Höhere Technische Bundeslehranstalt Wien 3, Rennweg
IT & Mechatronik

HTL Rennweg :: Rennweg 89b
A-1030 Wien :: Tel +43 1 24215-10 :: Fax DW 18

Diplomarbeit

Raven

ausgeführt an der
Höheren Abteilung für Informationstechnologie Netzwerktechnik
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2014/2015

durch

Philipp ADAM
Manuel CASPARI
Nicolas LUKASCHEK

unter der Anleitung von

DI Dr. Ferdinand KASPER
DI August HÖRANDL
DI Herbert SASSHOFER

Wien, 12. Mai 2015

Kurzfassung

Die Idee ist, eine reine End-zu-End Verschlüsselung für verschiedene Dienste wie Facebook, SMS und E-Mail zu entwickeln. Hierbei werden die Schlüssel nur auf den zwei betroffenen Geräten gespeichert, die anschließend direkt miteinander kommunizieren können.

Hervorzuheben ist, dass kein neuer Dienst geschaffen, sondern auf bereits vorhandenen Strukturen aufgebaut werden soll. Das Ziel ist es, eine Software für Android und PC zu entwickeln, welche eine verschlüsselte Kommunikation über SMS und E-Mail anbieten. Für die PC-Version ist die Kommunikation über SMS, für beide der Nachrichtenaustausch über Facebook ein optionales Ziel.

Aufgrund der Intuitivität des Software-Interfaces ist ein User-Manual nicht vonnöten.

Abstract

The idea is the development of an end-to-end encryption software for several services like Facebook, SMS and email.The keys, which are used for the encryption, are stored only on the two devices, which communicate directly with each other.

It should be emphasized that instead of creating a new service, existing systems and structures shall be used. The goal is the development of an application for Android and PC, which must be able to communicate encrypted over SMS and email.. For the PC version communication over SMS is optional. Both have the optional goal to communicate over Facebook.

Due to the intuitiveness of the software interface a user manual is not needed.

Ehrenwörtliche Erklärung

Ich versichere,

- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am 12. Mai 2015

_____

Philipp Adam

_____

Manuel Caspari

_____

Nicolas Lukaschek

Präambel

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für „Ingenieurprojekte“ gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den berufsbildenden höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer:

- **Prof. DI Dr. Ferdinand Kasper** Hauptbetreuer
- **Prof. DI August Hörandl** Hauptbetreuer Stellvertreter
- **Prof. DI Herbert Sasshofer** Nebenbetreuer

Vorwort

Dieses Buch ist das Endergebnis unserer Diplomarbeit Raven, ausgeführt an der HTL 3 Rennweg im Schuljahr 2014/15. Im Nachfolgenden werden Sie eine Beschreibung und Dokumentation unserer Arbeit vorfinden, angefangen von der Planung bis zum fertigen Produkt. Wir haben beim Schreiben dieses Buches besonderen Wert auf Nachvollziehbarkeit und Genauigkeit gelegt.

Zudem liegt dem Bibliotheksexemplar eine CD bei, welche die Planungsdokumente, Konzepte und den Sourcecode dieses Projekts enthält.

Wir danken unseren Betreuern für ihre Unterstützung.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Hintergrund . . . 1
1.2. Idee . . . 2
1.3. Projektname . . . 3
1.4. Zielgruppe . . . 4
1.5. Motivation . . . 4
1.5.1. Philipp Adam . . . 4
1.5.2. Manuel Caspari . . . 5
1.5.3. Nicolas Lukaschek . . . 5
1.6. Erklärung zum Gendern . . . 6

2. Planung **7**
2.1. Zielsetzung . . . 7
2.1.1. Primäre Ziele . . . 7
2.1.2. Optionale Ziele . . . 8
2.1.3. Nicht Ziele . . . 8
2.2. Organisation . . . 9
2.2.1. Organigramm . . . 9
2.2.2. Verantwortlichkeiten . . . 10
2.3. Managementmethodik . . . 12
2.4. Umweltanalyse . . . 13
2.5. Risikoanalyse . . . 14
2.5.1. Grafische Darstellung . . . 14
2.5.2. Maßnahmenkatalog . . . 15

3. Realisierung **16**
3.1. Allgemeines . . . 16
3.1.1. E-Mail . . . 16
3.1.2. SMS . . . 17
3.1.3. Facebook-Nachrichten . . . 18
3.1.4. Verwendete Libraries . . . 24
3.1.5. Core Library . . . 26
3.2. Desktopanwendung . . . 51
3.2.1. Darstellung des GUI . . . 51
3.2.2. Aufbau der Anwendung . . . 59

3.3. Androidanwendung 61
3.3.1. Allgemein 61
3.3.2. Oberfläche und Design 61
3.3.3. Umsetzung 62
3.3.4. Android Beam 68

4. Ergebnisse **69**
4.1. Desktopanwendung 69
4.1.1. GUI 69
4.1.2. Funktionen 69
4.2. Androidanwendung 70
4.3. Lizenzierung 72

5. Evaluation und Resümee **73**
5.1. Resümee 73
5.2. Planung vs. Realisierung 75
5.3. Persönliche Eindrücke, Lessons Learned 76
5.3.1. Philipp Adam 76
5.3.2. Nicolas Lukaschek 76
5.3.3. Manuel Caspari 77

6. Ausblick **79**

A. Anhang **80**

B. Quellenverzeichnis **83**

Tabellenverzeichnis

2.1. Maßnahmenkatalog 15

3.1. Verwendete Libraries 24
3.2. Vereinfachter `KeySafe` 49

A.1. Autorenvermerk 82

Abbildungsverzeichnis

1.1. Altes Logo . . . 3
1.2. Neues Logo . . . 3

2.1. Projektorganigramm . . . 9
2.2. Umweltanalyse . . . 13
2.3. Grafische Darstellung der Risiken . . . 14

3.1. Facebook Message Send Request . . . 21
3.2. HTTP Pull . . . 22
3.3. `KeyEntity`-Aufbau . . . 29
3.4. Workflow der Verschlüsselung . . . 32
3.5. Schlüssel Austausch . . . 35
3.6. Primitiv-Wurzel zu 7 . . . 36
3.7. Austausch mittels Diffie-Hellman . . . 40
3.8. `ExchangeInformation`-Protokoll Felder . . . 42
3.9. Workflow QR Code . . . 45
3.10. Workflow NFC-Austausch . . . 47
3.11. Startmenü . . . 51
3.12. Hauptmenü . . . 52
3.13. Profilerstellung . . . 53
3.14. Im Hauptmenü eingeloggt . . . 54
3.15. Schlüsselaustauschmethoden . . . 55
3.16. Schlüsselaustauschmethode „Manual“ . . . 55
3.17. Schlüsselaustauschmethode „Automatic“ . . . 56
3.18. Schlüsselaustauschmethode „QR-Key Exchange“ . . . 57
3.19. E-Mail verfassen . . . 58
3.20. Anmeldebildschirm . . . 62
3.21. SMS-Ansicht . . . 63
3.22. NavigationDrawer . . . 64
3.23. Nachrichten-Ansicht . . . 66

A.1. Plakat für den TOFT . . . 81

Listings

3.1. Nachrichten verschlüsseln 34
3.2. Nachrichten entschlüsseln 34
3.3. Diffie-Hellman Austausch 39

4.1. Code-Präambel 72

1. Einleitung

1.1. Hintergrund

Die Hintergrundgeschichte, die uns auf diese Idee brachte, war eine relativ einfache und noch immer in den Medien heiß diskutierte. Zunächst war ein Antrieb der aufgedeckten NSA-Spionageskandal. Dieser machte der gesamten Menschheit bewusst, dass kein digitaler Schritt unbemerkt bleibt und von NSA und Konsorten ausgewertet wird. Zudem war ebenfalls die Monopolstellung durch Facebook, insbesondere nach dem Kauf von WhatsApp, ein Hauptgrund. Es gilt als allgemein bekannt, dass Facebook & Co mit der NSA kooperieren, obgleich sie diese Tatsache vehement abstreiten.

Es mag zwar bereits diverse Verschlüsselungsmethoden für Onlinekommunikation und Datenübertragungen geben, allerdings haben diese oftmals sogenannte Backdoors („Hintertüren“) eingebaut. Dadurch ist es Geheimdiensten möglich, die Verschlüsselung mit relativ geringem Aufwand zu brechen, wodurch der eigentliche Sinn der Verschlüsselung ad absurdum geführt wird. Folglich ist es Spionagekonzernen möglich, den gesamten weltweiten Nachrichtenverkehr, von der Initiierung weg bis zur Terminierung, mitzulesen und in die Privatsphäre jedes Einzelnen einzugreifen.

Bereits vor 250 Jahren hat man sich Gedanken zu dieser Thematik gemacht:

“Those who surrender freedom for security will not have, nor do they deserve, either one.” [28]

1.2. Idee

Die Idee ist, eine echte End-zu-End Verschlüsselung für verschiedene Dienste, wie „Facebook“, SMS und E-Mail zu entwickeln. Hierbei werden die Schlüssel nur auf den zwei betroffenen Geräten gespeichert, die anschließend direkt miteinander kommunizieren können. Die Unterstützung eines dedizierten Servers wird dabei nicht benötigt.

Gerade in Zeiten von Datenschutz- und NSA-Skandalen werden die Rufe nach einer einfachen und sicheren Verschlüsselung immer lauter. Genau das möchten wir mit unserer Anwendung erreichen: eine einfach zu bedienende Software, mit der man über verschiedene Dienste sicher kommunizieren kann. Dies soll sowohl auf einem herkömmlichen PC, als auch auf der mobilen Plattform Android funktionieren. Unsere Anwendung dient hierbei als Multi-Messenger, welcher all diese Dienste vereint.

Bei der Verschlüsselung werden wir auf herkömmliche Methoden verzichten und mehrere, sichere Verfahren kombinieren, um die höchstmögliche Sicherheit gewährleisten zu können.

1.3. Projektname

Der Name des Projektes sollte einerseits die Flexibilität und einfache Bedienungsweise, andererseits auch den Sicherheitsaspekt des fertigen Produktes widerspiegeln. Die zündende Idee hatte schließlich Manuel Caspari, da in seiner Lieblingsserie „Game of Thrones“ Nachrichten nicht mittels Brieftaube, sondern mittels „Briefrabe“ übermittelt werden.

Der Rabe symbolisiert einerseits Mobilität, andererseits auch etwas Geheimnisvolles, ist also perfekt geeignet für eine mobile Verschlüsselungsanwendung. Das Logo stammt von Manuels Freund Michael Kroißmayer.

Abbildung 1.1.: Altes Logo

Abbildung 1.2.: Neues Logo

1.4. Zielgruppe

Primär möchten wir mit unserer Anwendung Privatpersonen ansprechen, die Wert auf ihre Privatsphäre im Internet legen. Seit dem Bekanntwerden der NSA-Spionageskandale ist das Bewusstsein für gesicherte Kommunikation über unsichere Medien stark gestiegen. An diesem Punkt möchten wir ansetzen und dem „Otto Normalverbraucher“ eine unkomplizierte Möglichkeit geben, seine privaten Nachrichten auch wirklich privat zu halten.

