Diplomarbeit

Immanuel Scholz

9. Oktober 2004

Immanuel Scholz
Matrikel: 2658156
geb. am: 16.10.1978

Diplomarbeit

Verschlüsselte SMS auf Basis des vorgeschlagenen Standards SEMS

Zielstellung:
Der Bedarf ab 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

Vorwort

Die Kommunikation mit einem mobilen Endgerät, einem Handy ist in der heutigen Gesellschaft nicht mehr wegzudenken. Vom Schulkind über Studenten, Erwachsenen selbst bis zu Rentnern steigt die Zahl der benutzten Handys stetig an. Vier von fünf Deutschen besitzen bereits eins, viele sogar mehrere. Der Absatzmarkt blüht, es werden täglich mehr Handys verkauft und mehr Anrufe mit Handys getätigt.

In einigen Kreisen wie beispielsweise für Geschäftsleute, Reporter oder andere hochmobile Angestellte ist das Handy längst von einem netten Gimmik zu einer notwendigen Bedingung geworden. Gerade in diesen Bereichen ist sichere Kommunikation erforderlich. Das Risiko der Spionage ist für drahtlose Nachrichtenkommunikation besonders hoch, da hier die technischen Rahmenbedingungen für ein Abhören sehr einfach herzustellen sind.

Der Schock war nicht allzu groß, als 1998 die GSM–Verschlüsselung gebrochen wurde. Ein paar Meldungen in Fachzeitschriften, ein wenig Verkehr in Mailinglisten und im Usenet. Die Probleme gingen unter, erreichten längst nicht die Medienwirksamkeit der „Aufladbaren Telefonkarten“, die etwa zeitgleich die Zeitungen aufwühlten. Doch mit der Zeit sind Telefonzellen und deren Kartenautomaten immer unwichtiger, das Mobiltelefon wird immer bedeutender. UMTS ist längst noch nicht so weit, wie es versprochen wurde. 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 beschäftigt sich mit grundlegeneden Mechanismen der mobilen Kommunikation. Es stellt verschiedene Szenarien des wirtschaftlichen Nutzens der SMS vor und definiert Anforderungen zu deren Erfüllung. Schließlich untersucht Kapitel 1 bestehende Standards und Produkte ob diese Anforderungen zutreffen. Kapitel 2 behandelt den „Secure Message Standard“. Erst wird die Struktur und der Aufbau von SEMS beschrieben, es folgt eine Kryptoanalyse des Standards. Schließlich werden Änderungen präsentiert die notwendig waren bzw. verworfen wurden. Kapitel **??** stellt den praktischen Teil dar. Es wird die im Rahmen dieser Arbeit entwickelte SEMS Java Implementation vorgestellt. Kapitel **??** schließlich bietet 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 sind und Kriterien für eine Nutzung herausgestellt. Schließlich 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.

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).
- SC-SC - Übertragung innerhalb des Telekommunikationsnetzes; es werden meist dieselben Leitungen wie für Festnetzkommunikation benutzt.
- SC-MS - Übertragung an das Zielhandy, was auch als SM MT bezeichnet wird (Short Message Mobile Terminated).

Der letzte Schritt kann dabei auch verzögert stattfinden, etwa weil das Zielhandy nicht erreichbar ist.

1.2 GSM-Schwächen

Kurznachrichten werden zusammen mit der Sprachkommunikation über das GSM Netz übertragen.[1] 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 Service Center 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 (stark) oder GSM/A5.2 (schwach) Verschlüsselungsstandard. Die Authentisierung erfolgt nach GSM/A3, die Schlüsselerzeugung nach GSM/A8. Die Algorithmen sind von der GSM-Kommision 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. Darüber hinaus verwenden viele Telekommunikationsanbieter nur eine reduzierte Schlüssellänge von 54 Bit statt der vorgeschriebenen 64 Bit, gerüchterweise um nationalen Geheimdiensten das Mithören zu ermöglichen. [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 die Erhaltung und Bereitstellung des Services. Ein Beispiel ist die normale, private Urlaubs-SMS mit einem Handy. Die Zahl der Kunden und die

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

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 Anspruck 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 eine B2B-Anwendungen wäre 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 4 von 5 Bundesbürgern ein Handy, wobei lediglich jeder zweite Haushalt (53%) 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 jedoch, sobald die Kommunikation kommerzielle Interessen betrifft. So zeigt eine Studie der Firma Fittkau & Maaß GmbH aus dem Jahre 2001 (Abbildung 1.1) 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 Nuzung 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 Markentwicklung im Bereich Betrieb zu Kunde (B2C) stattfinden wird (Abbildung 1.2). Wie Abschnitt 1.2 zeigt, ist eine unberechtigte Nutzung oder ein Ausspähen vertraulicher

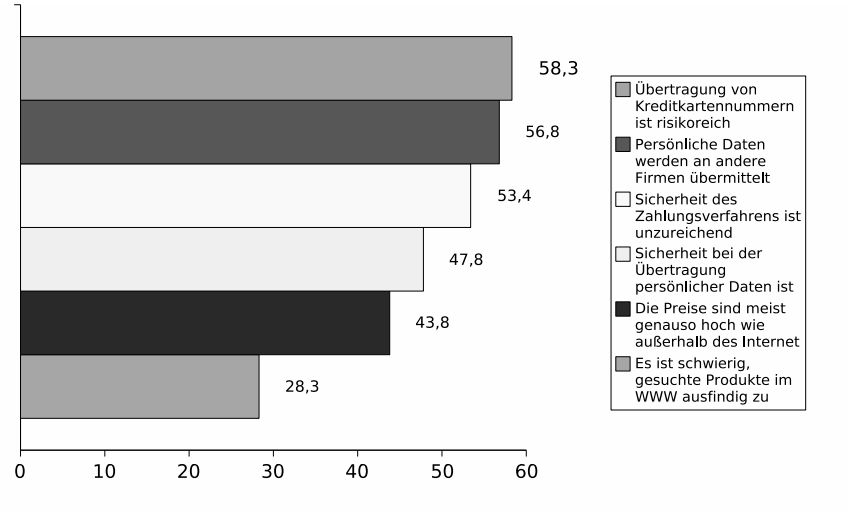

Abbildung 1.1: Sorgen bei kommerzieller Internetbenutzung

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

Wie auch gezeigt wurde werden zwar 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

