGUI.show();
                        else if (c.getLabel().equals("lists"))
                                listsGUI.show();
                        else if (c.getLabel().equals("addresses"))
                                phoneBookGUI.show();
...
                } else if (display == optionsDlg) {
                        updateOptions();
                        mainDisplay.setCurrent(main);
                } else if (display == infoDlg)
                        mainDisplay.setCurrent(main);
                else if (display == errorDlg)
                        mainDisplay.setCurrent(errorReturn);
        }
}
```

Abbildung 3.14: Sourcecode Ausschnitt aus `seji.SeJI`

Abbildung 3.15 zeigt Ausschnitte der Klasse `seji.data.payload.Key`, die ebenfalls öffentliche Variablen besitzt. Die Entscheidung, statt der für Java typischen `get` und `set` Funktionen wurde aufgrund von Performancebedenken getroffen, die sich allerdings nicht bestätigten. Die Abbildung verdeutlicht ebenfalls die Verwendung der Serialisierungsklassen `BitChipper` und `BitStapler`. Etliche Datenklassen verfügen über Funktionen mit dem Namen `addToBitStapler` und über Konstruktoren, die unter anderem einen `BitChipper` als Parameter erhalten. Mit diesen Funktionen werden die Klassen in SeJI serialisiert – sowohl für den Transport innerhalb einer SMS als auch zur persistenten Speicherung im Handy.

Die letzte Abbildung 3.16 zeigt Teile der Klasse `seji.crypto.Crypto`. Es sind die Implementationen der Ver- und Entschlüsselung mit RSA zu sehen. Die Abbildung verdeutlicht den Umgang mit `BigInteger`, der zentralen Klasse der kryptographischen Funktionen. Die Performancetests ergaben die größten Flaschenhälse in Operationen dieser Klasse (siehe auch Abschnitt

```
publicclass Key extends SignablePayload {

        public boolean sendMeYours;
        public long phoneNumber;
        public int expiration;
        public String name;
        public byte[] key;

        public Key(BitChipper chipper, int size)
                        throws IllegalTransportObjectException {
                super(size);
                int pos = chipper.pos;
                int signatureType = chipper.next(2);
                int signatureCount = chipper.next(5);
                signature = new SignatureInfo(signatureType, signatureCount, chipper);
                sendMeYours = chipper.next(1) == 1;
...
                key = new byte[remainingBits / 8];
                chipper.next(key);
        }

        public void addToBitStapler(BitStapler stapler) {
                stapler.add(2,4); // payload data type
                signature.addToBitStapler(stapler, true);
                stapler.add(sendMeYours?1:0, 1);
                stapler.add(2,3); // key type (2 instead of 1 for exponent used in SeJI)
...
                stapler.add(key);
        }
}
```

Abbildung 3.15: Sourcecode Ausschnitt aus `seji.data.payload.Key`

```
publicclass Crypto {

        public BigInteger msg;
        public int size;

        /**
         * Encrypt with RSA.
         */
        public void encrypt(PublicKey key) {
                msg = msg.modPow(Key.e, key.n);
        }

        /**
         * Decrypt with RSA using chinese remainder theorem.
         */
        public void decrypt(PrivateKey key) {
                BigInteger m1 = msg.modPow(key.dp, key.p);
                BigInteger m2 = msg.modPow(key.dq, key.q);
                BigInteger h = key.u.multiply(m2.subtract(m1)).mod(key.q);
                msg = m1.add(h.multiply(key.p)).mod(key.n);
        }
...
}
```

Abbildung 3.16: Sourcecode Ausschnitt aus `seji.crypto.Crypto`

3.6.3), so dass besondere Aufmerksamkeit auf die effiziente Benutzung von `BigInteger` gelegt wurde. An der Implementation von `BigInteger` selber wurde nichts geändert, mit einer Ausnahme: Eine notwendige Funktion zur Interpretation als unsigniertes `byte`-Array wurde hinzugefügt.

3.4 Testklassen

Für alle Klassen, mit Ausnahme der Oberflächenklassen unter `seji.gui` wurden Testklassen erstellt. Um nicht durch die Geschwindigkeit des Java ME Emulators beschränkt zu sein, laufen die Tests in der „normalen" Java Umgebung JDK 1.4.2. Die Testklassen wurden mit Hilfe von JUnit erstellt. Der Quelltext ist auf der beiliegenden CD unter `SeJITest/` gespeichert. Die Oberflächenklassen lassen sich leider nicht durch JUnit-kompatible Funktionen testen, da weder JUnit unter Java ME, noch MIDP2.0 unter Java 1.4.2 lauffähig ist.