Unsere Anwendung ist nicht dafür konzipiert worden, streng vertrauliche Nachrichten von Entscheidungsträgern sicher zu übermitteln. Es ist nicht vorgesehen, einen langfristigen Produktsupport, Wiederherstellungsmöglichkeiten für invalide Nachrichten oder ein User-Manual anzubieten. Wir garantieren nicht, dass jede Nachricht immer fehlerfrei und lesbar entschlüsselt werden kann, daher ist unsere Anwendung bei wichtigen Nachrichten mit Vorsicht zu genießen.

1.5. Motivation

1.5.1. Philipp Adam

Meine Hauptmotivation für dieses Projekt rührt daher, dass mit „Raven“ eine Software von großem praktischen Nutzen geschaffen werden soll.

Mit diesem Projekt soll das Chatten und Mailen genauso einfach wie bisher gestaltet werden, allerdings soll der Inhalt für NSA & Co verborgen bleiben. Die Tatsache, dass ich mit meinem Team der breiten Masse eine solche Anwendung zugänglich machen kann, hebt meine Motivation für dieses Projekt ungemein.

Ein weiterer Grund meiner Motivation ist, dass ich mit meinen langjährigen Freunden Manuel Caspari und Nicolas Lukaschek, die ich seit der ersten Klasse kenne, zusammen-

arbeiten kann. Mir macht es Spaß, die in fünf Jahren erlernten Kenntnisse produktiv einzusetzen und in diese Diplomarbeit einfließen zu lassen.

1.5.2. Manuel Caspari

Ich war furchtbar entsetzt, als der große NSA-Überwachungsskandal publik wurde. Dabei wurde aufgezeigt, dass der gesamte weltweite Internetverkehr überwacht wird.

Ich wollte meine Fähigkeiten als Programmierer und Netzwerktechniker dazu einsetzten, meine privaten Nachrichten durch Verschlüsselung auch privat zu halten. Schnell war die Idee eines verschlüsselten Multi-Messengers geboren, welcher persönliche Nachrichten für Dritte (also die NSA) unlesbar machen sollte. Mit dieser Diplomarbeit möchte ich ein Tool zum Schutz der Privatsphäre kreieren und damit der totalen Internetüberwachung den Kampf ansagen.

1.5.3. Nicolas Lukaschek

Meine persönliche Motivation für unser Projekt „Raven“ besteht zu einem großen Teil aus dem Interesse, nach jahrelangem Lernen von theoretischen Grundlagen aus den Bereichen Programmierung und Netzwerksicherheit endlich etwas in die Praxis umsetzen zu können.

Ich wollte schon immer einen eigenen Messenger für die mobile Plattform Android schreiben. Dieses Projekt bietet mir die Möglichkeit, mich im Rahmen der Diplomarbeit der Programmierung einer App zu widmen. Leider kam dieses Kapitel im Regelunterricht nie zur Sprache.

Ich möchte das bereits erlangte Wissen mit den Erkenntnissen aus dieser Diplomarbeit verbinden und somit meinen Horizont im Bereich der Softwareentwicklung maßgeblich erweitern.

1.6. Erklärung zum Gendern

Im vorliegenden Dokument wird im Zusammenhang mit dem Benutzer der Anwendung durchgängig die männliche Form verwendet. Grund dafür ist der Erhalt des Leseflusses. Im Sinne des Gleichbehandlungsgesetzes sind diese Bezeichnungen als nicht geschlechtsspezifisch zu betrachten.

2. Planung

2.1. Zielsetzung

2.1.1. Primäre Ziele

2.1.1.1. Androidanwendung

Gefordert wird eine Androidanwendung, welche Schlüssel generieren und lesen kann und verschlüsselte Nachrichten sendet und empfängt. Der Schlüssel soll durch das Verhashen von Zufallszahlen gebildet werden. Die Hashfunktion bekommt als Input vom System bereitgestellte Zufallszahlen (/dev/random). Der Schlüsselaustausch zwischen Android-Geräten muss sowohl mittels QR-Code (Quick Response Code), als auch NFC (Near Field Communication) möglich sein.

Beim Lesen des Schlüssels soll die Telefonnummer des Kommunikationspartners mit dem gelesenen Schlüssel in Verbindung gebracht werden. Sendet A nun beispielsweise eine SMS an B, so ist diese mittels zweier Verschlüsselungsalgorithmen zu verschlüsseln und an B zu übermitteln. B soll die SMS mit demselben Schlüssel entschlüsseln. Es findet also eine symmetrische Verschlüsselung statt.

Die Anwendung muss zur Fertigstellung auf Deutsch und Englisch verfügbar sein.

2.1.1.2. Desktopanwendung

Gefordert wird eine Desktopanwendung, welche Schlüssel generieren und lesen kann und mit diesen Nachrichten über das Internet (via IP) verschlüsselt sendet und empfängt.

Mit der Anwendung soll eine textbasierte, verschlüsselte Kommunikation über E-Mail möglich sein. Der Schlüsselaustausch zwischen den Kommunikationspartnern soll dabei im Voraus (pre-shared) erfolgen, oder es findet ein asymmetrischer Schlüsselaustausch (mittels Diffie-Hellman) statt. Der Schlüssel muss dabei ebenfalls zufällig erzeugt werden.

Mithilfe der Java Klasse `SecureRandom` werden sogenannte „cryptographically strong random numbers" generiert, welche als Basis für die Schlüsselerzeugung herangezogen werden sollen.

Funktional sollen Desktopanwendung und Android-App gleichwertig sein, wobei die Desktopanwendung natürlich weder SMS versenden, noch QR-Codes lesen können muss.

2.1.2. Optionale Ziele

Sowohl Androidanwendung, als auch Desktopanwendung können über Facebook verschlüsselte Nachrichten austauschen.

2.1.3. Nicht Ziele

Es wird ein User-Manual geschaffen, welches dem User grundlegende Anwendungsfunktionen erklärt. Hauptgrund dafür sind die Einfachheit und Intuitivität des Software Interfaces, welche zusätzliche Erklärungen überflüssig machen sollen.

Zusätzlich wird ein langfristiger Produktsupport für beide Anwendungen angeboten.

2.2. Organisation

2.2.1. Organigramm

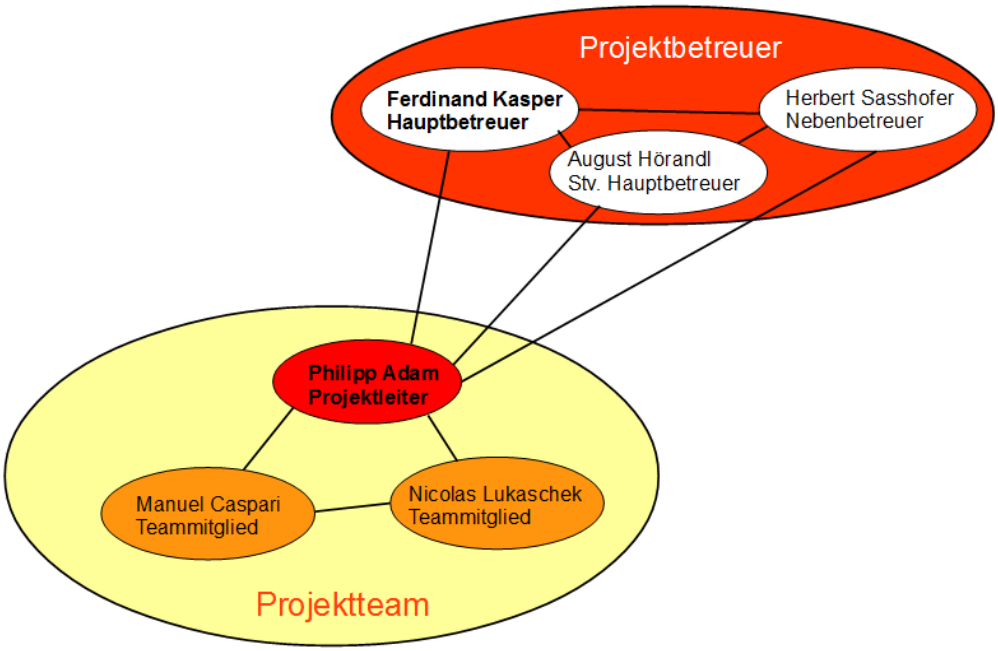

Abbildung 2.1.: Projektorganigramm

Das Organigramm zeigt Außenstehenden auf einen Blick die Organisationsstruktur innerhalb des Projekts auf. Die zwei wesentlichen Personengruppen im Projekt waren dabei das Projektteam und das Betreuerteam. Anzumerken ist allerdings, dass die Projektteammitglieder dabei weitgehend autonom gearbeitet haben und die Betreuer nur bei Problemfällen und Statusupdates hinzugezogen wurden.

2.2.2. Verantwortlichkeiten

2.2.2.1. Philipp Adam

Der Hauptfokus meines technischen Beitrages hat in der Entwicklung der Desktopanwendung gelegen. Ich habe die grafische Oberfläche entworfen, umgesetzt und die dahinterstehende Programmlogik implementiert. Weiters habe ich den Schlüsselaustausch zwischen Desktopanwendung und Androidanwendung mitentwickelt.

Zudem habe ich die Position des Projektleiters ausgeübt. Daher hat die Planung und Dokumentation des Projektes zusätzlich meine Arbeitszeit in Anspruch genommen. Daher habe ich zusätzlich den primären Ansprechpartner für Betreuer und externe Personen dargestellt.

2.2.2.2. Manuel Caspari

Mein Aufgabenbereich als stellvertretender Projektleiter hat einerseits Teile der Planung, als auch ein Großteil der eigentlichen Programmierung der Software umfasst. Hauptsächlich habe ich die Androidanwendung und die Schlüsselaustausch-Protokolle entwickelt.. Darunter sind der Schlüsselaustausch mittels Near Field Communication (NFC) oder Diffie-Hellman und das Handshake-Konzept gefallen. Zusätzlich habe die Programmteile für das Erzeugung der Schlüsselpaare, die Ver- und Entschlüsselung der Nachrichten, sowie Schnittstellen für die verschiedenen APIs geschreiben.

Zudem habe ich mich um die Verwaltung und Instandhaltung des Git-Servers, welcher für das Merging des Codes verwendet wurde, gekümmert.

2.2.2.3. Nicolas Lukaschek

Ich, habe mich in dieser Diplomarbeit als Mitarbeiter, der an beiden Fronten bestmöglich mitgeholfen hat gesehen. Sowohl beim Programmieren, als auch beim Projektmanagement selbst habe ich mich eingebracht. Meine geplante Ressourcenverteilung hat sich auf 50 Prozent für jede Seite belaufen, wobei bei Bedarf und Notwendigkeit ebenfalls eine kurzzeitige Änderung des Verhältnisses vorgenommen werden konnte.