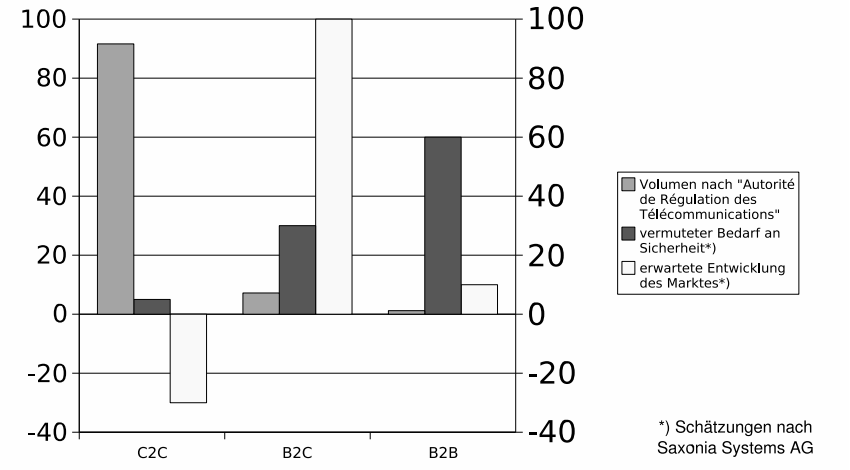

Abbildung 1.2: Marktverteilung und Sicherheitsbedürfnis von SMS

herausgearbeitet, die an Verschlüsselungssoftware zu stellen sind.

1.5.1 Sicherheit

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

1.5.2 Öffentlich

Die Verschlüsselungsprotokolle von GSM haben wieder einmal 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 und nicht patentiert sein. Diese Forderung ist besonders für den B2C Bereich wichtig, da hier soviel Kunden wie nur möglich erreicht werden sollen.

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 bis zu 10 Sekunden ist hier vollkommen akzeptabel.

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 Platformen entwickelt und nun notdürftig an die leistungsschwacheren Handys angepasst. Die Mindestanforderungen sollten nicht über einem Speicherverbrauch von etwa 256KB RAM und einem persistenten Speicher von 1 MB liegen.

1.6 Standards und Produkte

Folgender 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 mathematischer 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 Securitylösung 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.[2] Allerdings enthält die Suite auch eine Komponente zur SMS Verschlüsselung. Auf Anfrage teilte Pointsec mit, dass die Verschlüsselung der SMS symmetrisch mittels AES und vorher ausgetauschten Schlüsseln geschieht.

1.6.5 Kannel

Das OpenSource Programm „Kannel“ ([?]) ist ein WAP- und SMS Gateway. Mittels 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, kurz SEMS, ist ein vorgeschlagener Standard der Universität Mannheim, NAH6 und der freien Universität Amsterdam. Er

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

arbeitet asynchron mit RSA und zwei verschiedenen Schlüssellängen. SEMS ist auf die minimale Anzahl zu versendender SMS optimiert. Nach Angaben der Entwickler existiert bisher keine vollständige Implementationen des Standards.

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 1.3 zeigt tabellarisch, wie sich die verschiedenen untersuchten Standards und Produkte in Bezug auf die geforderten Kriterien verhalten.

Produkt	sicher	öffentlich	frei	asymmetrisch	Geschwindigkeit	Hardware
Fortress SMS	ja, AES	nein	nein	nein	schnell	ok
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 1.3: 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

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 alle nicht-professionellen Angreifer. Zudem benötigt man hier – eine genügend kurze Nachricht vorrausgesetzt – nur eine SMS. Demgegenüber ist die hohe Sicherheitsstufe mit 2208 Bit Schlüssellänge selbst für die Zukunft ausreichend gegen das Durchprobieren aller Schlüssel.

Unabhängig von der Schlüssellänge können Nachrichten signiert, verschlüsselt oder signiert und verschlüsselt übertragen werden. Signaturen können dabei entweder direkt in der SMS enthalten oder als eigene SMS angehängt werden.[2] 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 oder Schlüsselwiderruf mit bis zu 32 angehängten Signaturen 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] 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 bietet sich die unverschlüsselten Modi an.

2.2.2 Mittlere Sicherheit

Die mittlere Sicherheitsstufe bietet mit einer Schlüssellänge von 1120 Bit und dem 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 professionele Angreifer mit größerem Etat für Hardware sicher sein, so ist die Benutzung der hohen Sicherheitsstufe erforderlich. Die Schlüssellänge beträgt hier 2208 Bit was als ausreichend gilt. Das hat allerdings zur Folge, dass jede Nachricht stets mindestens zwei Kurznachrichten zum Versand benötigt.

2.2.4 Unsigniert

Kurznachrichten können unsigniert verschlüsselt versendet werden. Dabei richtet sich die Anzahl der benötigten SMS an der Sicherheitsstufe des Empfängers. Besitzt der Empfänger beispielsweise einen Schlüssel mittlerer Sicherheitsstufe, so wird eine SMS mit maximal 989 Bits Nachricht benötigt.

Allerdings is das unsignierte Verschlüsseln von SMS generell mit Bedacht zu verwenden, da der Absender einer solchen Nachricht nicht sicher zuordenbar ist.[3]

[3]Der Absender einer SMS läßt sich mit nur wenig Aufwand fälschen.

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 Sicherheitsstufe. 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 Mistrauen 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	2-3 SMS
ausgelagerte Signatur hoher Sicherheit	3 SMS	3 SMS	3-4 SMS

Abbildung 2.1: Notwendige Zahl der SMS

Die Zahlen beziehen sich dabei auf die mindestens notwendige Anzahl. Für die ausgelagerte Signatur hängt die Zahl der SMS von der Verschlüsselungsstufe der vorangegangenen SMS ab.

