Hochschule für Angewandte Wissenschaften Hamburg

Hamburg University of Applied Sciences

Bachelorarbeit

Markus Schüring

Verschlüsselung textbasierter Kommunikation auf Android Endgeräten

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Markus Schüring

Verschlüsselung textbasierter Kommunikation auf Android Endgeräten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Abgegeben am 16.08.2010

Markus Schüring

Thema der Bachelorarbeit
Verschlüsselung textbasierter Kommunikation auf Android Endgeräten
Stichworte
Android, Kryptographie, SMS, E-Mail, RSA, AES
Kurzzusammenfassung
Mobile Kommunikation in Schrift und Wort ist heute eine Selbstverständlichkeit. Über die Sicherheit machen sich viele jedoch keine Gedanken. Wenn jemand zum Mobiltelefon greift denkt er im Normalfall nicht darüber nach, ob jemand seine SMS-Nachricht oder E-Mail, die er gerade versendet hat, mitlesen kann.
In dieser Arbeit soll gezeigt werden, was im Bereich der Verschlüsselung auf mobilen Plattformen möglich ist. Exemplarisch soll dies hier anhand der Android Plattform und den Kommunikationsformen SMS und E-Mail gezeigt werden. Zu diesem Zweck soll ein Prototyp erstellt werden, der es möglich macht, SMS Nachrichten zu verschlüsseln und zu senden, sowie diese zu empfangen und entschlüsseln. Des weiteren soll der Kernbestandteil der Anwendung in eine Bibliothek ausgelagert werden. Zu den Kernbestandteilen gehören die Ver- und Entschlüsselung, die Konvertierung in einen für die Nachricht entsprechenden Zeichensatz, gegebenenfalls Kompression der Nachricht und die Schnittstelle zum Absenden der Nachrichten.

Markus Schüring

Title of the paper
Encryption of textbased communication on Android devices
Keywords
Android, cryptographie, sms, email, rsa, aes
Abstract
Nowadays mobile communication is everywhere, but hardly anyone cares about its security and risks. Somebody sending an email or short message, does not think about someone else reading his message.
This thesis will cover possibilities of cryptography on mobile communication devices. For this purpose, a prototype for the Android platform capable of sending and receiving encrypted short messages will be demonstrated. The core components of this prototype will be outsourced in a library. The core components are cryptography, converting messages in an appropriate character set, compression - if needed, and an interface to send the message as short message or email.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Motivation . . . 1
1.2. Zielsetzung . . . 2
1.3. Aufbau der Arbeit . . . 2

2. Grundlagen **4**
2.1. Open Handset Alliance . . . 4
2.2. Android Plattform . . . 5
2.2.1. Android OS . . . 5
2.2.1.1. Linux Kernel . . . 5
2.2.1.2. Android-Laufzeitumgebung . . . 6
2.2.1.3. Standardbibliotheken . . . 7
2.2.1.4. Der Anwendungsrahmen . . . 7
2.2.1.5. Anwendungsschicht . . . 8
2.2.2. Android Anwendungen . . . 8
2.2.2.1. Activity . . . 9
2.2.2.2. Service . . . 9
2.2.2.3. Content Provider . . . 10
2.2.2.4. Broadcast Receiver . . . 10
2.2.2.5. Zusammenspiel der Komponenten . . . 10
2.3. Ausgewählte Kommunikationsformen . . . 11
2.3.1. SMS . . . 11
2.3.2. E-Mail . . . 12
2.3.3. Gemeinsamkeiten . . . 13
2.4. Ausgewählte Angriffstechniken . . . 13
2.4.1. IMSI-Catcher . . . 13
2.4.2. Sniffing . . . 14
2.4.3. Diebstahl von Endgeräten . . . 16
2.4.4. Schadsoftware . . . 17
2.5. Kryptographie . . . 18
2.5.1. Definition . . . 18
2.5.2. Verfahren . . . 18
2.5.2.1. Symmetrische Verfahren . . . 18

2.5.2.2. Asymmetrische Verfahren 19
2.5.2.3. Zusammenfassung 21
2.5.2.4. AES 22
2.5.2.5. RSA 24

3. Analyse **25**
3.1. Verfügbare Systeme 25
3.2. Zielgruppe 26
3.3. Bedrohungsanalyse 28
3.4. Anforderungen 31
3.4.1. Funktionale Anforderungen 32
3.4.2. Nicht funktionale Anforderungen 34

4. Architektur und Entwurf **38**
4.1. Architektur der Anwendung 38
4.1.1. Sender 39
4.1.2. Empfänger 41
4.1.3. Zusammenspiel Sender und Empfänger 42
4.1.4. A/T Komponenten 43
4.1.5. GUI Entwurf 43
4.2. Realisierung der Anwendung 46
4.2.1. Prototyp: SMS Verschlüsselung mit AES 47
4.2.2. Bewertung des Prototypen 51

5. Bibliothek **53**
5.1. Idee einer Bibliothek 53
5.2. Realisierung der Bibliothek 54
5.2.1. Prototyp der Bibliothek 54
5.3. Bewertung 59

6. Test **61**
6.1. Testkonzept und Testdaten 61
6.2. Exemplarische Testumsetzung für die Bibliothek 63

7. Fazit und Ausblick **65**
7.1. Zusammenfassung 65
7.2. Vision und Ausblick 66

A. Anwendungsfälle **68**

B. Codeschnipsel aus der Bibliothek **70**

C. DVD **72**

Abbildungsverzeichnis **73**

Literaturverzeichnis **75**

1. Einleitung

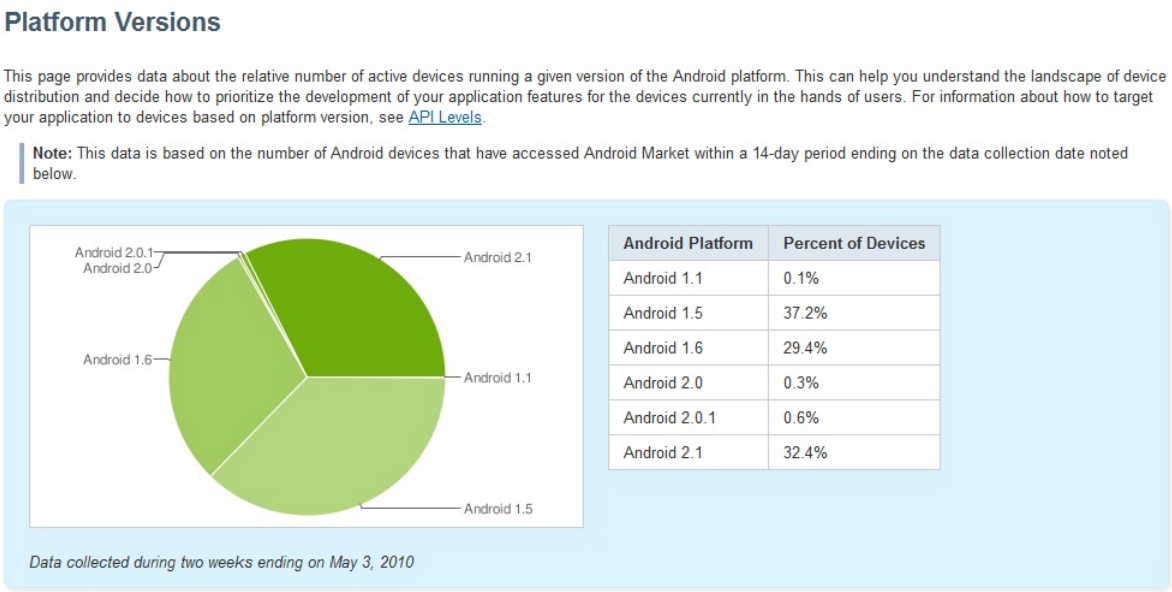
1.1. Motivation

Mobile Kommunikation ist heute allgegenwärtig, egal ob schriftlich über SMS-Nachrichten und E-Mail oder mündliche Kommunikation über das Handynetz beziehungsweise Voice-Over-IP. Über die Sicherheit macht sich dabei der private Nutzer kaum Gedanken. Anders sieht es jedoch in der Industrie, der Politik oder dem Militär aus. Hier ist es oftmals sehr wichtig, dass die Kommunikation - egal in welcher Form - sicher ist. Das heißt, dass sie nicht abgehört oder mit gelesen werden kann. In der Politik oder dem Militär spielt Geld oftmals keine Rolle, so dass Geräte angeschafft werden können, die höchsten Sicherheitsanforderungen genügen und dementsprechend teuer sind. Schaut man sich aber ein mittelständisches Unternehmen an, so stehen dort nur begrenzte finanzielle Mittel zur Verfügung. Trotzdem möchte man ein gewisses Sicherheitsniveau, bezogen auf die firmeneigene IT, erreichen. Zu dieser IT Ausstattung können Arbeitsplatzrechner, eigene Server für E-Mail und Daten aber auch Mobiltelefone gehören. Für die PCs und Server gibt es unzählige Softwarelösungen im kostenpflichtigen, sowie kostenlosen Bereich. Bei den Mobilgeräten ist die Auswahl schon begrenzter. Es gibt hier sowohl Hardware- als auch Softwareanbieter. Als Beispiel für einen Hardwareanbieter ist hier das Kryptohandy „TopSec GSM“ der Firma Rohde & Schwarz (siehe [42]) zu nennen, welches unter anderem von der Bundesregierung eingesetzt wird. Bei den Softwareanbietern tritt oft das Problem auf, dass die Software nur für ein bestimmtes Telefon, beziehungsweise Softwareplattform erhältlich ist. Die Software „GLK CryptoGSM“ (siehe [35]) beispielsweise läuft nur auf Telefonen der Firma Nokia und dann auch nur auf einigen Geräten, wie dem Modell 6630 oder dem N90.

Mit der Android Plattform ist nun ein OpenSource Betriebssystem für Mobilgeräte auf dem Markt, welche es erlaubt eine für diese Plattform standardisierte Softwarelösung zu schaffen, die auf einer Vielzahl von Geräten lauffähig ist, wenn ein Android OS installiert ist. Allerdings bleibt die oben genannte Einschränkung, dass Software nur für bestimmte Plattformen erhältlich ist, bestehen. Eine Android Anwendung ist nicht auf einem anderen Betriebssystem lauffähig. Die Android Plattform bietet den Vorteil, dass man auf bereits etablierte kryptographische Verfahren, wie AES und RSA zurückgreifen kann. Entweder werden die Java eigenen Klassen, wie JCE (Java Cryptography Extension) oder externe Klassen, wie Bouncy Castle genutzt.

1.2. Zielsetzung

In dieser Arbeit soll gezeigt werden, was im Bereich der Verschlüsselung auf mobilen Plattformen möglich ist. Exemplarisch soll dies hier anhand der Android Plattform und den Kommunikationsformen SMS und E-Mail gezeigt werden. Zu diesem Zweck soll ein Prototyp erstellt werden, der es möglich macht, SMS Nachrichten zu verschlüsseln und zu senden, sowie diese zu empfangen und entschlüsseln. Des weiteren soll der Kernbestandteil der Anwendung in eine Bibliothek ausgelagert werden. Zu den Kernbestandteilen gehören die Ver- und Entschlüsselung, die Konvertierung in einen für die Nachricht entsprechenden Zeichensatz, gegebenenfalls Kompression der Nachricht und die Schnittstelle zum Absenden der Nachrichten. Dazu soll das Android SDK (SDK: Software Development Kit) und damit auch das Java SDK verwendet werden. Aufgrund dessen, dass Android 1.6 auf rund einem drittel (29,4%) aller Android Geräte läuft (siehe 1.1) und auch mein Entwicklergerät (T-Mobile G1 alias HTC Dream) mit Android 1.6 ausgestattet ist, wird die Anwendung auch für diese Version entwickelt.



Platform Versions

This page provides data about the relative number of active devices running a given version of the Android platform. This can help you understand the landscape of device distribution and decide how to prioritize the development of your application features for the devices currently in the hands of users. For information about how to target your application to devices based on platform version, see API Levels.

Note: This data is based on the number of Android devices that have accessed Android Market within a 14-day period ending on the data collection date noted below.

Android Platform	Percent of Devices
Android 1.1	0.1%
Android 1.5	37.2%
Android 1.6	29.4%
Android 2.0	0.3%
Android 2.0.1	0.6%
Android 2.1	32.4%

Data collected during two weeks ending on May 3, 2010

Abbildung 1.1.: Plattform Versionen

1.3. Aufbau der Arbeit

Die Arbeit beginnt in Kapitel 2 mit einigen Grundlagen. Hier soll kurz erläutert werden, was das Android Betriebssystem ist und aus welchen Komponenten eine Android Anwendung aufgebaut ist. Des weiteren soll kurz erklärt werden, was eine E-Mail und eine SMS Nachricht ist und welche Gemeinsamkeiten sie haben. Zum Schluss dieses Kapitels soll noch auf

kryptographische Verfahren eingegangen werden. Der Kryptographieabschnitt beginnt mit einer allgemeinen Einleitung zu symmetrischen und asymmetrischen Verfahren und endet mit einer etwas genaueren Erklärung der beiden in der Anwendung verwendeten Verfahren AES und RSA.

In Kapitel 3 wird es eine kurze Analyse von bereits vorhandenen Software Lösungen geben und eine Anforderungsanalyse an die eigene Software. Die Anwendungsfälle dazu finden sich im Anhang A wieder.

In Kapitel 4 wird es um die eigentliche Anwendung gehen. Im ersten Abschnitt steht die Architektur und der Entwurf des Prototypen Vordergrund. Im zweiten Abschnitt soll es um die umsetzung des Prototypen gehen. Er beschränkt sich auf die Verschlüselung von SMS-Nachrichten mittels des AES-Algorithmus.

Kapitel 5 wird die aus dem Prototypen zu entwickelnde Bibliothek beschrieben und exemplarisch Umgesetzt.

Das sechste Kapitel (6) behandelt das Testkonzept für den Prototypen und die Bibliothek, sowie die Testumsetzung in Form von Unittests der Bibliothek.

Im siebten und letzten Kapitel (7) soll ein Fazit über die Anwendung und die Bibliothek gezogen werden. Dabei sollen auch die Probleme, die sich im Laufe der Arbeit ergeben haben beleuchtet werden. Zum Schluss wird es noch einen kleinen Ausblick geben, der aufzeigen soll, was vielleicht noch machbar wäre, beziehungsweise in welche Richtung die Anwendung/Bibliothek weiter entwickelt werden könnte.

2. Grundlagen

In diesem Kapitel sollen die Grundlagen, die für diese Arbeit notwendig sind, kurz erläutert werden. Dazu gehören die Beschreibung des Android Betriebssystem und der Aufbau einer Android Anwendung. Anschließend folgt eine kurze Beschreibung schriftlicher Kommunikationsformen, wie zum Beispiel SMS-Nachrichten, und es werden Grundlagen der Kryptographie anhand zweier Algorithmen - AES und RSA - erläutert.

2.1. Open Handset Alliance

Die Android Plattform wurde zunächst von der gleichnamigen Firma Android entwickelt, welche 2003 gegründet und 2005 von Google gekauft wurde. Im November 2007 wurde unter der Führung der Google Inc. die Open Handset Alliance gegründet. Sie ist ein Konsortium diverser Firmen aus dem IT-Bereich. Zu den Firmen gehören Softwareunternehmen (Google, ebay), Hardware- beziehungsweise Gerätehersteller (Intel Corporation, HTC), Netzbetreiber (T-Mobile, China Mobile) und auch Marketingfirmen (Wind River Systems, Teleca). Das Betriebssystem Android ist dabei das Hauptprodukt dieses Konsortium. Anfang 2010 wurde die Version 2.1 veröffentlicht. Android wird unter der GPLv2 ([18]) und der Apache 2.0 Lizenz ([19]) entwickelt.

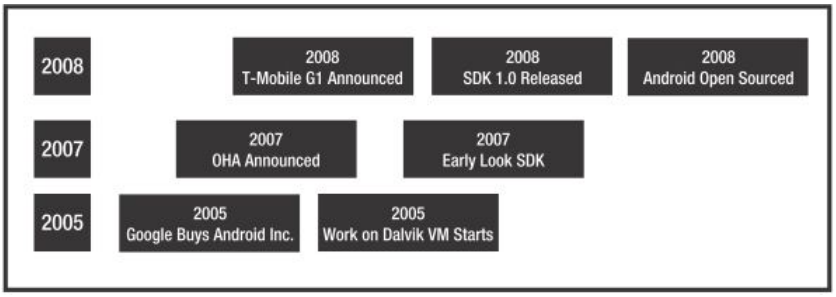


Abbildung 2.1.: Zeitlinie (Quelle: [43])

2.2. Android Plattform

In diesem Kapitel soll zunächst das Android Betriebssystem und anschließend die Komponenten einer Android Anwendung erläutert werden. Die Quellen für die Unterkapitel 2.2.1 und 2.2.2 sind im Wesentlichen folgende: [21, 7, 43]

2.2.1. Android OS

In diesem Abschnitt soll das Android Betriebssystem erläutert werden. Auf der nachfolgenden Grafik (2.2) ist das Schichtenmodell des Android Betriebssystem abgebildet. Die einzelnen Schichten werden anschließend kurz erläutert. ([21, 7, 43])

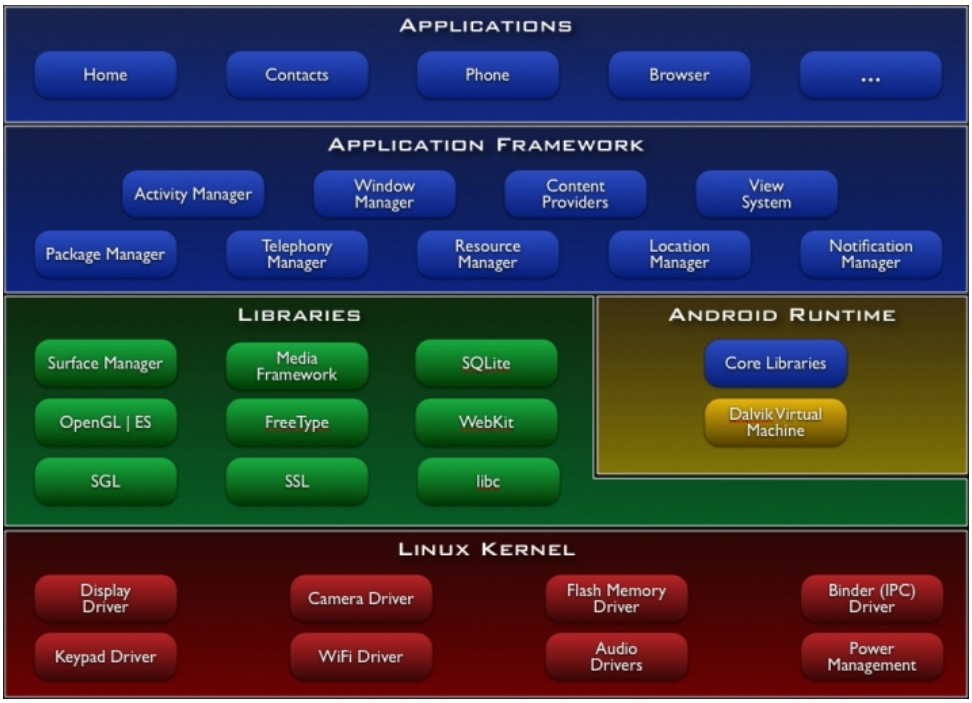


Abbildung 2.2.: Android Betriebssystem (Architektur) (Quelle: [21])

2.2.1.1. Linux Kernel

Das Android OS basiert auf einem Linux-Kernel. Aktuell (April 2010) wird die Version 2.6 eingesetzt. Er verwaltet den Speicher, sämtliche Prozesse und die Netzwerkkommunikation. Des Weiteren bildet der Kernel eine Hardwareabstraktionsschicht für die Software und stellt die Gerätetreiber für das System zur Verfügung.

2.2.1.2. Android-Laufzeitumgebung

Die Laufzeitumgebung stellt neben den Laufzeitbibliotheken auch die Dalvik Virtual Machine (DVM; [10]) zur Verfügung. Die DVM wurde von einem Google-Mitarbeiter namens Dan Bornstein entwickelt. Sie bildet das Herzstück der Laufzeitumgebung. Basierend auf der JVM Apache Harmony (Open-Source-Version der Java Virtual Machine) wurde sie dahingehend optimiert, dass multiple Instanzen der DVM auf Geräten mit wenig Hauptspeicher lauffähig sind. Jeder Android Anwendung wird eine eigene Instanz der DVM zur Verfügung gestellt. Dieses Verfahren kostet zwar einiges an Ressourcen ,zum Beispiel mehr Arbeitspeicher, bietet aber auch einige Vorteile. Zum einen müssen sich die Anwendungen keinen gemeinsamen Speicherbereich teilen; Stürzt eine Anwendung, aus welchen Gründen auch immer, ab stirbt nur ihre eigene VM und lässt alle anderen unbeeinflusst weiter arbeiten. Auch die Daten auf dem Gerät, zum Besipiel das Telefonbuch, sind besser geschützt, da keine Anwendung ohne Erlaubnis auf diese Zugreifen darf (wie Zugriffe erlaubt werden wird im weiteren Verlauf der Arbeit noch erläutert.). Des Weiteren kann keine Anwendung auf den Speicherbereich einer anderen Anwendung zugreifen. Dieses wird durch die DVM verhindert. Man bekommt dadurch eine erhöhte Sicherheit und auch eine bessere Verfügbarkeit.

Zu beachten ist, dass die JVM (Java Virtual Machine) nicht gleich der DVM ist. Android Anwendungen werden zwar in Java programmiert und auch zur Entwicklungszeit in Java-Bytecode kompiliert, aber auf der DVM läuft ein eigenständiger Bytecode, der so genannte dex-Code (siehe Abbildung 2.3). Mit dem dx-Tool, welches in dem Android Development Kit enthalten ist, werden Java Class Dateien in DVM kompatiblen dx-Code übersetzt. Doch warum wird dieser Aufwand betrieben? Das hat zum einen lizenztechnische Gründe, zum anderen aber, dass eine „normale“ JVM die Architektur eines mobilen Gerätes nicht vollständig ausnutzt. Bei der Entwicklung von J2ME (Java Platform 2, Micro Edition) hat Sun sich hauptsächlich darum gekümmert, dass Java auf mobilen Geräten lauffähig wird. Es wurden also viele Dinge aus den Java Bibliotheken einfach gestrichen und Java somit abgespeckt. Bei der DVM ist man einen anderen Weg gegangen. Man hat sich zu Nutze gemacht, dass in vielen Geräten Mikroprozessoren der Firma Arm Ltd. zum Einsatz kommen. Auf diese registerbasierten RISC-Prozessoren stützt man sich. Die DVM ist, im Gegensatz zur JVM, in der Lage auf diese Register zuzugreifen und somit die volle Leistung der Prozessoren auszunutzen. ([7, 43, 10])

Die DVM wird ausdrücklich nicht als Java-VM bezeichnet, da der Name Java von Sun geschützt ist. Da sie, wie eben erläutert, keinen Java-Bytecode verarbeitet fällt sie nicht unter eine Sun Lizenz. Man schreibt seine Anwendung wie gewohnt in Java, erzeugt seinen dx-Code und packt die dx-Dateien in eine apk-Datei (Android Package) zusammen. Diese apk-Datei ist das Gegenstück zum jar-File (Java ARchive). Über den Android Marktplatz oder über USB, beziehungsweise Internetdownload lädt man die Datei auf sein Gerät und führt sie aus.

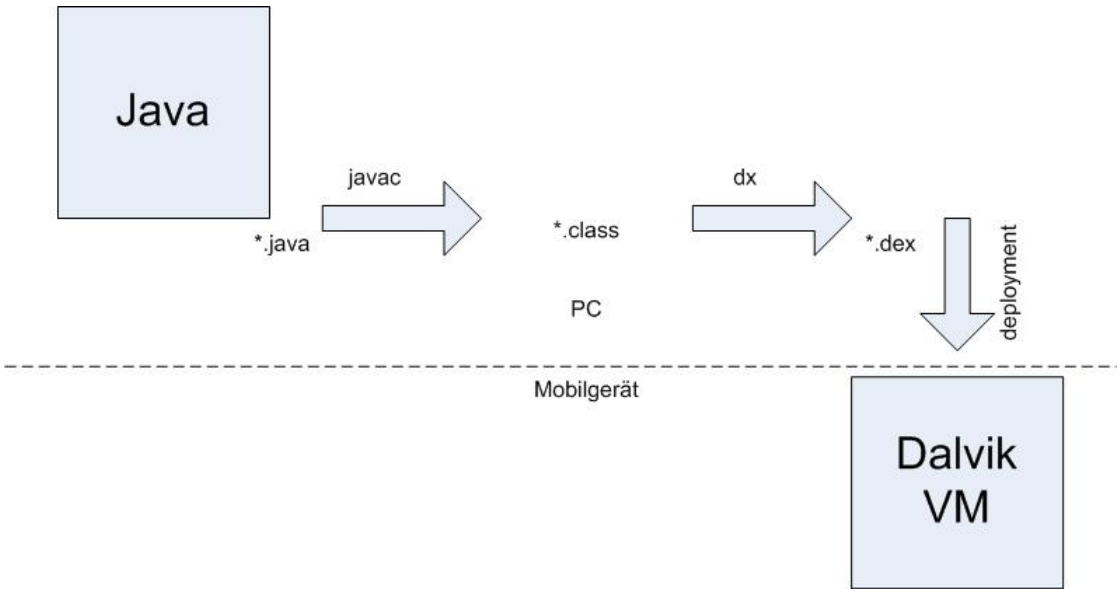


Abbildung 2.3.: Von *.java nach *.dex (Quelle: [7])

2.2.1.3. Standardbibliotheken

Über die Bibliotheken greifen der Anwendungsrahmen und die Anwendungsschicht auf bereits vorhandene Kernfunktionalitäten zu. Diese liegen als C/C++ Bibliotheken vor. Dazu gehören Funktionalitäten wie Datenbanken (SQLite), 3D-Grafikbibliotheken (SGL (2D) und OpenGL 1.0 (3D)), Multimedia-Verwaltungen, Webzugriff (LibWebCore) und einige mehr. Als Entwickler kann man für seine Anwendung auf diese Ressourcen zurückgreifen. ([7, 21])

2.2.1.4. Der Anwendungsrahmen

Der Anwendungsrahmen bildet den Unterbau für Android Anwendungen. Er enthält die androidspezifischen Klassen und abstrahiert die darunter liegende Hardware. Zu den Klassen gehören unter anderem diejenigen, die für die Oberflächengestaltung zuständig sind. Android liefert hier eigene Klassen, da die aus der Java-Welt stammenden Klassen Swing und SWT nicht zur Verfügung stehen. Desweiteren sind dort diverse Manager-Klassen angeordnet, wie zum Beispiel der Activity Manager, der unter anderem den Lebenszyklus von Prozessen verwaltet, oder der Notification Manager, der es möglich macht, dass eine Mitteilung in der Notification Bar (Abbildung 2.4) eingeblendet wird, wenn eine neue SMS eingetroffen ist. ([7])

Abbildung 2.4.: Android 1.6 Homescreen

2.2.1.5. Anwendungsschicht

Auf dieser Schicht befinden sich die Android Anwendungen. Das können Anwendungen sein, die von Google oder einem Gerätehersteller stammen oder zum Lieferumfang gehören. Zum Beispiel Google Maps oder Anwendungen, die den Kontakt zu „sozialen Netzwerken“ möglich machen, aber auch Eigenentwicklungen beziehungsweise Programme von anderen Drittanbietern. Auf dieser Schicht findet die Interaktion von Benutzer und Maschine/Gerät statt, aber auch Kommunikation zwischen verschiedenen Anwendungen. ([7])

2.2.2. Android Anwendungen

Eine Android Anwendung besteht aus mehreren Komponenten mit ihren jeweiligen Aufgabengebieten. Es sind im Wesentlichen vier verschiedene Komponenten: Activity, Service, Content Provider und Broadcast Receiver. Neben diesen Komponenten gibt es noch andere wichtige Klassen wie Intents und das Android Manifest.