Neben diversen Dokumenten im Bereich des Projektmanagements habe ich im Bereich des Programmierens hauptsächlich das GUI (Deutsch/ Englisch, Anzeigen der entschlüsselten SMS) erarbeitet.

2.3. Managementmethodik

Dieses Projekt wurde in einer „light“-Version von SCRUM verwaltet. Unter „light“ ist zu verstehen, dass, ähnlich wie in SCRUM, Sprints von definierter Dauer (meist ein Monat) mit einem definiertem Ziel angelegt wurden. Allerdings wurde dabei bewusst auf das „Daily SCRUM Meeting“ verzichtet, sowie der „Sprint Review“ erheblich eingeschränkt, da diese Instrumente des SCRUM aus Sicht des dreiköpfigen Teams schlichtweg zu viel des Guten gewesen wären. Dazu kommt noch, dass dieses Projekt keinen Auftraggeber (Product Owner) im klassischen Sinne hatte und das Team den „Product Backlog“ selbst priorisieren musste.

Zudem mussten für dieses Projekt die Diplomarbeitsrichtlinien der HTL Rennweg eingehalten werden. Diese beinhalteten unter anderem das Erstellen von typischen Wasserfallplänen, wie PSP oder Meilensteinplan. Dadurch wurde das Erstellen dieser Pläne zur Pflichtübung, aus der das Projektteam nur wenig praktischen Nutzen gezogen hatte. Aus der Sicht von SCRUM führen statischen Planungsmethoden die eigentliche Agilität von SCRUM ad absurdum.

Weiters wurde im Abstand von zwei Wochen ein Meeting mit den Projektbetreuern angesetzt. Dabei hat dieses Meeting entweder das Ende eines Sprints und den Anfang eines neuen markiert oder es diente zur Kommunikation des Sprintzwischenstandes mit den Projektbetreuern. Gegengleich zu den Meetings wurde alle zwei Wochen eine Managementsummary an die Betreuer geschickt, in der kurz der aktuelle Projektstatus sowie die „Next Steps“ gelistet waren.

Die Koordination innerhalb des Projektteams erfolgte mittels einer Facebook- und Telegram-Gruppe. Zudem wurde Redmine für das Erstellen und Tracken einzelner Tasks verwendet.

2.4. Umweltanalyse

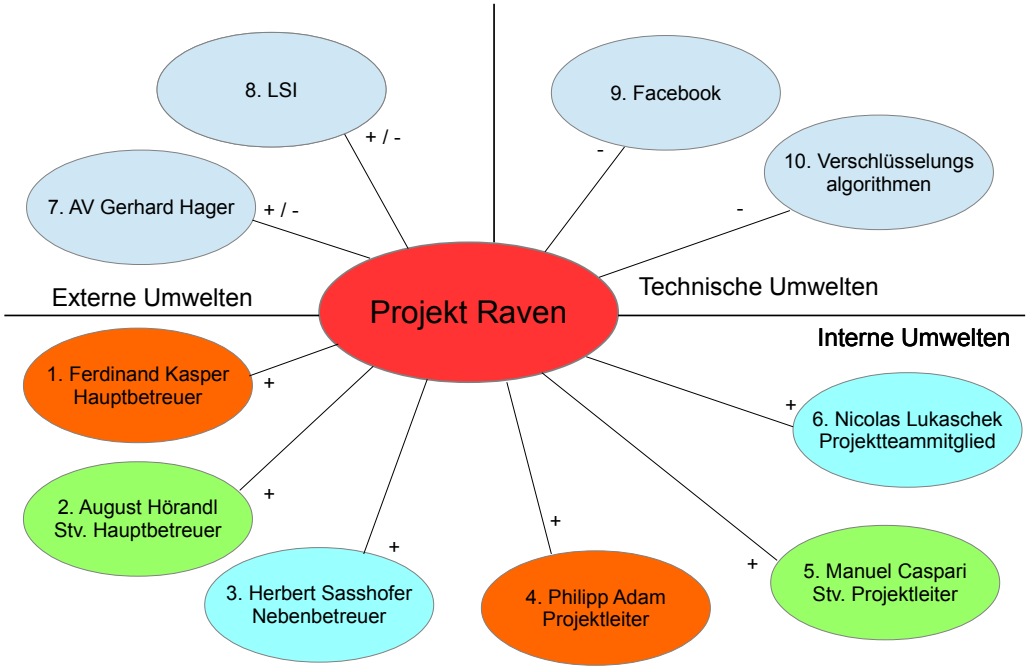

Abbildung 2.2.: Umweltanalyse

Die projektbeeinflussenden Faktoren gliedern sich in drei wesentliche Gruppierungen. Interne, externe und technische Umwelten müssen in der nachfolgenden Risikoplanung bedacht werden. Hierbei ist besonderes Augenmerk auf den technischen Aspekt gelegt worden, da dieser die Realisierung und damit den Erfolg des Projektes am ehesten gefährden kann.

2.5. Risikoanalyse

2.5.1. Grafische Darstellung

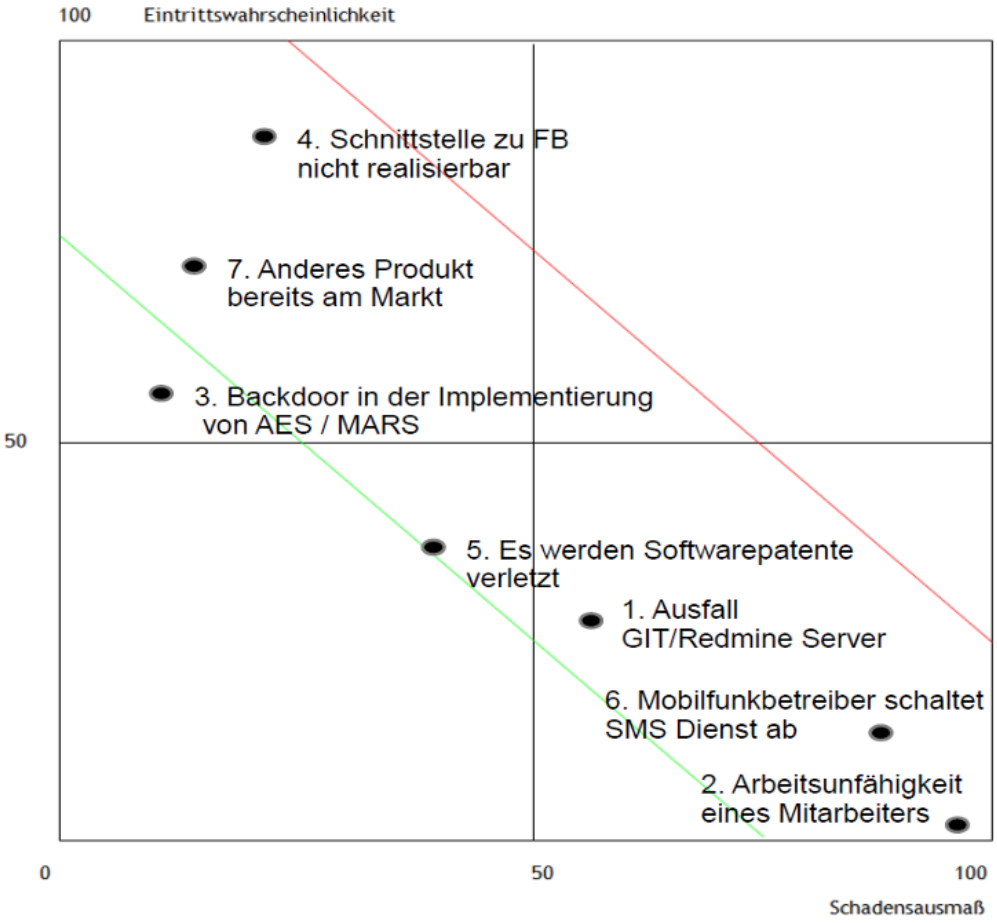

Abbildung 2.3.: Grafische Darstellung der Risiken

2.5.2. Maßnahmenkatalog

#	Bezeichnung	Beschreibung des Risikos	Maßnahmen
1	Ausfall GIT/Redmine Server	Der zur Verfügung gestellte Server für Redmine & GIT fällt aus, alle darauf gespeicherten Daten gehen verloren.	Es sollen in regelmäßigen Abständen ein Backup dieser beiden Dienste des Servers erstellt werden.
2	Arbeitsunfähigkeit eins Mitarbeiters	Ein Mitarbeiter des Projektteams verunglückt so schwer, dass er nicht mehr in der Lage ist, am Projekt weiterzuarbeiten.	
3	Backdoor in der Implementierung von AES / MARS	Es wird publik, dass die NSA ein Backdoor in der Javaimplementierung der verwendeten Verschlüsselungsalgorithmen AES und MARS eingebaut hat. Dadurch wäre der die Verschlüsselung der Nachrichten obsolet.	Es werden andere Verschlüsselungsalgorithmen eingesetzt
4	Schnittstelle zu FB nicht realisierbar	Es ist technisch nicht möglich, eine Schnittstelle zu Facebook mit vertretbarem Aufwand zu etablieren.	Es wird auf andere Social Networks mit bereits bestehender Java API zurückgegriffen
5	Es werden Softwarepatente verletzt	Die technische Umsetzung des Projektes verstößt gegen Softwarepatente Dritter.	Um rechtliche Schwierigkeiten zu vermeiden, soll der Code so angepasst werden, dass keine Softwarepatente mehr verletzt werden.
6	Mobilfunkbetreiber schaltet SMS Dienst ab	Ein großer österreichischer Mobilfunkbetreiber sieht den SMS-Dienst als Verlustgeschäft und obsolet an und schaltet ihn daraufhin ab. Dadurch kann die Anwendung nicht mehr Nachrichten über SMS übertragen.	
7	Anderes Produkt bereits am Markt	Es exisitiert bereits vor der Veröffentlichung der Software ein Produkt mit gleichen oder weitgehend ähnlichen Funktionalitäten am Markt.	

Tabelle 2.1.: Maßnahmenkatalog

3. Realisierung

3.1. Allgemeines

Die Desktop als auch die Androidanwendung wurden mithilfe des Entwicklungstools Eclipse umgesetzt. Dabei setzten wir auf die Sprache Java, da diese plattformunabhängig operiert und weit verbreitet ist. Wir haben dabei stets auf eine gewisse Konsistenz geachtet, um die selben Funktionen und Tools auf allen Geräten verfügbar stellen zu können.

Dabei haben wir die wichtigsten Grundfunktionen in eine eigene Core-Library ausgelagert, welche schlussendlich nurmehr in die App und die Desktopanwendung eingebunden werden musste.