Da die Testklassen unter der „normalen“ Java-Umgebung laufen, sind sie nicht zur Messung von Performanceeigenschaften einsetzbar. Die Operationen in der Testumgebung laufen etwa 100 bis 300 mal schneller als im Emulator.

3.5 Unimplementierte Funktionen

Wie zu Beginn dieses Kapitels bereits erwähnt wurde, sind nicht alle Funktionen von SEMS in der Beispielimplementation enthalten. Folgende Funktionen stehen in SeJI nicht zur Verfügung:

- Die hohe Sicherheitsstufe ist nur ansatzweise implementiert. Die Datenstrukturen stehen zur Verfügung, Schlüssel hoher Sicherheitsstufe können generiert werden. Der Versand jeglicher Daten über die hohe Sicherheitsstufe ist jedoch deaktiviert.
- Ausgelagerte Signaturen sind nicht möglich. Damit können auch keine Schlüssel mit Signaturen versehen werden (das Versenden mittels integrierter Signatur ist möglich).
- Schlüssel können nicht widerrufen werden.

3.6 Performance

Ein wichtiges Augenmerk bei der Arbeit mit leistungsschwachen Endgeräten wie Mobiltelefone ist die Geschwindigkeit der Applikation. Dauern Prozesse zu lange, nützen auch gute Sicherheitseigenschaften nichts.

War in der Vergangenheit der persistente Speicherbedarf einer Handy-Applikation noch von großer Bedeutung, rückt dieses Kriterium mehr und mehr in den Hintergrund. Aktuelle Handys wie das Ericsson P900 verfügen über 32MB Flash-Speicher bei Auslieferung, eine Aufrüstung ist problemlos möglich. Persistenter Speicherplatz schränkt einzelne Anwendungen nicht mehr ein, wenn sie nicht übermäßig verschwenderisch damit umgehen.

3.6.1 Akzeptable Grenzen

Akzeptable Höchstwerte für Geschwindigkeit zu finden, ist nicht einfach – die in dieser Arbeit gesteckten Grenzen sind sehr großzügig gefasst, was hervorheben soll, dass der Autor gute Sicherheitseigenschaften höher als gute Performance der Anwendung schätzt. Je nach Klientel sollten diese Grenzen variiert werden. Beispielsweise sollte ein signiertes Auktionsangebot innerhalb weniger Sekunden versendet werden können, während die Stimmabgabe per SMS bei einer politischen Wahl durchaus auch mehrere Minuten dauern darf.

Einige Grenzwerte für verschiedene Operationen wurden bereits in Abschnitt 3.1 angedeutet. Die Werte in Tabelle 3.17 sind ein Beispiel.

Anwendungsfall	Grenze
Schlüsselgenerierung	höchstens 5 Stunden
Schlüssel signieren / versenden	höchstens 5 Minuten
SMS versenden	höchstens 30 Sekunden
SMS empfangen und dekodieren	höchstens 2 Minuten

Abbildung 3.17: Akzeptable Grenzen für die Performance

Die lange Dauer der Schlüsselgenerierung rührt daher, dass dieser Prozess nur einmal zur Installation der Software notwendig ist. Der Unterschied zwischen „SMS versenden“ und „SMS empfangen“ ist daraus herzuleiten, dass manchmal ein Anwender ein Handy nach Versand einer SMS ausschalteten möchte und auf den erfolgreichen Versand warten muss, aber der Empfang – für den Empfänger unerwartet – auch verzögert stattfinden kann.

3.6.2 Messungen an SeJI

Alle Messungen wurden mit einem Prototypen von SeJI auf dem Java ME Emulator von Sun WTK2.1 durchgeführt. Kurze, nicht dokumentierte Vergleiche mit einem Nokia 7660 zeigten, dass die Performanceeigenschaften des Emulators vom Hersteller gut abgebildet wurden.

Die erste Messreihe in Abbildung 3.18 zeigt die Dauer verschiedener typischer Funktionen in SeJI.