Mit Intents (Absichtserklärungen) kann zwischen den Komponenten kommuniziert werden. Es können Nachrichten und Daten ausgetauscht werden, aber auch andere Activitys gestar-

tet werden. Dabei gibt es drei verschiedene Arten von Intents. Es gibt Explizite und Implizite Intents, sowie Broadcast Intents; zu letzteren später mehr. Bei einem Expliziten Intent ist dem Programmierer die Empfängerkomponente bereits bekannt. Sie wird direkt im Quellcode mit ihrem Namen (Empfaenger.class) angegeben. Sollte dem Entwickler hier ein Fehler unterlaufen, tritt dieser erst zur Laufzeit auf. Bei dem Impliziten Intent ist die Empfängerkomponente nicht direkt bekannt. Man schickt den Intent los, in der Hoffnung, dass sich jemand angesprochen fühlt und die Aufgabe ausführt. Meistens werden diese genutzt um Komponenten anderer Anwendungen anzusprechen, wie zum Beispiel einen Mailclient.

Damit das Ganze System funktioniert und die Anwendung weiß, wann sie auf welche Nachrichten reagieren soll bedarf es einer Deklaration im so genannten Android Manifest. In dieser XML-Datei werden die Berechtigungen und Intent-Filter gesetzt. Berechtigungen bedeuten hier, auf welche Systemressourcen, wie zum Beispiel das Telefonbuch oder die Berechtigung SMS-Nachrichten zu verschicken, die Anwendung zugreifen darf. Intent-Filter sezten fest, was auf welche Nachricht hin zu tun ist, beziehungsweise welche Komponente gestartet werden soll.

2.2.2.1. Activity

Eine Activity ist eine für den Nutzer sichtbare Komponente. Sie verwaltet die GUI, stellt sie dar und reagiert auf Eingaben, beziehungsweise Befehle des Nutzers und zeigt ihm auch die entsprechenden Ergebnisse an. Eine Android Anwendung kann mehrere Activitys haben. Sie wird entweder in Java programmiert, was allerdings sehr unkomfortabel ist, oder über eine XML-Datei definiert, welche beim Start eingelesen wird. Die XML-Datei ist wie das Android Manifest eine Ressource. Sie wird vom Android Ressourcencompiler aapt (Android Asset Packaging Tool) eingelesen. Das aapt macht dann aus dieser Definition Objekte oder Objektbäume, welche dann in Bytecode übersetzt und in die Anwendung integriert werden. Gleichzeitig erzeugt das aapt die „R-Klasse“. Diese dient als Schlüsselwert-Speicher über den die Zugriffe auf die Ressourcen vollzogen werden.

2.2.2.2. Service

Manche Programmteile brauchen keine grafische Oberfläche. Das sind zumeist Hintergrundprozesse, die über einen definierten Zeitraum aktiv sind. Ein gutes Beispiel ist das Abspielen von Musik oder eine aufwändige Berechnung, die aber den Benutzer bei seiner Arbeit nicht unterbrechen oder stören soll.

2.2.2.3. Content Provider

Content Provider stellen, wie der Name schon sagt, Inhalte bereit. Wenn eine Anwendung Daten laden oder speichern möchte, geschieht dies über diese Komponente. Ein Content Provider stellt hierzu die Verbindung zum Beispiel zu einer zugrunde liegenden Datenbank her. Dabei ist es egal, ob diese auf dem Gerät vorliegt ,als interne SQLite Datenbank, oder über das Internet erreichbar ist, zum Beispiel eine MySQL Datenbank. Über Berechtigungen kann er diese Daten beliebig vielen Anwendungen zur Verfügung stellen. Der Zugriff geschieht über Intents oder Content Resolver. Ein Content Resolver dient zur Implementierung von synchronen und asynchronen Zugriffe auf Content Provider.

2.2.2.4. Broadcast Receiver

Broadcast Receiver lauschen auf Intents, die auf Systemebene verschickt werden, so genannte Broadcast Intents. Diese Intents informieren Anwendungen über diverse Dinge, wie zum Beispiel, dass der Akku Ladestand niedrig ist, dass eine SMS eingetroffen ist oder keine Verbindung zum Netz mehr besteht. Broadcast Receiver fangen diese Nachrichten ein und verarbeiten sie in der Art und Weise, wie es der Programmierer wünscht. Das kann durch Aufrufen einer Activity geschehen, um eine Nachricht anzuzeigen oder irgendein Hintergrundprozess wird ausgeführt, der vielleicht nur eine kleine Notification setzt. Notifications sind eine Art von Systemnachrichten, mit denen man den Nutzer auf etwas hinweisen möchte. Diese tauchen in der Notification Bar (siehe Abbildung 2.4) auf. Wenn man auf sie klickt wird (meistens) eine Anwendung gestartet.

2.2.2.5. Zusammenspiel der Komponenten

In den Vorangegangenen Unterkapiteln wurden die einzelnen Komponenten, die eine Androidanwendung enthalten kann, vorgestellt. In den einzelnen Vorstellungen wurde auch kurz erwähnt, wie diese Komponenten zusammenarbeiten, die hier noch einmal zusammengefasst werden sollen. Es ist möglich über Methodenaufrufe und deren Rückgabewerte Daten beziehungsweise Werte auszutauschen. Es können Intents genutzt werden um andere Activities zu starten und ihnen Daten mitzugeben. Intents werden aber auch von Broadcastreceivern genutzt um erhaltene Nachrichten weiterzugeben. Aus der Java Welt ist dann noch IPC (Inter Prozess Communication) bekannt. Android beherrscht auch dieses und hält sich dabei an den IDL Standard (Interface Definition Language). Mit der letzten Methode ist es möglich über Prozessgrenzen hinweg zu kommunizieren.

2.3. Ausgewählte Kommunikationsformen

In Kapitel 2.3.1 und 2.3.2 sollen die beiden im Prototypen und der Bibliothekt verwendeten Kommunikationsformen SMS und E-Mail erläutert werden.

2.3.1. SMS

Umgangssprachlich wird mit SMS die Nachricht selbst gemeint. Die Abkürzung SMS steht allerdings nicht für die Nachricht, sondern für den Dienst: Short Message Service. Wenn in dieser Arbeit von der Nachricht die Rede ist, wird sie als SMS-Nachricht bezeichnet. Die erste Version des SMS Standards wurde 1989 verabschiedet. Die erste SMS-Nachricht wurde im Dezember 1992 von einem PC an ein Mobiltelefon versendet. Entwickelt wurde der SMS Standard hauptsächlich von Friedhelm Hillebrand (Deutsche Bundespost) und Bernard Ghillebaert (France Télécom). Ursprünglich waren diese Nachrichten nur ein Nebenprodukt des GSM Netzes, sie sollten dazu dienen, Netzstörungen an die Nutzer zu melden.

Aufbau:

Eine SMS Nachricht darf maximal 160 Zeichen lang sein, wenn eine 7-Bit Kodierung genutzt wird, und maximal 70 Zeichen lang sein, wenn 16-Bit Unicode genutzt wird. Diese Beschränkung ergibt sich aus der maximalen Nutzdatenlänge des MAP (Mobile Application Part), welches Teil des Signalisierungssystems Nummer 7 ist ([54]). Neben diesen beiden Varianten, um Texte zu übertragen, gibt es noch eine 8-Bit Kodierung für „Datennachrichten“, zum Beispiel Bildmitteilungen oder Klingeltöne.

Eine SMS Nachricht besteht aus folgenden Teilen: Header und Body.

Header:

Der Header einer SMS enthält unter anderem folgende Daten (siehe [23]): (Nummern Typ - Service Center Nummer, PDU Typ, Nummern Typ - Empfänger Nummer, Kodierung)

Den Nummerntyp und die Nummer des Service Centers, welches für die Übertragung der Nachricht zuständig ist. Diese (Telefon-)Nummer ist beim Kunden immer dem entsprechenden Provider gemäß voreingestellt, kann aber oft von Hand geändert werden. Der PDU Typ (Protocol Data Unit) gibt an, ob Optionen wie Empfangsbestätigung aktiv sein sollen oder nicht. Dann wird noch die Empfängernummer und ihr Typ (national oder international) angegeben und welche Zeichenkodierung im SMS-Body angewendet wird.

Body:

Bei der 7-Bit Kodierung handelt es sich um einen für das GSM Netz entwickelten Zeichensatz, dem GSM 03.38 Zeichensatz (siehe [31]). Damit sind 7 Bit/Zeichen × 160 Zeichen =

1.120 Bit möglich. Es lassen sich nur „lateinische Buchstaben“ abbilden mit einem Zeichenvorrat von 128 Zeichen. Es gibt jedoch Möglichkeiten den Zeichensatz durch verschiedene Mechanismen zu erweitern (siehe [4]). Bei der 16-Bit Kodierung wird auf ein auf BMP beschränktes UTF-16, einer Unicode Kodierung, zurückgegriffen. BMP bedeutet „Basic Multilingual Plane“ und enthält hauptsächlich Schriftsysteme, die aktuell in Gebrauch sind. Unicode Nachrichten werden für alle nicht „lateinischen“ Alphabete benötigt. ([1])

2.3.2. E-Mail

Die E-Mail ist, wie die SMS Nachricht, kein von Anfang an geplantes Produkt. Vielmehr etablierte sie sich durch das Benutzerverhalten im ARPA Net. Als Erfinder der E-Mail gilt Ray Tomlinson. In den 60ern gab es Mailboxsysteme mit denen man über das CPYNET Protokoll mit Programmen wie SNDMSG mit Benutzern von Großrechnern kommunizieren konnte. 1971 führte Tomlinson einige Änderungen an dem Protokoll durch, unter anderem wurde auch das @-Symbol eingeführt, und passte auch SNDMSG an. Nun konnte man Nachrichten an Nutzer durch das Netz schicken mittels des Nutzernamens und dem Hostnamen (nutzer@host.de).

In den 80er Jahren entwickelten sich neben dem (ARPA)Internet weitere Netzwerkssysteme, wie BTX (Bildschirmtext), Mailbox Systeme oder X.25 (Protokollfamilie großräumige Computernetze über das Telefonnetz). Aus dem Jahr 1982 stammt das RFC 822, in dem beschrieben wurde wie eine „ARPA Textmessage“ aus zusehenhat. Die aktuelle Fassung ist das RFC 5322 aus dem Jahr 2008. Mitte der 90er wurden die meisten der anderen Systeme von der uns heute bekannten E-Mail verdrängt. Die erste Mail in Deutschland traf am 3. August 1984 um 10:14 MEZ an der Uni Karlsruhe ein.

Eine E-Mail besteht aus folgenden Hauptteilen (gemäß RFC5322 [38]): Dem Header und dem Body.

Header:

Der Header einer E-Mail hat folgende Inhalte, die nach einem festen Muster aufgebaut sind. Es kommt zuerst der Feldname, gefolgt von dessen Inhalt und einem CRLF (carriage return & line feed). Der Feldinhalt muss in US-ASCII angegeben sein. Angegeben werden der Absender, das Datum und die Uhrzeit wann die E-Mail abgeschickt wurde, der Empfänger und ein automatisch erzeugte ID. Darüber hinaus können noch weitere Informationen hinzugefügt werden. Meistens wird der Weg, den eine Mail durch das Internet zurücklegt, im Header protokoliert. Weiter können noch eine Antwortadresse und CC beziehungsweise BCC Empfänger definiert werden.

Body:

Der Body enthält die eigentliche Nachricht. Er wird mit einer Leerzeile vom Header getrennt. Der Inhalt darf nur aus 7-Bit-ASCII Zeichen bestehen. Werden andere Zeichen, wie zum Beispiel deutsche Umlaute, verwendet, muss die E-Mail vor dem Versenden entsprechend umkodiert werden (Base64). Ebenso müssen Dateianhänge entsprechend behandelt werden. Das führt dazu, dass der Body etwas größer wird als die eigentliche Nachricht.

2.3.3. Gemeinsamkeiten

SMS Nachricht und E-Mail haben eine wichtige Gemeinsamkeit. Bei beiden wird der Body, der jeweils die Nachricht enthält, im Klartext übertragen [12]. Jeder Knotenpunkt, den die Nachricht passiert, kann den Inhalt mitlesen oder gar manipulieren, ohne das der Empfänger etwas davon bemerkt.

2.4. Ausgewählte Angriffstechniken

In Kapitel 2.3 wurde gezeigt, dass E-Mail und SMS Nachrichten im Klartext übertragen werden. Es wurde auch darauf hingewiesen, dass diese Nachrichten an diversen Stellen mitgelesen werden können. In diesem Abschnitt soll kurz angerissen werden, anhand einiger ausgewählter Beispiele, mit welchen Methoden und Techniken dieses möglich ist. Diese Auflistung hat jedoch nicht den Anspruch der Vollständigkeit, da es noch diverse andere Möglichkeiten gibt. Exemplarisch sollen hier vier Techniken eines Angriffes erläutert werden.

2.4.1. IMSI-Catcher

Ein IMSI-Catcher ist ein Gerät mit dem es möglich ist, ausgehende Telefonate und SMS Nachrichten von GSM Mobiltelefonen abzuhören. Eines der ersten Geräte wurde von der schon genannten Firma Rohde & Schwarz hergestellt. Mittlerweile gibt es mehrere Hersteller weltweit.

Der IMSI-Catcher simuliert eine reguläre GSM Zelle. Bei einer maximale Leistung von 25 Watt kann damit in ländlichen Gegenden ein Radius von bis zu 30 km um den IMSI-Catcher herum abgedeckt werden. Man stellt das Gerät so ein, dass es in der aktuellen Umgebung das stärkste Signal aussendet. Somit melden sich alle Mobiltelefone im Sendebereich des IMSI-Catcher bei diesem an. An dieser Stelle wird eine Schwachstelle des GSM Netzes ausgenutzt. Es wird bei GSM nur eine einseitige Authentifikation vorgenommen. Das heißt, dass sich nur der Nutzer gegenüber dem Netz beweisen muss, nicht aber das Netz gegenüber dem Nutzer. Ist die gefälschte Funkzelle einmal eingerichtet fängt sie alle Geräte im

Sendebereich ein und protokolliert IMSI und IMEI Nummer mit. Diese Nummern sind eindeutige Identifikationsnummer. Die IMSI (International Mobile Subscriber Identity) ist auf der SIM-Karte (Subscriber Identity Module) gespeichert und identifiziert den Nutzer, die IMEI (International Mobile Station Equipment Identity) ist eine 15 stellige Seriennummer, die jedes GSM- oder UMTS-Endgerät eindeutig identifiziert. Beide Nummer sind, wie ihre Namen aussagen, international eindeutig. Des weiteren können alle abgehenden Gespräche und Textnachrichten (zum Beispiel SMS Nachrichten) mitgehört beziehungsweise mitgelesen werden. Eingehende Gespräche oder Nachrichten können nicht abgefangen werden, da dem GSM Netz die falsche Zelle nicht bekannt ist. (siehe [30, 55, 20, 48, 12])

Seit dem 14. August 2002 ist der Einsatz von IMSI-Catchern in Deutschland den Strafverfolgungsbehörden erlaubt [48].

Abbildung 2.5.: IMSI-Catcher (Quelle: [20])

2.4.2. Sniffing

Mit einem Sniffing Programm ist es möglich den Datenverkehr eines Netzwerkes mitzuschneiden und zu analysieren. Sniffer werden in der Regel dazu genutzt, um Probleme im eigenen Netzwerk zu finden oder nach verdächtigen Inhalten zu suchen. Diese Programme lassen sich allerdings auch zur Spionage verwenden. So ist es möglich unverschlüsselte Verbindungen abzuhören und entsprechende Inhalte, wie Benutzernamen und Passwörter, herauszufiltern. Diese Programme können sowohl innerhalb eines Netzwerkes eingesetzt werden, zum Beispiel ein PC im Firmennetzwerk, als auch an Knotenpunkten, an denen viele Informationen weitergeleitet werden, zum Beispiel Gateway-Rechner.

Ein solches Programm ist zum Beispiel Wireshark. Die mit Wireshark gewonnenen Daten werden in Paketform dargestellt (TCP Dump) und auf Wunsch gefiltert. An einem kleinen Beispiel soll gezeigt werden, wie unverschlüsselter E-Mailverkehr mit diesem Programm belauscht werden kann.

Testweise wird eine E-Mail von der Adresse „mrlstrike@gmx.de“ an „schuer_m@informatik.haw-hamburg.de“ geschickt[1].

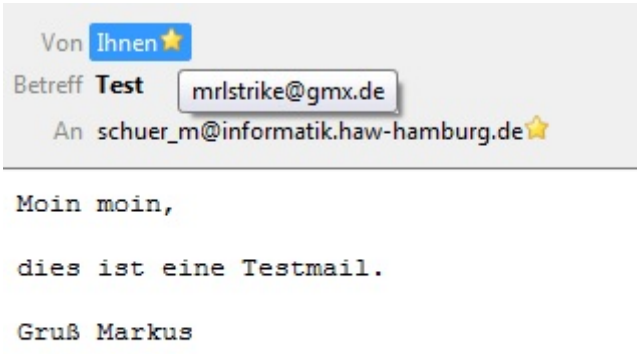


Abbildung 2.6.: E-Mail

Die Nachricht wird unverschlüsselt versendet. Das betrifft sowohl die Verbindung zum E-Mail Server als auch die Nachricht selbst. Damit die Daten schneller gefunden werden können wird noch ein Filter bei Wireshark auf Port 25 (Simple Mail Transfer Protocol (SMTP)) eingestellt. Nun lässt sich die verschickte E-Mail recht schnell finden.

[1] Beide Adressen gehören dem Autor dieser Arbeit.

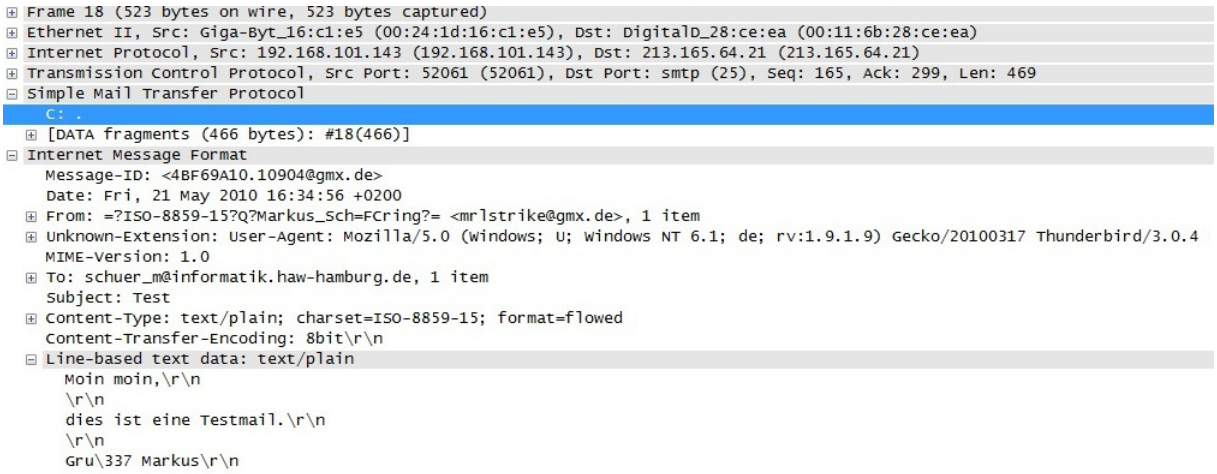


Abbildung 2.7.: Wireshark Protokoll

In der Abbildung 2.7 lässt sich sehr schön die in Kapitel 2.3.2 beschriebene Struktur einer E-Mail erkennen. Von oben nach unten: Message-ID, Datum, Absender, Empfänger, die Betreffzeile und der Body mit der eigentlichen Nachricht im Klartext. Zusätzlich erfährt man hier noch, dass die Nachricht mit dem E-Mail Client Mozilla Thunderbird in der Version 3.0.4 verschickt wurde und, dass Windows in der Version NT 6.1 auf dem Computer läuft. Mit Hilfe dieser zusätzlichen Informationen lassen sich weitere Angriffe auf den betroffenen Nutzer vorbereiten.

2.4.3. Diebstahl von Endgeräten

Eine weitere Gefahrenquelle für sensible Informationen ist der Diebstahl von Endgeräten. Dies gilt sowohl für mobile Computer (Notebooks oder Netbooks) als auch für Mobiltelefone. Für Computer gibt es einige Möglichkeiten seine Daten vor unberechtigten Zugriffen zu schützen. Als Beispiel kann man die gesamte Festplatte verschlüsseln. Entweder mit einer Funktion des Betriebssystems oder einer Zusatzsoftware wie TrueCrypt. Für Mobiltelefone gibt es ähnliche Möglichkeiten. So bietet Windows Mobile zum Beispiel an die SD-Karte des Gerätes zu verschlüsseln. Damit sind aber Informationen, die auf dem internen Speicher liegen, noch nicht verschlüsselt. Hier muss der Nutzer wieder auf Zusatzsoftware zurückgreifen, sofern es ein Angebot für sein Gerät gibt. Für die meisten „klassischen“ Mobiltelfone ist dies nicht der Fall. Bei aktuellen Smartphones ist das Angebot deutlich größer. Ein weiteres Problem ist, dass die viel Mobiltelefone aus Bequemlichkeit nicht mit einer PIN oder einem Passwort gesperrt sind, wenn sie im eingeschalteten Zustand in der Tasche oder auf dem Schreibtisch liegen. Somit hat jeder, der Zugriff zu dem Gerät hat, die Möglichkeit auf alle Daten zuzugreifen. Das ist auch dann möglich, wenn eine Software installiert wur-

de, die den internen Speicher verschlüsselt, da diese Art von Software sehr oft an die PIN Eingabe gekoppelt ist. Aber auch bei gesperrten Telefonen ist man vor Angriffen nicht unbedingt sicher. Zwar verweigern die meisten Geräte im gesperrten Zustand einen Kontakt mit einem fremden Computer aber auch hier gibt es Mittel und Wege die Geräte auszutricksen. Jüngstes Beispiel ist das iPhone von Apple. Hier gelang es das iPhone trotz Sperrung auszulesen. Dazu wurde das iPhone während des Bootvorganges mit einem dem Gerät unbekannten PC mittels USB verbunden worauf das iPhone sehr oft die Verbindung zuließ und man ein vollständiges Backup erstellen konnte. In diesem Backup konnten dann Notizen, SMS-Nachrichten und sogar Passwörter im Klartext gefunden werden. ([25]) Dieses Verfahren funktioniert allerdings nur, wenn das iPhone im ungesperrten Zustand heruntergefahren wurde.

2.4.4. Schadsoftware

Bei aktuellen Smartphones ist es oft üblich die Grundfunktionen mit weiteren Programmen, Apps genannt, zu erweitern. So kann es passieren, dass man sich ohne es zu wissen Schadsoftware auf dem Gerät installiert. Diese Software kann dann nach Belieben Daten auf dem Gerät auslesen und über das Internet verschicken. Bei der Installation von Android Applikationen wird der Nutzer darauf hingewiesen auf welche Daten und Funktionen eine Anwendung zugreifen möchte (siehe Abbildung 2.8). Damit ist allerdings noch nicht ausgeschlossen, dass die Anwendung heimlich Dinge macht, die der Nutzer nicht wahrnimmt. So könnte das in Abbildung 2.8 gezeigte Programm eines sein, welches Daten auf der SD-Karte speichert (zum Beispiel ein Browser) aber gleichzeitig die SD-Karte ausliest und die Daten über das Internet verschickt.

Abbildung 2.8.: Android Installationshinweise

2.5. Kryptographie

In den vorangegangenen Kapiteln wurden E-Mail und SMS-Nachrichten, sowie mögliche Angriffstechniken eingeführt. In diesem Kapitel sollen nun die Grundlagen der Verschlüsselung gelegt werden. Zunächst werden symmetrische und asymmetrische Verfahren allgemein erläutert, anschließend wird je ein Vertreter aus diesen Klassen beschrieben.