3.1.1. E-Mail

Einer der wichtigsten Aspekte unserer Anwendung ist das Versenden und Erhalten von verschlüsselten und unverschlüsselten E-Mails. Dank der Einfachheit des Einbindens von eigenen Libraries konnte problemlos die JavaMail-API verwendet werden, sodass nur noch die Herausforderung der Implementierung zu bewältigen war.

Zu Beginn wird man sofort mit den drei Protokollen des heutigen E-Mail-Verkehrs konfrontiert: Dem **Internet Message Access Protocol (IMAP)**[11], **Post Office Protocol 3 (POP3)** [13]und **Simple Mail Transfer Protocol (SMTP)** [12].

Die ersten beiden dienen zum Abrufen von E-Mails, während das letztere zum Senden dient. Somit stand fest, dass das Abrufen von E-Mails mittels beiden Protokollen, sowohl

IMAP als auch POP3, implementiert werden muss und der Benutzer im Endeffekt die Wahl hat, welches der beiden er zum Abrufen verwendet. Nach der Spezifikation eines gültigen Profils kann der Benutzer anschließend E-Mails versenden bzw. seine empfangenen Nachrichten vom zuvor angegebenen Server abfragen. Zum Versenden von verschlüsselten E-Mails muss vor dem Absenden noch ein Schlüsselaustausch mit dem Gesprächspartner durchgeführt werden. Dies kann auf drei Arten geschehen:

1. Automatisch
2. Manuell
3. QR-Code.

Bei Methode eins wird nach Angabe der Zieladresse (somit des Gesprächspartners) eine Schlüsselaustauschanfrage an diesen geschickt. Beim nächsten Nachrichtenabruf des Partners antwortet dieser dann automatisch auf die Anfrage. Bei anschließender Nachrichtenabfrage am eigenen Konto wird die Antwort dann erhalten und die Kommunikation kann von nun an verschlüsselt stattfinden.

Die zweite Methode, „Manuell“, kann man ebenfalls als statisches Festlegen eines Schlüssels bezeichnen. Auf beiden Seiten muss zum jeweils gegenüberliegenden Gesprächspartner derselbe Schlüssel angegeben werden, mit dem die Nachricht dann auf einem Gerät verschlüsselt und am anderen entschlüsselt wird.

Die letzte Methode dient zum Erstellen eines QR-Codes, der dann vom Ziel mit Hilfe der Androidanwendung eingescannt werden kann. Der Schlüssel wird hierbei offline übertragen.

3.1.2. SMS

Das Versenden und Empfangen von SMS-Nachrichten stellt einen wichtigen Teil unserer App für Android dar. Es sollen sowohl unverschlüsselte als auch verschlüsselte Nachrichten

gespeichert und jederzeit gelesen werden können.

Schon früh war klar, dass auch Funktionen, wie zum Beispiel Emoticons, unterstützt werden sollten, sodass der Benutzer im Endeffekt keinen Featureverlust im Vergleich zur Standard-SMS-App von Android hinnehmen muss. Das Verfassen und Lesen von Nachrichten sowie das Einfügen von Smilies laufen ident zur Standardapp von Android ab.

3.1.3. Facebook-Nachrichten

Die Umsetzung der Facebook-Implementation erfolgte durch eine eigens entwickelte Schnittstelle. Wir haben uns für diesen Schritt entschieden, da die offizielle Facebook-API für den Zugriff zur Chatfunktion am 31.03.2015 endgültig eingestellt wurde. Uns blieben somit nur mehr zwei Optionen: Entweder auf die Implementierung der Facebook-Chat-Funktion zu verzichten oder sich selbst einen Weg um diese Beschränkung herum zu bauen. Wir haben uns für zweiteres entschieden.

Die Grundidee dieser Schnittstelle war es, auf die Methoden der mobilen Facebook-Seite zuzugreifen. Somit war unser erster Schritt das Sammeln notwendiger Informationen zum Ablauf und der Verarbeitung von Requests an den Facebook-Server. Da es hierfür keine offizielle Dokumentation gibt, musste diese Erkenntnis durch Sniffing der Pakete gewonnen werden.

Der nächste Schritt war es nun, ebenfalls Requests selbst zu erstellen und diese an den Server zu schicken. Problematisch hierbei war, dass wir Cookies und ein CRSF-Token, anstatt einer Signatur, wie es bei APIs im Normalfall üblich ist, verwenden mussten.

Der dritte und letzte Schritt war das Schreiben einer Schnittstelle in Java, mit der wir dann einfach über die Androidanwendung Requests verschicken und Responses verarbeiten konnten.

3.1.3.1. Anmeldung bei Facebook-Servern

Um Zugriff auf die Chatfunktion erhalten zu können, müssen wir uns zunächst mit einem Facebook-Account authentifizieren. Wir schicken hierbei denselben Anmelde-Request, der auch bei der mobilen Seite verwendet wird. Im Prinzip handelt es sich hierbei um einen einfachen Formular-Request, welcher die Felder E-Mail und Passwort erwartet. In der Antwort des Facebook-Servers erhalten wir nun im Header die Informationen zu dem Cookie, welches am Client gesetzt werden soll. Im Fall von Android wird dieses Cookie automatisch von einem Cookie-Manager gespeichert, jedoch wurde für einen möglichen Einsatz auf Desktopgeräten noch zusätzlich eine Methode implementiert, um das Cookie auch hier speichern zu können.

Das Cookie selbst ist jedoch nur ein Teil, um einen erfolgreichen Request für Chats absetzen zu können. Um sich vor sogenanntem „Cross-Site-Request-Forgery“ (CSRF) zu schützen, verwendet Facebook, wie auch viele andere Dienste, ein CSRF-Token. Dieses wird nach der Anmeldung im HTML-Quellcode versteckt und muss nun an alle HTTP-Requests als Feld angehängt werden. Andernfalls wird der Request verworfen und zählt als ungültig. Dies verhindert, dass ein Angreifer über ein iFrame auf einer beliebigen Seite Zugriff auf Funktionen von Facebook hat. Ansonsten wäre es beispielsweise möglich, über eine kompromittierte Seite Statusupdates auf Facebook abzusetzen oder ähnliche Anfragen zu stellen.

Somit suchen wir nach der Anmeldung im Quellcode automatisiert nach diesem Token und speichern es temporär in einer Variable. Hierbei kommt die Library Jsoup zum Einsatz, die uns das einfache Parsen des HTML-Codes ermöglicht. Im Formularfeld mit der ID „`fb_dtsg`“ finden wir nun genau dieses Token und können den Attribut-Wert auslesen.

Ein weiteres benötigtes Feld ist die eigene User-ID. Diese wird ebenfalls für das Absetzen viele Requests benötigt und lässt sich aus dem HTML-Quellcode unter der ID „USER_ID“ herauslesen. Um noch dem Nutzer die Option zu bieten, sein Facebook-

Profilbild verwenden zu können, speichern wir den Verweis an dieser Stelle temporär mit.

Somit fehlt uns nun nur noch der letzte Teil der Initiierung, das Token, um sich von Facebook wieder abmelden zu können. Die Seite „logout.php“ erwartet zusätzliche Tokens, die, ähnlich wie bei dem CSRF, versteckt im HTML-Quellcode mitgeschickt werden. Damit wir beim Abmelden nicht zusätzliche Requests abschicken müssen bis es möglich ist, sich tatsächlich auszuloggen, wird das Token ebenfalls temporär gespeichert.

Wir erfüllen nun alle Vorraumsetzungen, um erfolgreich Requests am Facebook-Server absetzen zu können und haben die Initiierung abgeschlossen.

3.1.3.2. Senden einer Nachricht

Zum Senden von Nachrichten wird ein Request an `facebook.com/messages/send` abgesetzt. Dabei akzeptiert der Server die POST-Attribute: User-ID, CSRF-Token, Thread-ID, Body (Nachricht) sowie die ID des Gegenübers. Wie bereits im Anmeldeverfahren beschrieben, erhalten wir die User-ID und das CSRF-Token nach der Anmeldung, und diese werden temporär von unserer Schnittstelle gespeichert.

Der Body beinhaltet die eigentliche Nachricht und wird als Attribut im POST angehängt. Schlussendlich wird noch die ID oder die TID des Gespräches benötigt. Dabei ist es egal, ob direkt die TID, welche jede einzelne Kommunikation mit einer eindeutigen ID versehen hat, oder direkt die Facebook Nutzer-ID des Gegenübers verwendet wird.

Abbildung 3.1.: Facebook Message Send Request

Daraufhin erhalten wir von dem Facebook-Server eine JSON-Response. Ob der Request erfolgreich war, können wir an dem HTTP-Response-Header oder an der Angabe des Fehlers im JSON erkennen.

Unsere entwickelte Schnittstelle füllt automatisch die temporär gespeicherten Felder im Request aus. Somit müssen für das Senden einer Nachricht nur mehr die ID, TID und der Body übergeben werden.

3.1.3.3. Benachrichtigungsdienst

Viele Nutzer sehen das Benachrichtigungssystem mittlerweile als selbstverständlich an. Eine SMS-Anwendung, die den Nutzer nicht über neue Nachrichten benachrichtigt, wird heute kaum noch Abnehmer finden. Dieselbe Problematik hatten wir ebenfalls bei unserer Androidanwendung. Zwar war es möglich, Facebook-Nachrichten zu lesen und zu schreiben, doch würde der Nutzer nicht über eine neu eintreffende Nachricht informiert werden. Aus diesem Grund haben wir das System und die benötigten Schnittstellen wieder komplett selbst erstellt, um diesen Service auch für Facebook zur Verfügung stellen zu können.

Facebook verwendet bei PC und mobilen Browsern ein Pull-System. Am Client wird also ein Script ausgeführt, das automatisiert alle 54 Sekunden eine Anfrage an den Channel schickt. Um einen Channel beitreten zu können, wird an `facebook.com/a/channel /reconnect.php` ein Request mit der eigenen User-ID verschickt und man erhält nun als Antwort die Channel-URL.