Alle Messungen wurden mit Operationen der mittleren Sicherheitsstufe

Vorgang	benötigte Dauer
verschlüsselt senden	1,7s
verschlüsselt und signiert senden	9,0s
verschlüsselt empfangen	7,5s
verschlüsselt und signiert empfangen	8,9s / 16,9s
Nachricht signieren	7,2s
Schlüssel generieren	936,5s

Abbildung 3.18: Dauer verschiedener Operationen in SeJI

durchgeführt. Für die hohe kann ein etwa vier mal längere Zeitdauer angenommen werden (RSA-Operationen benötigen bei doppelter Schlüssellänge die vierfache Zeit). Eine Besonderheit stellt der Empfang von signiert und verschlüsselten SMS dar. Da hier nach der Signaturbildung das höchstwertigste Bit gelöscht wird, muss dieses vor dem Test der Signatur geraten werden. Wurde das Bit falsch geraten, so muss der Vorgang wiederholt werden und die Dauer steigt entsprechend um die Zeit einer Entschlüsselungsoperation. Die Schlüsselerzeugung ist nur ein sehr ungenauer Wert. Je nachdem, wie schnell eine Primzahl gefunden wird, kann diese Zahl stark variieren. In dem abgebildeten Versuch waren 248 Versuche notwendig um alle beide Primzahlen zu finden. Jeder Primzahlentest dauert etwa 3,7 Sekunden und durchschnittlich muss man nur $(ln(2) * 528)/3 = 122$ Tests durchführen.[6]

Die zweite Messreihe misst die Anteile, die verschiedene Komponenten an der Zeitdauer des Sendens einer verschlüsselten und signierten SMS haben. Abbildung 3.19 zeigt, dass der überwiegende Teil der Rechenzeit an die RSA-Operationen fällt.

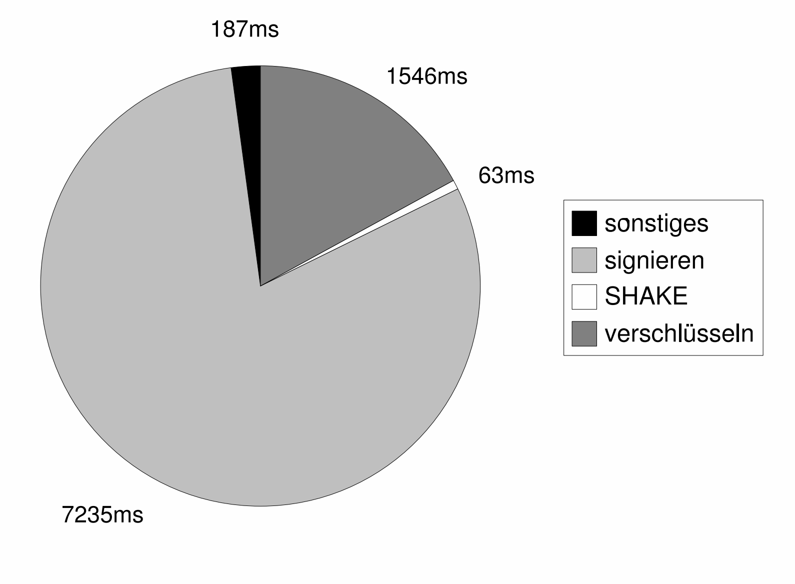

Abbildung 3.19: Aufteilung der Rechenzeit eines Sendevorgangs

Optimierungen sind also vor allem hier sinnvoll. Die Komponente „sonstiges" beinhaltet beispielsweise die I/O-Vorgänge des Versendens oder der Zugriff auf den persistenten Speicher des Handy um die benötigten Schlüssel auszulesen.

3.6.3 Messungen einzelner kryptographischer Operationen

Es wurden zwei Messungen einzelner Kryptooperationen vorgenommen. Diese verdeutlichen das Verbesserungspotential einzelner Implementationen und Parameter des Standards.

[6]Schätzung der Primzahlenverteilung nach Kranakis.

Die erste Messung bezieht sich auf die Unterschiede in der Geschwindigkeit einer öffentlichen RSA-Operation[7]. Ziel war es festzustellen, welchen Einfluss die Wahl des öffentlichen Exponenten auf die Geschwindigkeit hat. Abbildung 3.20 zeigt das Ergebnis.