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.[4]

Unverschlüsselt, unsigniert

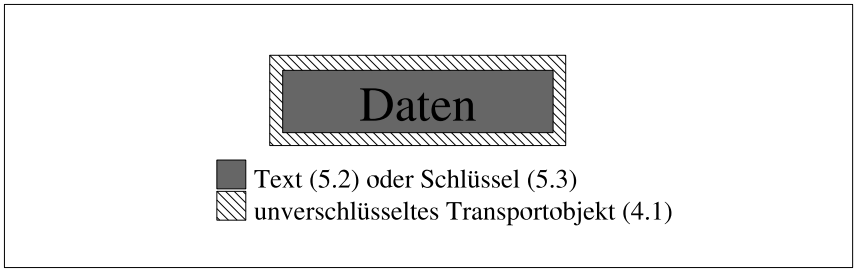

Abbildung 2.2: Unverschlüsselt, unsigniert

Unverschlüsselte und unsignierte SMS entsprechen am ehesten den herkömmlichen Kurznachrichten, lediglich Protokollvorteile wie die verschiedenen Kodierungen oder das Aufteilen von Nachrichten über mehrere SMS können genutzt werden[5].

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

Unverschlüsselt, integrierte Signatur mittlerer Sicherheitsstufe

Dieser Modus stellt den Standardweg dar, eine Kurznachricht signiert und mit wenig Overhead zu versenden. 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.

[4] Diese Kombination hat nur Nachteile gegenüber der ausgelagerten Signaturvariante selber Sicherheitsstufen.

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

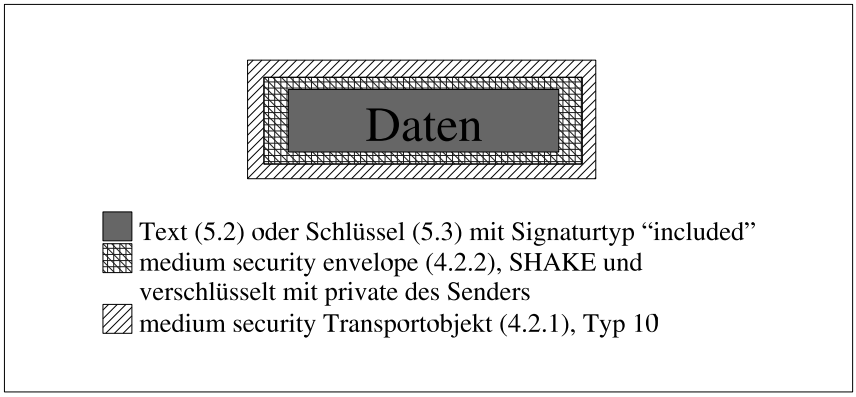

Abbildung 2.3: Unverschlüsselt, integrierte Signatur mittlerer Sicherheitsstufe

Verschlüsselung und ausgelagerte Signatur mittlerer Sicherheitsstufe

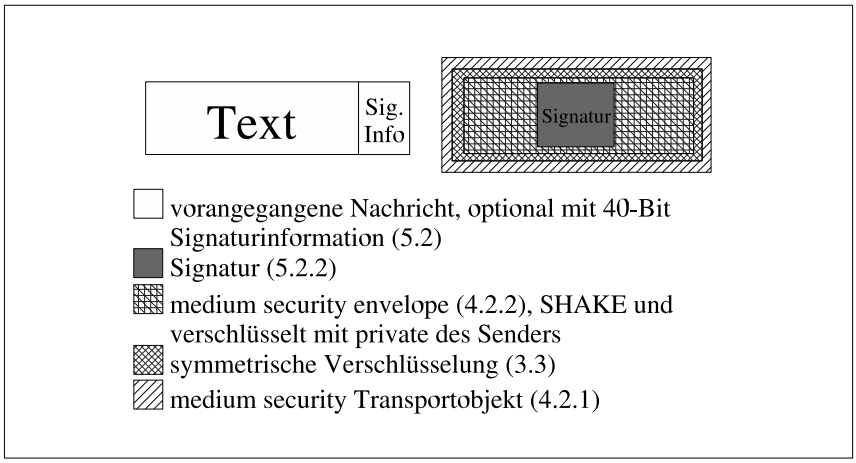

Abbildung 2.4: Verschlüsselung und ausgelagerte Signatur mittlerer Sicherheitsstufe

Hier wird erst ein Hashwert der vorangegangenen Nachricht gebildet, welcher dann signiert und mit AES verschlüsselt wird. Der Schlüssel und IV des symmetrischen Verfahrens 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[6] zwei SMS als Signatur, was wohl ein Grund für die Wahl eines symmetrischen Verfahrens war.

Außerdem ist auf der Abbildung 2.4 zu erkennen, dass bei ausgelagerten

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

Signaturen viel Platz innerhalb des envelope ungenutzt bleibt. Die eigentlichen Nutzdaten, die in den Signierprozess eingehen ist nur eine 160 Bit große SHA-1 Ausgabe.

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, Schlüsselmanagment
- Sonstige Probleme

2.3.1 Sicherheit der benutzten Komponenten und deren Wechselwirkung

SEMS setzt verschiedene kryptographische Verfahren und Algorithmen ein. Sowohl der zu benutzende Algorithmus als auch deren Arbeitsmodus und 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 sind zahlreiche Probleme und Schwächen bekannt, denen sich auch SEMS stellen muss. Da dies wohl den offensichtlichsten und umfangreichsten Aspekt der Kryptoanalyse darstellt, ist dem Problem „RSA direkt“ ein eigener Abschnitt 2.4 gewidmet.

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} ist heutzutage ausreichend sicher. Außerdem enthält die Eingabe zu SHA immer Redundanz, was die Suche nach Kollisionen erschwert.[7] Es ist bis heute kein erfolgreicher An-

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

griff 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 ausgelagerten Signaturen eingesetzt wird, wird mittles verschiedener Konkatenation und Addition modulo zwei aus zwei SHA-1 Operationen gewonnen. Die Sicherheit dieser AES-Operation könnte als durch SHA-1 abgeschwächt werden. Eine hinreichende Forderung an das Hashverfahren wäre hier eine Ausgabe, die in vertretbarem Aufwand nicht von einer Pseudozufallszahl unterscheidbar ist.[8] 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 CBC Modus mit einer Schlüssellänge von 256 Bit und einem IV von 128 Bit verwendet. Der Standard lautet hier wie folgt (|| ist die Konkatenation):

$$K_1 = SHA1(0x00 \mid\mid envelope)$$

$$K_2 = SHA1(0x0000 \mid\mid envelope)$$

$$Schlüssel = K_1 \mid\mid K_1 \; XOR \; K_2$$

$$IV = K_2\{letzten \; 128 \; bits\}$$

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.

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.

[8] Wobei die Eingabe keine Zufallszahl sein muss.