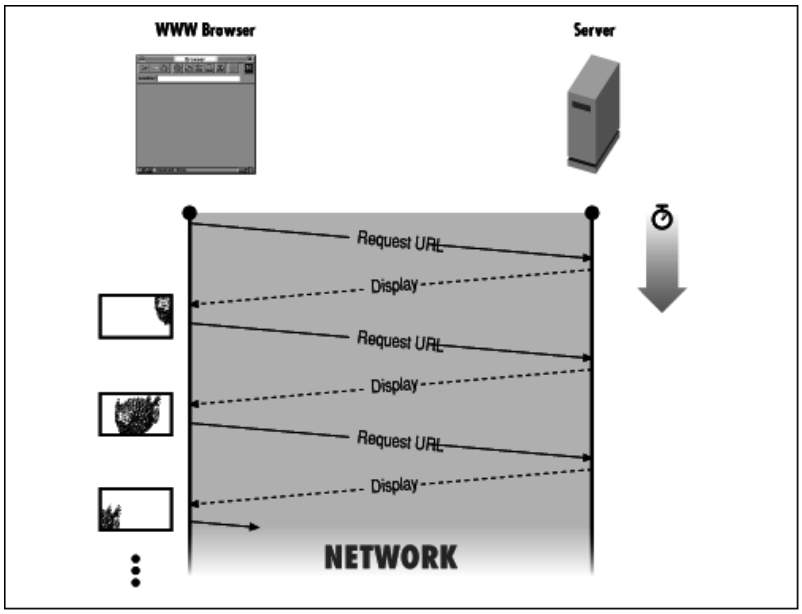

Abbildung 3.2.: HTTP Pull

Der Request wird 54 Sekunden offen gehalten. Falls während dieser Zeitspanne eine neue Nachricht eintrifft, antwortet der Facebook Server sofort. Folglich wird der Request dann vom Client verarbeitet und in unserem Fall an die höheren Layer, die Androidanwendung, weiter gegeben. Nachdem eine Antwort empfangen wurde, wird erneut ein Pull-Request gesendet und wieder offen gehalten. Sollte nun innerhalb von 54 Sekunden keine Nachricht empfangen werden, schickt der Facebook-Server einen Heartbeat, um dem Client zu signalisieren, dass die Verbindung nicht abgerissen ist, aber auch keine neue Nachricht empfangen wurde. Sobald ein Heartbeat empfangen wurde, gilt dieser Request wieder als abgeschlossen, und unser System sendet erneut eine Anfrage.

Über diesen Pull-Channel können jedoch nicht nur Nachrichten, sondern auch andere Informationen empfangen werden. Beispielsweise kann Facebook über diese Schnittstelle signalisieren, ob die vom Nutzer gesendete Nachricht bereits von seinem Gegenüber gelesen wurde oder eine Nachricht vom Gegenüber zurzeit eingetippt wird. Wir erkennen automatisch, um welchen Nachrichten-Typ es sich handelt und erzeugen nun ein geeignetes `Facebook-Object`, ein von uns implementiertes Interface zum Verarbeiten der Daten.

3.1.4. Verwendete Libraries

Folgende Libraries Dritter wurden im Zuge der Erstellung von Raven verwendet:

Name	Lizenz
Liquid Look And Feel	LGPL
Spongeycastle	MIT
ZXING	Apache License v2.0
Apache Commons Validator	Apache License v2.0
Emojicon	Apache License v2.0
Floating Action Button	MIT
Material Design Library	Apache 2.0 Licence
Jsoup	MIT
Trace	MIT
Java Mail for Android	GPL 2
Appcompat	Apache License v2.0
Google Play Services	Apache License v2.0
Android Async HTTP	Apache License v2.0

Tabelle 3.1.: Verwendete Libraries

[14]Emojicon: Mit dieser Library wird uns das Darstellen und Versenden von Smileys ermöglicht. Sie kommt bereits bei vielen anderen Anwendungen, wie beispielsweise Telegram, zum Einsatz. Der Entwickler Hieu Rocker veröffentlichte die Library 2014 unter der Apache Lizenz auf GitHub.

[15]Floating Action Button: Google setzte mit Android Lollipop auf eine vollkommen neue Oberfläche namens Material Design. Jedoch gibt es für Entwickler nur wenige offizielle Libraries, um neue Funktionen zur Verfügung zu stellen. Viele Entwickler setzen manche Funktionen selbst um, was wiederum zu einer Inkonsistenz unter verschiedenen Apps führt. Aus diesem Grund setzten wir bei dem Floating Action Button, der kleine blaue Button am unteren Bildschirmrand, auf die Implementierung des Entwicklers Oleksandr Melnykov.

[16]Material Design Library: Aus den gleichen Gründen haben wir bei weiteren Lollipop Funktionen ebenfalls eine bereits von Entwicklern akzeptierte Implementierung gesetzt. Diese Library bietet eine Vielzahl an Android 5.0 Funktionen, welche bis Android

2.2 abwärtskompatible sind. Geschrieben wurde sie von dem Entwickler unter dem Pseudonym navasmdc.

[17]Jsoup: Ist eine Library zum Verarbeiten von HTML in Java. Wir haben sie großteils bei der Entwicklung unserer Facebook-Schnittstelle verwendet. Entwickelt und unter der MIT Lizenz veröffentlicht wurde sie von Jonathan Hedley.

[18]Trace: Bei Trace handelt es sich um eine Library zum automatisierten Verschicken von Absturzberichten. Während der frühen Entwicklungsphase kam diese Library bei Tests zum Einsatz. Google selbst bietet zwar eine Möglichkeit, Absturzberichte automatisch an die Entwickler zu verschicken, jedoch wird dies erst nach der Veröffentlichung der App ermöglicht. Damit wir bereits vorab immer über mögliche Fehler informiert sind, haben wir diese Library in der Testphase verwendet. Sie wurde von Mads Kristiansen entwickelt und unter der MIT Lizenz veröffentlicht.

[19]Java Mail for Android: Dabei handelt es sich um eine Portierung der Java-Mail Funktionalitäten auf Android. Im Grunde bietet sie den selben Funktionsumfang, wie beispielsweise E-Mails mittels POP3 oder IMAP zu empfangen und mittels SMTP zu versenden. Jon Simon veröffentlichte die Library 2009.

[20]Appcompat: Ist eine offizielle Library von Google, um die neusten Android-Funktionen auch unter älteren Versionen bereitstellen zu können. Sie beinhaltet die grundlegenden Funktionen für das Material Design, nicht aber spezifische Implementierungen, wie den Floating Action Button.

[21]Google Play Services: Eine weitere offizielle Google Library mit einer Vielzahl an Funktionen und Schnittstellen zu diversen Google-Diensten. Wir verwenden sie hauptsächlich für die Erhebung von Statistiken mittels Google Analytics.

[22]Android ASync HTTP: Hierbei handelt es sich um eine Library zum Absetzen von HTTP-Requests, die jedoch auch asynchron verschickt werden können. Wir verwenden sie für den Facebook-Benachrichtigungsdienst, da dieser in einem periodischen Intervall

Requests verschicken und bearbeiten muss. Um andere Requests nicht innerhalb dieses Zeitraums zu blockieren, wird diese Library herangezogen. Sie wurde unter der Apache 2.0 Lizenz von James Smith veröffentlicht.

[23]Liquid Look and Feel: Da wir die Desktopanwendung nicht im standardmäßigen Metallic-Look von Swing veröffentlichen wollten, haben wir dieses alternative „Look and Feel“ gewählt. Entwickelt wurde es von den Pseudonymen „mikeai“ und „xendren“.

[24]Spongeycastle: Für sämtliche kryptografische Vorgänge greifen wir auf diese Library zurück. Sie ermöglicht das Generieren und Verarbeiten von „kryptografisch starken“ Zahlen. Entwickelt wurde sie von dem Pseudonym „rtyley“.

[25]ZXING: Mit dieser Library werden QR-Codes in der Desktopanwendung generiert. Entwickelt wurde sie von Sean Owen.

[26]Apache Commons Validator: Diese Library kann eine Vielzahl von Gültigkeits-Checks durchführen. In der Desktopanwendung wird sie zum Überprüfen der eingegebenen E-Mail-Adressen verwenden. Entwickelt wurde sie von der Apache Software Foundation.

3.1.5. Core Library

3.1.5.1. Idee

Unsere Anwendung hat zum Ziel, auf möglichst vielen Plattformen zu laufen. Einerseits lässt sich dies leichter durch die Programmiersprache Java bewerkstelligen, dennoch müssen einige Optimierungen für spezifische Plattformen manuell erledigt werden. Hier kommt unsere Core-Library ins Spiel. Die Idee war es, die Grundfunktionen, wie Verschlüsselung der Nachrichten, Schlüsselaustausch, Schlüsselspeicherung und etwaige andere wichtigen Grundkomponenten in einer eigenen Library zur Verfügung zu stellen. Diese wird gleich von Beginn an für mehrere Plattformen optimiert und muss schlussendlich nur in die Android-App bzw. die Desktopanwendung eingebunden werden.

3.1.5.2. Funktionen

Folgende Anforderungen wurden an die Schlüssel gestellt:

1. Kryptographisch starke Zufallszahlen
2. Kryptographisch starke Hashfunktion, die einen 512-Bit-Hashwert liefert

Diese zwei Punkte sind ausschlaggebend für die Sicherheit der gesamten Verschlüsselung innerhalb der Anwendungen. Zu beachten gilt, dass der Risikofaktor bei schwachen Zufallszahlen aufgrund ihrer Berechenbarkeit sehr hoch ist. Infolge würde das zu berechenbaren Schlüsseln führen, was wiederum den gesamten kryptographischen Ansatz zunichte machen würde. Aus diesem Grund erzeugen wir, mit Hilfe der `SecureRandom`-Klasse, kryptographisch sichere Zufallswerte. Hierbei werden Daten von `/dev/random` herangezogen und inkludieren somit verschiedene hardwarenahe Inputs und Interrupts, Benutzereingaben und Netzwerkpakete, die möglichst unvorhersagbar sind.

Eine weitere Möglichkeit zu Generierung von Zufallszahlen wären Zufallszahlengeneratoren auf Hardware-Basis. In den Intel-Prozessorgeneration Ivy-Bridge [27] und aufwärts ist ein solcher Hardware-Zufallsgenerator bereits implementiert. Dabei wird ein Flip-Flop in einen quasi stabilen Zustand gebracht, der nun je nach elektromagnetischen Störungen sich entweder auf den Zustand null oder eins festsetzt.

Neben einem starken Zufallszahlenerzeuger darf auch eine starke Hashfunktion nicht fehlen. Hierbei stießen wir auf Blake, eine kryptographische Hashfunktion von Jean-Philippe Aumasson, Luca Henzen, Willi Meier und Raphael C.-W . Phan [1]. Blake war ein Finalist der „NIST hash function competition“ [1] und verlor nur knapp gegen Keccak. Er wurde durch den Sieg zum offiziellen SHA-3 Standard erklärt. Allerdings war die Auswahl für den SHA-3-Standard in der Kryptographieszene hart umstritten, da im Nachhinein

[1] Wettbewerb zur Bestimmung von SHA-3

sicherheitsrelevante Änderungen zugunsten der Performance vorgenommen wurden. Bruce Schneier, Kryptographie- und Computersicherheitsexperte, warnt zur Vorsicht:

> „Normally, this wouldn't be a big deal. But in light of the Snowden documents that reveal that the NSA has attempted to intentionally weaken cryptographic standards, this is a huge deal. There is too much mistrust in the air. NIST risks publishing an algorithm that no one will trust and no one (except those forced) will use"[5]

Aus diesem Grund ist unsere Wahl auf Blake gefallen, welcher offiziell aus Performancegründen gegen Keccak verloren hatte.