2.5.1. Definition

Kryptographie ist eine Wissenschaft, die sich mit Methoden und Verfahren der Ver- und Entschlüsselung von Daten beschäftigt. Sie ist ein Teilgebiet der Computersicherheit. Kryptographische Methoden werden nicht nur zur Ver- und Entschlüsselung von Daten, sondern auch zum Signieren, Authentifizieren und Sichern von Übertragungskanälen genutzt [44, 12].

2.5.2. Verfahren

Bei den kryptographischen Verschlüsselungsmethoden unterscheidet man zwischen symmetrischen und asymmetrischen Verfahren, welche nachfolgend erläutert werden.

2.5.2.1. Symmetrische Verfahren

Bei symmetrischen Verfahren wird zur Ver- und Entschlüsselung der gleiche Schlüssel verwendet. Vorteil eines solchen Verfahrens ist seine Geschwindigkeit, die es zum Beispiel möglich macht große Datenmengen in relativ kurzer Zeit zu verarbeiten. Zur Anwendung kommen dabei Stromchiffren, bei denen jedes Zeichen des Klartextes einzeln verschlüsselt (oder entschlüsselt) wird oder Blockchiffren, bei denen die Daten in einer definierten Blockgröße verschlüsselt (oder entschlüsselt) werden. Ein großer Nachteil der symmetrischen Verfahren ist der Schlüssel, welcher allen Beteiligten bekannt sein muss. Der Schlüssel muss vor Anwendung des Verfahrens über einen sicheren Kanal oder persönlich ausgetauscht werden. Wird der Schlüssel kompromittiert sind alle bisher ausgetauschten Nachrichten für Unberechtigte lesbar. Mathematisch funktioniert ein symmetrischer Verschlüsselungsalgorithmus wie folgt [44]:

Verschlüsselung: $e(m,k) = c$

und Entschlüsselung: $d(c,k) = m$,

wobei gilt: e = Verschlüsselungsfunktion (encrypt); d = Entschlüsselungsfunktion (decrypt); m = Klartext (message); c = Chiffretext (chiffre); k = Schlüssel (key).

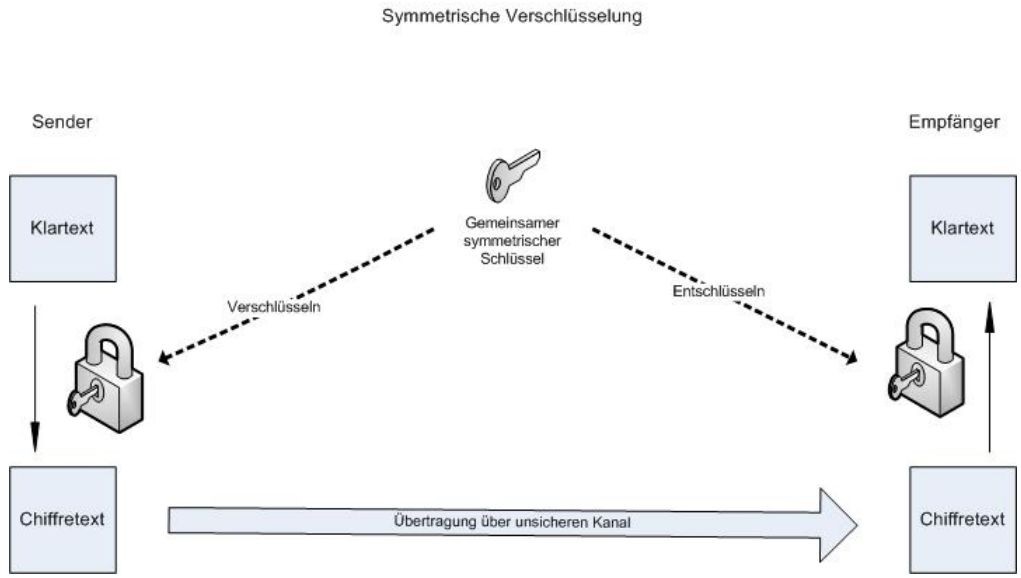


Abbildung 2.9.: Symmetrische Verschlüsselung

Eines der ältesten symmetrischen Verfahren ist die Caesar-Chiffre, eine einfache „Verschiebechiffre“, beziehungsweise Substitutionschiffre. Bei der Caesar-Chiffre wird dem Klartextalphabet ein Chiffrealphabet gegenüber gestellt. Beim Verschlüsseln werden entsprechend des Chiffrealphabetes die Buchstaben im Klartext ersetzt (siehe Abbildung 2.10).

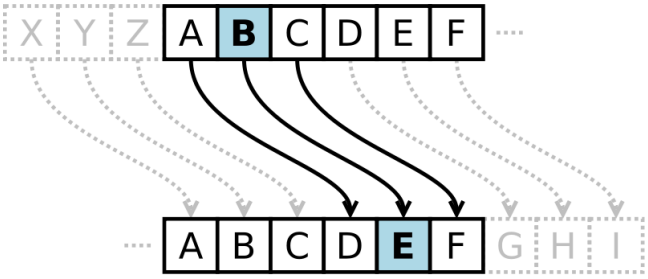


Abbildung 2.10.: Caesar-Chiffre (Quelle: [57])

Weitere Beispiele für symmetrische Verfahren sind: (Triple-) DES, AES, Kasumi, RC2, RC5 und RC6.

2.5.2.2. Asymmetrische Verfahren

Bei dieser Art der Kryptographie besitzt jeder Anwender ein eigenes Schlüsselpaar. Einen privaten, geheimen Schlüssel (private Key) und einen öffentlichen Schlüssel (public Key).

Der öffentliche Schlüssel kann jedem frei zur Verfügung gestellt werden. Sei es auf einem „Keyserver“ oder als Textfile im Netz. Mit diesem Schlüssel kann jeder dem Inhaber des Schlüssels eine verschlüsselte Nachricht oder Daten zukommen lassen. Nur mit dem privaten Schlüssel lässt sich diese Nachricht, beziehungsweise Daten, entschlüsseln. Man kann sich also sicher sein, dass nur der Empfänger die Nachricht lesen kann. Für das Ver- und Entschlüsseln gilt (nach [44]):

$c = e(a,m)$und $m = d(x,c)$, wobei

a der öffentliche Schlüssel und x der private Schlüssel ist; e die Verschlüsselungsfunktion, d die Entschlüsselungsfunktion, m der Klartext und c der Chiffretext ist.

Das asymmetrische Verfahren ermöglich auch das Signieren, sprich digitale Unterschreiben von Nachrichten oder Daten. Dazu verschlüsselt man die Nachricht mit dem privaten Schlüssel und hängt diesen Datensatz an die eigentliche Nachricht mit dran. Mit dem öffentlichen Schlüssel kann nun verifiziert werden, ob die Nachricht von dem besagten Nutzer stammt und ob die Nachricht eventuell verändert worden ist. Eine elektronische Signatur ist wie folgt definiert (siehe [12]):

$S = E(M,K_D)$; Die Signatur S entsteht durch Verschlüsseln der Nachricht M mit dem privaten Schlüssel K_D.

$D(S,K_E)$; Die Signatur S wird nun auf der Empfängerseite mit dem öffentlichen Schlüssel $K_E$entschlüsselt.

Nun wird das übermittelte Dokument M mit $D(S,K_E)$ verglichen. Wenn S korrekt ist gilt: $M = D(S,K_E)$.

Beispiele für asymmetrische Verfahren sind: RSA, Rabin, Elgamal.

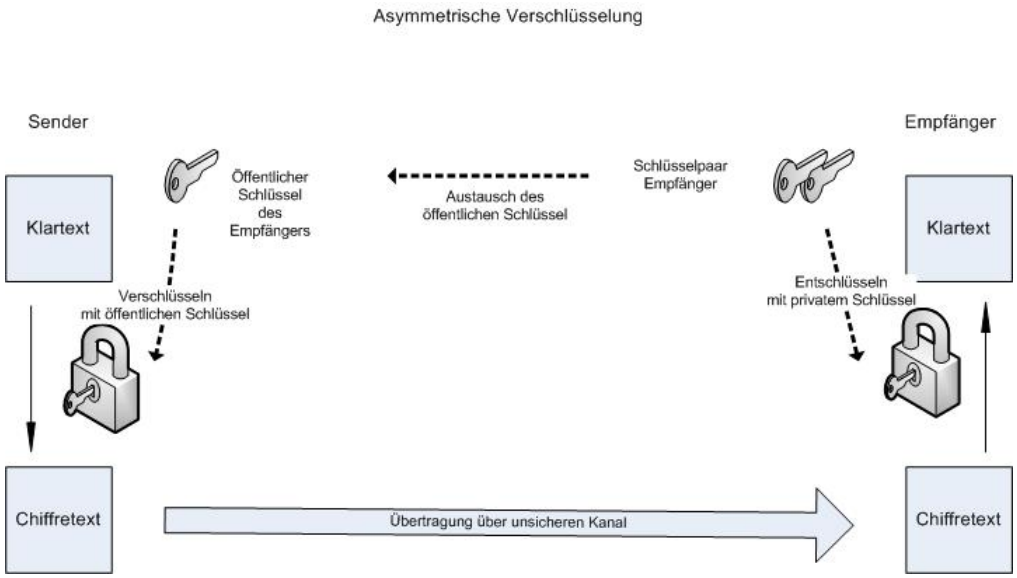


Abbildung 2.11.: Asymmetrische Verschlüsselung

2.5.2.3. Zusammenfassung

Die beiden eben beschriebenen Verfahren sind die wichtigsten Verfahren der modernen Kryptographie [44]. Jedoch stehen sie nicht in Konkurrenz zueinander, wie man vermuten könnte, sondern ergänzen sich durch ihre Unterschiede recht gut. Den wichtigsten Unterschied bildet die mathematische Grundlage. Symmetrische Verfahren beruhen auf relativ einfachen mathematischen Verfahren und sind daher leichter zu implementieren und sind auch in ihrer Anwendung sehr schnell. Asymmetrische Verfahren beruhen auf komplexen mathematischen Strukturen. Sie sind daher in der Implemetierung recht aufwändig und fehleranfällig. Durch die Komplexität ist auch deren Anwendung im Gegensatz zu den symmetrischen Verfahren recht langsam. Als Beispiel ist der RSA Algorithmus etwa um den Faktor 1000 langsamer als AES.

Daher kombiniert man in der Praxis beide Verfahren. Man nutzt einen asymmetrischen Algorithmus um den Schlüssel auszutauschen, den man später für das symmetrische Verfahren, mit dem man die eigentliche Daten verschlüsselt, nutzen möchte.

Nachfolgend werden die Algorithmen AES und RSA erläutet. AES wird Bestandteil des Prototypen und der Bibliothek sein, der RSA Algorithmus wird nur in der Bibliothek zum Einsatz kommen.

2.5.2.4. AES

Der AES-Algorithmus (Advanced Encryption Standard) wurde im Oktober 2000 als Nachfolger des DES-Algorithmus (Data Encryption Standard) von der NIST (National Institute of Standards and Technology) als Standard eingeführt. Entwickelt wurde AES von Joan Daemen und Vincent Rijmen, daher wird er auch Rijndael-Algorithmus genannt.

Beim AES handelt es sich um eine Blockchiffre, genauer um eine SP-Chiffre, dass heißt es werden abwechselnd Substitutions- und Permutationsschritte durchgeführt. Die Anzahl der Runden hängt dabei von der Schlüssellänge ab. Bei 128 Bit sind es 10 Runden. Der AES-Algorithmus funktioniert grob wie folgt:

Zunächst wird der Klartext in Blöcke aufgeteilt. Im Gegensatz zum (reinen) Rijndael nutzt der AES eine feste Blocklänge von 128 Bit. Jeder dieser Blöcke wird in einer 4x4 Matrix abgespeichert, wobei jeder Eintrag 1 Byte groß ist. Als nächstes werden folgende Transformationen angewendet (X steht für die Anzahl der Runden, die entsprechend der Schlüssellänge nötig sind) (siehe auch: [12, 44, 28]):

1. Der erste Rundenschlüssel wird mittels XOR mit dem Block verknüpft
2. Es werden X-1 Runden mit folgenden Transformationen durchgeführt:
   a) Substitution (SubBytes): Jedes Byte aus dem Block wird durch einen Eintrag aus der S-Box ersetzt. Die S-Box ist eine Substitutionstabelle.
   b) Permutation (ShiftRow): Durch eine Linksrotation werden alle Zeilen vermischt. Zeile eins bleibt wie sie ist, Zeile zwei wird um eine Stelle nach links verschoben, die Zeile drei um zwei Stellen und die vierte Zeile um drei Stellen.

a0	b0	c0	d0		a0	b0	c0	d0
a1	b1	c1	d1	==>	b1	c1	d1	a1
a2	b2	c2	d2		c2	d2	a2	b2
a3	b3	c3	d3		d3	a3	b3	c3

   c) Diffusion (MixColumn): Dieser Schritt sorgt wie ShiftRow für eine Durchmischung der Blöcke. Bei MixColumn werden die Spalten durchmischt. Durch eine Multiplikation jeder Spalte mit einem festgelegten Polynom wird dafür gesorgt, dass jedes Byte mit jedem anderen Byte der Spalte in „Wechselwirkung“ tritt. Wenn b ein Byte ist, dann ist die Multiplikation (●) wie folgt definiert (siehe [12]):
   2 ● b := Linksshift von b, falls b<128
   := (Linksshift von b) $\bigoplus$ 00011011, falls b>=128
   3 ● b := (2 ● b) $\bigoplus$ b
   Die genaue mathematische Definition des ShiftRow findet sich in [28] auf Seite 12 wieder.

d) Schlüsselverknüpfung (AdRoundKey): In dieser Phase wird ein Rundenschlüssel (Subkey) mittels bitweise XOR zur 4x4 Matrix addiert.

3. In der letzten Runde wird 2c ausgelassen.

Für das Ver- und Entschlüsseln sind X so genannte Rundenschlüssel zu erzeugen. Die Rundenschlüssel werden durch die Schlüsselexpansion gewonnen. Jeder Subschlüssel hat 16 Bytes und hat die selbe Form wie die 4x4 Matrix. Anhand der Abbildung 2.12 kann man sich deutlich machen, wie die einzelnen Subschlüssel erzeugt werden (hier wird von einem 128 Bit Schlüssel ausgegangen).

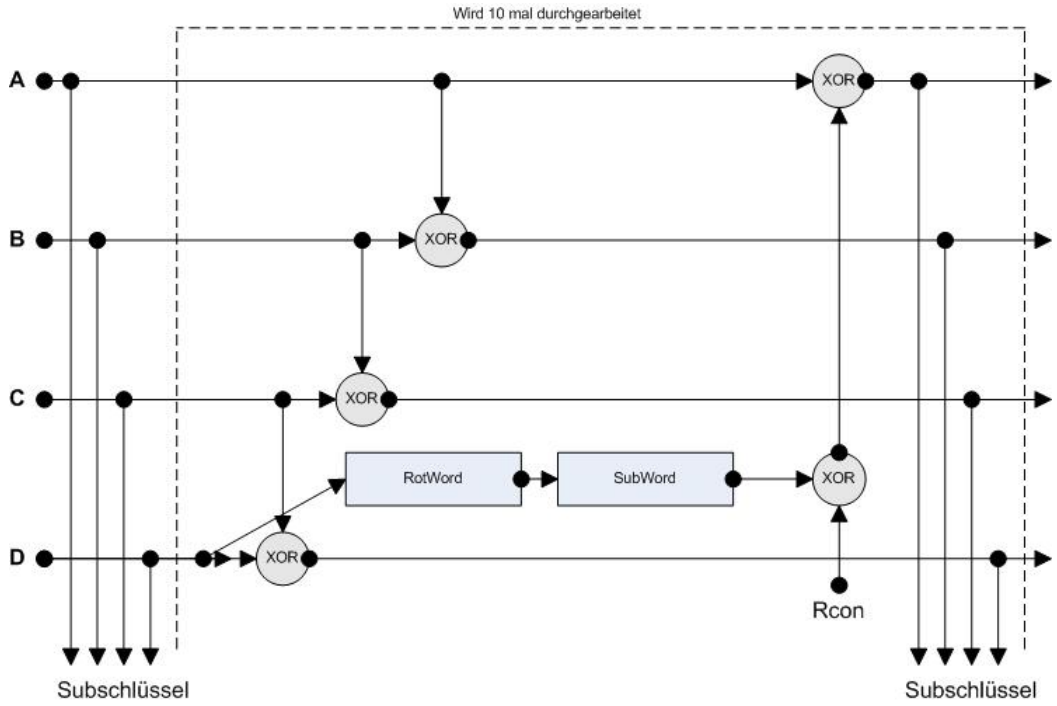


Abbildung 2.12.: Schlüsselexpansion

Die 16 Bytes werden auf die variablen A,B,C und D verteilt. Der erste Subschlüssel ist gleich dem AES Schlüssel. Alle weiteren Schlüssel werden wie in dem Bild beschrieben verarbeitet, wobei neben der XOR Verknüpfung noch zwei Funktionen und eine Konstante (Rcon) eine Rolle spielen. RotWord führt eine byteweise Linksrotation durch und SubWord ersetzt jedes der vier Bytes durch einen entsprechenden Eintrag in der S-Box. Es handelt sich hierbei um die gleiche S-Box wie bei SubBytes. ([44])

2.5.2.5. RSA

Ronald L. Rivest, Adi Shamir und Leonard M. Adleman entwickelten 1977 das RSA Verfahren. Es handelt sich hierbei um ein asymmetrisches Verfahren, welches auf dem Faktorisierungsproblem (Einweg-Funktion) beruht. Im Gegensatz zum Diffie-Hellman Verfahren kann man mit RSA nicht nur Schlüssel austauschen, sondern direkt Daten verschlüsseln und/oder signieren. Der RSA-Algorithmus ist wie folgt aufgebaut (siehe [12, 40]):

1. Wähle zwei große Primzahlen p und q und berechne $n = pq$. Der Wert n wird auch als RSA-Modul bezeichnet.
2. Wähle $d \in [0,n-1]$, so dass gilt: d ist relativ prim zu $\varphi(n) = (p-1) * (q-1)$, dass heißt $ggT(\varphi(n),d) = 1$, wobei $\varphi(n)$ die Euler´sche Funktion bezeichnet und ggT den größten gemeinsamen Teiler meint.
3. Wähle $e \in [0,n-1]$ mit $e * d = 1 \, mod \, \varphi(n)$. Das heißt e ist das multiplikative Inverse modulo $\varphi(n)$ zu d.
4. (e,n) ist der öffentliche Schlüssel (public Key)
5. (d,n) ist der geheime Schlüssel (private Key)
6. Verschlüsseln: $M \in [0,n-1] : E(M) = M^e \, mod \, n$.
7. Entschlüsseln: $D(C) = C^d \, mod \, n$.

Damit das Verschlüsseln und Signieren in der Praxis schnell ist, wird für den öffentlichen Schlüssel (e,n) ein kleiner Wert gewählt. Für Operationen, die den öffentlichen Schlüssel benutzen werden $O(k^2)$ Schritte benötigt und für Operationen mit dem privaten Schlüssel $O(k^3)$ Schritte benötigt, wobei k die Anzahl der Bits im binär repräsentierten Modul n entspricht (siehe [12]). Um die Sicherheit des Verfahrens gewährleisten zu können, müssen die Werte p, q und $\varphi(n)$ geheim gehalten werden.

3. Analyse

In diesem Kapitel soll zunächst gezeigt werden, welche Softwarelösungen es schon im Bereich der Verschlüsselung von schriftlichen Nachrichten gibt. Die Tabelle 3.1 gibt eine kurze Übersicht welche Programme erhältlich sind und was für Eigenschaften sie haben. Anschließend folgt eine Beschreibung der anvisierten Zielgruppe einer solchen Software und zum Schluss werden die Anforderungen an die Anwendung aufgeführt.

3.1. Verfügbare Systeme

Mittlerweile befinden sich schon einige Programme im Angebot für Android-Geräte (gefunden über [6]). Die meisten beschränken sich auf symmetrische Verfahren wie DES oder AES. Über die GUI werden Nachricht und Passwort eingegeben und dann in den Chiffretext überführt. Die meisten setzten dabei auf eine HEX-Codierung. Der Chiffretext wird dabei in einen String aus HEX-Symbolend übersetzt.

Desweiteren sind viele Programme reine Verschlüsselungsprogramme ohne Funktionen, die das Senden von Nachrichten erlauben. Sie nutzen einfach die zugrunde liegenden Android Komponenten. Das können entweder die Standardprogramme zum Senden von Nachrichten sein oder Anwendungen von Drittanbietern. Das heißt für diese Programme auch, dass

Spezifikation	SMS	E-Mail	symm.	asymm.	Telefonbuch zugriff	direkt senden und empfangen
Aois CryptText	$(\surd)$	$(\surd)$	$\surd$	—	—	—
messySMS	$\surd$	—	$\surd$	—	—	$(\surd)$ nur senden möglich
TXTcrypt	$(\surd)$	$(\surd)$	$(\surd)$	—	—	—
TextSecure 0.3	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$
BA Schüring	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$(\surd)$

Tabelle 3.1.: Verfügbare Anwendungen für Android

eine eingehende verschlüsselte Nachricht zunächst im regulären Posteingang landet. Anschließend muss die Nachricht von Hand in das Programm kopiert werden oder von dem Programm entsprechend ausgelesen werden.

Während an dieser Arbeit geschrieben wurde hat die Firma WHISPER SYSTEMS das Programm TextSecure in der Verison 0.3 vorgestellt. Im Gegensatz zu allen anderen Programmen setzt TextSecure auf das OTR-Protokoll (Off-the-Record) und Elliptische-Kurven-Kryptographie (ECC: Elliptic Curve Cryptosystems) (siehe [24, 51]). In den FAQs von WHISPER SYSTEMS heißt es, dass die verschlüsselte Nachricht komprimiert wird um den begrenzten Platz einer SMS-Nachricht nutzen zu können, Zitat:

> „TextSecure aims to adapt the principles of OTR (forward-security and deniability) to the SMS environment by compressing the message format to fit in the limited space provided by an SMS message.“ [51]

Zusammenfassend kann man sagen, dass die meisten Programme, welche von privaten Programmierern erstellt wurden, mit den gleichen Problemen zu kämpfen haben. Das sind der SMS-Zeichensatz und die meist durch die gewählte Kodierungsart resultierende Länge der verschlüsselten Nachricht. WHISPER SYSTEMS scheint hier einen vielversprechenden Weg zu verfolgen.

SMS Verschlüsselung für andere Systeme

Windows Mobile: xSMS

Symbian: SafeTxT

3.2. Zielgruppe

Zu Anfang dieser Arbeit wurde schon etwas zum Thema der Zielgruppe gesagt. Eine Privatperson hat in der Regel kein Interesse daran seine Nachrichten zu verschlüsseln. Sei es, weil Sie nicht interessiert, dass jemand seine E-Mails mitliest oder weil ihm die Handhabe einer Software zu kompliziert ist. In den meisten Fällen wird er gar nicht wissen, dass seine Nachricht für jeden lesbar ist. In der Wirtschaft, Politik und dem Militär sieht das, wie bereits erwähnt, ganz anders aus.

Im Grundlagenkapitel wurde bereits deutlich gemacht, dass E-Mails und SMS Nachrichten im Klartext übermittelt werden. Das heißt, dass an jedem Knotenpunkt, an dem die Nachricht vorbeikommt, die Möglichkeit besteht diese zu lesen oder zu manipulieren, ohne dass der Empfänger davon etwas mitbekommt. Vergleicht man das mit dem normalen Postwesen, so ist eine E-Mail oder SMS Nachricht nichts anderes als eine Postkarte. Es kommen also alle

bereits erwähnten Personengruppen als Zielgruppe in Betracht, die eine mehr, die andere etwas weniger.

Folgende Szenarien sind denkbar:

Szenario 1: Privatpersonen

Wenn wir Grüße aus dem Urlaub schicken, so greifen wir gerne zur Bildpostkarte. Der Inhalt dieser Postkarte ist für jeden lesbar. In den meisten Fällen ist die dort enthaltene Information für einen Fremden recht uninteressant. Anders sieht es jedoch bei anderen postalischen Schreiben aus. Sei es ein Trauerbrief, eine Kündigung oder ein persönlicher Liebesbrief. Hier möchte sicherlich niemand, dass ein Fremder diese Mitteilungen liest. Sie sind eben vertraulich und meist sehr persönlich. Im Umgang mit E-Mail und SMS Nachrichten sind viele Menschen deutlich freizügiger. Ohne Nachdenken wird die Nachricht schnell eingetippt und abgeschickt. Dass diese Nachricht, die vielleicht ein Außenstehender nicht lesen soll, wie eine Postkarte ist, wissen viele nicht oder ignorieren es.

Auf SMS Nachrichten bezogen mag eine Verschlüsselung vielleicht überzogen sein, da meist nur unwichtige Botschaften übertragen werden. Bezogen auf den E-Mailverkehr sieht das ein wenig anders aus. Zumindest eine digitale Unterschrift sollte benutzt werden, wenn der Inhalt dieser Nachricht einwandfrei einer bestimmten Person zugeordnet werden soll oder gar muss.

Szenario 2: Wirtschaft

Industriespionage ist nicht erst seit heute ein großes Problem. Im digitalen Zeitalter hat sich dieses Problem noch verschärft. Viele Unternehmen sind schon Opfer so genannter Hacker-Attacken geworden. Bei diesen Angriffen wurden nicht selten sensible Dokumente gestohlen. Doch nicht nur von Außen drohen Gefahren. Auch im Unternehmen kann der „Feind“ sitzen. Nicht selten nehmen Mitarbeiter nach einer Kündigung sensible Daten mit nach Hause oder zu ihrem neuen Arbeitgeber. Die Gründe hierfür können sehr unterschiedlich sein. Daher macht es Sinn nicht nur über Sicherheitsmaßnahmen nachzudenken, die gegen Angriffe von Außen schützen, sonder auch über solche, die gegen Angriffe von innen schützen. Ein Baustein ist dabei die E-Mail- und Datenverschlüsselung. Nur berechtigte Personen dürfen somit die entsprechenden Daten und Mails sehen, beziehungsweise lesen.