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. Damit ist die Ausgabe von SHAKE mindestens so zufällig wie die Ausgabe von SHA-1.[9]

2.3.2 Sicherheit der Schlüssel, Schlüsselmanagment

Nicht nur die Sicherheit der verwendeten Verfahren und Algorithmen ist wichtig für ein kryptographisches System, sondern auch, falls es denn damit arbeitet, das Managment der Schlüssel. Wärend die meisten untersuchten Kryptosysteme für Kurznachichten überhaupt nicht oder nur sehr spärlich auf das Schlüsselmanagment 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.

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.

- In der Beschreibung der Operationsmodi von SEMS wird SHAKE nur für unverschlüsselte Transportobjekte erwähnt. Dies ist so selbstverständlich nicht sicher. SHAKE muss immer auf den Klartext angewendet werden, bevor mit diesem eine RSA Operation durchgeführt wird.
- An einigen Stellen werden Referenzen auf falsche Strukturen genannt, vermutlich aufgrund von Kopier- und Einfügeoperationen. So soll z.B. in Abschnitt 2.4.3 des SEMS-Standards eine Signatur hoher Sicherheitsstufe in ein Umschlag mittlerer Stufe eingebettet werden. Dies ist so natürlich nicht sicher und sicher 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 Bits zu spezifizieren. Ein Umschlag, der mehr als 2048 Nutzdaten enthält,

[9] Dies gilt nur unter der Vorraussetzung, 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.

kann am Empfänger nicht korrekt dargestellt werden. Der nicht nutzbare wird dennoch übertragen und mitsigniert, was man beispielsweise als „unsictbares Kleingedrucktes“ in einer Signatur misbrauchen kann.[10]

- 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.[11] 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 Benutzung von PKCS#1 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 Kurznachichten untersucht.

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

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

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

oder TLS/SSL-Funktionalität zur Verfügung stellen (z.B. [BC]). Eine Implementation eines neuen Standards macht Anwendungen, die darauf bauen 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 immernoch 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, der diese Zahl der versendeten Nachrichten minimiert notwendig.

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 dafür bekannt. Tabelle 2.5 listet einige solche Angriffe und Sicherungsmöglichkeiten dagegen auf.

Angriff	Bedingung	Sicherungsmöglichkeit
[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 2.5: Angriffe auf RSA mit kleinem öffentlichen Exponenten

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 Exponenten.

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 von 2^{2208} 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.

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 Performence eine stark untergeordnete Rolle, die Übertragung der SMS selbst beansprucht 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

Es war eine Änderung am Standard zwingend notwendig um die Implementation in Java zu ermöglichen. Der „User Data Header“, der zur Addressierung 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.[12]

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

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-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 8-Bit SMS an einen Zielport versand mindestens 133 Bytes Daten enthalten kann.[13] Eine Applikation, die auf unterschiedlichen WMA-Platformen 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 addressiert werden. Allerdings existieren verschiedene MIDP-Erweiterungen der Hersteller, die es ermöglichen, entweder unaddresierte 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 werden muss. Der Nachteil, man verliert die MIDP-Standardkonformität, wiegt jedoch weit mehr auf.

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

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 bedeuted, dass Nutzer der Implemenation 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 was etwa 7% entspricht. Es muss dafür auch die Länge des Schlüssels auf 1056 bzw. 2096 Bit gekürzt werden, da durch eine RSA Operation auf dem security envelope mit einem vollen Schlüssel die Originallänge widerhergestellt 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 Konsistenzüberprüfung 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 müsste noch die Schlüssellänge um ebenfalls 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 Packet mittlerer Sicherheitsstufe müssen noch die Nutzdaten auf 1045 bzw. 965 Bit verringert werden. Bei Packeten hoher Sicherheit wird die doppelte Zahl an Bit eingespart.[14] Die Prüfsumme und weitere 80 Bit Nutzdaten fallen im High Security Envelope weg.

Die Schlüssel verkürzen sich ebenfalls auf 1056 bzw. 2096 Bits. Dafür muss 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 Bits.
- 4.2.1 Medium security packet format – Die Länge des „Medium security crypto envelope“ von 1112 auf 1056 Bits.

[14] Da beide versendete SMS einen UDH benötigen.

- 4.2.2 Medium security crypto envelope – Das Feld CRC wegstreichen und den Payload von 989 auf 965 Bits.
- 4.3.1.1 und 4.3.1.2 High security packet – Den „Security crypto envelope“ von 1104 auf 1048 Bits.
- 4.3.2 High security crypto envelope – Das Feld CRC wegstreichen und den Payload von 2084 auf 2004 Bits.[15]
- 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, das 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.[16]

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.

Die Wahl des Exponenten $e = 3$ folgte lediglich aus Performancegründen.[17] 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 ([SEMS], Abschnitt 5.3) bereits vor. 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.

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

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

[17] Auf Nachfragen bei den Entwicklern.

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 Empfängfer vorliegt. Der Autor dieser Arbeit teilt diese Meinung nicht. In der Beispielimplementation werden bei Empfang einer integriert-signierten SMS stehts alle im Addressbuch 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“[18] 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. Leider waren zur Abgabe dieser Arbeit keine Klärungen durch SEMS-Entwickler vorliegend. Folgende Vermutungen wurden getroffen:

- 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.

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

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

- 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 enstanden 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 großen Teil aus Diskussionen in Emails 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.[19]

Demgegenüber stehen gravierende Sicherheitsbedenken. Das Schlüsselmanagment 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.[20]

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

[20] Außer man bedient sich einer dritten, vertrauenswürdigen Instanz als Mittler. Es fallen

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 lange Schlüssel in der Elliptische Kurvenkryptographie entsprechen vermutlich 1024 Bit langen 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.[21] 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, gibt es bis heute noch nicht.

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.

Neuste Bestrebungen in der Faktorisierung ([ShTr03], [DS81]) lassen jedoch Zweifel aufkommen, ob 1120 Bit lange genug ausreichend sein werden.

dann Kosten für zusätzliche SMS von und zu dieser Instanz an.

[21] 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.

Da der Standard sich auf eine feste Bitzahl beschränkt muss diese so gewählt sein, dass auch in Zukunft keine Sicherheitsbedenken die Akzeptanz trüben.