Implementierung

Aufgrund der doppelten Verschlüsselung in unserer Anwendung verwenden wir die 512-Bit-Implementierung des Algorithmus, wobei wir die ersten 256-Bit für den ersten Schlüssel, die zweiten 256-Bit für den zweiten Schlüssel verwenden. Zudem ermöglichen wir mit einer eigenen Klasse für die Schlüsselerzeugung, noch zusätzlich das Hinzufügen von Noncen. Schlussendlich werden über diesen Zufallspool, mit Hilfe der Blake-Hashfunktion, die zwei Schlüssel erzeugt und mit ihren Metadaten in einem `KeyEntity`-Objekt gespeichert.

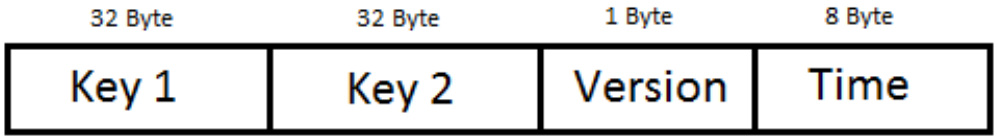

Abbildung 3.3.: `KeyEntity`-Aufbau

3.1.5.3. Verschlüsselung

Allgemein

Ein wesentlicher Aspekt der Core-Library sind die zahlreichen implementierten Verschlüsselungsalgorithmen. Wir greifen hierzu auf die quelloffene Library „Bouncycastle“ zurück. „Bouncycastle“ ist eine umfangreiche Kryptographie-Schnittstelle, die sowohl für Java als, auch C# verfügbar ist. Auf Android ist eine abgewandelte Form, welche nur die wichtigsten Funktionen beinhaltet, bereits standardmäßig integriert.

Da wir für unsere Zwecke jedoch mehr als nur die wichtigsten Funktionen benötigen, Stichwort „Schlüsselaustausch mittels Diffie-Hellman“, muss diese Schnittstelle zusätzlich noch in unserem Core integriert werden. Dies stellte eine Herausforderung dar. Aufgrund des Packagenamens konnte die Library nicht in das Projekt eingebunden werden. Die ebenfalls quelloffene Library namens „Spongeycastle“, eine wiederum abgewandelte Form von „Bouncycastle“ mit einem anderem Packagenamen, konnte uns hier zumindest unter

Android Abhilfe schaffen.

Jedoch standen wir mit dieser Library bei Desktopgeräten erneut vor einem Problem, da Java von Grund auf nur gewisse signierte Kryptographieerweiterungen zulässt. Da „Spongeycastle“ eben keine solche Signatur aufwies, verweigerte Java uns den Zugriff auf diese Schnittstelle. Somit verwenden wir nun beide Libraries, „Bouncycastle“ unter Desktopgeräten, „Spongeycastle“ unter Android.

Technisch

Im Laufe dieses Projekts haben wir uns mit zahlreichen Verschlüsselungsalgorithmen auseinandergesetzt und experimentiert. Wir wollten eine Mischung zwischen bewährten Methoden und relativ neuen, vielversprechenden Algorithmen einsetzen. Uns war gleich von Anfang an klar, dass wir uns nicht auf einen einzigen Algorithmus verlassen, sondern, wie auch beispielsweise bei Truecrypt, mehrere Verfahren kombinieren wollten.

Unser erstes Verfahren haben wir auch schnell gefunden: Advanced Encryption Standard, kurz AES. AES ist eine von Joan Daemen und Vincent Rijmen entwickelte Blockchiffre und wurde erstmals 1998 unter dem Namen Rijndael veröffentlicht. [3]

Im Jahr 2000 wurde diese durch das National Institute of Standards and Technology (NIST) als offizieller Nachfolger für DES (Data Encryption Standard) erklärt und erhielt seine Bezeichnung AES. Obwohl dieser Algorithmus mittlerweile seit 15 Jahren standardisiert ist, gibt es bislang nur wenige theoretische Angriffe. Allerdings wurde durch geleakte Dokumente bekannt, dass die National Security Agency (NSA) in ihrem neuen Utah Data Center mit enormer Rechenleistung bereits daran arbeitet, die AES-Verschlüsselung zu brechen.

Inwiefern es der NSA mittlerweile gelungen ist, bleibt unbekannt.

Wir verwenden in unserer Diplomarbeit die stärkste Implementierung dieses Algorithmus, AES-256. Doch da wir das theoretische Risiko ausschließen wollten, dass es womöglich

doch gelungen ist, einen praktikablen Angriff auf AES durchzuführen, haben wir den Algorithmus mit einem relativ jungen, aber sehr starken Verfahren kombiniert.

Dieses Verfahren trägt den Namen Threefish, welches von Bruce Schneier, Niels Ferguson, Stefan Lucks, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas und Jesse Walker entwickelt wurde. Im Gegensatz zu AES, welches eine maximale Schlüssellänge von 256-Bit ermöglicht, fängt Threefish bei 256-Bit gerade erst einmal an. [4]

Ebenfalls unterstützt werden allerdings auch Schlüssellängen mit 512 sowie 1024-Bit. Zurzeit ist nur ein theoretischer Angriff für Threefish bekannt, welcher das Knacken eines Schlüssels bei 32 Runden (wohl angemerkt, dass die Standardimplementierung 72 Runden vorsieht!) in 2^{226}Kombinationen statt 2^{256}ermöglicht, in der Realität daher kein sicherheitstechnisch relevanter Angriff.

Mit diesem Wissen gewappnet kamen wir zu dem Entschluss, dass eine Mischung aus diesen beiden Varianten für unser Vorhaben bestens geeignet war.

Verschlüsselung einer Nachricht

Die vom Anwender eingegebene Nachricht soll nun mit diesen zwei Algorithmen verschlüsselt werden. Hierzu gehen wir in folgenden Schritten vor:

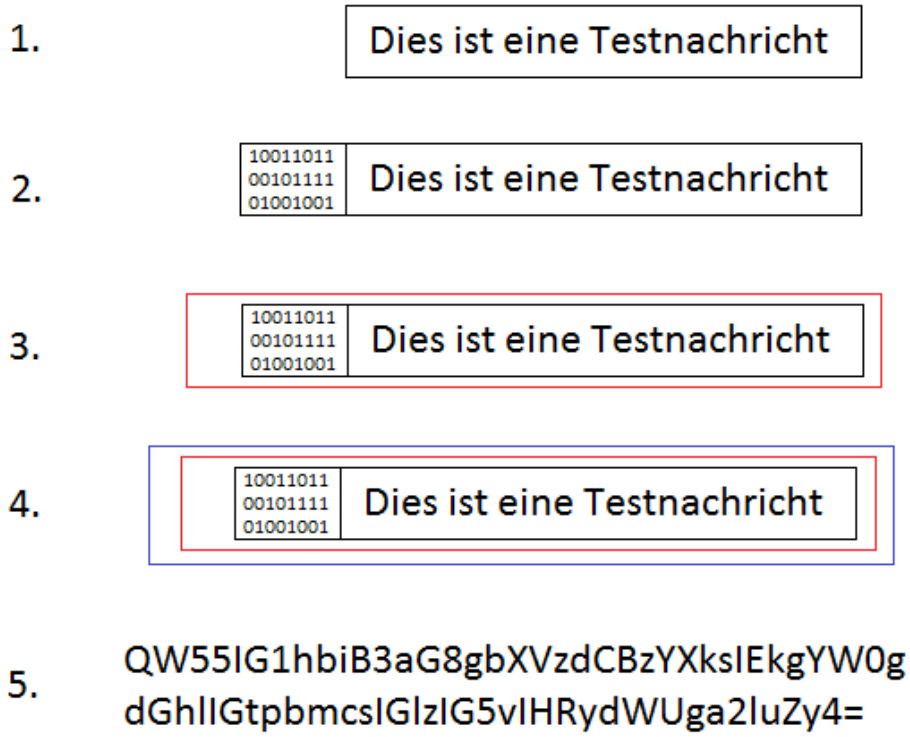

Abbildung 3.4.: Workflow der Verschlüsselung

1. Eine Klartextnachricht wird eingegeben, in diesem Fall „Dies ist eine Testnachricht“
2. In diesem Schritt fügen wir eine Nonce hinzu. Diese wird für jede Nachricht zufällig generiert und ist 2^{24}Bit lang. Ihre Aufgabe liegt darin, dass ein Klartext mit dem gleichen Schlüsseln nicht jedes Mal die gleiche, verschlüsselte Zeichenfolge ergibt. Ansonsten könnte man bei häufigen Wörtern wie „Hallo“ Rückschlüsse ziehen. Mit dieser Nonce wird das verhindert.
3. In diesem Schritt wird nun die gesamte Nachricht, inklusive Nonce, mit Threefish

verschlüsselt.

4. Der verschlüsselte Text aus Schritt 3 wird nun erneut, dieses Mal jedoch mit AES, verschlüsselt. Hierbei ist zu erwähnen, dass sowohl ein eigener Schlüssel für Threefish als auch für AES verwendet wird. Jede Nachricht wird also mit zwei verschiedenen Verfahren mit zwei verschiedenen Schlüsseln verschlüsselt.

5. Das Ergebnis aus dem 4. Schritt wird in Base64 kodiert. Die Bytes werden also mithilfe von Base64 in Buchstaben, Zahlen und wenige Sonderzeichen umgewandelt.

Die Entschlüsselung einer Nachricht läuft prinzipiell genau anders herum ab:

1. Den Base64 String wieder in eine Bytefolge umwandeln.

2. Nun mit dem AES-Schlüssel die Nachricht mithilfe von AES entschlüsseln.

3. Mithilfe des Threefish-Schlüssels das Ergebnis aus dem zweiten Schritt mit Threefish entschlüsseln.

4. Die Nonce entfernen (die ersten drei Byte).

5. Die Nachricht ist erfolgreich entschlüsselt und lesbar.

Im Core wurde die Verschlüsselung so umgesetzt, dass wir jeweils die Verfahren mittels Bouncycastle bzw. Spongeycastle ansprechen und mithilfe einer `Message`-Klasse umsetzen. Diese `Message`-Klasse erfüllt jeweils die fünf Schritte für die Verschlüsselung, sowie auch die anderen fünf Schritte für die Entschlüsselung. Das hat den Vorteil, dass das Verschlüsselung und Entschlüsseln kompakt von einer Klasse erledigt wird, die noch dazu komplett plattformunabhängig läuft.

Somit können wir in der Anwendung verschlüsselte Nachrichten mit folgendem Befehl erzeugen:

```
Message mes = new Message("Dies␣ist␣eine␣Testnachricht");
String encrypted = mes.encryptMessage(key);
```

Listing 3.1: Nachrichten verschlüsseln

`key` ist ein `KeyEntity`-Element, welches aus dem Schlüsselspeicher geladen wird

Die Entschlüsselung wird dann wie folgt umgesetzt:

```
Message mes = new Message("RGllcyBpc3QgZWljaHJpY2h0");
String plain = mes.decryptMessage(key);
```

Listing 3.2: Nachrichten entschlüsseln

Schlüsselaustausch

Wir stellen in unserer Anwendung mehrere Möglichkeiten zum Schlüsselaustausch zur Verfügung. Im folgenden Abschnitt werden diese und deren Ablauf genau erläutert.