Entsprechende Verschlüsselungssoftware auf dem Diensthandy oder PDA dürfen da nicht fehlen. Auch wenn sich der Mitarbeiter im Außendienst befindet, sollte er sicher Kommunizieren können. Interessant wird dies vor allem im Ausland. Wenn der dortige Provider unverschlüsselte Nachrichten mitprotokollieren sollte und findige Mitarbeiter diese zu Geld machen wollen ist der beste Schutz eine entsprechende Verschlüsselung. Hier könnte es auch interessant werden SMS Nachrichten, sofern sie vertrauliche Dinge beinhalten, wie zum Beispiel eine kurze Preisabsprache mit der Zentrale, zu verschlüsseln.

Szenario 3: Politik und Behörden

In der Politik sieht die Sachlage ähnlich der Wirtschaft aus. Auch hier ist es in der Vergangenheit zu „Internetattacken“ auf Regierungseinrichtungen gekommen. Auch hier gibt es schützenswerte Daten. Es werden E-Mails zwischen Behörden und Ministerien ausgetauscht, die zweifelsfrei zugeordnet werden müssen und gegebenenfalls vor den Blicken Unberechtigter geschützt werden müssen. In den meisten Behörden wird ein Mobilgerät wohl eher selten für dienstliche E-Mails genutzt werden. Anders jedoch bei Beamten oder Ministern, die sich sich regelmäßig im Ausland aufhalten. Hier kann es durchaus wichtig sein, dass jede Kommunikation, egal ob schriftlich oder mündlich, abhörsicher ist.

3.3. Bedrohungsanalyse

In der Bedrohungsanalyse werden die in Kapitel 2.4 genannten Angriffstechniken aufgegriffen und zueinander in ein Verhältnis gesetzt um die Bedrohung zu analysieren die von ihnen einzeln oder in Kombination ausgeht. Die Grafik 3.1 zeigt einen Bedrohungsbaum mit einer Eintrittswahrscheinlichkeit der einzelnen Bedrohungen, welche im einzelnen Erläutert werden. Die Datenlage bezogen auf mobile Geräte ist noch sehr dünn, daher beziehen sich die „Gefahrenwerte“ (gering, mittel, hoch) auch auf Daten, die in Bezug auf stationäre Computer erhoben wurden.

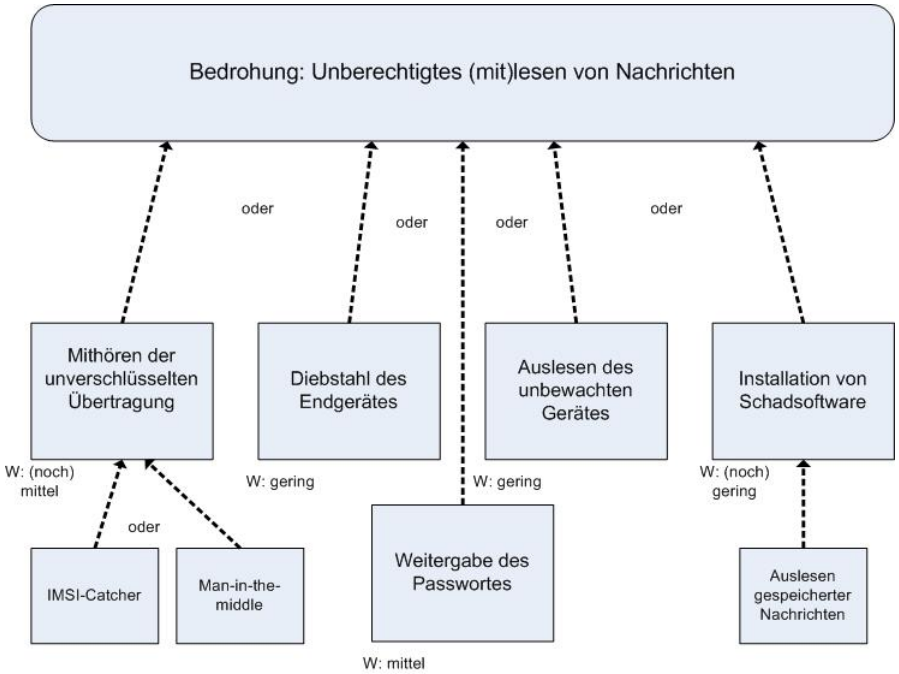


Abbildung 3.1.: Bedrohungsbaum

Ähnlich wie bei einem stationären Rechner entwickeln sich für mobile Endgeräte mehr und mehr Bedrohungen. Dazu gehören neben der klassischen Malware (Viren, Würmer und Trojaner) oder Sniffing-Angriffe auch „neue“ Bedrohungen. Zu den „neuen“ Bedrohungen gehört zum Beispiel der Diebstahl des Endgerätes.

Bei der Malware gibt es seit 2004 eine starke Zunahme, wie in der Grafik 3.3 zu sehen ist. Verstärkt wird dieser Trend durch die zunehmende Standardisierung der Betriebssysteme auf Mobiltelefonen, wie den Grafiken 3.2 entnommen werden kann (siehe auch [15, 37]). Momentan hat Symbian den größten Marktanteil.

Worldwide Smartphone Sales to End Users by Operating System in 1Q10 (Thousands of Units)

Company	1Q10 Units	1Q10 Market Share (%)	1Q09 Units	1Q09 Market Share (%)
Symbian	24,069.8	44.3	17,825.3	48.8
Research In Motion	10,552.6	19.4	7,533.6	20.6
iPhone OS	8,359.7	15.4	3,848.1	10.5
Android	5,214.7	9.6	575.3	1.6
Microsoft Windows Mobile	3,706.0	6.8	3,738.7	10.2
Linux	1,993.9	3.7	2,540.5	7.0
Other OSs	404.8	0.7	445.9	1.2
Total	**54,301.4**	**100.0**	**36,507.4**	**100.0**

Source: Gartner (May 2010)

Abbildung 3.2.: Marktanteile von Betriebssystemen aus Mobiltelefonen (Quelle: [8])

Somit ist es nicht verwunderlich, dass die bisher bekanntesten Viren und Würmer alle auf Geräten mit dem Symbian Betriebssystem aufgetreten sind. Als Beispiel ist hier der Wurm „CommWarrior“ zu nennen ([14]). Dieser Wurm verbreitete sich über MMS (Multimedia Messaging Service) und Bluetooth und führt an einem bestimmten Datum ein Geräte-Reset aus. Der erste Wurm, welcher sich über das iPhone verbreitete, wurde im November 2009 entdeckt. Man kann davon ausgehen, dass auch Android Geräte in Zukunft betroffen sein werden.

Mobile Malware Summary

Type	Year 2004	2005	2006	2007	2008	2009	2010	Total
Garbage			8					8
Riskware			1		1	8		10
Spyware			5	15	6		1	27
Trojan	11	105	160	23	13	24	32	368
Virus	14	19	17	6				56
Worm				2	8	6	22	38
HackTool							4	4
Backdoor							1	1
Total	**25**	**124**	**191**	**46**	**28**	**38**	**60**	**512**

Platform	Year 2004	2005	2006	2007	2008	2009	2010	Total
iPhone						2		2
J2ME			2		2	7	1	12
PocketPC	1		1	2	7	8	15	34
Symbian	24	124	188	44	19	21	44	464
Total	**25**	**124**	**191**	**46**	**28**	**38**	**60**	**512**

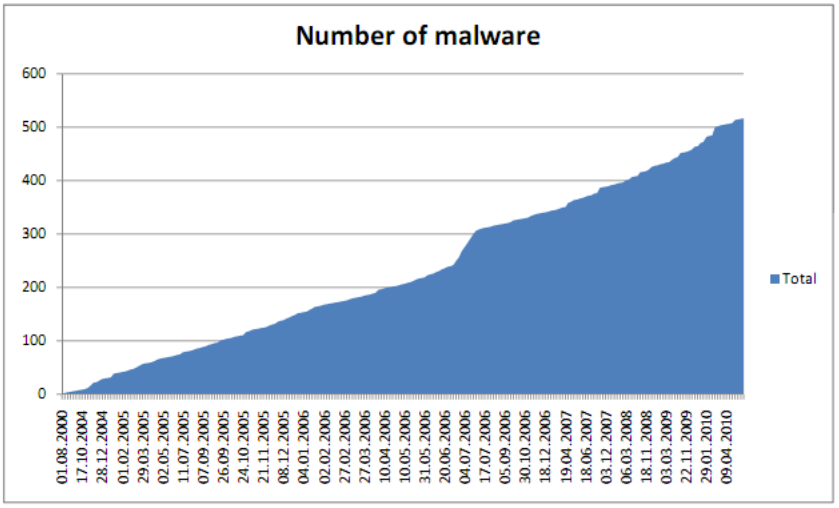


Abbildung 3.3.: Number of Mobile Malware (Quelle: [16])

Angesicht der steigenden Anzahl von Malware ist es nur eine Frage der Zeit, bis Programme auftauchen, die unbemerkt vom Nutzer Daten ausspähen und an unbefugte Dritte weitergeben. Dazu kommt noch das veränderte Nutzungsverhalten mit den aktuellen Geräten, das dem Verhalten an einem stationären Computer immer ähnlicher wird. Für die Bedrohungsanalyse ist die Gefahr durch Malware Schaden zu erleiden noch gering einzuschätzen.

Neben der Malware hat auch die Computerkriminalität stark zugenommen. Im Vergleich von 2008 auf 2009 um 17,7%. Interessant ist hier der Anteil der Fälle, die unter den Punkt Ausspähen und Abfangen von Daten fallen. Hier hat es einen Anstieg von 48,7% gegeben (siehe Grafik 3.4 und [11] auf Seite 7). Die Zahl von rund 11500 erfassten Fällen mag nicht alarmierend klingen, aber die Dunkelziffer dürfte um einiges höher liegen. Hierzu gibt es leider keine Angaben ([34, 36]).

II Kurzinformation „Polizeiliche Kriminalstatistik 2009“

Fall- und Tatverdächtigenentwicklung in Kürze

T1)*

Inhalt	Anzahl		Veränderung gg. Vorjahr		Aufklärungs-quote in %	
	2009	2008	absolut	in %	2009	2008
Straftaten insgesamt						
erfasste Fälle	6 054 330	6 114 128	-59 798	-1,0		
aufgeklärte Fälle	3 368 879	3 353 473	15 406	0,5	55,6	54,8
Die insgesamt positive Fallentwicklung der vergangenen Jahre setzt sich deutlich fort.						
Kontoeröffnungs- und Überweisungsbetrug	20 915	16 039	4 876	30,4	69,2	69,2
Wirtschaftskriminalität	101 340	84 550	16 790	19,9	91,7	92,5
Computerkriminalität	74 911	63 642	11 269	17,7	37,5	40,3
darunter:						
Ausspähen, Abfangen von Daten	11 491	7 727	3 764	48,7	22,4	29,0
Veruntreuungen	33 744	32 379	1 365	4,2	97,7	98,1
Insolvenzstraftaten nach StGB	5 152	5 129	23	0,4	99,7	99,9

Abbildung 3.4.: Ausschnitt aus der PKS 2009 (Quelle: [11])

Die meisten Zahlen und Statistiken beziehen sich auf „klassische Computer“. Darunter fallen auch Notebooks, so dass es auf diese mobilen Geräte bezogen keine seperaten Fakten gibt. Ebenso lassen sich kaum Angaben über andere mobile Geräte, wie Mobiltelefone oder Organizer finden. Daher ist die Bedrohung noch als mittel einzuschätzen.

Eine weitere Bedrohung geht von den Nutzern selbst aus. Laut einer Studie ([33]) im Auftrag des Bitkom geben 37% aller Deutschen (über 14 Jahre) ihre privaten Passwörter an Dritte weiter, am Arbeitsplatz tut dies zudem etwa jeder Dritte auch. Ist das Passwort erst einmal weitergegeben, hat der eigentliche Besitzer keine Kontrolle mehr über dieses. Auch hier ist eine mittlere Bedrohung zu sehen.

Eine weitere Studie der Bitkom zeigt auf, dass mittlerweile jeder zweite Internetnutzer (ab 14 Jahre) von Computerkriminalität betroffen ist ([9]). Das heißt, dass nicht nur Firmen sich gegen Angriffe auf vertrauliche Daten schützen müssen, sondern auch der private Nutzer immer mehr in das Visir von Krimminellen rückt. Im Großen und Ganzen muss man die Bedrohungslage für Mobiltelefone schon als mittel einstufen. Virenscanner und sichere Kominikationskanäle (SSL, https) düften in Zukunft auf dem Mobiltelefon genauso wichtig sein wie auf dem normalen Computer.

3.4. Anforderungen

In diesem Kapitel wurde bereits gezeigt, welche Zielgruppe in Betracht gezogen wird. In diesem Abschnitt sollen die funktionalen und nicht funktionalen Anforderungen an die Anwendung aufgeführt werden. Sie sollen den mögliche Funktionsumfang der Anwendung beschreiben.

3.4.1. Funktionale Anforderungen

Die funktionalen Anforderungen sollen eine Beschreibung der Anwendung liefern. Meistens werden Anforderungen nur sehr allgemein formuliert. Bei den funktionalen Anforderungen soll aber ein möglichst genaues Bild der Anwendung gezeichnet werden. Das heißt, dass versucht werden sollte alle möglichen Ein- und Ausgaben zu beschreiben sowie die Zustände des System vorher und nachher. Auch sollen mögliche Fehlerfälle und ihre Behandlung betrachtet werden. ([46])

Eine Möglichkeit diese Anforderungen festzuhalten bieten Anwendungsfälle. Sie beschreiben die Anforderungen wie folgt:

Nr. n	Name des Anwendungsfalles
Vorbedingung	Die Vorbedingung, die erfüllt sein muss.
Beschreibung	Beschreibung des Ablaufes
Ausnahmen	Ausnahmen, Fehler und ihre Behandlung
Nachbedingung	In welchem Zustand befindet sich die Anwendung am Ende

Eine Ergänzung dazu stellen die Anwendungsfalldiagramme dar. Sie sollen eine kurze Übersicht über die vorhandenen Anwendungsfälle liefern.

Auf Seiten des Anwenders steht die Benutzbarkeit im Vordergrund. Die folgenden funktionalen Anforderungen spiegeln sich auch in den Anwendungsfällen wieder, welche sich im Anhang A befinden.

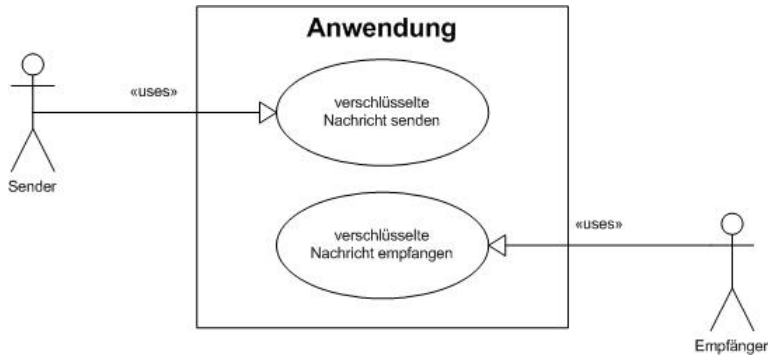


Abbildung 3.5.: Senden und Empfangen einer Nachricht

Beim Programmstart wird zunächst die Oberfläche für SMS Nachrichten geladen. Hier hat der Benutzer die Möglichkeit den Empfänger seiner Nachricht aus dem Telefonbuch seines Geräts zu wählen oder aber die Telefonnummer von Hand einzutragen. Für den E-Mail Versand wird ähnlich verfahren. Name und Telefonnummer/E-Mail Adresse des Empfängers sollten nun angezeigt werden. Ist ein Feld im Telefonbuch nicht gesetzt, so bleibt das entsprechende Feld in der Anwendung leer und es wird eine Fehlermeldung an den Benutzer

herausgegeben. Anschließend wird die eigentliche Nachricht in das dafür vorgesehen Feld eingegeben. Jetzt besteht noch die Möglichkeit das Passwort für die Verschlüsselung einzugeben. Alternativ könnte über ein Public-Privat-Key Verfahren nachgedacht werden. In diesem Fall würde der öffentliche Schlüssel des Empfängers aus dem entsprechenden Telefonbucheintrag gelesen werden und zur Verschlüsselung verwandt werden. Über den Button „Senden“ wird die Nachricht verschickt. Sollte hierbei, oder an anderer Stelle ein Fehler, auftreten soll dies dem Benutzer mitgeteilt werden. Die Bereits eingegebenen Daten sollten, wenn ein Fehler auftritt, erhalten bleiben.

Beim Empfang einer Nachricht soll dem Anwender dieses über ein kleines Bild in der Notification Bar angezeigt werden. Zusätzlich könnte ein Klingelton abgespielt werden oder man lässt das Mobiltelefon kurz vibrieren. Um die Nachricht zu lesen tippt der Anwender auf das Bild in der Notification Bar und startet somit die Anwendung. Es sollen auf dem Display die Telefonnummer, E-Mail Adresse oder der Name des Senders angezeigt werden, sowie die noch verschlüsselte Nachricht. Nach Eingabe des Passwortes und drücken des „Entschlüsseln“ Buttons soll die entschlüsselte Nachricht angezeigt werden. Bei einer asymmetrischen Verschlüsselung würde der Schritt mit der Passworteingabe entfallen. Im Fehlerfall sollten auch hier die bereits vorhandenen Daten erhalten bleiben.

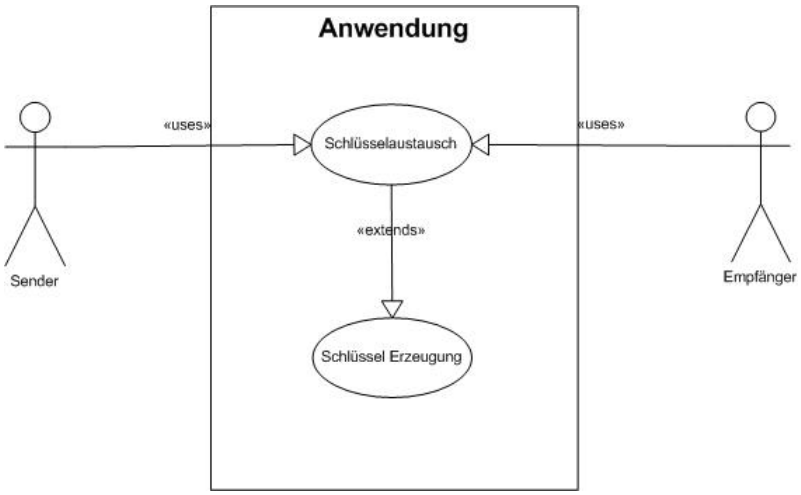


Abbildung 3.6.: Schlüsselaustausch

Um ein Public-Private-Key Verfahren nutzen zu können müssen vorher das Schlüsselpaar erzeugt und die öffentlichen Schlüssel der Kommunikationspartner ausgetauscht werden. Das Schlüsselpaar wird direkt auf dem Mobilgerät erzeugt. Dazu wird ein entsprechender Menüeintrag aufgerufen. Für den Austausch stehen mehrere Verfahren zu Verfügung. Die einfachste Variante wäre es den öffentlichen Schlüssel in einer ASCII-Datei zu speichern

und von Hand zu verteilen oder auf einem Webspace zu hinterlegen. Eine andere Möglichkeit wäre einen Schlüsselserver zu nutzen. Beide Verfahren haben aber den Nachteil, dass der Schlüssel händisch in das dazu vorgesehene Feld im Adressbuch kopiert werden müsste. Unterläuft dem Nutzer dabei ein Fehler, so ist der Schlüssel unbrauchbar. Besser wäre es, wenn dem Nutzer diese Arbeit abgenommen würde. Dazu könnte man den Schlüssel automatisiert aus einer ASCII-Datei oder dem Schlüsselserver einlesen. Alternativ wäre es aber auch denkbar den Schlüssel direkt auszutauschen. Dieses könnte mit einer SMS oder E-Mailnachricht gemacht werden.

3.4.2. Nicht funktionale Anforderungen

Genau so wichtig wie die funktionalen Anforderungen sind die nicht-funktionalen Anforderungen. Diese Anforderungen entstehen aus den funktionalen Anforderungen und beschreiben Dinge, die die Anwendung nur indirekt betreffen und deren innere Abläufe beschreiben, wie zum Beispiel Zuverlässigkeit oder Leistung und Effizienz. Eine umfangreiche Liste solcher Anforderungen können bei [41] und [49] oder in der ISO/IEC 9126 nachgelesen werden.

Für die Anwendung dieser Arbeit sind folgende Punkte von Interesse:

Zuverlässigkeit (Systemreife, Wiederherstellbarkeit, Fehlertoleranz)

Für den Nutzer ist die Zuverlässigkeit sehr wichtig. Dort steht zunächst der subjektive Eindruck für den Nutzer im Vordergrund: „Die Anwendung funktioniert“.

Zur Zuverlässigkeit gehört unter anderem die Fehlertoleranz. Das umfasst zwei Teile. Auf Eingabefehler des Benutzers sollte sehr tolerant reagiert werden. Fehlt eine Eingabe soll der Benutzer darauf hingewiesen werden und seine bisherigen Eingaben sollten erhalten bleiben. Bei einem internen Fehler oder einer Störung von Außen muss das Programm in einen konsistenten Zustand überführt werden. Ist zum Beispiel „kein Netz vorhanden“ könnte dem Benutzer angeboten werden die Nachricht zu verschicken, wenn das Netz wieder verfügbar ist oder aber der Benutzer möge es später noch einmal selber versuchen, die Nachricht zu verschicken. Fehlt der öffentliche Schlüssel des Empfängers einer Nachricht kann dem Benutzer angeboten werden, die Nachricht mit einem Passwort zu verschlüsseln oder je nach Infrastruktur den Schlüssel zu holen, beziehungsweise anzufordern. Tritt ein Fehler beim Entschlüsseln auf, so wird ein Hinweis an den Nutzer ausgegeben, die verschlüsselte Nachricht muss aber erhalten bleiben. Wird das Programm durch einen Anruf unterbrochen, sollte nach Beendigung des Gespräches das Programm mit allen bisherigen Eingaben wieder verfügbar sein. Damit die Daten wieder hergestellt werden können müssen die Entsprechenden Methoden des Android Framework implementiert werden. Das sind die onPause() und die onSaveInstanceStae(Bundle outState) [7]. Mit diesen Methoden können zum Beispiel bereits nelegte Eingabefelder gesichert und nach Wiederherstellung des Programmes wieder befüllt werden. Für diese Funktionalität hat der Programmierer der Anwendung zu sorgen.

Aussehen und Handhabung (Look and Feel)

Besonders bei kleinen Bildschirmen mit einer verhältnismäßig geringen Auflösung ist Aussehen und die Handhabung besonders wichtig. Das HTC Dream zum Beispiel hat eine Auflösung von 320 x 480 Pixel, das Motorola Milestone 480 x 854 Pixel. Gegenüber einem Arbeitsplatzrechner ist das sehr wenig. Auch die Bildschirmdiagonale ist deutlich geringer. Die meisten Geräte haben eine Diagonale von 3,2 - 3,7 Zoll (8,13 - 9,4 cm). Bei Arbeitsplatzrechnern sind 22 Zoll oder größer üblich. Ist die grafische Oberfläche zu voll oder größer angelegt als der Bildschirm, so beeinträchtigt dieses die Nutzbarkeit der Anwendung erheblich. Entweder verliert der Nutzer durch vieles scrollen die Übersicht oder die einzelnen Bildschirmelemente lassen sich nicht mehr vernünftig bedienen. Das heißt, dass die Benutzeroberfläche einfach und mit aussagekräftigen Elementen bestückt werden sollte. Ein eindeutiges Icon, das zum Beispiel ein Telefonbuch zeigt, ist hier deutlich angebrachter, als ein Button auf dem „Telefonbuch“ steht. Dieses Vorgehen schafft Platz auf der Oberfläche für Elemente, die sich zum Beispiel nicht durch ein Bild darstellen lassen. Die Menüleiste sollte, solange sie nicht gebraucht wird, ausgeblendet werden und erst mit drücken des Menüknopfes am Gerät angezeigt werden.

Leistung und Effizienz (Antwortzeiten, Ressourcenbedarf, Wirtschaftlichkeit)

Neben der Handhabung steht für den Nutzer auch die Reaktionsgeschwindigkeit, beziehungsweise die Antwortzeiten der Anwendung im Vordergrund. Es wird ihm nur schwer zu vermitteln sein, wenn er zum Beispiel 10 Sekunden lang warten muss nachdem er eine Nachricht verschickt hat, damit das Gerät wieder reagiert. Wenn eine Operation viel Rechenzeit in Anspruch nimmt, sollte sie im Hintergrund ausgeführt werden.

Des weiteren muss der Ressourcenbedarf der Anwendung im Auge behalten werden. Das gilt sowohl für Arbeitsspeicher und CPU, als auch für die begrenzte Akkulaufzeit. Im Gegensatz zu einem Desktop PC steht verhältnismäßig wenig Arbeitsspeicher und eine geringe Prozessorleistung zur Verfügung. Die Geräte der ersten Generation (zum Beispiel HTC Dream und Magic) haben insgesammt 192 MB RAM, von dem aber nur 20 MB - 50 MB zur freien Verfügung stehen. Auch ist der Prozessor mit 528 MHz relativ langsam. Bei aktuellen Geräten (zum Beispiel HTC Desire oder Motorola Milestone) stehen 256 MB (Milestone) bis 576 MB (Desire) RAM und eine 600 MHZ (Milestone), beziehungsweise eine 1 GHz CPU (Desire) zur Verfügung. Hier stehen teilweise über 100 MB RAM zur freien Verfügung. [1] Ein moderner Arbeitsplatzrechner hat im Durchschnitt 3-4 GB von denen circa 2 GB zur freien Verfügung stehen. Wenn die Anwendung nicht gebraucht wird, sollte sie beendet werden und nicht weiter im Hintergrund aktiv sein und somit Arbeitsspeicher belegen.