Der Vorteil durch die Einschränkung der Komplexität scheint hier auch fraglich zu sein, da keine kryptographischen Algorithmen direkt betroffen sind. Was einfacher werden würde ist die Implementation der nicht-kryptographischen Komponenten des Standards.

2.6.4 1024 und 2048 Bit Schlüssel zulassen

1120 und 2208 Bit sind eine ungewöhnliche Schlüssellänge. Auch die in SeJI benutzte Schlüssellänge von 1056 bzw. 2096 Bit ist nicht üblich. Es ist unwahrscheinlich, dass viele 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, wärend andere Standards wie PKCS dies nicht vorschreiben oder sogar verbieten.[22] Ein Import von Schlüsseln wäre also nur für diejenigen möglich, die ebenfalls diesen Exponenten einsetzen.

Außerdem ist die Vorgehensweise, geheime Schlüssel über Rechnergrenzen hinweg zu transportieren kritisch. Besonders bei der Übertragung auf ein mobiles Gerät muss dies extrem sorgfältig geschehen, damit der Schlüssel nicht während des Transportes 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 unorthodoxen Schlüsselgenerierung. Außerdem würde 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 vorangeganene 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

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

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[23] und der ursprüngliche Schlüssel bereits abgelaufen oder zurückgezogen sein kann.

[23]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 mithilfe des sogenannten „Wasserfallmodell“ 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 ebenfalls drei Komponenten von SeJI abgebildet werden:

- Senden von SMS. Nachrichten verschiedener Typen können gesendet werden. Die Nachricht wird dabei entsprechend gewählten Einstellungen kodiert.
- Empfang von SMS. Es können empfangene Nachrichten dekodiert werden. Da die Kodierung kann aufgrund von fehlenden oder ungültigen

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

- Schlüsselmanagment. Schlüssel können in SEMS generiert, signiert, widerrufen und versendet werden. Schlüssel sind stets Personen und Telefonnummern zugeordnet. Eine Addressbuchverwaltung mit integriertem Schlüsselmanagment bietet sich hier an.

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

3.1.1 Schlüsselgenerierung

Der Nutzer kann eigenen privaten Schlüssel generieren. Die Schlüsselerzeugung findet vollständig auf dem Mobiltelefon statt und darf lange dauern, da sie nur einmalig geschieht. Der Nutzer darf beliebig viele eigene Schlüssel besitzen. Als Parameter sind auszuwählen:

- Sicherheitsstufe. Entweder mittlere Stufe, was einem 1120-Bit Schlüssel entspricht oder hohe Stufe, was 2208 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 Widerrufungen können versendet und gelöscht werden.

3.1.4 Das Addressbuch

Der Nutzer kann alle Einträge in seinem Addressbuch 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. Der Name muss nicht mit den Namen aus angehangen Schlüsseln übereinstimmen. Jeder Name darf nur einmal im Addressbuch 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 angeigt. 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 seperaten 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.
- Schlüssel können in das Addressbuch übernommen werden. Dazu kann ein bestehender Eintrag ausgewählt oder ein Neuer erzeugt werden.

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

- 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 handhaben. Diese werden direkt in Java Klassen umgesetzt. Jede dieser Datenstrukturen ist dabei fähig, sich aus einer Bitfolge zu konstruieren und in eine Bitfolge zu serilisieren. Auf den Strukturen 4.2.2 und 4.3.2 („envelopes“) können zudem kryptographische Operationen durchgeführt werden.

Verwaltung der Addressen und SMS Listen

Da in MIDP keine Schnittstelle zum Handy-internen Addressbuch existiert, muss eine Personenverwaltung selbst implementiert werden. Die Verwaltung des Addressbuches 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 Menuführung 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 Klassenmodelles für die verschiedenen Komponenten. Die Datenklassen von SEMS lassen sich recht einfach auf ein objektorientiertes Modell abbilden, da sie im Standard bereits nach einer Vererbungshirarchie angeordnet sind. Abbildung 3.1 zeigt ein UML-Diagramm der SEMS-Datenklassen. Ein `Data`-Objekt ist in ein `Envelope` eingebettet, welches wiederum in ei-

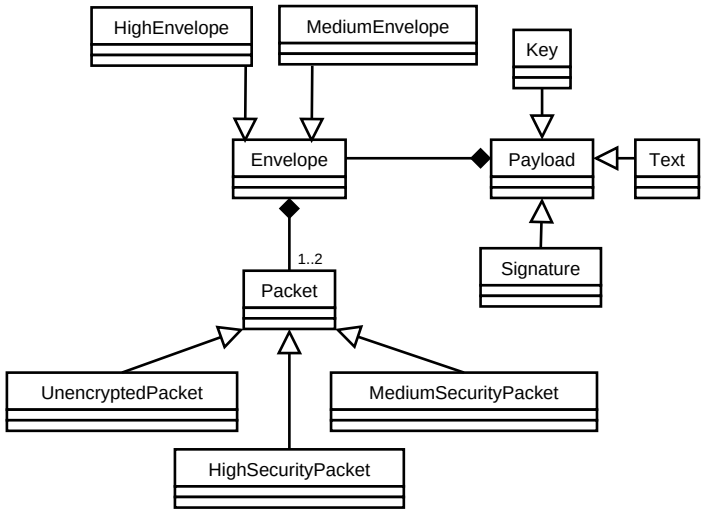

Abbildung 3.1: Klassenmodell für die Datenklassen von SEMS

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 Addressverwaltung sind ebenfalls Datenklassen. Abbildung 3.2 zeigt eine Übersicht über diese Komponente. Einige Informationen sind hier nocheinmal modeliert, zum Beispiel die öffentlichen Schlüssel. Sowohl `Key` in der Sems Komponente als auch `PublicKey` innerhalb der Addressdatenklassen enthalten Informationen über den Schlüssel wie der öffentliche Modulus. Dies ist gewollt, da so einerseits die Vererbungshirarchie vereinfacht wird (von wem sollte `Key` aus SEMS erben? Von `Data` oder `Key` in der Addressdatenkomponente?) und andererseits das Design der Application

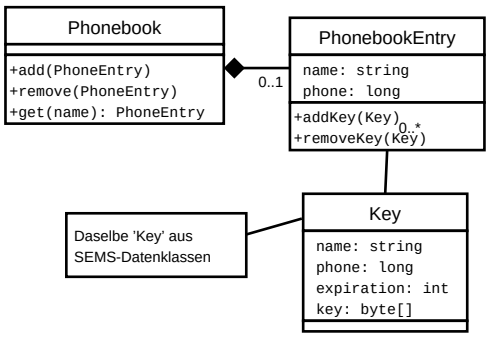

Abbildung 3.2: Klassenmodell für die Datenklassen der Addressverwaltung

vom SEMS Standard gelöst wird. Sollten sich später die Strukturen ändern, so beeinflusst dies nicht das Speicherformat der SeJI Applikation.