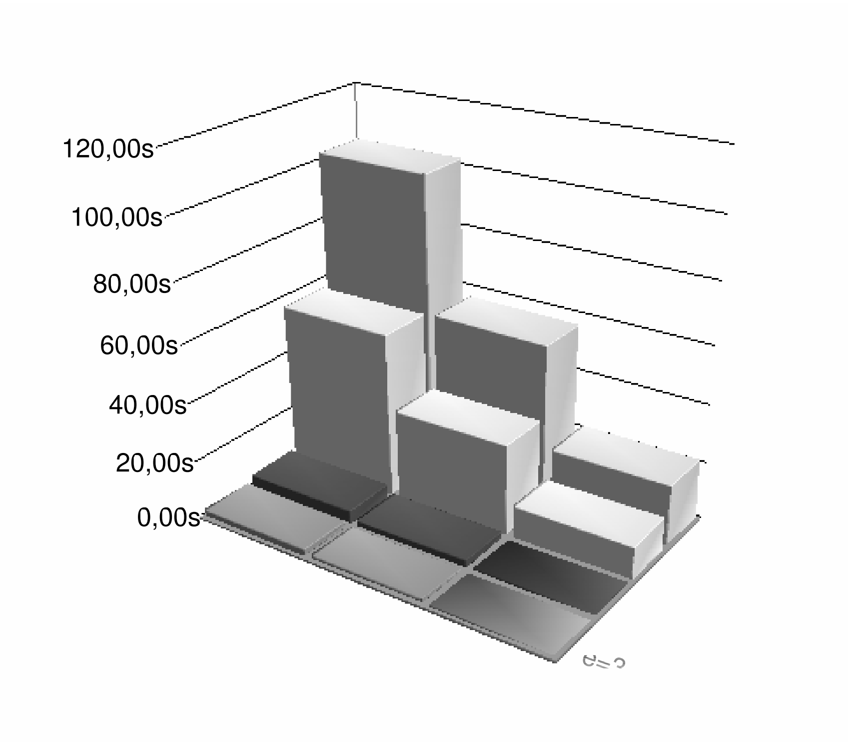

Abbildung 3.20: Einfluss des öffentlichen Exponenten auf RSA

Die Grafik zeigt die Dauer (Y-Achse) verschiedener RSA-Operationen (Z-Achse) auf unterschiedlichen Handys (X-Achse). Der Modulus der Operation hat die Länge 2208 Bit. Die vorderste Zeile zeigt die Geschwindigkeit für den in SEMS vorgeschriebenen Exponenten von 3. Erwartungsgemäß ist hier die Geschwindigkeit sehr hoch, die Ergebnisse bewegen sich im Bereich der minimal messbaren Werte. Auch bei dem in der vorliegenden Arbeit vorgeschlagenen Exponenten von $e = 0x4000000800001$ (zweite Zeile) ist die Geschwindigkeit noch ausreichend. So dauert auf einem Nokia 7650 die Operation knapp 800ms gegenüber etwa 450ms für $e = 3$. Ein zufälliger Exponent (letzte Zeile) hingegen ist nicht ratsam, hier liegt die Dauer der Operation deutlich im zweistelligen Sekundenbereich.

Die Grafik verdeutlicht zudem, wie stark der Einfluss des Mobiltelefons

[7]Diese wird zum Test einer integrierten Signatur und zum Verschlüsseln einer Nachricht benötigt.

auf die zu erzielende Geschwindigkeit ist. Die ersten Messungen entstanden auf einem Siemens S55. Dieses zwar weit verbreitete, aber eher betagte Handy braucht für eine Operation mit $e = 3$ über 14 Sekunden, für ein zufälliges e sogar über 20 Minuten.[8]

Die zweite Messung zielt auf die Unterschiede in der Implementation der Klasse `BigInteger`. `BigInteger` ist die zentrale Komponente der RSA-Operationen und der größte Flaschenhals in der Performance der Applikation. Abbildung 3.21 zeigt die Geschwindigkeit verschiedener RSA-Operationen mit zwei unterschiedlichen `BigInteger`-Klassen.