Folgende Methoden stehen zur Verfügung:

- Automatischer Austausch (Diffie-Hellman)
- Austausch mittels QR Code
- Austausch mittels NFC

Automatischer Austausch (Diffie-Hellman)

Allgemein

Der automatische Austausch wird mittels des Diffie-Hellman-Verfahrens durchgeführt. Dabei handelt es sich um ein Schlüsselaustauschverfahren, welches über ein unsicheres Medium (z.B. das Internet) sicher durchgeführt wird. Entwickelt wurde es von Whitfield Diffie und Martin Hellman im Jahre 1976, wodurch das Verfahren auch seinen Namen erhielt. [6]

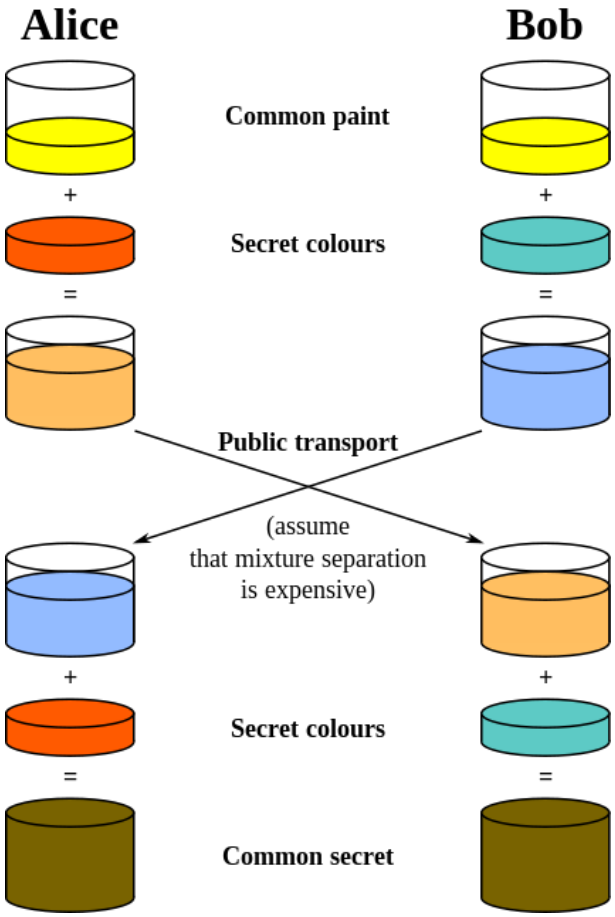

Abbildung 3.5.: Schlüssel Austausch
[6]

Wir gehen nun von zwei Personen aus, einerseits Alice und andererseits Bob. Die beiden möchten nun sicher über das Internet kommunizieren, können den Schlüssel daher

nicht einfach über dieses unsichere Medium verschicken. Daher wollen sie mit Hilfe des Diffie-Hellman-Verfahrens einen gemeinsamen Schlüssel errechnen.

Hierzu wählen beide eine zufällige Primzahl p sowie eine Primitivwurzel q zu p. Für ein einfaches Testbeispiel wählen wir für p die Primzahl 7. Eine Primitivwurzel zu der Primzahl 7 wäre beispielsweise die Zahl 3, da:

$$
\begin{array}{rcrcrcrcrcl}
3^1 & = & 3 & = & 3^0 \times 3 & \equiv & 1 \times 3 & = & 3 & \equiv & 3 \pmod 7 \\
3^2 & = & 9 & = & 3^1 \times 3 & \equiv & 3 \times 3 & = & 9 & \equiv & 2 \pmod 7 \\
3^3 & = & 27 & = & 3^2 \times 3 & \equiv & 2 \times 3 & = & 6 & \equiv & 6 \pmod 7 \\
3^4 & = & 81 & = & 3^3 \times 3 & \equiv & 6 \times 3 & = & 18 & \equiv & 4 \pmod 7 \\
3^5 & = & 243 & = & 3^4 \times 3 & \equiv & 4 \times 3 & = & 12 & \equiv & 5 \pmod 7 \\
3^6 & = & 729 & = & 3^5 \times 3 & \equiv & 5 \times 3 & = & 15 & \equiv & 1 \pmod 7
\end{array}
$$

Abbildung 3.6.: Primitiv-Wurzel zu 7
[7]

Alle Zahlen $q^n mod(p)$ sind also wieder Zahlen zwischen 1 bis p in einer Restklasse.

Diese beiden Zahlen, p und q, machen sich Alice und Bob nun über dieses unsichere Medium aus. Falls ein Angreifer diese Zahlen mitliest, kann er dennoch nichts damit anfangen.

Alice generiert nun eine zufällige Zahl a, beispielsweise 6. Diese Zahl kennt nur Alice und sie wird nie über das Internet übertragen.

Daraufhin berechnet Alice A, wobei $A = g^a mod(p)$. Setzt man unsere Beispielswerte ein, so ergibt sich $A = 3^6 mod(7) = 1$. Alice sendet also die Zahl 1 an Bob.

Bob berechnet nun ebenfalls eine zufällige Zahl b, beispielsweise 15. Ebenfalls wie im Schritt zuvor, kennt diese Zahl nur Bob selbst, und sie verlässt nie Bobs Computer. Daraufhin berechnen wir mit derselben Formel B. $B = g^b mod(p) = 3^{15} mod(7) = 6$. Bob sendet also an Alice die Zahl 6.

Alice nimmt nun den von Bob empfangenen Wert B und führt folgende Berechnung durch: $s = B^a mod(p)$. Dies ergibt sich zu $s = 6^6 mod(7) = 1$.

Bob führt nun ebenfalls dieselbe Berechnung mit A von Alice durch: $s = A^b mod(p) = 1^{15} mod(7) = 1$.

Alice und Bob haben nun denselben Wert für s. Dies ist kein Zufall, sondern beruht auf der Mathematik hinter diesem Verfahren. Alice und Bob haben sich so gegenseitig, ohne es jemals im Internet bekannt gegeben zu haben, einen sicheren Schlüssel ausgemacht.

In der Realität werden jedoch nicht so einfache Zahlen verwendet, sondern Zahlen im mehr als 300-stelligen Bereich. In dem oben durchgerechneten Beispiel ist es weiters bloß Zufall, dass A und s denselben Wert angenommen haben. Mit größeren Zahlen ist es nahezu unmöglich, denselben Wert für A oder B wie s zu erhalten.

Ein Angreifer kann zwar nun die Zahlen B, A, p und q mitlesen, kann aber ohne die geheimen Zahlen a und b nicht die notwendige Berechnung für s durchführen, da $s = A^b mod(p)$ ergibt. Jedoch könnte ein Angreifer sich in diesen Prozess zwischenschalten, einen Man-In-the-Middle-Angriff durchführen. Hierzu würde ein Angreifer dem User Bob vortäuschen, dass er selbst Alice sei und schickt ihm sein generiertes A. Das selbe führt der Angreifer nun auch mit Alice durch, wodurch der Angreifer nun eine verschlüsselten Kanal sowohl mit Alice, als auch Bob geformt hat.

Schreibt Alice nun eine Nachricht an Bob, so empfängt der Angreifer diese, entschlüsselt sie mit seinem Schlüssel, verschlüsselt sie mit Bobs Schlüssel und schickt dieses an Bob. Der Angreifer kann also bequem, in beide Richtungen, mitlesen. Dieser Angriff ist aber nur möglich, wenn Alice und Bob sich gegenseitig nicht authentifizieren können. Der Grundstein dieses Verfahrens liegt darin, dass Alice und Bob nach dem Diffie-Hellman-Austausch denselben Schlüssel haben. Dies ist, falls sich ein Angreifer dazwischen schaltet, nicht der Fall.

Aus diesem Grund ist es ratsam, dass Alice und Bob ein weiteres Verfahren, eine digitale Signatur, verwenden um festzustellen, ob sie wirklich miteinander kommunizieren oder über den Angreifer. Sind die Schlüssel von Alice und Bob nicht gleich, ist es mehr

als nur wahrscheinlich dass bei dem Austausch ein Angreifer seine Hand im Spiel hatte. Sind die Schlüssel jedoch gleich, ist die Kommunikation sicher.

Umsetzung

Wir verwenden diesen Schlüsselaustausch sowohl in der Android-App als auch in der Desktopanwendung. Die Funktion wird von der BouncyCastle-Library bereitgestellt, wobei wir statt der Primzahlenimplementierung (siehe 3.1.5.3 auf Seite 29) die Elliptic-Curve-Implementierung verwenden. Das hat vor allem Performancegründe, da das Finden von großen Primzahlen (im Bereich von 1024 - 2048-Bit) ein aufwendiges Unterfangen ist. Bei einer Elliptic-Curve erhalten wir ebenfalls Werte für p und q, die in einer Restklasse liegen, jedoch mit viel weniger Rechen- bzw. Suchaufwand.

Wir setzen hierbei auf die Secp256k1-Kurve. Diese wird auch von der Cryptocurrency Bitcoin für das Erstellen der Adressen verwendet. Diese Kurve ist nicht standardisiert und wurde vermutlich genau aus diesem Grund von Satoshi Nakamto, dem Bitcoin-Entwickler, verwendet. Andere Standards könnten womöglich Backdoors beinhalten, weswegen wir uns ebenfalls gegen diese entschieden haben. Da diese Kurve mittlerweile durch Bitcoin ihre Stärke bewiesen hat, waren wir ebenfalls zuversichtlich, dass sie für unsere Zwecke ebenfalls ausreichen würde. Vor Bitcoin kam diese Kurve kaum in der Realität zum Einsatz.

Im Gegensatz zu manch anderen Elliptic-Curves, wurde diese in einer speziellen, nicht zufälligen Form gewählt. Dies macht sie zum einen viel performanter, anderseits vermindert dies auch drastisch das Risiko, dass der Entwickler womöglich ein Backdoor implementiert hat.