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-

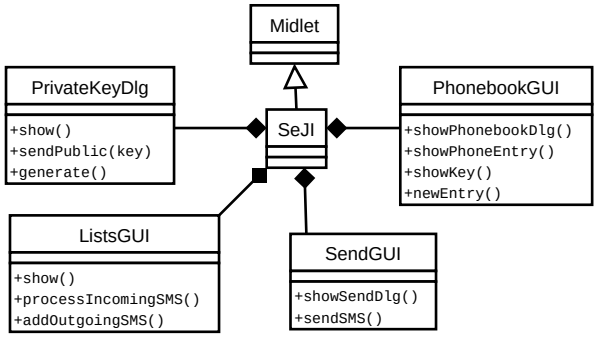

Abbildung 3.3: Klassenmodell für die Oberflächenklassen

zeige eingegangener Nachrichten und das Addressbuch existiert jeweils eine Oberflächenklasse die wiederum mehrere Dialoge enthält. Somit wurde ein Kompromis 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, die Java Midlets zur Verfügung stehen beschränkt. Diese sind:

- Textbeschriftungen und Bilder zur Darstellung von Informationen. Texte können als Laufschrift animiert werden.
- Auswahllisten geben die Möglichkeit, aus mehreren Alternativen auszuwählen
- Menupunkte 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 Menupunkte als numerierte Liste auf dem Display erscheinen, die über die Betätigung der entsprechenden Nummer ausgewählt werden. Falls das Handy Kurzwahltasten unterstützt könnten die Menupunkte 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 vorangehängt, 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.

Hauptmenu

Das Hauptmenu 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 Addressbuch verwaltet.

[4] Verschlüsselte SMS, für die kein geheimer Schlüssel oder SMS mit integrierter Signatur für die kein Testschlüssel zur Verfügung steht können nicht ausgewertet werden.

Abbildung 3.4: Skizze zum „Hauptmenu“

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

Send

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

- das Hauptmenu. Es werden Textnachrichten versendet.
- die Outbox. Es werden ausgelagerte Signaturen versendet.
- die Addressverwaltung. Es können Schlüssel und Schlüsselwiderrufe versendet werden.

Lists

Es können verschieden Listen betrachtet werden. Über Menupunkte 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 Menupunkte gelöscht und betrachtet werden. Schlüssel können zudem importiert werden.

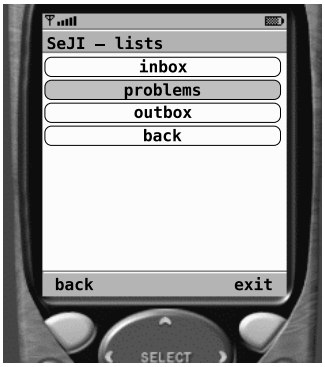

Abbildung 3.5: Skizze zum Menu „Lists“

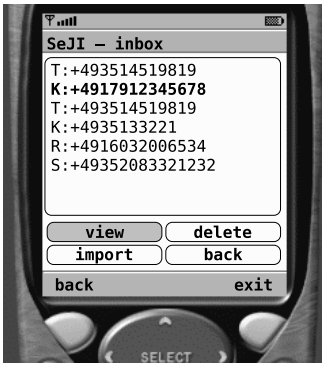

Abbildung 3.6: Skizze zur Liste „Inbox“

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 Menupunkte 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.

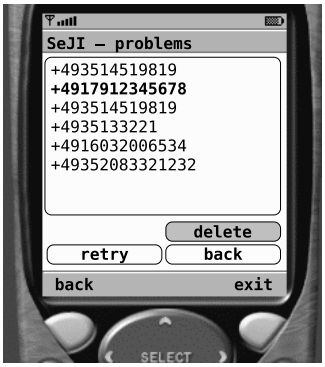

Abbildung 3.7: Skizze zur Liste „Problems“

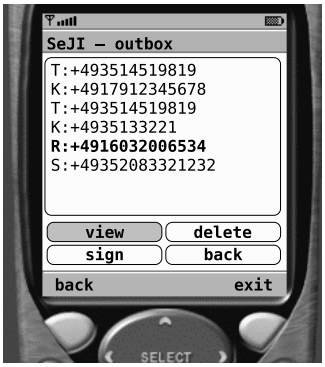

Abbildung 3.8: Skizze zur Liste „Outbox“

Addresses

Die Addressliste beinhaltet alle Telefonbucheinträge, die innerhalb von SeJI aufgenommen wurden. Über Menupunkte 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 Menupunkt 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, ansonst um einen öffentlichen Schlüssel. Schlüssel können angezeigt und gelöscht werden. Über einen Menupunkt können zudem neue Schlüssel generiert werden.

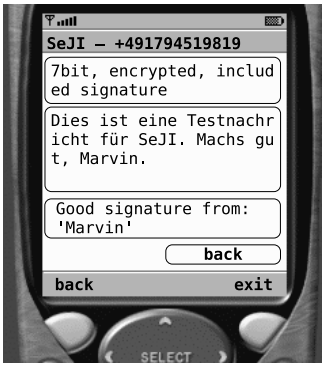

Abbildung 3.9: Skizze zur Funktion „SMS anzeigen“

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 Menupunkt 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“ Menuoption 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“

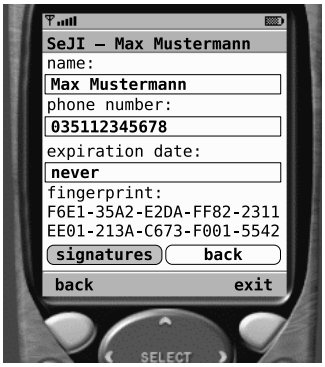

Abbildung 3.12: Skizze zur Funktion „Schlüssel anzeigen“

zwecken kryptographischer Anwendungen auf Java-Handys, als Beweis der Implementierbarkeit von SEMS und als Messplatform 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.