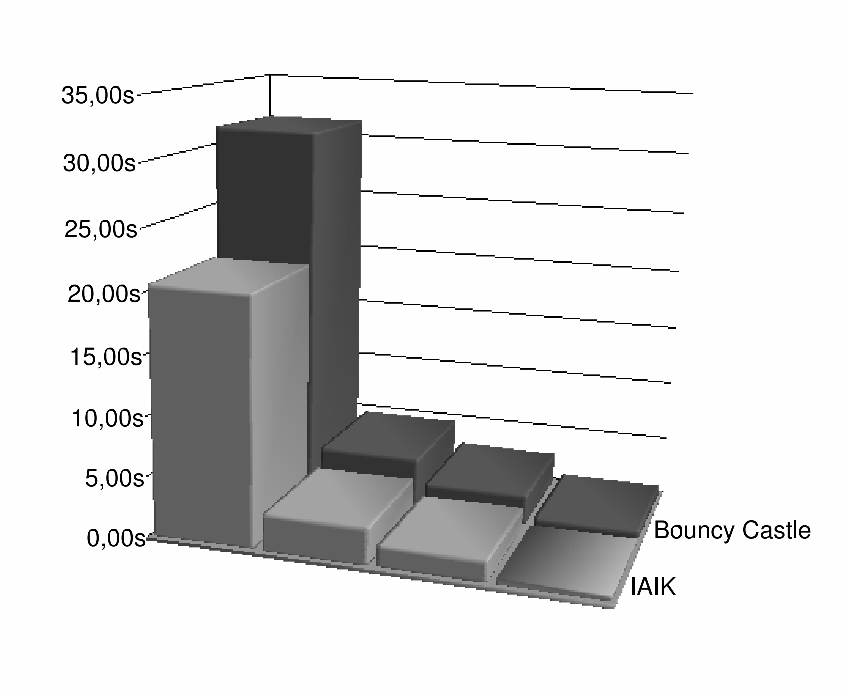

Abbildung 3.21: Einfluss der `BigInteger`-Klasse auf RSA

Die x-Achse unterscheidet die RSA-Operationen (v.l.n.r.): Entschlüsselung mit 2208 Bit langen Schlüsseln, Entschlüsselung mit 1112 Bit langen Schlüsseln, Verschlüsselung mit 2208-Bit Schlüsseln und Verschlüsselung mit 1112-Bit Schlüsseln. Die vordere Zeile wurde mit einer kommerziellen Implementation des „Institute for Applied Information Processing and Communication" ([IAIK]) vorgenommen, die hintere Zeile mit der freien Implementation der Gruppe „Bouncy Castle" ([BC]). Mit dieser Implementation arbeitet auch SeJI (siehe Abschnitt 3.3). Laut Angaben von IAIK arbeitet die hauseigene Implementation mit dem JNI, wodurch ein Geschwindigkeitsvorteil von etwa 50 Prozent bei allen Operationen erzielt wird.

[8] Ein Siemens S55 implementiert nur das MIDP 1.0, ohne WMA. Somit ist dieses Handy ohnehin nicht für SeJI geeignet und wurde daher in der Grafik nicht berücksichtigt.

Kapitel 4

Zusammenfassung und Ausblick

Das letzte Kapitel gibt eine Zusammenfassung dieser Arbeit und einen Ausblick, welche Probleme noch nicht gelöst sind und wo noch Bedarf an weiterer Forschung besteht.

4.1 Zusammenfassung

Ziel der Absicherung von Verbindungsinhalten bei der mobilen Kommunikation im GSM-Standard war es, wenigstens den Sicherheitsstandard von Festnetzverbindungen zu erreichen. Dies gelang nur mangelhaft – die entsprechenden kryptographischen Routinen, die die letzte Strecke zum Handy absichern sollten, waren unzureichend und gelten mittlerweile als gebrochen. Dies trifft gerade die Kommunikation mit textuellen Inhalten wie SMS, da hier die Informationen besser analysiert werden können als bei gesprochenen Wörtern.

Kunden besitzen durchaus ein Sicherheitsbewusstsein, sobald es um kommerzielle Bereiche wie „Online Banking“ oder „Online Shopping“ geht. Solche Bereiche sind für das Handy noch unausgeschöpft. Die fehlenden Sicherheitsvoraussetzungen sind ein Grund dafür. Wirtschaftlich ist die SMS durchaus bedeutsam – auch in naher Zukunft ist nicht zu erwarten, das andere Kurznachrichten-Übertragungssysteme wie ICQ oder Email die Verbreitung und Akzeptanz von SMS auf Mobiltelefonen erreichen.

Es existieren verschiedene Produkte, die kryptographische Sicherheit für SMS versprechen. Von allen Untersuchten entspricht SEMS den Bedürfnissen an Sicherheit, Skalierbarkeit und Offenheit am besten.