Bei der Wirtschaftlichkeit sind zwei Punkte zu beachten. Zum einen das eben genannte Beenden, wenn die Anwendung nicht gebraucht wird, um Akku und Arbeitsspeicher zu schonen

[1] Die Werte über freien Arbeitsspeicher sind Erfahrungswerte diverser Nutzer, die sie in Internetforen veröffentlicht haben.

und zum anderen die Gebührenseite.

Der Akku eines Android Gerätes hält je nach Beanspruchung und Leistung des Akkus einen bis drei Tage.

Bei den Gebühren sind zwei Arten von Interesse. Die Online-Gebühren und die Kosten pro SMS. Für die SMS-Nachrichten ist darauf zu achten, dass diese möglichst kurz sind (keine „Multi-SMS“) um somit Gebühren zu sparen. Bei den Online-gebühren sollte unbedingt darauf geachtet werden, dass die Anwendung nur dann auf das Internet zugreift, wenn es notwendig ist und die Verbindung abbaut, wenn sie nicht mehr gebraucht wird. Das ist vor allem wichtig, wenn der Benutzer keine Flatrate hat oder sich im Ausland befindet, da dort zum Teil hohe Roaming Gebühren anfallen.

Wartbarkeit, Änderbarkeit (Analysierbarkeit, Stabilität, Prüfbarkeit, Erweiterbarkeit)

Die Anwendung sollte wenn möglich erweiterbar sein. Um dies zu ermöglichen sollten die einzelnen Komponenten nur lose miteinander gekoppelt werden und wenn möglich auf Standards setzen. Daher werden Komponenten, wie die Verschlüsselung von der Komponente in der die Nachrichten verschickt werden, getrennt. So kann auch ein anderes Protokoll, zum Beispiel ein Instant Messenger wie ICQ, auf diese Komponenten zurückgreifen und sich zunutze machen.

Portierbarkeit und Übertragbarkeit (Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit)

Bei der Portierbarkeit auf andere Plattformen dürfte es zu Schwierigkeiten kommen. Android Anwendungen werden in Java geschrieben, haben aber auch eigene Komponenten, die für Android entwickelt wurden. Sollte die Anwendung auf eine andere Plattform portiert werden müssten in jedem Fall die Benutzeroberfläche und die Schnittstelle zur Kommunikation ausgetauscht werden. Die Kommunikationsschnittstelle umfasst den SMS Manager, die E-Mail Komponente und auch den Zugriff auf die Datenbasis, wie dem Telefonbuch und dem Schlüsselspeicher.

Eine leichte Installation ist in jedem Fall gegeben. Entweder wird die Anwendung über den Android Markt installiert oder direkt durch das Ausführen der apk-Datei. Diese kann entweder im Internet auf einem Server liegen und wird dann auf der SD-Karte gespeichert oder wird mittels USB auf der SD-Karte gespeichert. Von dort kann sie dann mit einem File-Manager installiert werden.

Sicherheitsanforderungen (Vertraulichkeit, Informationssicherheit, Datenintegrität, Verfügbarkeit)

Zu den Sicherheitsanforderungen gehören mehrere Dinge. Verschlüsselt eingehende Nachrichten sollten nur temporär entschlüsselt und im verschlüsselten Zustand auf dem Gerät

gespeichert werden. Beim Beenden des Programmes müssen alle temporär gespeicherten Daten, wie Nachrichten und Passwörter gelöscht werden. Wird ein Public-Private-Key verfahren angewandt muss der private Schlüssel verschlüsselt auf dem Gerät abgelegt werden. Das Passwort für den privaten Schlüssel muss jedesmal beim Entschlüsseln von Nachrichten eingegeben werden.

Dadurch, dass die Nachrichten und Schlüssel verschlüsselt gespeichert werden, ist eine Sicherung bei Diebstahl gegeben.

Korrektheit (Ergebnisse fehlerfrei)

Um die Qualtität der Anwendung sicherzustellen müssen Tests durchgeführt werden. Das können unter anderem Unittests sein. Es können Ver- und Entschlüsselung geprüft werden, in dem ein Klartext eingegeben, danach einmal ver- und entschlüsselt und mit der Eingabe verglichen wird. Stimmt das Ergebnis kann davon ausgegangen werden, dass die Verschlüsselung fehlerfrei arbeitet.

Für eine fehlerfreie Übertragung der Nachrichten kann jedoch von Seiten der Anwendung nicht garantiert werden. Bei Überlastung des Netzes kann es durchaus vorkommen, dass eine Nachricht nur unvollständig übertragen wird.

Flexibilität (Unterstützung von Standards)

Die Anwendung wird an den Stellen, an denen es möglich ist auf Standards setzen. Bei der Verschlüsselung kommen der AES und der RSA Algorithmus zum Einsatz. Für die Kommunikation selbst wird auf die Standards von SMS und E-Mail gesetzt. Durch Verwendung von Standards wird auch die Portierbarkeit in weiten Teilen der Anwendung begünstigt. Auch die Erweiterbarkeit wird somit unterstützt.

4. Architektur und Entwurf

In den vorangegangenen Kapiteln wurde beschrieben, was eine SMS oder E-Mail-Nachricht ist, welche kryptographischen Verfahren es gibt und welche Anforderungen an eine Anwendung gestellt werden, die das Versenden von verschlüsselten Nachrichten ermöglicht. In diesem Kapitel soll diese Vorarbeit nun umgesetzt werden. Zunächst soll ein Prototyp dieser Anwendung erstellt werden, anschließend soll der Kernbestandteil zu einer Bibliothek umgebaut werden.

4.1. Architektur der Anwendung

In den nachfolgenden Abschnitten wird die Architektur der Anwendung beschrieben werden. Dabei werden zunächst der „Sender“ und der „Empfänger“ getrennt betrachtet, anschließend werden beide Komponenten zusammengebracht werden.

4.1.1. Sender

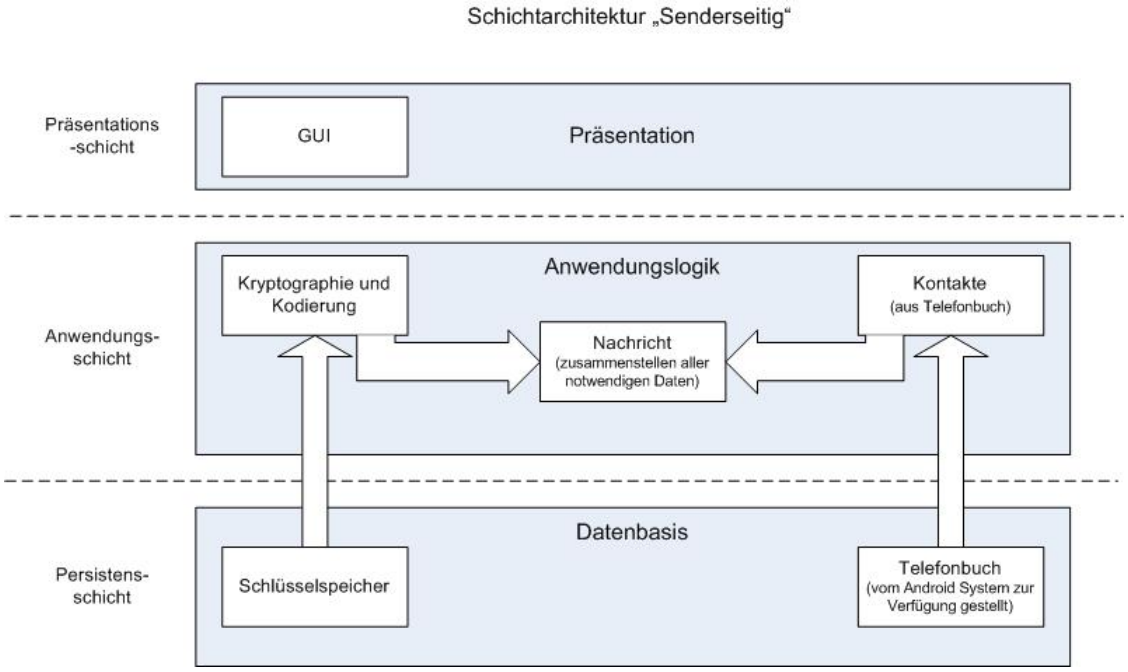


Abbildung 4.1.: Schichtarchitektur Sender

In Abbildung 4.1 ist der Schichtaufbau der geplanten Anwendung dargestellt. In dieser Grafik wurde die Empfängerkomponente noch nicht beachtet. Es handelt sich um eine drei Schichtenarchitektur. Jede Schicht greift auf die Daten der ihr zu Grunde liegenden Schicht zu. Diese Architektur soll die Trennung von Daten, Anwendungslogik und grafischer Oberfläche unterstützen, so dass es möglich ist einzelne Komponenten auszutauschen oder wieder zu verwenden. Die drei Schichtenarchitketur besteht aus folgenden Teilen: Der Persistensschicht, die die Datenbasis beinhaltet, der Anwendungsschicht in der sich die eigentliche Anwendung befindet und der Präsentationsschicht mit der Benutzeroberfläche.

Die Entscheidung für eine 3 Schichtenarchitektur liegt auch in der Struktur von Android Anwendungen. Über eine XML-Datei wird die grafische Oberfläche definiert. Somit liegt der Anwendung von Anfang an eine Trennung von Layout und Logik zugrunde. Damit wären bereits zwei der drei Schichten vorhanden. Die dritte Schicht, die Persistensschicht, wird zum Teil vom Android OS bereitgestellt, zum Beispiel das Telefonbuch, oder selbst angeleget, zum Beispiel ein „Datenspeicher“ auf der SD-Karte.

In der obersten Schicht, der Präsentationsschicht existiert einzig und allein die GUI (Graphical User Interface). Über die GUI soll die Interaktion mit dem Benutzer statt finden. Sie nimmt Eingaben des Nutzers entgegen, welche von der unter ihr liegenden Schicht verarbeitet werden. Ebenso macht sie den Nutzer auf Ereignisse, wie einer neu eingegangenen Nachricht

oder eines Fehlers, aufmerksam. Da wir momentan nur die Senderseite betrachten fällt das Benachrichtigen über neue Nachrichten weg und wird in der Empfängerkomponente behandelt.

In der unter der Präsentationsschicht liegenden Anwendungsschicht befindet sich der Kern der Anwendung. In diesem Teil der Anwendung findet der Zugriff auf die Datenbasis und die Verarbeitung der eingegebenen Daten, welche vom Benutzer geliefert werden statt. Die Anwendungslogik besteht im Wesentlichen aus drei Teilen. Hier bezeichnet als Kontakte, Nachricht und Kryptographie.

Der Block Kontakte greift auf das Telefonbuch des Benutzers zu. Es stellt dem Nachricht-Block die entsprechenden Daten zur Verfügung. Das ist der Name des Empfängers und die zugehörige Telefonnummer oder E-Mail Adresse.

Im Block Kryptographie wird die Nachricht, welche noch im Klartext vorliegt, verschlüsselt und in einen geeigneten Zeichensatz umkodiert. Anschließend wird die nun verschlüsselte und kodierte Nachricht an den Nachricht-Block weitergegeben. Wird zum Verschlüsseln ein öffentlicher Schlüssel benötigt, wird noch auf den Schlüsselspeicher zugegriffen.

Der Nachricht-Block verbindet nun alle vorliegenden Daten. Entsprechend der gewählten Versandart werden nun Nachricht und Empfänger an die entsprechenden Android Komponenten weitergegeben. Bei einer SMS-Nachricht würden nun Telefonnummer und Nachricht an den SMS-Manager weitergegeben, der die Nachricht dann verschickt.

In der Persistensschicht befindet sich neben dem Telefonbuch noch der Schlüsselspeicher. Das Telefonbuch wird vom Android Betriebssystem zur Verfügung gestellt. Der Schlüsselspeicher wird nur dann benötigt, wenn ein Public-Private-Key Verfahren angewendet werden soll. Der Schlüsselspeicher kann unterschiedlich aussehen. Dazu gibt es drei Möglichkeiten, die alle Vor- und Nachteile haben.

1. Speichern des öffentlichen Schlüssels im Notizfeld eines Telefonbuchkontaktes
2. Speichern des öffentlichen Schlüssels als ASCII-Datei auf der SD-Karte
3. Speichern des öffentlichen Schlüssels in einer anwendungsinternen SQL-Lite Datenbank

Die erste Möglichkeit hat den Vorteil, dass gleichzeitig mit dem Empfänger der Nachricht alle notwendigen Daten aus dem Telefonbuch geladen werden können. Name, Telefonnummer oder E-Mail-Adresse und der Schlüssel würden sofort vorliegen, ohne dass ein weiterer Zugriff auf einen anderen Speicher notwendig ist. Auch muss der Nutzer sich beim Synchronisieren seiner Daten, zum Beispiel mit seinem Google-Account, über die Sicherung aller öffentlichen Schlüssel keine Gedanken machen. Ein weiterer Vorteil ist, dass andere Anwendungen, die diesen Schlüssel gebrauchen könnten, diesen ohne großen Aufwand nutzen können, was zur Flexibilität beiträgt. Der Nachteil ist, dass das Notizfeld nicht mehr für den Nutzer zur Verfügung steht.

Die zweite Möglichkeit, den Schlüssel auf der SD-Karte zu speichern, ist ähnlich vorteilhaft wie die erste. Hier muss allerdings ein eindeutig identifizierbarer Name für die Datei vergeben werden, zum Beispiel die Telefonnummer. Neben dem Zugriff auf das Telefonbuch muss dann noch einmal auf die SD-Karte zugegriffen werden, um den Schlüssel einzulesen. Eine Datensicherung ist hier für den Nutzer recht einfach, muss aber von Hand gemacht werden.

Die dritte Möglichkeit hat gegenüber Möglichkeit zwei den Vorteil, dass der Zugriff schneller ist. Das Verfahren der Identifizierung des Schlüssels ist aber das Gleiche wie bei Möglichkeit zwei. Zur Datensicherung muss hier aber noch eine weitere Funktion in die Anwendung eingebaut werden, die die Schlüssel in einer geeigneten Art und Weise auf der SD-Karte speichert, damit sie vom Nutzer händisch gesichert werden können.

4.1.2. Empfänger

Die Empfängerseite sieht der Senderseite recht ähnlich. Da das meiste mit dem Sender übereinstimmt zeigt die folgende Grafik nur einen kleinen Ausschnitt.

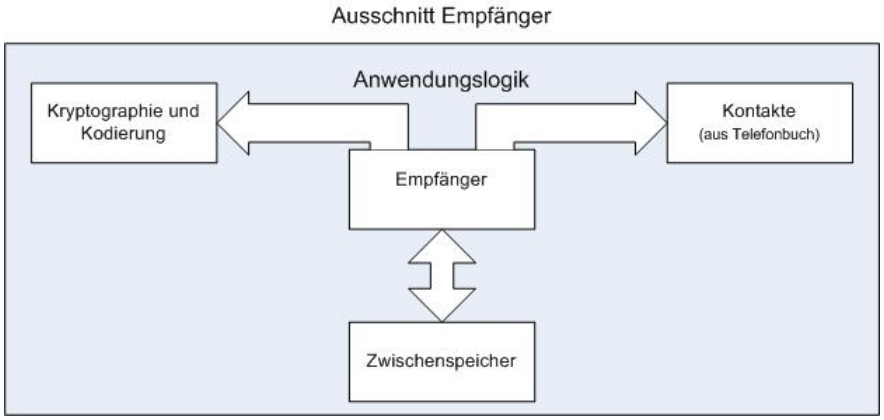


Abbildung 4.2.: Ausschnitt Empfänger

Der Ausschnitt in Abbildung 4 zeigt die Anwendungsschicht. Dieser Teil ist im Empfänger anders als beim Sender. Persistenzschicht und Präsentationsschicht sind gleich geblieben.

Die Empfängerkomponente sammelt die empfangenen Nachrichten ein. Dies geschieht mit Hilfe eines Broadcastreceivers. Anschließend wird der Nutzer über eine Notification über die eingegangene Nachricht informiert. In dieser Notification wird die Nachricht zwischengespeichert, bis sie aufgerufen wird. Des Weiteren zerlegt die Empfänger Komponente die Nachricht in ihre einzelnen Bestandteile, dem Absender und der eigentlichen Nachricht und gibt sie an die entsprechenden Anwendungsteile zur Verarbeitung beziehungsweise Auswertung weiter. An dieser Stelle muss eventuell ein Zwischenspeicher eingebaut werden, da

eine Notifiaction nur die zuerst eingegangene Nachricht speichert. Sollte eine zweite Nachricht eintreffen, bevor die erste bearbeitet wurde, geht sie zumindest für diese Anwendung verloren und müsste von Hand aus dem Android SMS Programm kopiert und entschlüsselt werden.

Über den Kryptographie-Block wird die Nachricht dann dekodiert und entschlüsselt. Je nachdem welches Verfahren genutzt wird (symmetrische oder asymmetrische Verschlüsselung) wird zusätzlich noch auf den Schlüsselspeicher zugegriffen. Wie dieser realisiert werden könnte wurde bereits in Kapitel 4.1.1 beschrieben. Im Zusammenspiel mit dem Kontakte-Block wird die Nachricht dann mit dem Namen des Absenders auf der grafischen Oberfläche angezeigt.

4.1.3. Zusammenspiel Sender und Empfänger

In den beiden vorangegangen Unterkapiteln (4.1.1 und 4.1.2) wurden die Sender- und die Empfängerkomponenten vorgestellt. Einzeln betrachtet wären beide unabhängig voneinander lauffähig. Es soll allerdings eine Anwendung entstehen, die beides beinhaltet. Würde man sie einfach kombinieren wären einige Komponenten mehrfach vertreten, daher ergibt es Sinn die entsprechenden Komponenten so zu gestalten, dass sie von beiden Teilen, dem Sender und dem Empfänger, genutzt werden können.

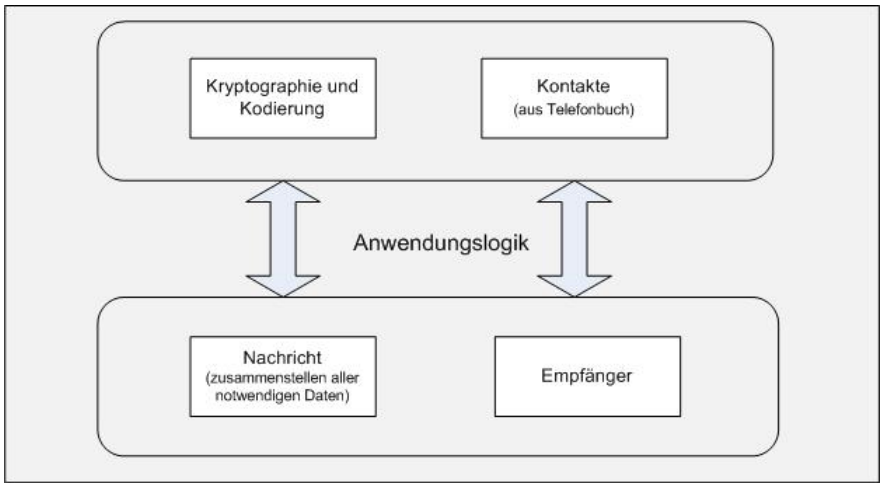


Abbildung 4.3.: Sender und Empfänger

Beide, Sender (hier Nachricht) und Empfänger, greifen auf die Koponenten zur Ver- und Entschlüsselung, sowie Kodierung und das Telefonbuch zu.

4.1.4. A/T Komponenten

Neben der bereits vorgestellten drei Schichten Architektur kann man die Anwendung auch in A und T-Komponenten zerlegen, wie es beim Architekturmodel Quasar der Fall ist [39, 45].

Die A-Komponenten beschreiben den fachlichen Anteil der Anwendung, welche unabhängig von der Technik ist. Das sind die Komponenten, die bei der migration auf ein anderes System oder eine andere Plattform weiter verwendet werden können. Zu diesen Komponenten gehören hier:

- Kommunikationsprotokolle (SMS, E-Mail, Google Talk, etc.)
- Kryptographie (AES, RSA)

Die T-Komponenten beschreiben den technischen Aspekt unabhängig von dem fachlichen Anteil. Das heißt, welche techniken werden konkret genutzt. Dazu gehören die Programmiersprache oder konkrete Formen des Datenzugriffes auf eine interne Speicherstruktur. Zu diesen Komponenten gehören hier:

- Plattform (hier Android)
- Zugriff auf Datenstrukturen (zum Beispiel das Telefonbuch)
- Programmiersprache (hier Java und Android SDK)

Bei der migration auf eine andere Plattform, wie zum Beispiel dem iPhone, müssten die T-Komponenten entsprechend angepasst werden.

4.1.5. GUI Entwurf

Die graphische Oberfläche für die Anwendung soll den Nutzer nicht mit Inhalten überfordern. Daher soll ein Layout gewählt werden, welches ihn zum Beispiel an ein standard E-Mail Formular oder eine SMS Eingabemaske erinnert. Für die Gestaltung einer GUI gibt es laut [47] unter anderem folgende Heuristiken:

- Verdichtete Informationen
  - alle Informationen innerhalb einer Ansicht
  - graphische Darstellung, die der Informationsstruktur entspricht
- Tabelle
  - Daten nach logische Kriterien sortieren
- Formular
  - standardisierte Dateneingabe

  - deutlich machen, welche Eingaben erwartet werden

Dieses sind die Heuristiken, die für die Anwendung am wichtigsten erscheinen. Alle weiteren Heuristiken können bei ([47] ab Seite 256) nachgelesen werden.

Für die GUI der Anwendung bedeutet dieses folgendes:

- Beim Start der Anwendung soll die GUI alle wichtigen Informationen anzeigen, die für den Versand einer Nachricht nötig sind. Ein Eingabefeld für den Empfänger, eines für die Nachricht und gegebenenfalls ein Feld für das Passwort. Es bietet sich an, neben dem Eingabefeld für den Empfänger einen Button zu Platzieren, der als Symbol ein Telefonbuch trägt, über den der Zugriff auf das Telefonbuch vollzogen wird.
- Wichtig ist auch, dass eine logische Struktur eingehalten wird, damit sich der Nutzer nicht erst neu Einarbeiten muss. Dazu verwendet man bereits bekannte Strukturen. Diese Strukturen lehnen sich zum Beispiel an E-Mail-Kontaktformulare an. Das heißt für Eingabefelder der GUI von Oben nach unten gesehen:

1. Empfänger
2. Betreffzeile (bei SMS-Nachrichten wird diese Zeile ausgeblendet)
3. Nachricht
4. Passwort (bei Public-Private-Key-Verfahren wird diese Zeile ausgeblendet)
5. Button Senden

Um sich eine kurze Übersicht über den bisherigen Entwurf zu machen oder Ideen festzuhalten ist eine kleine Freihandskizze durchaus sinnvoll (siehe Abbildung 4.4).

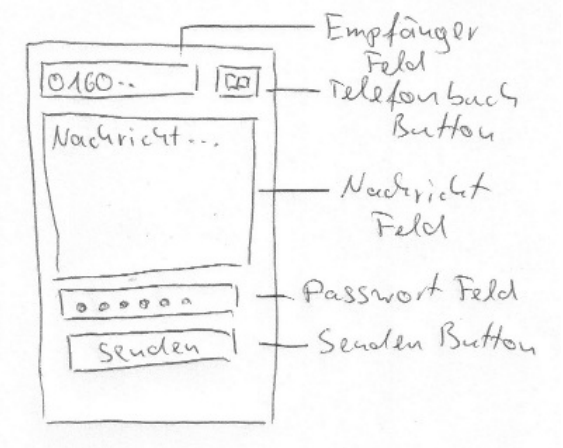


Abbildung 4.4.: Freihandskizze einer GUI

Weitere Menüstrukturen sollten, solange sie nicht gebraucht werden, verdeckt sein und erst durch das Betätigen eines Buttons oder Auslösen eines Events in den Vordergrund gebracht

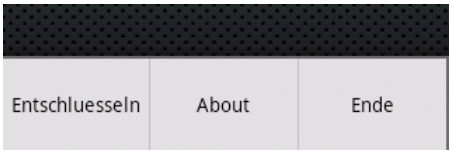


Abbildung 4.5.: Android Menü

werden. Bei einer Android-GUI definiert man dazu ein Menü, welches erst eingeblendet wird, wenn der Menü-Button auf dem Gerät gedrückt wird (siehe Grafik 4.5).

Die gleichen Anforderungen, wie für die Start-GUI, gelten auch für die Oberfläche, welche beim Empfang einer Nachricht gestartet wird.

Für den Prototypen sollen dabei noch die Oberflächen für den Empfang und das Entschlüsseln einer Nachricht definiert werden und eine Oberfläche, die eine Kurze Information über das Programm und eine kleine Anleitung liefert.

Die Oberfläche „About“ soll sich nur über das (Options)Menü aufrufen lassen. Die GUI selbst beinhaltet nur Text. Zunächst eine kurze beschreibung des Programmes, darauf folgend eine Kurzanleitung, wie das Programm zu bedienen ist.

Die dritte GUI ist die zum Entschlüsseln einer Nachricht. Sie wird entweder über das Betätigen der Notification oder durch den Button „Entschlüsseln“ im Menü gestartet. Die Oberfläche orientiert sich dabei an der Struktur der Start-GUI, die in Grafik 4.4 abgebildet ist. Das heißt von oben nach unten:

1. Absender
2. Betreffzeile (bei SMS-Nachrichten wird diese Zeile ausgeblendet)
3. Nachricht
4. Passwort (bei Public-Private-Key-Verfahren wird diese Zeile ausgeblendet)
5. Button Entschlüsseln

Das Feld „Absender“ wird automatisch mit dem Absender der Nachricht befüllt. Dort steht dann neben dem Namen auch die Absendernummer oder E-Mail-Adresse. Das Nachrichtenfeld enthält zunächst die verschlüsselte Nachricht, nach der Entschlüsselung wird dort der Klartext angezeigt.