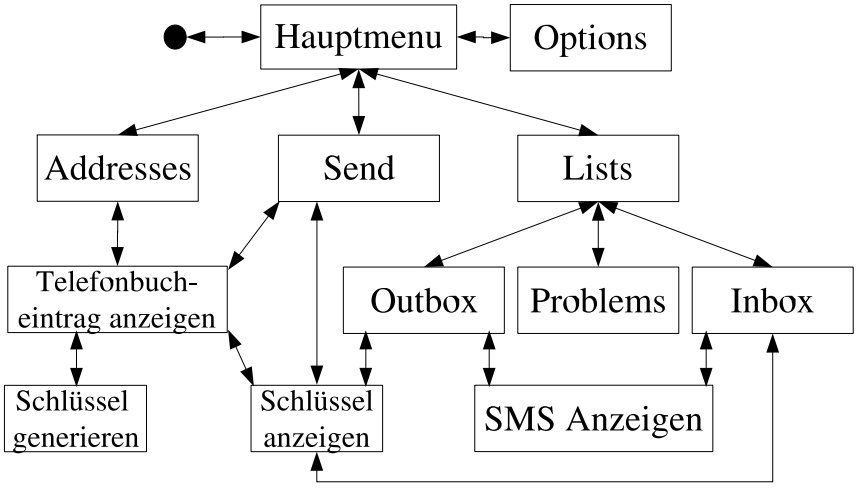

Abbildung 3.13: Übersicht aller Bildschirme in SeJI

tion `toUnsignedByteArray(int)` hinzugefügt, welche die Zahl als ein nicht signiertes `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 Hauptprogrammes 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 Hauptmenu oder den Optionsbildschirm. Im Konstruktor werden dann diese Variablen gesetzt, die Dialoge initialisiert und mit Menupunkten aufgefüllt. Die Funktion `commandAction`, deklariert im Interface `CommandListener` wird bei Anwahl eines Menupunktes durch den Benutzer angesprungen. `SeJI` ist selbst ein Lauscher für alle eigenen Menupunkte, die Unterscheidung welcher Menupunkt 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, d ie sich allerdings nicht bestätigten getroffen. Die Abbildung verdeutlich 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 ?? 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`

??), so dass besondere Aufmerksamkeit auf die effiziente Benutzung von `BigInteger` gelegt wurde.[6]

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

[6] An der Implementation von `BigInteger` selber wurde nichts geändert, mit einer Ausnahme: Eine notwendige Funktion zur Interpretation als unsigniertes `byte`-Array wurde hinzugefügt.

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, besonders I/O Zugriffe 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 Signature ist möglich).
- Schlüssel können nicht automatisch 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 Zielklientel sollten diese Grenzen variiert werden. Beispielsweise sollte ein signiertes Auktionsangebot

innerhalb weniger Sekunden versendet werden, 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 wären beispielsweise sinnvoll. Die lange Dauer der Schlüsselgenerierung rührt daher, dass dieser Pro-

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

zess 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 **??** zeigt die Dauer verschiedener typischer Funktionen in SeJI.

3.6.3 Messungen einzelner kryptographischer Operationen

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

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 3.18 zeigt das Ergebnis.

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

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

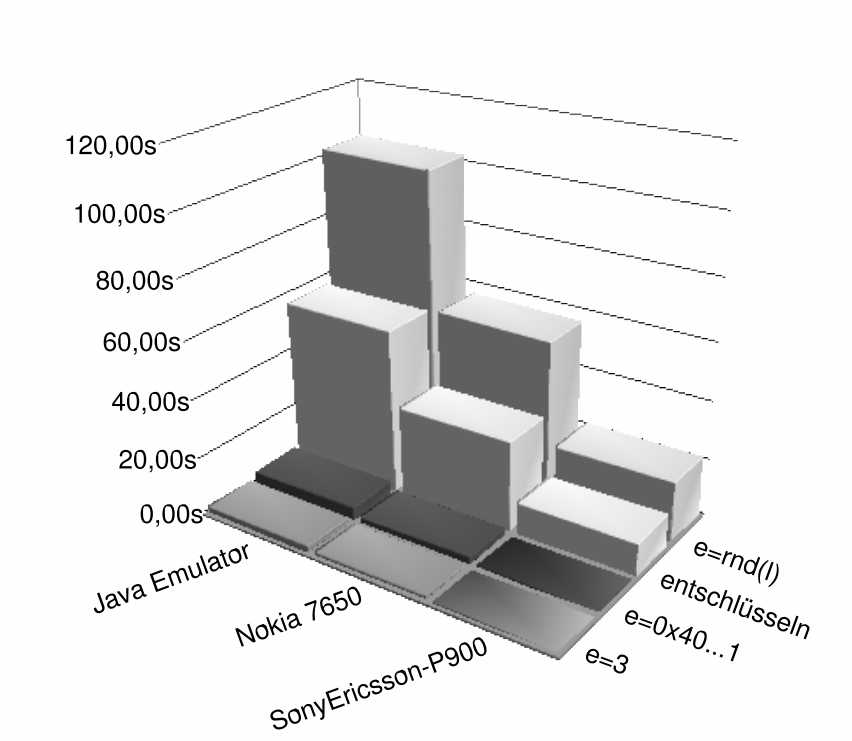

Abbildung 3.18: Einfluss des öffentlichen Exponenten auf RSA

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 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 $e = 3$ reichlich 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 3.19 zeigt die Geschwindigkeit verschiedener RSA-Operationen

[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.

mit zwei unterschiedlichen `BigInteger`-Klassen.