SEMS arbeitet asymmetrisch direkt auf RSA und bietet verschiedene Sicherheitsstufen, die sich durch die gewählte Schlüssellänge unterscheiden. Die mittlere Sicherheitsstufe (1120 Bit-Schlüssel) kommt unter bestimmten Bedingungen mit nur einer SMS aus, während die hohe Sicherheitsstufe (2208

Bit-Schlüssel) stets mehrere SMS benötigt.

Für eine Implementierung als Midlet unter Java ME war eine Änderung am Standard notwendig: der im SMS-Standard optionale „User Data Header“ muss in die Nutzdaten der SMS mit aufgenommen werden. Dadurch sind einige Strukturen von SEMS anzupassen. Eine weitere Änderung sieht die Benutzung eines anderen öffentlichen Modulus als den Vorgeschlagenen vor. Dadurch wird SEMS widerstandsfähiger gegenüber Angriffen bei teilweise bekanntem privaten Schlüssel.

Es wurden im Rahmen dieser Arbeit einige Änderungen untersucht und wieder verworfen. Die Benutzung der Asymmetrie ausschließlich zur Übertragung eines symmetrischen Schlüssels beinhaltet zu große Risiken des Schlüsselmanagements, der Zuordenbarkeit von Signaturen und bringt keine Vorteile der nutzbaren Nachrichtenlänge. Elliptische Kurven sind bisher auf nicht längentreue Algorithmen beschränkt, was die Zahl nutzbarer Zeichen einer Nachricht zu stark reduziert.

Die SEMS Java Implementation „SeJI“ ist eine freie Implementation des vorgeschlagenen Standards unter Java ME und MIDP Version 2.0. Die Applikation ist gegliedert in verschiedene Grundkomponente. Ein Adressbuch inklusive eines Schlüsselmanagements zum Generieren, Signieren und Widerrufen von Schlüsseln musste getrennt vom handyinternen Adressbuch programmiert werden. Mehrere Listen für eingehende, bereits versendete und nicht dekodierbare SMS ermöglichen die Ansicht und Bearbeitung von Nachrichten. Schließlich können auch SMS in unterschiedlichen Typen versendet werden.

Eine vollständige Umsetzung von SEMS sprengt den Rahmen einer Diplomarbeit, weshalb SeJI nur Teile des vorgeschlagenen Standards implementiert. Dennoch ist SeJI benutzbar, den wichtigsten wirtschaftlichen Anforderungen kommt die hier vorgestellte Implementation nach.

4.2 Ausblick

Die Weiterentwicklung von SeJI ist ein wichtiges Anwendungsgebiet. SEMS ist ein in sich gut durchdachter Standard, das Weglassen von Komponenten schränkt die Anwendbarkeit ein. Wesentlich problematischer wird eine unvollständige Implementation, wenn Kompatibilität mit anderen Applikationen wichtig wird, die ebenfalls SEMS implementieren. Auch die Performance von SeJI lässt sich steigern, durch direkten Zugriff auf herstellerspezifische API's oder Einsatz alternativer Java-Implementationen.[1] Die Sicherheit der aktuellen Implementation kann verbessert werden, beispielsweise durch eine zusätzliche, passwortgestützte Verschlüsselung des benutzten Dateisystems.

In dieser Arbeit wurden einige Ideen verworfen. Die Gründe dafür sollten periodisch auf Gültigkeit geprüft werden. Wenn längentreue, kryptographi-

[1] Zu Unterschieden in der Performance siehe auch Abschnitt 3.6.

sche Verfahren existieren, die auf elliptischen Kurven anwendbar sind, könnte eine Implementation auf dieser Basis effizienter und ressourcenschonender sein. Oder sollten externe Smartcards zur Speicherung von kryptographischen Schlüsseln und Durchführung von Operationen auch im Handybereich Einzug finden, so kann eine Anpassung der Schlüssellänge auf übliche Werte notwendig werden.

Ein weiteres Forschungsgebiet ist der Schutz der Verbindungsdaten bei einer Nachrichtenkommunikation. SEMS bietet hierfür nur einen rudimentären Ansatz, indem es den Versand von Dummy-Nachrichten ermöglicht.

Die Umsetzung der Implementation auf Handys ohne Java ist eine eher praktische Übung und wird wohl mit der rasant wachsenden Verbreitung von Java zunehmend bedeutungsloser. Eine Anpassung an herstellerspezifische API's kann jedoch sinnvoll sein, wenn dadurch Verbesserungen in der Benutzbarkeit oder Sicherheit zu erwarten sind.[2]

Schließlich macht diese Arbeit einige Annahmen über die wirtschaftliche Weiterentwicklung der SMS. Treffen diese Annahmen wider Erwarten nicht zu, so kann eine Anwendbarkeit von SEMS auf andere Nachrichtenprotokolle untersucht werden.