Zum GUI Entwurf gehört auch die Notification, da sie optisch darauf hinweist, dass eine neue Nachricht eingetroffen ist. Dazu wird ein passendes Icon gewählt, hier ein geschlossenes Vorhöngeschloss, welche darauf hinweist, dass eine verschlüsselte Nachricht eingegangen ist (siehe Abbildung 4.6). Zusätzlich wird die Nachricht selbst in der Notification angezeigt. Solange die Anwendung auf alle Nachrichten reagiert (nicht nur verschlüsselte) hilft dies dabei zu entscheiden, welche Anwendung man aus der Notificationliste auswählen soll, da auch der Android SMS-Manager (in Abbildung 4.6 schon gelöscht) eine Notification setzt.

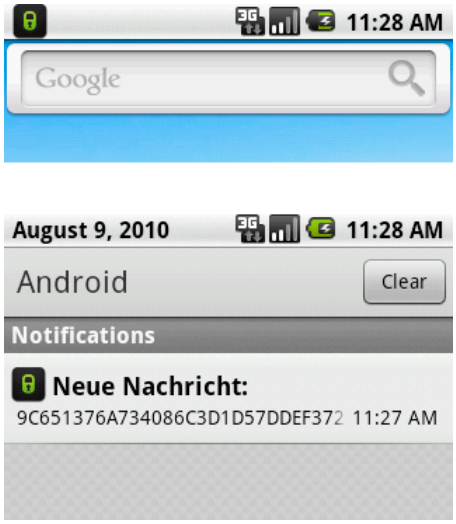


Abbildung 4.6.: Notification

Die hier gezeigte Anwendung ist durch die geringe Anzahl von grafischen Oberflächen noch recht übersichtlich. Werden es jedoch mehr Oberflächen ist es hilfreich sich einen Graphen zu Zeichnen, der die möglich Schachtelung der GUIs aufweist, beziehungsweise von welcher GUI aus eine andere Oberfläche zu erreichen ist. Für den hier vorgestellten Prototypen sieht der Graph wie folgt wie in Abbildung 4.7 gezeigt aus.

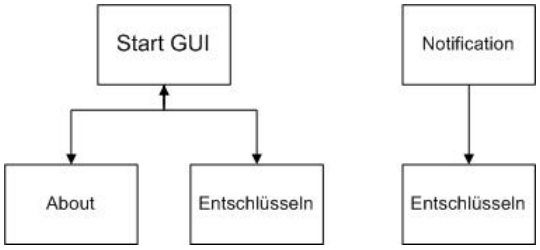


Abbildung 4.7.: GUI-Graph

4.2. Realisierung der Anwendung

In Kapitel 4.1 wurde die Architektur der Anwendung beschrieben. Für diese Arbeit soll allerdings nur ein Prototyp implementiert werden, der exemplarisch die Funktion der Verschlüsselung von SMS-Nachrichten zeigen soll.

4.2.1. Prototyp: SMS Verschlüsselung mit AES

In einem Prototypen sollen zunächst einmal die grundlegenden Funktionen getestet werde. Dazu gehören: SMS-Nachrichten senden und empfangen, die Nachrichten mit dem AES-Algorithmus und einem Passwort verschlüsseln und entschlüsseln. Die Anwendung sieht daher schematisch wie folgt aus:

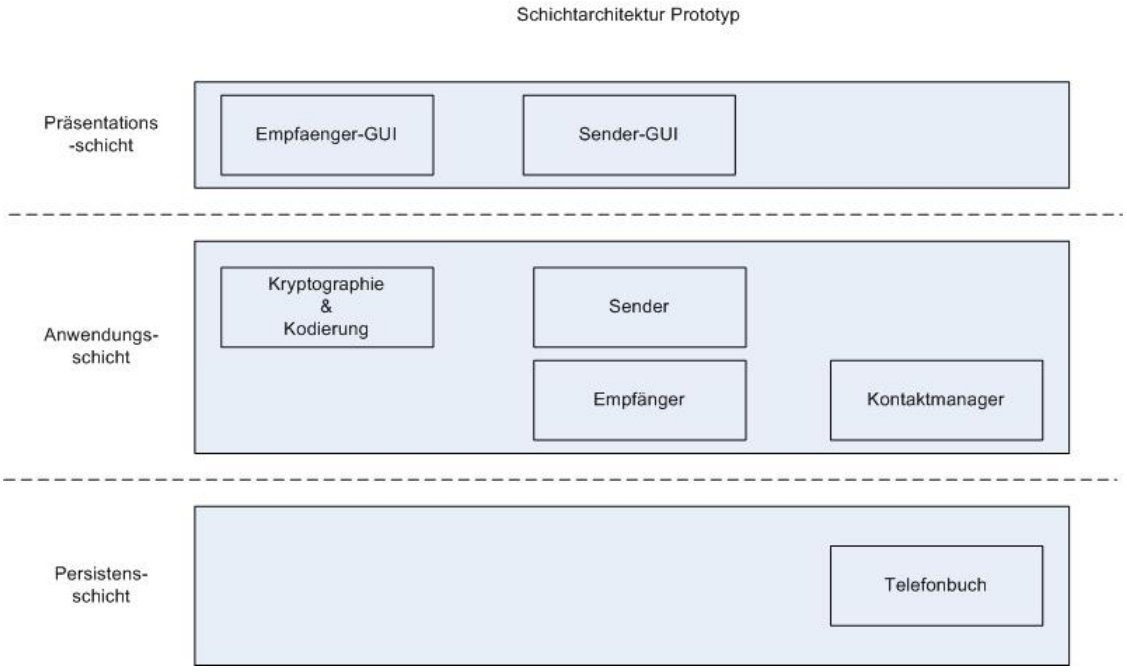


Abbildung 4.8.: Schichtmodell der Anwendung

In Abbildung 4.8 sind die einzelnen Komponenten aufgeführt. Jede dieser Komponenten besteht aus einer oder mehreren Klassen. Einzige Ausnahme bildet das Telefonbuch, welches vom Android Betriebssystem zur Verfügung gestellt wird. Durch diese Aufteilung sollen Flexibilität und Erweiterbarkeit geschaffen werden.

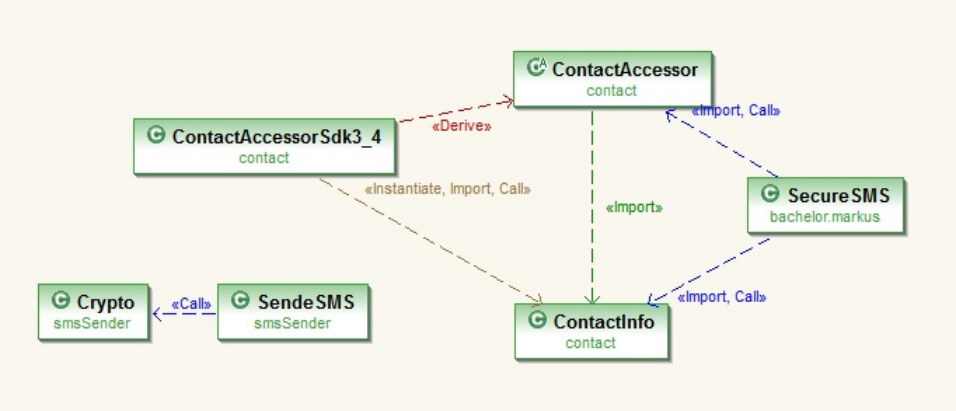


Abbildung 4.9.: Klassendiagramm der Anwendung

Wie in Abbildung 4.8 gezeigt besteht die Anwendung aus zwei, beziehungsweise drei, Schichten, wobei die Persistenzschicht von Android bereitsgestellt wird. Bleiben noch die Präsentationsschicht und die Anwendungsschicht.

Die Anwendug hat drei seperate grafische Oberflächen, beziehungsweise Views, wovon zwei für die Funktion der Anwendung bestimmt (Abbildung 4.10) sind und eine zur Information über den Prototypen (eine About Seite) dient.

Alle Views werden über XML-Files definiert und vom Android Ressourcencompiler ausgewertet (siehe 2.2.2.1). Durch die Definition über das XML-File erreicht man, wie bereits erwähnt, auch eine Trennung von Layout und Logik. Die „erste GUI“, welche direkt beim Start der Anwendung gezeigt wird (siehe Abbildung 4.10 links), dient zur Aufnahme der Daten um eine verschlüsselte Nachricht zu versenden. Über den Button Empfänger wird der Zugriff auf das Telefonbuch vollzogen und Name, sowie Telefonnumer herausgezogen. Der Name wird lediglich in der GUI angezeigt. Die Telefonnummer wird als String zwischengespeichert. Ebenso wird die Nachricht im Klartext als String gespeichert. Das Passwort wird der GUI im Klartext übergeben und mittels schwarzer Punkte im Eingabefeld unkenntlich gemacht. Intern wird es als Bytearray zwischengespeichert. Das hat gegenüber Strings den Vorteil, dass man es nach Beendigung des Sendevorganges überschreiben kann. Dies wird mit Arrays.fill() gemacht. Strings dagegen sind, nachdem sie erzeugt wurden, unveränderlich. Die Empfängeroberfläche verfährt mit dem Passwort genauso.

Mit dem „Drücken“ des Senden Buttons werden Telefonnummer des Empfängers, Passwort und Klartextnachricht an die Klasse SendeSMS übergeben. Die Klasse Crypto übernimmt nun die Verschlüsselung und Kodierung der Nachricht. Zuerst wird aus dem Passwort, welches als Bytearray vorliegt, mit dem KeyGenerator aus javax.crypto, welches seit JCE1.2 zum Lieferumfang von Java und somit auch zum Android-SDK gehört, ein Schlüssel für symmetrische Verschlüsselung erzeugt. Für den Prototyp wurde der AES-Algorithmus mit einer

Algorithm 4.1 Intent-Filter Deklaration

```
<receiver android:name="smsEmpfaenger.Empfaenger">
  <intent-filter>
    <action android:name= "android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

Schlüssellänge von 128 Bit ausgewählt. Anschließend werden der Schlüssel und die Klartextnachricht der Verschlüsselungsfunktion übergeben. Bevor die Nachricht verschickt wird, wird der verschlüsselte Text noch kodiert und in eine Folge von Hex-Symbolen umgewandelt. Dabei wird ein Chiffrezeichen in zwei Hex-Symbole übersetzt. Die Kodierung findet ebenfalls in der Klasse Crypto statt. Zuletzt wird die kodierte Nachricht mit der Empfängernummer an den Android SMS-Manager weitergegeben und verschickt.

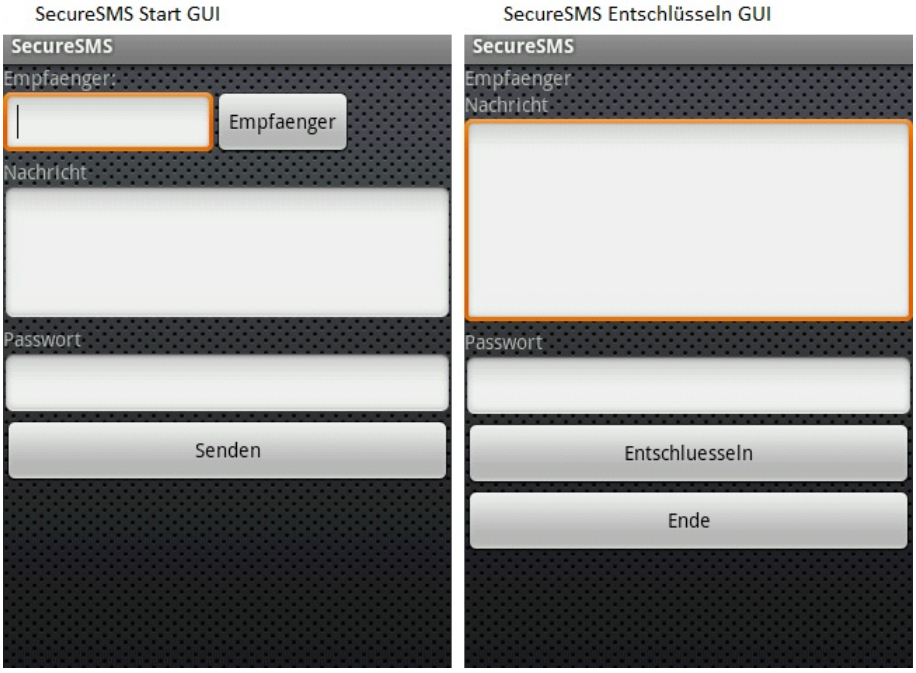


Abbildung 4.10.: SecureSMS GUI

Die Empfängerkomponente ist als BroadcastReceiver implementiert. Über das Android Manifest wird definiert, auf welches Ereignis, hier der Eingang einer SMS-Nachricht, reagiert werden soll. Der Empfänger reagiert dabei auf alle eingehenden SMS-Nachrichten. Der Quellcode (4.1) zeigt den entsprechenden Eintrag aus dem Manifest.

Nach Eingang der Nachricht setzt der Empfänger eine Notification, die in der Notification Bar angezeigt wird. Dieses geschieht über einen Intent. Dem Intent wird eine ID (ein Integerwert)

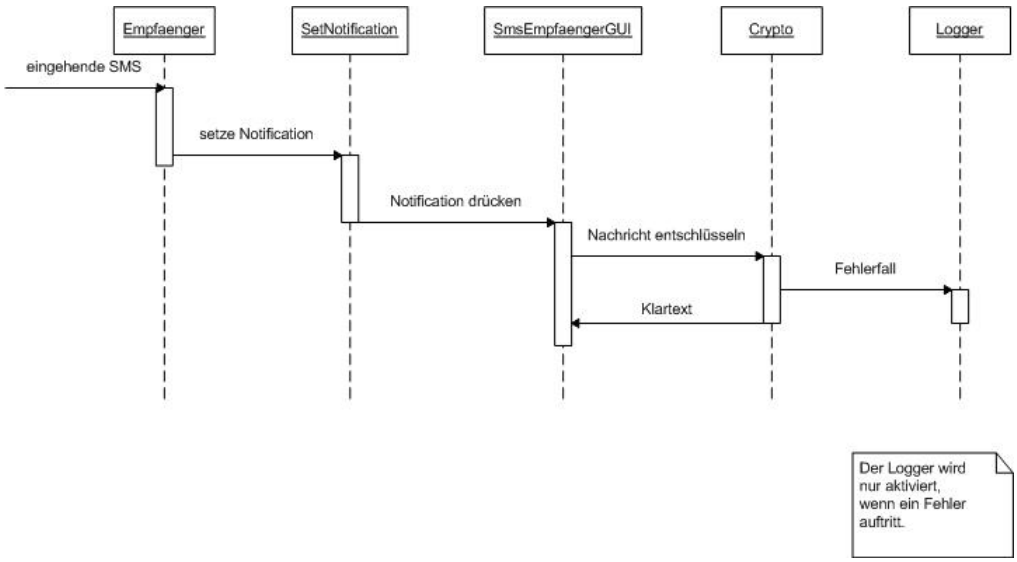


Abbildung 4.11.: Laufzeitdiagramm Empfänger

und der Message Body mit der verschlüsselten Nachricht mitgegeben. Die ID dient dazu die Notification später löschen zu können. Der Empfänger beendet sich selbst, nachdem er seine Aufgabe ausgeführt hat.

Um die Nachricht zu entschlüsseln wird die Notification aufgerufen. Diese startet über einen Intent die Anwendung mit der entsprechenden Oberfläche (Abbildung 4.10 rechts). Dem Intent wird wiederum die Nachricht mitgegeben und die ID der Notification, damit sie wieder gelöscht werden kann.

Nach dem Start der GUI wird die Notification gelöscht und das Nachrichtenfeld mit der verschlüsselten Nachricht gefüllt. Anschließend wird über den Button „Entschlüsseln“ nach der Eingabe des Passwortes die Nachricht entschlüsselt und angezeigt.

Der Prototyp wird durch das Betätigen des Ende-Button, beziehungsweise Betätigen des Zurück oder Home-Button auf dem Gerät, mit der Methode finish() beendet. Durch den Aufruf dieser Methode wird das Programm sauber beendet und der belegte Speicher wieder frei gegeben.

In den Abbildungen 4.11 und 4.12 ist zu sehen welche Klassen zur Laufzeit wann aktiv sind. Es wird auch dargestellt, welche Klasse eine andere Aufruft und ob es Rückgabewerte gibt. Zu beachten ist, dass nur die Klassen des Prototypen, mit einer Ausnahme, selbst betratrechtet wurden. Diese Ausnahme betrifft den Android SmsManager in der Abbildung 4.12. In der Grafik startet, beziehungsweise übergibt der SmsManager den Fehlercode an den Logger. Der Einfachheit halber wurde diese Darstellung gewählt, auch wenn es so nicht ganz richtig ist. Tatsächlich übergibt die den SmsManager umgebende Klasse - SendeSMS - den Fehlercode an den Logger (sieh Abbildung 4.9). Da von dieser Klasse aber keine Instanz

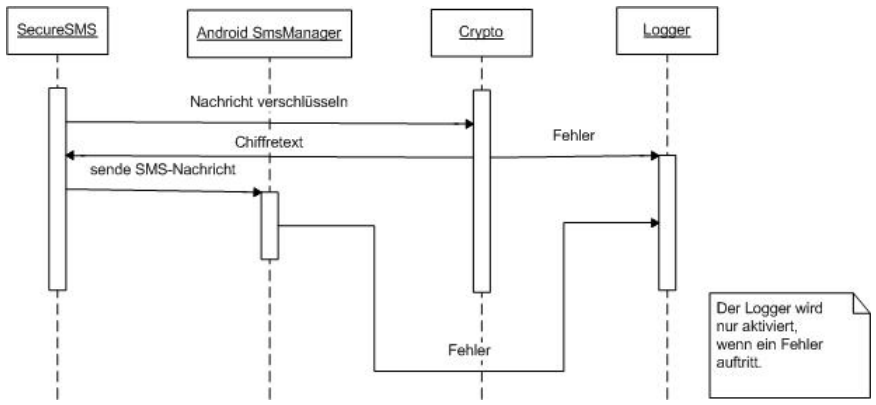


Abbildung 4.12.: Laufzeitdiagramm Sender

erzeugt wird, würde sie hier nicht auftauchen.

4.2.2. Bewertung des Prototypen

In dem Prototypen wurden die Grundfunktionen der späteren Bibliothek getestet. Der Prototyp ist in der Lage SMS-Nachrichten zu ver- und entschlüsseln, sowie diese Nachrichten zu senden und zu empfangen. Die geforderte Trennung von Layout, Logik und Daten wurde dabei eingehalten. Aus dem Prototypen lassen sich einige Verbesserungsvorschläge für die Bibliothek herausziehen.

Auf die Problematik mit der Kodierung wurde schon mehrfach hingewiesen. Daher sollte dieser Bestandteil des Prototypen in der Bibliothek von der „Verschlüsselungseinheit“ getrennt werden, so dass er, wenn gewünscht, gegen eine andere Klasse ausgetauscht werden kann. Auch wäre es so möglich verschiedene Kodierer für unterschiedliche Zwecke einzubinden.

Leider ist nicht ersichtlich in welcher Form das Passwort von Android zwischengespeichert wird, oder ob es überhaupt gespeichert wird.