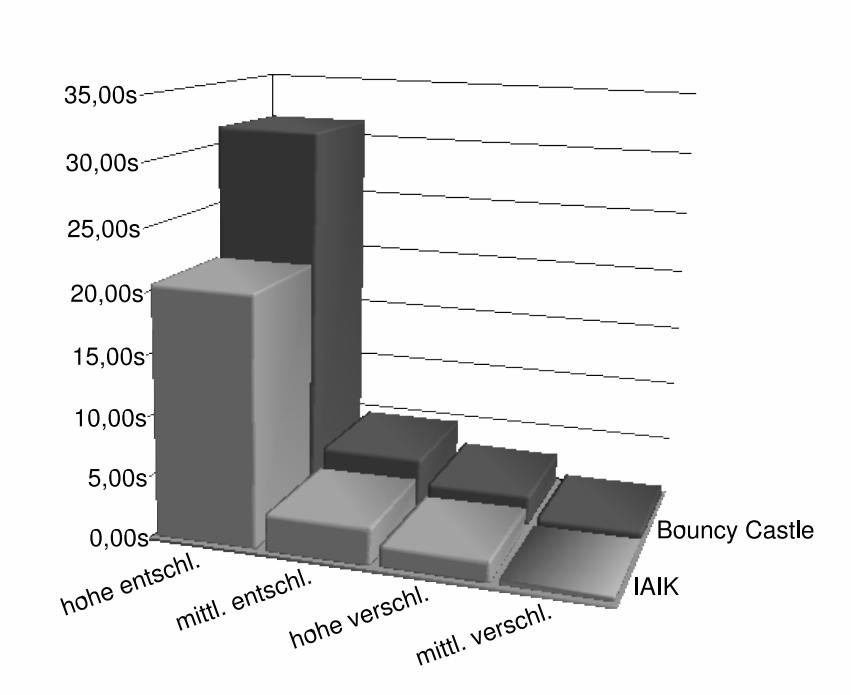

Abbildung 3.19: Einfluss der `BigInteger`-Klasse auf RSA

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 langen Schlüsseln und Verschlüsselung mit 1112 Bit langen Schlüsseln. Die vordere Zeile wurde mit einer kommerziellen Implementation des „Institute for Applied Information Processing and Communication“ ([?]) 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 heuseigene Implementation mit dem JNI, wodurch ein Geschwindigkeitsvorteil von etwa 50% bei allen Operationen erzielt wird.

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 ist.

4.1 Zusammenfassung

Ziel der Absicherung von Verbindungsinhalten bei der mobilen Kommunikation war es, wenigstens den Sicherheitsstandard von Festnetzverbindungen zu erreichen. Bedauerlicherweise gelang dies nur mangelhaft – die entsprechenden kryptographischen Routinen, die die letzte Streke zum Handy absichern sollten waren unzureichend und gelten mittlerweise als gebrochen. Dies trifft gerade die Kommunikation über textuelle Nachrichten, da hier die Inhalte besser analysiert werden können als bei gesprochenen Wörtern.

Die Kunden besitzen durchaus ein Sicherheitsbewustsein, sobald es um traditionell bedenkliche Bereiche wie „Online Banking“ oder „Online Einkaufen“ geht. Dieser Bereich ist wirtschaftlich noch unausgeschöpft. Die fehlenden Sicherheitsvorraussetzungen bei SMS sind ein Grund dafür. Und 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 kryptographisch starke Sicherheit für SMS bieten sollen. Den Bedürfnissen an Sicherheit, Skalierbarkeit und Offenheit als am besten entsprechend wurde SEMS herausgestellt.

SEMS arbeitet assymetrisch 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 Java Midlet war eine Änderung am Standard notwendig: der sogenannte „User Data Header“ muss in die Nutzdaten der SMS mit aufgenommen werden. Dadurch mussten einige Strukturen von SEMS angepasst werden. Eine weitere optionale Änderung sieht die Benutzung eines anderen öffentlichen Modulus als den vorgeschlagenen vor. Dadurch wird SEMS widerstandsfähiger gegenüber Angriffen bei teilweise bekannten Schlüsseln.

Es wurden im Rahmen der Diplomarbeit einige Änderungen untersucht und später wieder verworfen. Die Nutzung der Asymetrie ausschließlich zur Übertragung eines symmetrischen Schlüssels beinhaltet zu große Probleme des Schlüsselmanagments, der Zuordenbarkeit von Signaturen und bringen keine Vorteile der nutzbaren Nachrichtenlänge. Elliptische Kurven sind bisher auf nicht längentreue Algorithmen beschränkt, was die Länge einer nutzbaren Nachricht zu stark reduziert.

Die SEMS Java Implementation „SeJI“ ist eine freie Implementation des vorgeschlagenen Standards unter Java MIDP2.0. Die Applikation ist gegliedert in verschiedene Grundkomponente. Ein Addressbuch inklusive eines Schlüsselmanagments zum generieren, signieren und widerrufen von Schlüsseln musste getrennt vom handyinternen Addressbuch 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.

Leider sprengt eine vollständige Implementation von SEMS den Rahmen einer Diplomarbeit, weshalb SeJI unvollendet bleibt. Dennoch ist SeJI benutzbar, den wichtigsten wirtschaftlichen Anforderungen kommt die hier vorgestellte Implementation nach.

4.2 Ausblick

Die Weiterentwicklung von SeJI ist ein offensichtliches Anwendungsgebiet. SEMS ist in sich ein gut designter Standard und das Weglassen von Komponenten schränkt die Anwendbarkeit ein. Wesentlich problematischer wird eine unvollständige Implementation, wenn andere Kompabilität mit anderen Applikationen wichtig wird. Auch die Performance von SeJI läßt sich noch steigern, durch direkten Zugriff auf herstellerspezifische APIs oder Einsatz alternativer Java-Implementationen[1] Die aktuelle Implementation kann in vielen Gebieten noch verbessert werden, etwa durch eine zusätzliche, passwortgestützte Verschlüsselung des benutzten Dateisystems.

Die in dieser Arbeit verworfenen Ideen sollten periodisch auf Gültigkeit geprüft werden. Wenn längentreue, kryptographische Verfahren in Zusammenarbeit mit elliptischen Kurven existieren, könnte eine Implementati-

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

on auf dieser Basis effizienter und Ressourcenschonender sein. Oder sollten beispielsweise externe Smartcards zur Speicherung von kryptographischen Schlüsseln und 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 Nachrichtenkommunikation. SEMS bietet hierfür nur einen rudimentären Ansatz, indem es den Versand von Dummy-Nachrichten ermöglicht.

Die Umsetzung der Implementation auf Handys ohne Javafähigkeit 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]Es sind Einschränkungen zu beachten, die eventuell aufgrund der GPL-Lizenz für SeJI bestehen.

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`

[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`

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

Eidesstattliche Erklärung

Ich erkläre ehrenwörtlich, 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.