[2]Soll SeJI als Grundlage einer solchen Implementation dienen, sind Einschränkungen aufgrund der GPL-Lizenz zu beachten.

Abkürzungsverzeichnis

3DES Tripple DES. Siehe DES.

AES Advanced Encryption Standard. Der aktuelle Verschlüsselungsstandard Rijndael.

API Application Programming Interface. Programmierschnittstelle.

ASCII American Standard Code for Information Interchange. Amerikanischer Standard für die Zeichenkodierung.

B2B Buisiness To Buisiness. Betrieb-zu-Betrieb-Kommunikation.

B2C Buisiness To Customer. Betrieb-zu-Kunde-Kommunikation.

C2C Customer To Customer. Kunde-zu-Kunde-Kommunikation.

CFB Cipher Feedback. Ein Operationsmodus für Blockchiffren.

CRC Cyclic Redundancy Check. Ein Verfahren zur Berechnung von (nicht kryptographischen) Prüfsummen.

DES Data Encryption Standard. Seit 1973 geltender Verschlüsselungsstandard. Durch AES abgelöst.

DSA Digital Signature Algorithm. Ein amerikanischer Signaturalgorithmus von 1991.

ECC Elliptic Curve Cryptography. Kryptographieverfahren auf Basis elliptischer Kurven.

ElGamal Kryptosystem von Taher El Gamal von 1985.

EMS Enhanced Message Service. Erweiterung zu SMS.

ETSI European Telecommunications Standards Institute. Europäisches Standardisierungsinstitut für Telekommunikation.

GPL General Public License. Eine Softwarelizenz der Free Software Foundation.

GPRS General Packet Radio Service. Erweiterung zu GSM für schnellen Datentransfer.

GSM Global System for Mobile Communications, vorher Group Special Mobile (franz.). Ein Mobilfunkstandard.

ICQ Aussprache I see you. Netzwerkprotokoll von Mirabilis, jetzt AOL, zum Nachrichtenaustausch.

IV Initialisierungsvektor. Initiale Belegung von Blockchiffren in rekursiven Operationsmodi (siehe CFB, AES).

Java ME Java Micro Edition. Java Version für embedded systems.

Java SE Java Standard Edition. Java Version für Desktop Rechner.

JDK Java Development Kit. Java Entwicklungsumgebung.

JNI Java Native Interface. Ermöglicht die Einbindung von Objektcode in Java.

MIDP Mobile Information Device Profile. Eine API für Java auf embedded systems.

MMS Multimedia Messaging Service. Nachfolger von EMS. Ermöglicht den Versand von multimedialer Daten.

MS Mobile Station. Mobiles Endgerät.

PDA Personal Digital Assistant. Tragbarer Taschencomputer.

PGP Pretty Good Privacy. Verschlüsselungsstandard von Phil Zimmermann aus dem Jahre 1991. Vorrangig für Emails.

PIN Personal Identification Number. Passwortnummer zur Identifikation.

PKCS Public Key Cryptocraphic Standards. Sammlung von Standards zur asymetrischen Kryptographie. PKCS#1 behandelt RSA.

RSA Assymetrisches Kryptosystem benannt nach den Erfindern Rivest, Shamir und Adlermann.

SC Service Center. Im GSM-Standard ein Funkknoten oder Zwischenknoten.

SEMS Secure Message Standard. Holländisch-Deutsches, asymetrisches Kryptosystem.

SHA Secure Hash Algorithm. Amerikanische Hashfunktion von 1991.

SHAKE Some Hashes Again Known Excrements. Weiterentwicklung von Rüdiger Weis zu BEAST.

SMS Short Message Service. Bezeichnet sowohl das Übertragungsprotokoll als auch eine Übertragungseinheit von Kurznachrichten für Handys.

TCP/IP Transport Control Protocol / Internet Protocol. Wichtigstes Netzwerkprotokoll des Internet.

TLS/SSL Transport Layer Security / Secure Socket Layer. Sicherheitsprotokoll für HTTP. TLS ist die Weiterentwicklung von SSL.