```
byte[] passwort = passwortFeld.getText().toString().getBytes();
```

Es wird zwischndurch eine toString() Methode aufgerufen. Ob allerdings intern ein String gespeichert wird ist unklar. Wenn dem so sein sollte wäre das eine mögliche Schwachstelle. Diese könnte aber nur ausgenutzt werden, wenn das Programm manipuliert wird, da eine externe Anwendung auf den fremden Arbeitsspeicherbereich keinen Zugriff hat. Dies wird durch die Dalvik Virtual Machine verhindert. In der Bibliothek sollte trotzdem vermieden werden Passwörter als String zwischenzuspeichern.

Weggelassen wurde hier das Testen des Prototypen. Es müssten die internen Funktionen, wie zum Beispiel Ver- und Entschlüsselung, die graphische Oberfläche (Funktion und Usability) und das Verhalten der Anwendung im Telefonnetz bei Fehlern, die mit der Anwendung

direkt nichts zu tun haben, getestet werden. Für die Umsetzung der Bibliothek sollten Tests mit eingeplant werden.

Der in Kapitel 4.1.2 erwähnte „SMS-Zwischenspeicher“ wurde aufgrund Zeitmangels im Rahmen dieser Arbeit nicht weiter betrachtet.

5. Bibliothek

5.1. Idee einer Bibliothek

„Code reuse is one of the Holy Grails of computer programming.“ ([52])

Dieser Satz fasst sehr schön zusammen, worum es bei der Erstellung einer Bibliothek geht.

Der essentielle Anteil der in 4.2 gezeigten Anwendung besteht aus den Klassen und Methoden zur Kodierung, Verschlüsselung und Senden einer Nachricht. Es wäre schön, wenn diese Komponenten ohne großen Programmieraufwand für andere Anwendungen zugänglich gemacht werden könnten. Zu diesem Zwecke soll eine Bibliothek erstellt werden, die die entsprechende Methoden bereitstellt.

Ein wichtiger Bestandteil dieser Bibliothek ist, dass sie kompatibel zu bereits bestehenden Anwendungen ist. Das heißt, dass Methoden zur Verfügung gestellt werden müssen, die die gleiche Signatur haben, wie die entsprechende Android Methoden. Trotzdem soll es möglich sein über diese Methoden verschlüsselte Nachrichten zu schicken. In der Bibliothekt wird daher eine Methode vorkommen müssen, die wie folgt aussieht:

> sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Um diese Methode weiterhin anbieten zu können wird ein Wrapper um den SMS-Manager gelegt (siehe Abbildung 5.1).

Zusätzlich zur Rückwärtskompatibilität sollen Methoden bereit gestellt werden, um Anwendungen zu entwickeln, die das Versenden und Empfangen verschlüsselter Nachrichten ermöglichen sollen.

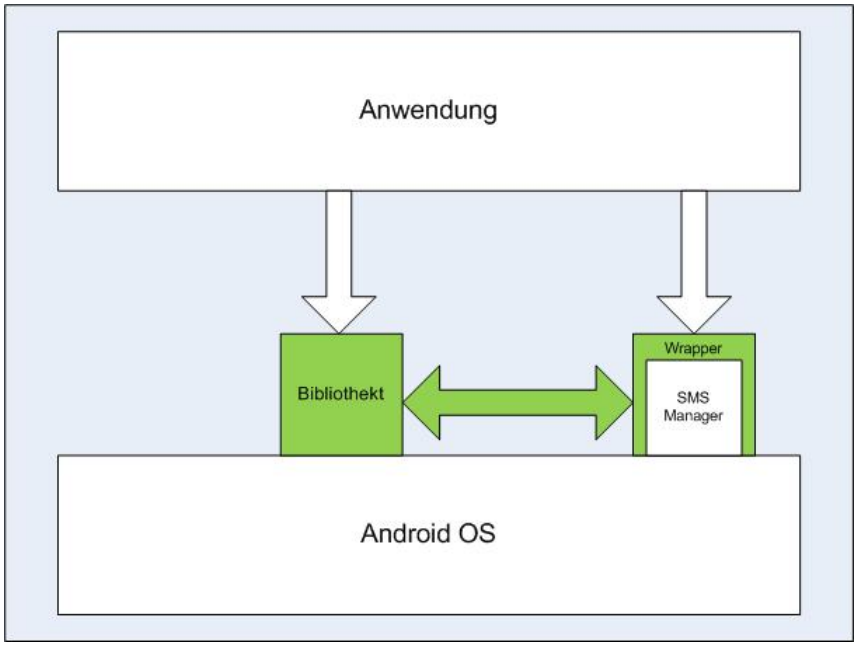


Abbildung 5.1.: Wrapper

5.2. Realisierung der Bibliothek

In diesem Abschnitt sollen die Kernbestandteile des Prototypen in eine Bibliothek umgebaut und ergänzt werden. In 5.2.1 soll die Bibliothek selbst beschrieben werden und in 6 werden mögliche Testfälle benannt.

5.2.1. Prototyp der Bibliothek

In Abbildung 5.1 wurde gezeigt, wo die Bibliothek im System angesiedelt werden soll. Sie wird direkt zwischen das Android Betriebssystem und die Anwendung gesetzt.

Abbildung 5.2.: Package Dependency

Die Bibliothek ist in zwei Teile unterteilt. Der Bibliothekt selber und dem Logger (siehe auch Abbildung 5.2).

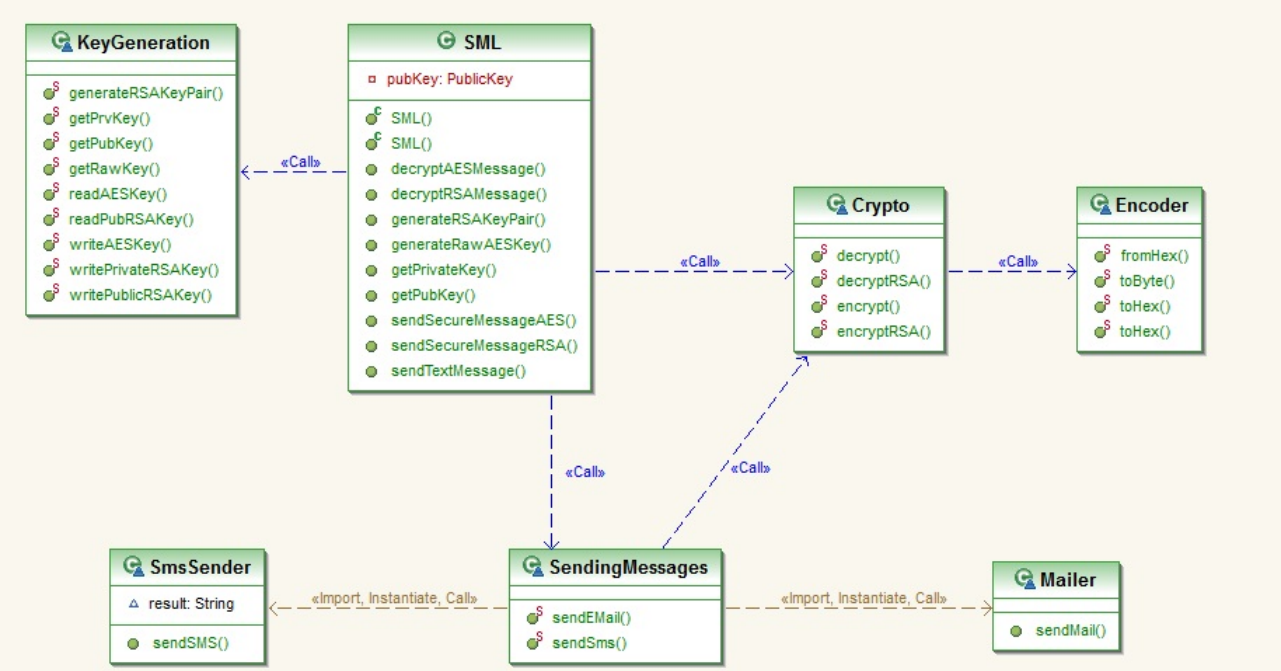


Abbildung 5.3.: Objektmodell der Bibliothek (Package de.markus.bachelor.lib)

Logger

Der Logger dient dazu, den späteren Anwender der Bibliothek bei der Fehlersuche zu unterstützen. Im Normalfall ist es dem Anwender nicht möglich in den Quellcode hineinzuschauen. Daher ist es wichtig, dass möglichst alle Situationen, in denen Fehler auftreten können, mit geloggt werden. Das hilft dabei herauszufinden, ob der Fehler beim Nutzer der Bibliothek liegt oder gar in der Bibliothek selbst.

In der Standardeinstellung schreibt der Logger die Logfiles mit Angabe des Datums auf die SD-Karte. Im Konstruktor kann dieser Pfad angepasst werden. Es wird sowohl eine kurze Beschreibung, wo der Fehler aufgetreten ist, als auch die Exception selbst in das Logfile geschrieben.

Bibliothek

Wie bereits in Kapitel 5 genannt muss folgende Methode weiterhin bereit gestellt werden (siehe auch[22]):

> sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)

Sie soll dazu dienen, dass bereits bestehende Anwendungen ohne Änderungen die Bibliothek nutzen können und somit um Verschlüsselung ergänzt werden können. Um dieses zu ermögliche wird ein Wrapper geschrieben. Hier treten jedoch zwei Probleme auf, die nur eingeschränkt gelöst werden können.

Zum einen ist es nicht möglich der Methode zusätzliche Parameter zu übergeben, wie zum Beispiel einen Schlüssel oder ein Passwort. Möchte man dennoch einen Schlüssel übergeben, so geschieht dies hier über einen Konstruktor, der wie folgt aussieht:

public SML(String loggerPath, String encryption, byte[] rawKey, PublicKey pubKey)

Für die Verschlüsselung sind die letzten drei Parameter von Interesse. Der Parameter „encryption“ nimmt den Verschlüsselungsalgorithmus als String auf, hier RSA oder AES. Entsprechend des Verschlüsselungsalgorithmus nimmt „rawKey“ einen AES-Schlüssel auf, beziehungsweise „pubKey“ einen öffentlichen RSA-Schlüssel. Der Schlüssel und der Verschlüsselungsalgorithmus werden in einer Variable zwischengespeichert und in der Methode sendTextMessage() entsprechend ausgewertet und eingebunden (siehe B.1 und B.2).

Nach Beendigung des Verschlüsselungsvorganges werden die zwischengespeicherten Schlüssel wieder gelöscht. Der AES-Schlüssel wird wie im Prototyp als Bytearray gespeichert und mit Arrays.fill() mit Nullen überschrieben. Der öffentliche RSA-Schlüssel wird als PublicKey (java.security.PublicKey) gespeichert und braucht nicht gelöscht zu werden, da er öffentlich ist. Somit wäre eine Kompromittierung dieses Schlüssels unkritisch.

Zum anderen ist es nicht möglich auf Exceptions zu reagieren, die eventuell auftreten können, wenn zum Beispiel etwas beim Ver- und Entschlüsseln schief läuft. Sollte ein Fehler auftreten und dieser innerhalb der Methode nicht behandelt werden können, so kann keine Exception nach außen gereicht werden, da sonst die Rückwärtskompatibilität nicht mehr gegeben ist. Das betrifft jedoch nicht die Exceptions, die vom Android SMS Manager geworfen werden. Alle anderen Fehler können nur über die Logfiles nachvollzogen werden.

Neben dem oben genannten Konstruktor gibt es noch zwei weitere Konstruktoren. Den Default-Konstruktor und einen weiteren, der als Parameter den Pfad des Loggers anpassen kann. Diese beiden Konstruktoren sind für Neuentwicklungen von Anwendungen vorgesehen.

Zusätzlich zur sendTextMessage-Methode gibt es zwei weitere Methoden, um verschlüsselte Nachrichten zu senden:

sendSecureMessageAES() und sendSecureMessageRSA()

Prinzipiell machen beide Methoden das Gleiche. Nur die Signaturen unterscheiden sich minimal. Wie die Namen schon andeuten ist die erste der beiden für AES und die zweite für RSA ausgelegt. Innerhalb dieser Methoden wird die Nachricht dann mit dem entsprechenden Schlüssel verschlüsselt, anschließend kodiert und gesendet. Ein wichtiger Bestandteil dieser beiden Methoden ist das Exception Handling. Hierbei muss unterschieden werden,

nach Exceptions, die behandelt werden können und denen, die eine Fehlermeldung nach außen geben.

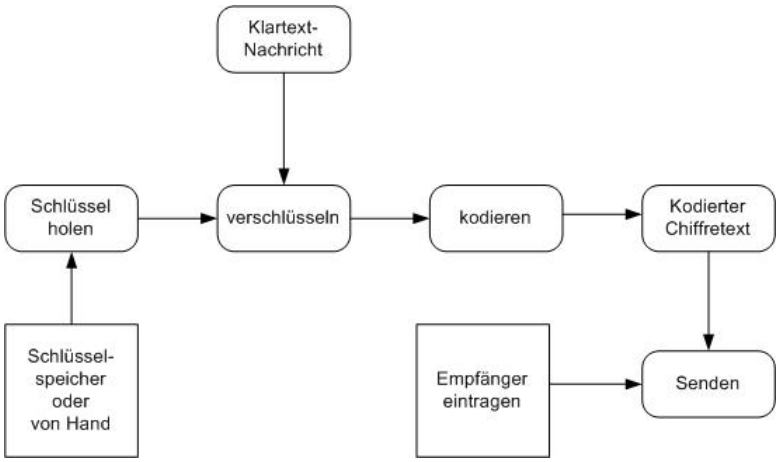


Abbildung 5.4.: Bibliothek Datenfluss Verschlüsselung

Für den Nutzer der Bibliothek sind letztere die wichtigeren. Sie sollen ihm dabei helfen Fehler, die aufgetreten sind, zu behandeln oder zu beseitigen. Dabei ist es wichtig, dass die Fehlermeldung aussagekräftig ist. Das Exception Handling hat dabei zwei Komponenten. Einmal eine Fehlermeldung als Rückgabewert in Form eines Strings. Der zweite Teil ist der schon erwähnte Logger.

Der Rückgabewert kann somit vom Anwender der Bibliothek genutzt werden, um Fehlersuche zu betreiben oder um selbstständig eine Fehlerbehandlung außerhalb der Bibliothek zu implementieren. Ebenso kann der Rückgabewert, wenn gewünscht, direkt an den Endanwender weitergeben werden, um ihn zum Beispiel auf einen möglichen Eingabefehler hinzuweisen. Ein konkretes Beispiel ist, wenn eine Nachricht mit einem RSA-Schlüssel verschlüsselt und verschickt werden soll, dieser aber nicht vorhanden ist. Das Programm muss in einem konsistenten Zustand bleiben und sollte dem Anwender darauf hinweisen, dass der öffentliche Schlüssel des Empfängers nicht vorhanden ist. Der Rückgabewert sieht dabei wie folgt aus:

Fehlertyp#kurze textuelle Beschreibung

Beispiel:

Schluesselfehler#Kein Schluessel übergeben

Der zweite Teil des Exception Handling ist der Logger, der zusätzlich, wenn gewünscht, die Fehlermeldungen in ein Logfile wegschreibt. Das Format bleibt dabei gleich, wird allerdings um Datum, Uhrzeit und die geworfende Exception ergänzt:

Datum#Uhrzeit#Fehlertyp#kurze textuelle Beschreibung#Java/Android Exception

Für die Entschlüsselung der Nachrichten stehen die beiden decrypt-Methoden (decryptAESMessage() und decryptRSAMessage()) zur Verfügung. Der Entschlüsselungsvorgang (Abbildung 5.5) läuft entsprechend der Verschlüsselung in umgekehrter Reihenfolge ab.

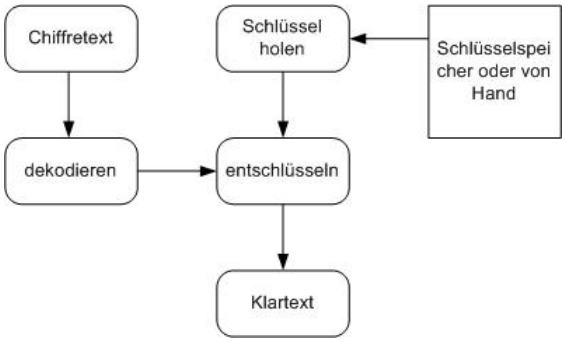


Abbildung 5.5.: Bibliothek Datenfluss Entschlüsselung

Der Encoder übersetzt alle verschlüsselten Nachrichten in einen String von Hex-Symbolen beziehungsweise von Hex zurück in einen „normalen“ String. Jedes Chiffresymbol wird hierbei durch zwei Hex-Symbole ersetzt. Ebenso werden die RSA-Schlüssel vom Encoder in das Hexformat überführt. Somit ist ein einheitlicher Zeichensatz auch systemübergreifend gegeben. Es wird dabei der ASCII-Zeichensatz verwendet, den jeder Computer oder jedes moderne Mobiltelefon versteht. Im Gegensatz zum Prototypen ist der Encoder hier eine eigene Klasse. Möchte man den Encoder austauschen, muss nur eine Klasse bearbeitet werden. Wie schon erwähnt, steigert dieses die Flexibilität, Änderbarkeit und Wartbarkeit der Bibliothek.

Die Klassen SmsSender und Mailer sind die letzten beiden in der Hierachie. Hier wird die bereits verschlüsselte und kodierte Nachricht versendet.

Der SmsSender übergibt die verschlüsselte Nachricht an den Android SMS-Manager ([22]). Bevor die Übergabe vollzogen wird muss noch geprüft werden, ob die Zeichenlänge nicht überschritten wird. Es kann dann entschieden werden, ob der Sendevorgang abgebrochen wird oder ob eine Multipart-SMS verschickt wird. Aktuell wird eine Fehlermeldung zurück gegeben, wenn die Nachricht die zulässige Länge von 160 Zeichen überschreitet.

Soll eine E-Mail verschickt werden, tritt die Klasse Mailer in Aktion. In der aktuell vorliegenden Version der Bibliothek wird die Nachricht an einen Intent weitergegeben, bei dem es sich um einen impliziten Intent handelt.

```
public void sendMail(Intent emailIntent){
    startActivity( Intent.createChooser( emailIntent, "Send mail..."));
```

}

Das heißt, dass die Bibliothek die Nachricht nicht selbst verschickt, sondern die Nachricht samt Empfänger von einem Programm verschicken lässt, welches der Endnutzer auf seinem Gerät installiert hat. Zum Besipiel der interne Google-Mail-Client oder einen anderen Client seiner Wahl. Der Code (B.3), wie der Intent befüllt wird, findet sich in Anhang B.

Neben diesen „Hauptmethoden“ gibt es noch einige Hilfsmethoden, die dem Anwender der Bibliothek die Benutzung einfacher machen sollen.

Die Klasse KeyGeneration stellt Methoden zur Verfügung, die dem Nutzer RSA-Schlüsselpaare oder AES-Schlüssel erzeugen. Für Neuentwicklungen hat dies den Vorteil, dass der Anwender der Bibliothek sich nicht selbst mit diesem Thema auseinander setzen muss. Der zweite Vorteil ist, dass die Schlüssel gleich in dem Format vorliegen, in dem sie die Bibliothek braucht. Die RSA-Schlüssel werden entweder als Schlüsselpaar oder mit den entsprechenden Methoden als öffentlicher und privater Schlüssel (java.security.PublicKey und java.security.PrivateKey) bereitgestellt. Die AES-Schlüssel werden als „rawKey“ in Form eines Bytearray geliefert.

5.3. Bewertung

In den vorangegangenen Kapiteln wurde jeweils das Design und die Architektur besprochen und die daraus folgende Umsetzung. In Kapitel 4.2.2 wurde bereits kurz bewertend auf den Prototypen eingegangen. Die Änderungsvorschläge aus diesem Kapitel wurden übernommen. Der Kodierer wurde von der Kryptographie getrennt und es wurde darauf verzichtet Passwörter als String zwischenzuspeicher.

Die in Kapitel 4.2.2 geforderten Tests wurden für die Bibliothek in Form von Unit-Tests umgesetzt.

Einige Ideen wurden im Laufe der Entwicklung verworfen, beziehungsweise nicht umgesetzt. Anfangs sollte es über Kompression der Klartextnachricht (Variante 1) oder des Chiffretextes (Variante 2) ermöglicht werden längere Klartexte per SMS zu verschicken. Die Resultate aus diesen Experimenten waren allerdings für kurze Texte, etwa 150 bis 200 Zeichen, unzureichend, so dass diese Idee wieder verworfen wurde.

Für beide Tests wurde ein Zufalls-String mit einer Länge von 150 Zeichen, beziehungsweise 200 Zeichen erzeugt. Ohne Kompression ergaben so 150 Klartextzeichen 320 Chiffrezeichen und 200 Klartextzeichen ergaben 416 Chiffrezeichen. Als Verschlüsselungsalgorithmus wurde AES gewählt, zur Kompression wurde GZIP (java.util.zip.GZIPInputStream ,bzw GZIPOutoutStream) verwendet. Als Kodierung wurden Hex-Symbole gewählt.

Bei Variante 1 wurde ein String mit 150 Zeichen erst komprimiert und dann verschlüsselt. Aus 150 Klartextzeichen ergaben sich so 288 Chiffrezeichen und aus 200 Klartextzeichen wurden 320 Chiffrezeichen.

Bei Variante 2 (erst verschlüsseln, dann komprimieren) gab es folgende Ergebnisse: 150 Klartextzeichen ergaben 366 Chiffrezeichen und 200 Klartextzeichen ergaben 462 Chiffrezeichen. Variante 2 ist somit noch unwirksamer als Variante 1.

Die Kompression verkleinert die zu übertragende Textmenge, jedoch nicht stark genug. Sollte Kompression später einmal eingesetzt werden, ist in jedem Fall Variante 1 zu bevorzugen.

6. Test

Um die Qualität des Prototypen und der Bibliothek gewährleisten zu können sind Tests durchzuführen. Dazu muss ein Testkonzept erstellt werden, das Ziel der Tests und die Testdaten erfasst werden. Ebenso muss die Testumgebung gewählt werden.

6.1. Testkonzept und Testdaten

Die Bibliothek soll in mehreren Stufen getestet werden.

1. Komponententest, beziehungsweise Unittest
2. Integrationstest
3. Systemtest

Für den Komponententest wird das JUnit Framework genutzt, welches für Android ein wenig angepasst wurde. In Abbildung 6.2 ist dieses Framwork abgebildet. [21]. Die einzelnen Unittests sind genauso aufgebaut, wie „normale“ Unittests, wie sie zum Beispiel aus JUnit bekannt sind. Als Testumgebung wird die Eclipse IDE mit den entsprechenden Android Plugins und der Android Emulator genutzt. Der Komponententest für die Bibliothek wird in Kapitel 6.2 genauer erläutert.

Für den Intergrationstest werden die einzelnen Komponenten zusammengeführt und getestet. Hierbei können zum Beispiel Schnittstellenfehler aufgedeckt werden. Bei diesem Test werden die einzelnen Komponenten Stück für Stück zusammengebaut und somit Schritt für Schritt getestet. Dieses Vorgehen nennt man inkrementelle Integration (nach [46]), welche auch in Grafik 6.1 abgebildet ist. Dieses Verfahren ist für die Testfolgen zwei und drei nur für den Prototypen durchführbar. Für die Bibliothek reicht ein etwas umfangreicherer Unittest aus, bei dem das Zusammenspiel der Kryptographiekomponente mit der Kodierkomponente getestet wird.

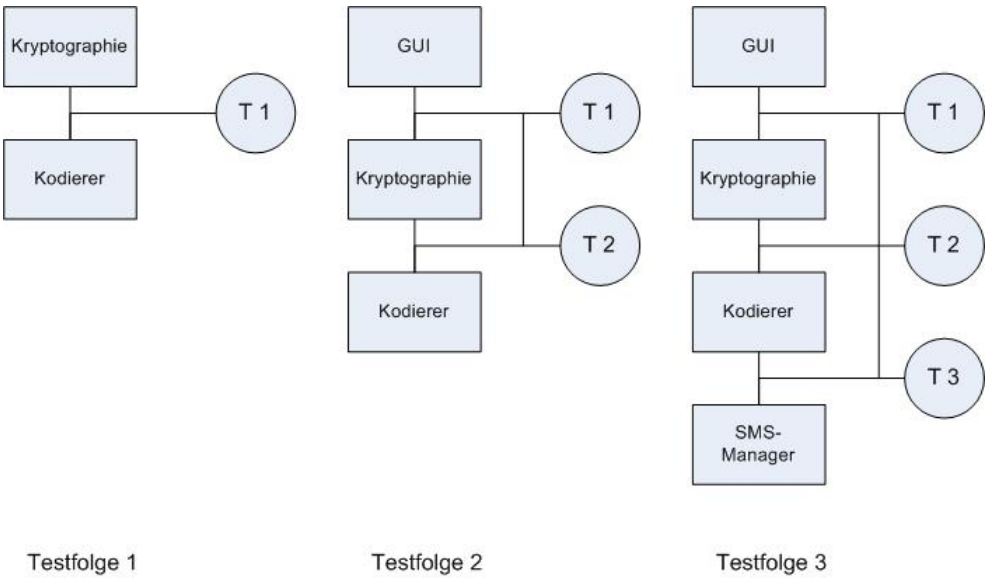


Abbildung 6.1.: Test durch inkrementelle Integration

Zuletzt wird ein Systemtest durchgeführt. Für die Bibliothek ist dieser Test nur bedingt möglich, da sie hierfür in eine Anwendung integriert werden müsste um das Zusammenspiel mit den Androidkomponenten, wie zum Beispiel dem SMS-Manager, testen zu können. Für eine vollständige Anwendung ist dies aber unabdingbar.

Beim Testen soll sich auch an den Anforderungen orientiert werden, da für den Endnutzer die Funktion der Anwendung im Vordergrund steht. Die einzelnen Komponenten mögen zwar getestet sein, wenn aber die Anforderungen nicht erfüllt werden oder nicht gegeben sind, habe die Tests ihr Ziel nicht erreicht. Zu den Anforderungen, die getestet werden müssen gehören unter anderem:

1. funktioniert die Ver- und Entschlüsselung und somit die Schlüsselerzeugung
2. funktioniert der Versand und der Empfang von Nachrichten
3. wird der richtige Empfänger ausgewählt (funktioniert der Telefonbuchzugriff richtig)
4. wie sind die Reaktionszeiten der Anwendung

Der erste Punkt kann mit Unittests und den Integrationstests überprüft werden. Das gilt für den Prototypen, als auch für die Bibliothek. Zweitens und drittens können über Systemtests geprüft werden. Zusätzlich kann die Funktionalität auch über Benutzertests geprüft werden. Dazu bearbeiten mehrere Nutzer Aufgaben an dem System und protokollieren vorhandene Fehler und Mängel. Gleichzeitig wird die Nutzbarkeit der Oberfläche überprüft.

Der vierte Punkt ist für den Anwender oftmals genauso wichtig wie die reine Funktionalität. Es müssen also Zeiten gemessen werden, wie lange es dauert, bis die Anwendung nach

betätigen einer Funktion wieder reagiert.

Um die Tests durchführen zu können müssen Testdaten ausgewählt werden. Für die Bibliothek können folgende Testdaten bereit gestellt werden:

- AES und RSA Schlüssel
- Klartext zum Verschlüsseln
- Chiffretext als Eingabeparameter (während des Test erzeugt)

Diese Testdaten können sowohl für den Komponententest, als auch für den Integrationstest genutzt werden.

Das Testziel ist hier, die Funktionalität der Bibliothek zu prüfen und mögliche Fehler in den Komponenten zu finden. Dabei geht es darum festzustellen, ob die einzelnen Funktionen der Bibliothek, zum Beispiel die Ver- und Entschlüsselung, funktionieren. Ein weiteres Ziel ist zu prüfen, ob die Schnittstellen zwischen den einzelnen Komponenten in Ordnung sind, zum Beispiel die Übergabe des Chiffretextes an den Kodierer. Zuletzt wird noch getestet ob die Systemkomponenten, wie der Android SMS-Manager, korrekt angesprochen werden.

6.2. Exemplarische Testumsetzung für die Bibliothek

Für die Bibliothek wurden exemplarisch Unit-Tests umgesetzt. Es empfielt sich ein zweites Projekt anzulegen um die eigentliche Anwendung, hier die Bibliothek, nicht unnötig mit Testcode zu vergrößern [7]. Der Test wird auf dem Gerät oder dem Emulator durchgeführt. Bei Ausführung des Tests wird automatisch die zu testende Anwendung und anschließend der Unittest selbst installiert.

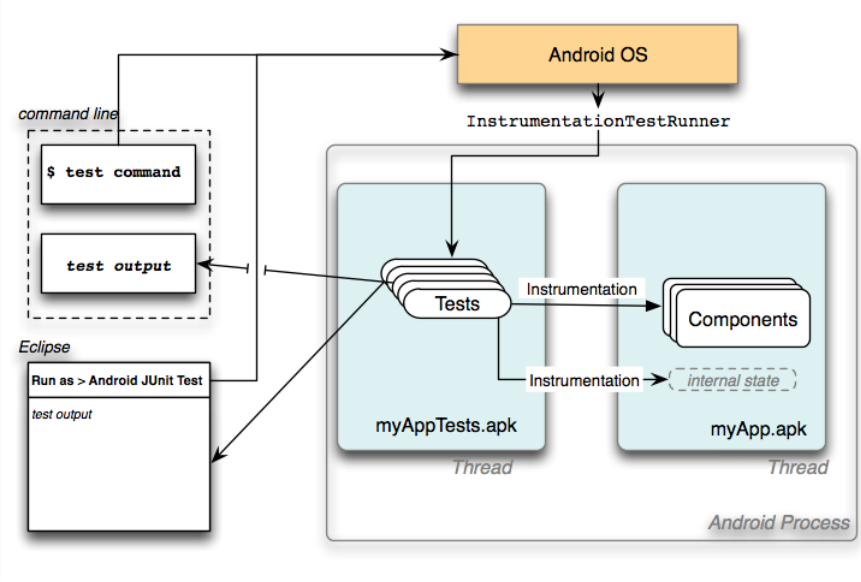


Abbildung 6.2.: Android JUnit Framework (Quelle: [21])

Von der Bibliothek können folgende Funktionen mit Unit-Tests überprüft werden werden:

1. Die Funktion der Verschlüsselungsalgorithmen AES und RSA
2. Die Erzeugung eines AES-Schlüssels und eines RSA-Schlüsselpaares
3. Korrektheit des RSA-Schlüsselpaares (über Verschlüsselungstest, siehe erstens)

Für die Tests werden zwei unterschiedliche AES-Schlüssel und zwei unterschiedliche RSA-Schlüsselpaare erzeugt. Mit diesen Schlüsseln kann nun getestet werden ob die Verschlüsselung und die dazu gehörende Entschlüsselung funktioniert, beziehungsweise durch die Eingabe eines falsche Schlüssels fehlschlägt. Zusätzlich müssen noch falsche oder fehlende Eingaben geprüft werden, ob mit der erwarteten Fehlermeldung reagiert wird.

Die Korrektheit der Bibliothek ist damit allerdings noch nicht bewiesen. Die Unit-Tests sagen nur aus, dass die Methoden für die eingegebenen Werte in der erwarteten Weise funktionieren. Es ist nicht möglich alle möglichen Eingaben zu testen.

Ergänzen um Test nach Anforderungen.

7. Fazit und Ausblick

In diesem Kapitel sollen die Ergebnisse dieser Arbeit zusammengefasst und bewertet werden. In Abschnitt 7.2 sollen kurz Themen vorgestellt werden, in welche Richtung die Arbeit weiter verfolgt werden könnte, die aber in dieser Arbeit nicht behandelt wurden.

7.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde untersucht, was im Bereich der Verschlüsselung von textbasierter Kommunikation auf mobilen Geräten möglich ist und wo Probleme auftreten. Exemplarisch wurde dafür ein Prototyp entworfen, der es ermöglicht SMS-Nachrichten zu verschlüsseln und anschließend zu senden, sowie deren Empfang und Entschlüsselung.

In Kapitel 2 wurden dazu die Grundlagen erläutert. Zunächst wurde die Android Plattform vorgestellt, anschließend die beiden in der Arbeit verwendeten Kommunikationsverfahren E-Mail und SMS-Nachricht. Dabei wurde festgestellt, dass beide Nachrichtentypen ihren Inhalt im Klartext übermitteln. Auch wurde festgehalten, dass sowohl E-Mail, als auch SMS-Nachrichten in einem für sie speziellen Zeichensatz kodiert werden. Der Body der E-Mail darf nur 7-Bit-ASCII Zeichen enthalten, der SMS-Body nur den GSM 03.38 Zeichensatz. Dessen Bedeutung wird in den Kapiteln 3 und 4 wieder aufgegriffen. Auf diesen Kenntnissen aufbauend wurden mögliche Angriffstechniken erläutert, die es Dritten ermöglichen unverschlüsselte Nachrichten auszuspähen. In der Bedrohungsanalyse (Kapitel 3.3) wurden diese Techniken wieder aufgegriffen und Analysiert. Im letzten Abschnitt des 2. Kapitels wurden kryptographische Verfahren, symmetrisch und asymmetrisch, erläutert. Zusätzlich wurden die Verfahren AES und RSA, welche im Prototypen und der Bibliothek zum Einsatz kommen, eingeführt.

In Kapitel 4 wurden die in Kapitel 3 aufgestellten Anforderungen in Architektur, einem Prototypen, sowie einer Bibliothek umgesetzt. Es wurde gezeigt, was machbar ist und welche Probleme es gibt. Ein Problem des Prototypen und der Bibliothek ist der SMS-Zeichensatz. Dem Android SMS-Manager muss ein Java-String übergeben werden. So ist es hier nicht möglich gewsen die Nachricht direkt in den SMS-Zeichensatz zu übersetzen, sondern es musste der Umweg über eine Hex-Kodierung genommen werden. In Kapitel 5 und 5.2 wurde auf die besondere Bedeutung das Exception-Handling hingewiesen. In Bezug auf die

Kompatibilität zu „alten“ Programmen wurde gezeigt, dass eine Fehlerbehandlung und das Werfen von Exceptions nur unzureichend umsetzbar ist.

Zusammenfassend wurde gezeigt, dass eine „einfache“ Art der Verschlüsselung gerade in Bezug auf SMS-Nachrichten sehr gut umsetzbar ist. Aufgrund der Besonderheiten dieses Kommunikationstypes ist eine professionelle Lösung jedoch nicht trivial. Die Firma WHISPER SYSTEMS ist mit Ihrer Anwendung „TextSecure“ auf einem interessanten Weg, den man weiter verfolgen sollte. WHISPER SYSTEMS hat angekündigt den Quellcode Ihrer Software offenzulegen.

7.2. Vision und Ausblick

Die bisher vorliegende Bibliothek bietet einige Basisfunktionen an. Sie ist aber in der Art angelegt, dass man sie erweitern kann. Dabei geht es nicht nur darum, die Bibliothek um neue Funktionen und Protokolle zu erweitern, sondern auch bestehende Strukturen umzustellen oder zu ergänzen.

Bei der SMS-Verschlüsselung ist es momentan nur möglich Nachrichten mit einer Länge von 160 Zeichen zu verschicken. Dabei ist es normalerweise möglich auch längere Nachrichten zu verschicken. Bei den so genannten Multi-SMS (Concatenated SMS, Long SMS) werden längere Texte als einzelne SMS-Nachrichten verschickt und beim Empfänger wieder zusammengebaut. Allerdings müssen beide Seiten dieses Verfahren unterstützen. Ein weiteres Verfahren, welches nicht implementiert wurde ist das der Daten-SMS oder auch MMS-Nachricht (Multimedia Messaging Service). Beide Verfahren wären eine sinnvolle Ergänzung für die Bibliothek.

Um aus dem Prototypen eine vollständige Anwendung zu machen würde er noch den in 4.1.2 genannten SMS-Zwischenspeicher benötigen, da es sonst passieren könnte, dass eine SMS-Nachricht „verloren“ gehen könnte. Die Nachricht würde zwar vom Android eigenen SMS-Empfänger gespeichert, müsste dann aber von Hand in die Anwendung kopiert werden um sie zu entschlüsseln. Da der Empfänger auf alle eingehenden Nachrichten reagiert könnte man über eine Möglichkeit nachdenken, ob es möglich wäre verschlüsselte Nachrichten von unverschlüsselten Nachrichten zu unterscheiden. Sollte dieses möglich sein, würde nur dann eine Notification gesetzt werden, wenn eine verschlüsselte Nachricht eingegangen ist.

Ein weitere Bestandteil könnte die Echtzeitkommunikation sein. Instant Messaging, wie ICQ, MSN oder Google Talk sind fester Bestandteil der heutigen Kommunikation. Es wäre schön, hier eine Möglichkeit bieten zu können, dass die Nachrichten verschlüsselt übertragen werden. Denkbar wäre eine Kommunikation, die über ein vorher ausgetauschtes Passwort gesichert wird und dann einen synchronen Verschlüsselungsalgorithmus nutzt, aber auch eine asynchrone Verschlüsselung wäre denkbar. Letzteres wäre sogar die elegantere Lösung,

da nur öffentliche Schlüssel über eine geeignete Infrastruktur ausgetauscht werden müssten und man so immer sicher kommunizieren könnte ohne vorher Passwörter tauschen zu müssen.

Im Verlauf der Arbeit wurde mehrfach darauf hingewiesen, dass die verwendete Hex-Kodierung gerade für SMS-Nachrichten ein großes Problem ist, da der Klartext in der Länge etwa verdoppelt wird. Es wäre also im Sinne der späteren Anwender sich dieses Problemes anzunehmen und eine andere Kodierungsform zu finden. Sei es eine Übersetzung in einen anderen Zeichensatz oder durch eine geeignete Kompression der Nachricht. Das Thema Kompression wurde bereits in Kapitel 5.3 angesprochen und auch die Gründe dafür, warum sie nicht zum Einsatz gekommen ist.

Eine Idee, die während der Bearbeitung dieser Arbeit aufkam, war die Entschlüsselung der Nachricht mit einem Passwort oder (öffentlichen) Schlüssel mit einer weiteren Komponente zu verbinden oder zur Bedingung zu machen. Ein Beispiel wäre, dass die Nachricht nur an eine bestimmten GPS-Koordinate entschlüsselt werden darf. Damit könnten unterschiedliche Einsatzzwecke geliefert werden, wie zum Beispiel Spiele, bei denen der Spieler gewisse Koordinaten ablaufen muss und dort immer neue Aufgaben gestellt bekommt.

Eine Frage, die während der Bearbeitung dieser Arbeit offenblieb, ist: Wie weit darf eine Bibliothek in die Konfiguration eingreifen?“. Gemeint ist damit, macht es vielleicht Sinn einen eigenen Mailclient oder einen eigenen Dienst zur Versendung von SMS Nachrichten zu implementieren? Wird die Bibliothek zu umfangreich nimmt man dem späteren Nutzer eventuell Freiheiten seine Anwendung nach seinen Wünschen zu gestalten. Ist die Bibliothek zu „klein“ muss er eventuell zuviel selbst gestalten, so dass sich der Einsatz der Bibliothek nicht lohnt.

Ein anderer Punkt, der nicht behandelt wurde, ist die rechtliche Seite der kryptographischen Bestandteile der Bibliothek [13]. In einigen Ländern ist der Einsatz von starken kryptographischen Verfahren nicht gestattet oder unterliegt hohen Restriktionen. In den USA zum Beispiel fällt Kryptographie unter das Waffengesetz und damit unter strenge Exportkontrollen und in Frankreich gab es von 1990 bis 1996 ein Gesetz, dass jeden Nutzer von asymmetrischer Kryptographie verpflichtete seinen privaten Schlüssel bei einer „vertrauenswürdigen Behörde“ zu deponieren. In Deutschland gibt es bisher kein Kryptographiegesetz.

A. Anwendungsfälle

Senden und Empfangen von SMS und E-Mail Nachrichten

Nr 1	Senden einer verschlüsselten SMS/E-Mail mit Passworteingabe
Vorbedingung	Das Passwort muss dem Sender und dem Empfänger bekannt sein
Beschreibung	Der Benutzer wählt aus dem Telefonbuch den Empfänger aus. Anschließend wird die Nachricht im Klartext und das Passwort eingegeben. Zuletzt wird auf den Senden Button gedrückt, um die Nachricht abzuschicken.
Ausnahmen	E1: Wenn kein Netz vorhanden ist, soll dies dem Benutzer mitgeteilt werden. E2:(nur bei SMS Nachrichten) War die SMS Nachricht zu lang soll dies dem Benutzer mitgeteilt werden. E3: Wenn die Telefonnummer/E-Mail Adresse nicht vorhanden ist, soll eine Fehlermeldung angezeigt werden.
Nachbedingung	Sendebestätigung anzeigen. Bei einem Fehler sollten die Daten zwischengespeichert werden.
Nr 2	Empfangen einer verschlüsselten SMS/E-Mail mit Passworteingabe
Vorbedingung	Das Passwort muss dem Sender und dem Empfänger bekannt sein
Beschreibung	Der Benutzer bekommt einen Hinweis in der Statusleiste, dass eine neue Nachricht eingetroffen ist. Mit dem Finger wird die Statusleiste nach unten gezogen und auf die Nachricht getippt. Nun startet das Programm und die Telefonnummer des Absender und die verschlüsselte Nachricht werden angezeigt. Es wird jetzt das Passwort eingegeben und auf den Entschlüsseln Button getippt. Anschließend wird die entschlüsselte Nachricht angezeigt.
Ausnahmen	E1: War das Passwort falsch soll eine entsprechende Fehlermeldung angezeigt werden.
Nachbedingung	keine

Nr 3	Senden einer verschlüsselten SMS/E-Mail mit PublicKey
Vorbedingung	Das Passwort muss dem Sender und dem Empfänger bekannt sein
Beschreibung	Der Benutzer wählt aus dem Telefonbuch den Empfänger aus. Anschließend wird die Nachricht im Klartext eingegeben. Zuletzt wird auf den Senden Button gedrückt, um die Nachricht abzuschicken.
Ausnahmen	E1: Wenn kein Netz vorhanden ist, soll dies dem Benutzer mitgeteilt werden. E2: Es ist kein öffentlicher Schlüssel des Empfängers vorhanden, dieses sollte dem Benutzer mitgeteilt werden. E3: (nur bei SMS Nachrichten) War die SMS Nachricht zu lang, soll dies dem Benutzer mitgeteilt werden.
Nachbedingung	Sendebestätigung anzeigen. Bei einem Fehler sollten die Daten zwischengespeichert werden.
Nr 4	Empfangen einer verschlüsselten SMS/E-Mail mit PrivateKey
Vorbedingung	Der öffentliche Schlüssel muss ausgetauscht worden sein.
Beschreibung	Der Benutzer bekommt einen Hinweis in der Statusleiste, dass eine neue Nachricht eingetroffen ist. Mit dem Finger wird die Statusleiste nach unten gezogen und auf die Nachricht getippt. Nun startet das Programm und die Telefonnummer des Absenders und die verschlüsselte Nachricht werden angezeigt. Mit dem Antippen des Entschlüsseln Button wird der intern gespeicherte private Schlüssel aufgerufen und die Nachricht entschlüsselt. Anschließend wird die entschlüsselte Nachricht angezeigt.
Ausnahmen	E1: Fehler beim Entschlüsseln der Nachricht, soll dem Benutzer mitgeteilt werden.
Nachbedingung	keine
Nr 5	Empfangen einer verschlüsselten SMS/E-Mail mit PrivateKey
Vorbedingung	Der öffentliche Schlüssel muss ausgetauscht worden sein.
Beschreibung	Aufrufen des Menüpunktes .... Auf den Button Schlüsselpaar erzeugen tippen.
Ausnahmen	E1: Erzeugung fehlgeschlagen, sollte dem Benutzer mitgeteilt werden
Nachbedingung	Schlüsselpaar liegt auf der SD-Karte des Mobilgerätes
Nr 6	Schlüsselaustausch
Vorbedingung	Schlüsselpaar muss erzeugt worden sein.
Beschreibung	geeignete Art der Übertragung (E-Mail, SMS, KeyServer,....)
Ausnahmen	
Nachbedingung	Schlüssel ist ausgetauscht und ist im Notizfeld des entsprechenden Kontaktes gespeichert

B. Codeschnipsel aus der Bibliothek

In Anhang B sind einige Codeschnipsel aus der Bibliothek untergebracht, auf die in diser Arbeit eingegangen wird. Der vollständige Quellcode befindet sich auf der beiliegenden DVD.

Algorithm B.1 Konstruktor