UDH User Data Header. Header für verschiedene Metainformationen zu SMS. Liegt im Nutzdatenbereich.

UMTS Universal Mobile Telecommunications System. Weiterentwicklung von GSM.

WAP Wireless Application Protocol. Protokollsammlung ähnlich HTTP und HTML für Handys.

WMA Wireless Message API. Java-API zur Ansteuerung von SMS auf Mobilgeräten.

WTK Wireless Toolkit. Entwicklungswerkzeuge von Sun für Java ME.

WTLS Wireless Transport Layer Security. Abgespeckte TLS-Version für Handys.

Literaturverzeichnis

[SEMS] Secure Message Standard Version 1.0
`http://www.nah6.com/projects/sems/sems.pdf`

[TS23.040] Technical realization of the Short Message Service (3GPP TS 23.040 Version 6.3)

[3GPP] 3rd Global Partnership Project
`http://www.3gpp.org`

[ETSI] European Telecommunications Standards Institute
`http://www.etsi.org`

[ART] Autorité de Régulation des Télécommunications
`http://www.art-telecom.fr`

[VATM] Verband der Anbieter von Telekommunikations- und Mehrwertdiensten
`http://www.vatm.de`

[ID21] Initiative D21 (http://www.initiatived21.de). Siehe auch
`http://www.heise.de/newsticker/meldung/50546`

[SecMIDP] A Brief Introduction to Secure SMS Messaging in MIDP
`http://nds1.forum.nokia.com/nnds/ForumDownloadServlet?id=3621&name=A_Brief_Introduction_to_Secure_SMS_Messaging_in_MIDP_en.pdf`

[SecMes] SecureMessenger
`http://securesms.sourceforge.net`

[GSMHack] Cryptolabs GSM research papers
`http://www.cryptolabs.org/gsm`

[QuasiSMS] Secure SMS messaging using Quasigroup encryption and Java SMS API
`http://www.cs.uku.fi/research/publications/reports/A-2003-1/page187.pdf`

[FortSMS] Fortress SMS
`http://www.fortressmail.net/fortress_sms.htm`

[Kannel] Kannel WAP Gateway
`http://www.kannel.org`

[T9Enc] SMS Encryption through T9
`http://www.esato.com/archive/t.php/t-53528`

[BSVK] Bürgerinitiative für Sicherheit im Versand von Kurznachrichten
`http://www.bsvk.de`

[RSAAttack] Possible Attacks on RSA
`http://members.tripod.com/irish_ronan/rsa/attacks.html`

[20YearRSA] Twenty years of attacks on the RSA cryptosystem
`http://crypto.stanford.edu/~dabo/papers/RSA-survey.ps`

[BC] Bouncy Castle
`http://www.bouncycastle.org`

[Has88] Hastad. „Solving simultaneous modular equations of low degree.“ SIAM Journal of Computing, 1988.

[CFPR96] Coppersmith, Franklin, Patarin, Reiter. „Low-exponent RSA with related messages.“ Eurocrypt 96, vol. 1070, Springer Verlag 1996.

[Cop97] Coppersmith. „Small solutions to polynomial equations, and low exponent RSA vulnerabilities.“ Journal of Cryptology 10, 1997.

[CS98] „A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack“ (Cramer, Shoup, Springer-Verlag, 1998).

[SHAKE] Rüdiger Weis. Weiterentwicklung von BEAST (Lucks, „Conference on Communication and Multimedia Security“, Chapman & Hall, 1996)

[WMA] Wireless Messaging API Version 1.0
`http://java.sun.com`

[BDF98] Boneh, Durfee, Frankel. „An attack on RSA given a fraction of the private key bits.“ Lecture Notes in Computer Scine. Springer Verlag 1998.

[Blö03] Blömer. „New Partial Key Exposure Attacks on RSA.“ Crypto 2003

[ShTr03] A. Shamir, E. Tromer. „Factoring Large Numbers with the TWIRL Device.“
`http://www.wisdom.weizmann.ac.il/~tromer/papers/twirl.pdf`

[DS81] Chaos Computer Club, Datenschleuder Nr. 81
`http://ds.ccc.de/081/byebye512bit`

[IAIK] Institute for Applied Information Processing and Communication
`http://jce.iaik.tugraz.at`

[GPL] General Public Licence
`http://www.gnu.org/copyleft/gpl.html`

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.