```
public SML(String loggerPath, String encryption, byte[] rawKey, PublicKey pubKey) throws
NoSuchAlgorithmException{
  logger = new Logger(loggerPath);
  if (encryption == "AES" && rawKey != null){
    algorithmus = encryption; this.rawKey = rawKey;
  }else if(encryption == "AES" && pubKey != null){
    algorithmus = encryption; this.pubKey = pubKey;
  }else{
    throw new NoSuchAlgorithmException("NoSuchAlgorithm: "+ encryption);
  }
}
```

Algorithm B.2 Wrappermethode für sendTextMessage

```
public void sendTextMessage (String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent){
//key für AES oder RSA einlesen
 if(algorithmus == "AES"){
 try {
  SendingMessages.sendSms(destinationAddress, text, rawKey, null);
 }catch (Exception e) {
   //Es kann keine Exception für Kryptographie geworfen werden, da nur Exceptions des Android SMS Manager geworfen werden können. e.printStackTrace();
 }
 }else if(algorithmus == "RSA"){
 try {
  SendingMessages.sendSms(destinationAddress, text, null, pubKey);
 } catch (Exception e) {
   //Es kann keine Exception für Kryptographie geworfen werden, da nur Exceptions des Android SMS Manager geworfen werden können. e.printStackTrace();
 }
 }else{
          //keine Verschlüsselung SmsManager sms = SmsManager.getDefault(); sms.sendTextMessage(destinationAddress, scAddress, text, sentIntent, deliveryIntent);
 }
}
```

Algorithm B.3 Impliziter E-Mail Intent

```
/* Create the Intent */
final Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
/* Fill it with Data */
emailIntent.setType("plain/text");
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL,newString[]{receiver}); emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, encrypted);
/* Send it off to the Activity-Chooser */
Mailer mail = new Mailer();
mail.sendMail(emailIntent);
```

C. DVD

Inhalt der DVD

- Bachelorarbeit als PDF Dokument
- Prototyp aus dieser Arbeit
- Bibliothek aus dieser Arbeit
- Quellen, die mir als digitale Dokumente vorlagen (PDF, HTML, TXT)

Abbildungsverzeichnis

1.1. Plattform Versionen 2

2.1. Zeitlinie (Quelle: [43]) 4
2.2. Android Betriebssystem (Architektur) (Quelle: [21]) 5
2.3. Von *.java nach *.dex (Quelle: [7]) 7
2.4. Android 1.6 Homescreen 8
2.5. IMSI-Catcher (Quelle: [20]) 14
2.6. E-Mail 15
2.7. Wireshark Protokoll 16
2.8. Android Installationshinweise 17
2.9. Symmetrische Verschlüsselung 19
2.10. Caesar-Chiffre (Quelle: [57]) 19
2.11. Asymmetrische Verschlüsselung 21
2.12. Schlüsselexpansion 23

3.1. Bedrohungsbaum 28
3.2. Marktanteile von Betriebssystemen aus Mobiltelefonen (Quelle: [8]) 29
3.3. Number of Mobile Malware (Quelle: [16]) 30
3.4. Ausschnitt aus der PKS 2009 (Quelle: [11]) 31
3.5. Senden und Empfangen einer Nachricht 32
3.6. Schlüsselaustausch 33

4.1. Schichtarchitektur Sender 39
4.2. Ausschnitt Empfänger 41
4.3. Sender und Empfänger 42
4.4. Freihandskizze einer GUI 44
4.5. Android Menü 45
4.6. Notification 46
4.7. GUI-Graph 46
4.8. Schichtmodell der Anwendung 47
4.9. Klassendiagramm der Anwendung 48
4.10. SecureSMS GUI 49
4.11. Laufzeitdiagramm Empfänger 50

4.12. Laufzeitdiagramm Sender 51

5.1. Wrapper 54
5.2. Package Dependency 54
5.3. Objektmodell der Bibliothek (Package de.markus.bachelor.lib) 55
5.4. Bibliothek Datenfluss Verschlüsselung 57
5.5. Bibliothek Datenfluss Entschlüsselung 58

6.1. Test durch inkrementelle Integration 62
6.2. Android JUnit Framework (Quelle: [21]) 64

Literaturverzeichnis

[1] E. Wilde (Internet Engineering Task Force (IETF)). Uri scheme for global system for mobile communications (gsm) short message service (sms), Januar 2010.

[2] 3GPP. Short message service (3gpp2 c.s0015-0), 1999.

[3] 3GPP. Technical realization of the short message service, März 2007. 3GPP TS 23.040 V7.0.1.

[4] 3GPP. Alphabets and language-specific information, September 2009. 3GPP TS 23.038 V9.0.0.

[5] AndroidZoom. Androidzoom. Webseite, 2010. URL http://www.androidzoom.com/. Abgerufen am: 30.05.2010 um 15:30Uhr.

[6] AndroLib. Androlib. URL http://www.androlib.com/.

[7] Marcus Pant Arno Becker. *Android Grundlagen und Programmierung*. dpunkt, 2009.

[8] Christy Pettey Ben Tudor. Gartner says worldwide mobile phone sales grew 17 per cent in first quarter 2010, Mai 2010. URL http://www.gartner.com/it/page.jsp?id=1372013. Abgerufen am: 12.07.2010 um 18:00 Uhr.

[9] Bitkom. Presseinformation internet-kriminelle weiten aktivitäten aus, 2009. URL http://www.bitkom.org/files/documents/BITKOM_BKA_Presseinfo_IT-Kriminalitaet_08_10_2009.pdf.

[10] Dan Bornstein. Dalvik virtual machine. Webseite, 2008. URL http://www.dalvikvm.com/. Brief overview of the Dalvik virtual machine and its insights. (Abgerufen am: 19.07.2010 um 12 Uhr).

[11] Bundesministerium des Inneren. Polizeiliche kriminalstatistik 2009, 2009.

[12] Claudia Eckert. *IT-Sicherheit*. Oldenbourg Wissensch. Vlg, 2008.

[13] Jörg Ertl (Hochschule Esslingen). Weltweiter Überblick zur rechtlichen und politischen situation der kryptographie. Webseite, 2010. URL http://www.it.fht-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/ws9798/html/krypt_recht_pol/krypt_recht_pol-2.html. Abgerufen am: 21.07.2010 um 10 Uhr.

[14] F-Secure. Worm:symbos/commwarrior, 2005. URL http://www.f-secure.com/v-descs/commwarrior.shtml.

[15] F-Secure. F-secure zieht bilanz der virenentwicklung im 1. halbjahr 2006, Juni 2006. URL http://www.f-secure.com/de_DE/about-us/pressroom/news/2006/fs_news_20060627_16.

[16] F-Secure. Number and types of mobile malware, 2010. Grafiken auf Anfrage bei F-Secure per E-Mail von Marvin Jackson (Pre-Sales Engineer) erhalten.

[17] David Flanagan. *Java in a Nutshell*. O'Reilly, 2005.

[18] Free Software Foundation. Gnu general public license, version 2, Juni 1991. URL http://www.gnu.org/licenses/gpl-2.0.html.

[19] The Apache Software Foundation. Apache 2.0 lizenz, Januar 2003. URL http://www.apache.org/licenses/LICENSE-2.0.html.

[20] Dirk Fox. Der imsi-catcher, 2002. Datenschutz und Datensicherheit 26 (2002) 4.

[21] Google Inc. Android developers. Webseite, Mai 2010. URL http://developer.android.com.

[22] Google Inc. Android referenz sms managaer. Webseite, 2010. URL http://developer.android.com/reference/android/telephony/SmsManager.html. Abgerufen am: 11.07.2010 um 12:30 Uhr abgerufen.

[23] GsmFavorites. Sms packet format. Webseite, Mai 2010. URL http://www.gsmfavorites.com/documents/sms/packetformat/. Abgerufen am: 15.05.2010 um 11:00 Uhr.

[24] heise. Android-software zur verschlüsselung von telefonaten und sms, Mai 2010. URL http://www.heise.de/ct/meldung/Android-Software-zur-Verschluesselung-von-Telefonaten-und-SMS-10094html. Abgerufen am: 20.07.2010 um 17:15 Uhr.

[25] heise Security. iphone-leck weitet sich aus [update]. Webseite, Mai 2010. heise Security Meldung vom 31.05.2010; Abgerufen am: 01.06.2010 um 16:30Uhr.

[26] Freidhel Hillebrand. *GSM and UMTS: The Creation of Global Mobile Communication*. Wiley, 2001.

[27] David Hook. *Beginning Cryptography with Java*. wrox, 2005.

[28] Vincent Rijmen Joan Daemen. Aes proposal: Rijndael, März 2009. Version 2.

[29] Christian Bettstetter Christian Hartmann Jörg Eberspächer, Hans-Jörg Vögel. *GSM Architecture, Protocols and Services*. Wiley, 2001.

[30] Sascha Krissler Karsten Nohl. *iX Ausgabe 5 (Mai 2010)*. Heise, 2010. Artikel: Geheimnislos; Verschlüsselung von Handy-Gesprächen knacken (S. 97 - 99).

[31] Markus Kuhn Ken Whistler, Kent Karlsson. Gsm 03.38 to unicode, November 2009. Übersetzung von GSM 03.38 nach Unicode.

[32] Jonathan Knudsen. *Java Cryptograühy*. O'Reilly, 1998.

[33] Ulrike Kuhlman. Die schwachstelle vor der tastatur. *heise.de*, 1: 1, 2010. URL `http://www.heise.de/newsticker/meldung/Die-Schwachstelle-vor-der-Tastatur-1016121.html`. heise Artikel Abgerufen am: 06.06.2010 um 12 Uhr.

[34] Hajo Köppen. Der computerkriminelle: Griff in die tasten. Zeitschrift Computer und Arbeit, April 2007. Fachhochschule Gießen-Friedberg.

[35] Belcher GI Lda. Cryptogsm. Webseite, Mai 2010. URL `http://www.cryptogsm.co.uk/`. Abgerufen am: 15.05.2010 um 11:00 Uhr.

[36] Norbert Pohlmann Markus a Campo. Computerkriminalität fakten und zahlen, Dezember 2002. URL `http://www.internet-sicherheit.de/de/forschung/publikationen/buecher-als-pdfs/virtual-private-networks/dokument-detail/virtual-private-networks-309-318/`.

[37] Matthew Miller. Google android smacks down windows mobile in latest gartner data. Webseite, Mai 2010. URL `http://www.zdnet.com/blog/cell-phones/google-android-smacks-down-windows-mobile-in-latest-gartner-data/3829`. Abgerufen am: 12.07.2010.

[38] Ed. (Qualcomm Incorporated) P. Resnick. (rfc 5322) internet message format, Oktober 2008.

[39] Gerhard Pews. Vorlesung TU Darmstadt im WS 2009/2010: Software-Engineering in der industriellen Praxis, Oktober 2009.

[40] A. Shamir R.L. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, 1977.

[41] James & Suzanne Robertson. Requirements specification template. Paper, März 2010. URL `http://www.volere.co.uk`. Auflage 15.

[42] Rohde & Schwarz International. Rohde & schwarz international. Webseite, Mai 2010. URL `http://www2.rohde-schwarz.com/en/products/secure_communications/voice_and_data_encryption/TopSec_GSM.html`. Abgerufen am: 10.05.2010 um 15:30 Uhr.

[43] Dave MacLean Sayed Y. Hashimi, Satya Komatineni. *Pro Android 2*. Apress, 2010.

[44] Klaus Schmeh. *Kryptografie*. dpunkt, 2007.

[45] Johannes Siedersleben. *Moderne Software-Architektur*. dpunkt, 2004.

[46] Ian Sommerville. *Software Engineering*. Pearson Studium, 2001.

[47] Dr. Gernot Starke. *Effektive Software-Architekturen*. Carl Hanser Verlag, 2008.

[48] Daehyun Strobel. Imsi catcher. Seminararbeit (Ruhr-Universitšat Bochum), 2007.

[49] James Robertson Suzanne Robertson. *Mastering the Requirements Process*. Addison-Wesley, 2006.

[50] WHISPER SYSTEMS. Textsecure and faq, 2010. URL `http://www.whispersys.com/support.html#6`. Abgerufen am: 20.07.2010 um 17 Uhr.

[51] Whisper Systems. Whisper systems. Webseite, 2010. URL `http://whispersys.com/index.html`. Abgerufen am: 30.05.201 um 15:45 Uhr.

[52] Greg Travis. Build your own java library. IBM Webseite (PDF), Mai 2001. URL `ibm.com/developerWorks`.

[53] FINN TROSBY. Sms, the strange duckling of gsm. Telektronikk 3.2004, 2004.

[54] INTERNATIONAL TELECOMMUNICATION UNION. Introduction to ccitt signalling system no. 7, März 1993. URL `http://www.itu.int/rec/T-REC-Q.700-199303-I/en`. Recommendation Q.700 (03/93).

[55] Gerd Kramarz von Kohout. Imsi-catcher: Gsm unsicherheit in der praxis, 2001. URL `http://ftp.ccc.de/congress/2001/mp3/vortraege/tag2/saal2/28-s2-1300-IMSI-Catcher.mp3`. Vortrag beim CCC Kongress 2001 (Tonmitschnitt (MP3 Datei)) Gerd Kramarz von Kohout arbeitet bei der DeTeMobil in der Abteilung technisches Sicherheitsmanagement In diesem Vortrag wird über den Einsatz von Geräten gehen, die in der Patentschrift DE 197 49 388 C 2 (Patentinhaber: Siemens AG, 1997) sowie in der Offenlegungsschrift DE 199 20 222 A 1 (Anmelder: Rohde & Schwarz, 1999) beschrieben sind. Im Detail geht es um - technischen Wirkungsweise & protokollbedingte Grundlagen - Auswirkungen auf den Netzbetrieb und eingebuchte Teilnehmer der Einsatzregion - juristische Grundlagen (Frequenzverordnung, polizeilicher Notstand, Terrorgesetz) - Erfahrungen aus der Sicht des Netzbetriebs MP3 Datei am 21.05.2010 herunter geladen.

[56] Martin E. Hellman Whitfield Diffie. New directions in cryptography. IEEE Transactions on Information Theory, 22(6):644-654„ November 1976.

[57] Wikipedia. Caesar3.svg. Webseite, August 2010. URL `http://de.wikipedia.org/wiki/Datei:Caesar3.svg`. Abgerufen am: 02.08.2010 um 15 Uhr.

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 16.08.2010 Markus Schüring