In unserer Core-Library verwenden wir eine eigene Klasse, welche den Austausch mittels Diffie-Hellman noch um einiges vereinfacht. Möchte man mit unserer Library nun einen Schlüsselaustausch starten, wird nur folgender Code benötigt:

```
ECDHExchange alice = new ECDHExchange();
// Initialisiert das DH Verfahren für Alice

ECDHExchange bob = new ECDHExchange();
// Initialisiert das DH Verfahren für Bob

/* Mittels .getEncodedPublicKey() erhält man den
öffentlichen Teil (A bzw B) im Base64 Format. Dieser wird über
das Internet übertragen und anschließend von dem gegenüber
mit .importPublicKey(PublicKey) importiert.
*/

PublicKey bob_pub = alice.importPublicKey(Base64.decode(
                        bob.getEncodedPublicKey(), Base64.NO_WRAP));
PublicKey alice_pub = bob.importPublicKey(Base64.decode(
                        alice.getEncodedPublicKey(), Base64.NO_WRAP));

// Mit .generateSharedSecret(PublicKey_des_Gegenübers)
// kann man nun den gemeinsamen Schlüssel (s) erstellen

alice.generateSharedSecret(bob_pub);
bob.generateSharedSecret(alice_pub);

// Gibt man nun den geheimen Schlüssel aus, so sieht
//man, dass sowohl Alice, als auch Bob
//den selben Schlüssel erechnet haben.

System.out.println("Alice␣Secret:␣" + alice.getSharedSecret());
```

```
System.out.println("Bob Secret: " + bob.getSharedSecret());
```

Listing 3.3: Diffie-Hellman Austausch

Angewandt sieht der automatische Austausch in unserer Anwendung folgendermaßen aus:

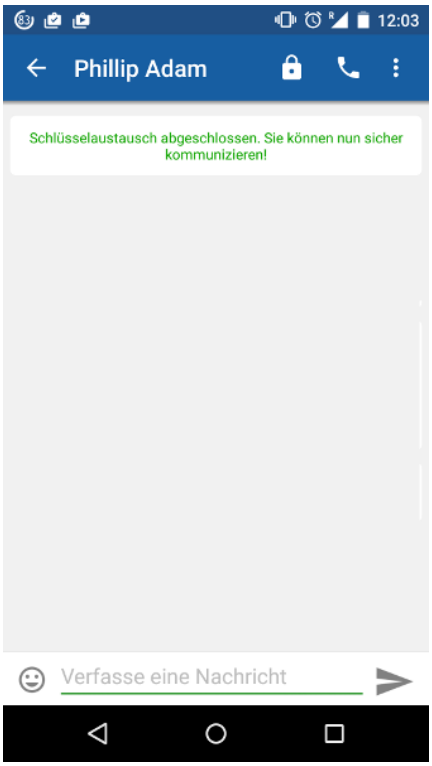

Abbildung 3.7.: Austausch mittels Diffie-Hellman

Zunächst schickt also der Nutzer, der die Verschlüsselung initialisieren möchte, seinen öffentlichen Teil an den Empfänger. Dieser erhält daraufhin eine Benachrichtigung, dass Nutzer A mit ihm verschlüsselt kommunizieren möchte. Öffnet nun Nutzer B die Nachrichtenansicht, so wird automatisch von unserer Anwendung ebenfalls der Diffie-Hellman-Austausch initialisiert, sowie von Nutzer A beantwortet. Gleichzeitig, da der Nutzer B nun über den öffentlichen Teil von A verfügt, kann er auch seinen gemeinsamen Schlüssel generieren und legt ihn, zugeordnet zu Nutzer A, in seinem Speicher ab.

Öffnet Nutzer A die Anwendung, erkennt sie, dass der Schlüsselaustausch von B beantwortet wurde. Somit kann A auch mit dem öffentlichen Teil von B seinen gemeinsamen Schlüssel generieren und abspeichern. Im Endeffekt verfügen nun Nutzer A und B über den gemeinsamen Schlüssel, der Austausch ist damit abgeschlossen.

Im Hintergrund werden also folgende zwei öffentliche Teile verschickt:

Algorithmus 3.1 Schlüsselaustauschmitteilung, wie sie über ein Medium übertragen wird

```
Alice: %MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE
0SAB89HQwP9n1eR7293unR9S7YOxoGvERbu7sP
w9ZqNxvvLXwbSVB9B2UQn1HOVP/k1uk+Tj1mqkMd91TgjWvA==
Bob: %MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE
BOptKJ/VFO9tjeyp9NtQENbqCJCoeZg4ClKKWbQfM
yhPAWq+XD12tOLYTU4CnZgi/MmgFoOIGjRx2Uq1L3R/ng==
```

Wie man sieht, startet bei uns ein Schlüsselaustausch mit einem „%“. Auf dieses Zeichen haben wir uns festgelegt. Die einzige Bedingung war, dass dieses Zeichen nicht im Base64-Zeichensatz vorkommt. Somit können wir recht einfach erkennen, ob es sich um eine Nachricht oder einen Schlüsselaustausch handelt. Beginnt die Nachricht mit einem „%“ und sind alle folgenden Zeichen im Base64 Format kodiert, so handelt es sich um einen Austausch. Damit der Nutzer von diesen Buchstaben, Zahlen und Sonderzeichengemisch nicht verwirrt wird, stellen wir einen Austausch in unserer Anwendung nicht als Text dar, sondern als Mitteilung zum Schlüsselaustausch (siehe 3.7 auf der vorherigen Seite).

Austausch mittels QR-Code

Hintergedanke

Die Idee hinter dem QR-Code-Austausch und NFC-Austausch war es, eine Möglichkeit zu bieten, den Schlüssel komplett offline und ohne Abhängigkeit von Primzahlen zu übertragen. Dies bedeutet natürlich nicht, dass die Stärke von mathematischen Verfahren, die auf dem Produkt zweier Primzahlen basieren, angezweifelt wird. Herkömmliche

Computer sind nicht in der Lage, ein Produkt zweier sehr großer Primzahlen in seine Primfaktoren in endlicher Zeit zu zerlegen. Die Rede ist hier von Zahlen im mehreren hundertstelligen Bereich.

Lediglich ein Quantencomputer wäre in der Lage, eine Primfaktorzerlegung effizient mithilfe des Shor-Algorithmus durchzuführen. Jedoch gibt es nach momentanen Stand der Technik keinen funktionierenden Quantencomputer, auf dem der Shor-Algorithmus vollständig ausgeführt werden kann.

Natürlich wären Organisationen, wie die NSA, besonders deswegen an einem Quantencomputer interessiert. Die Realisierung solcher Systeme ist jedoch schwer abzusehen.

Mit dem Offline-Austausch mittels QR-Code oder NFC wollen wir besonders sicherheitskritischen Benutzern eine alternative Möglichkeit zum Schlüsselaustausch bieten.

`ExchangeInformation`-Protokoll

Um den Schlüssel offline übertragen zu können, benutzen wir ein eigens entwickeltes Protokoll namens `ExchangeInformation`. Dieses ermöglicht es, den Schlüssel (sogenannte `KeyEntity`, bestehend aus Key- und Meta-Informationen) und notwendige Austauschinformationen zu übertragen.

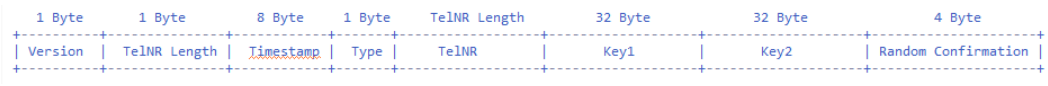

Abbildung 3.8.: `ExchangeInformation`-Protokoll Felder

1. Das Version-Feld: Dieses Feld gibt an, um welchen Austausch es sich handelt. Unsere App stell hierzu drei Kommunikationsmethoden zur Verfügung: SMS, Facebook und E-Mail. Dabei entspricht SMS dem Wert 0x01, Facebook dem Wert 0x02 und E-Mail dem Wert 0x03. In unserem Protokoll ist dieses Feld ein Byte groß, was

255 möglichen Kommunikationsmethoden entsprechen würde. Wir haben uns also genug Platz für Erweiterungen reserviert.

2. Das Primary-ID-Length-Feld: In unserem Beispiel in der Grafik entspricht dies die Anzahl an Stellen der Telefonnummer des eigenen Geräts. Die Länge wird benötigt, um in Folge die ID aus der `ExchangeInformation` entnehmen zu können. Dieses Feld ist wieder ein Byte groß. Das bedeutet, dass wir IDs bis zu 255 Zeichen unterstützen (im Falle von Telefonnummern, Facebook-IDs und E-Mail Adressen mehr als ausreichend).

3. Das Timestamp-Feld: Dieses Feld ist ein Teil der `KeyEntity` und gibt an, wann der Schlüssel erzeugt wurde. Dieser Wert wird in Millisekunden seit dem 01.01.1970 angegeben, es handelt sich demnach um eine Long-Variable. Somit entspricht die Länge des Felds acht Byte.

4. Das Typ-Feld: Ein weiteres Meta-Information Feld, welches für die `KeyEntity` benötigt wird. Dieses Feld gibt an, welcher Verschlüsselungsalgorithmus angewandt werden soll. Diese Variable wird zusammen mit den Schlüsseln gespeichert, damit beide Partner auch dasselbe Verfahren (im Falle unterschiedlicher Androidanwendungsversionen) verwenden. Die Standard-Verschlüsselung ist AES und Threefish mit einem von der Blake-Hashfunktion generierten Schlüssel. Dieses Verfahren wird durch den Wert 0x05 repräsentiert.

5. Das Primary-ID Feld: Dieses Feld entspricht der Größe, die im vorhergehenden Length-Feld definiert wurde. In diesem Feld wird die eigene Telefonnummer, im Falle von SMS, oder aber auch die eigene Facebook-ID oder E-Mail-Adresse eingetragen. Diese wird benötigt, um den Partner den „Random-Confirmation-Code (RCC)“ zuschicken zu können.

6. Die Felder `Key1` und `Key2`: Diese zwei Felder beinhalten die eigentlichen Schlüssel.

Wie bereits in erwähnt, verwenden wir zwei verschiedene Verschlüsselungsverfahren mit zwei unterschiedlichen Schlüssel. Diese werden mithilfe der Hashfunktion Blake-512 erzeugt. Wir erhalten demnach einen Hash der Länge 64-Byte. Diese 64Byte werden in zwei Teile, je 32Byte (256Bit), aufgeteilt. Somit wird die erste Hälfte für das erste Verschlüsselungsverfahren, die zweite Hälfte für das zweite Verschlüsselungsverfahren verwendet.

7. Das Random-Confirmation-Code-Feld (RCC): Der Random-Confirmation-Code ist ein 4-Byte großer Zufallswert. Für Ihn gibt es also 4294967296 mögliche Kombinationen. Dieser Code wird für die Zuordnung, aber auch für zusätzliche Sicherheit benötigt. Sobald der Partner eine `ExchangeInformation` erhalten hat, schickt er diesen Code an die Primary-ID in der `ExchangeInformation`. Das andere Gerät hat sich den selbst generierten RCC gemerkt, vergleicht ihn mit dem empfangenen und hat nun die Sicherheit, dass der Schlüsselaustausch auch mit der richtigen Person vollzogen wurde.

Umsetzung

Der Austausch läuft in folgenden Schritten ab:

1. Gerät A ist in diesem Fall der Ersteller des Codes. Das bedeutet, dass Gerät A den Schlüssel generiert und mithilfe unseres `ExchangeInformation`-Protokolls kodiert. Die erhaltenen Bytes werden in Base64 kodiert und anschließend in einen QR Code umgewandelt. In diesem Schritt erzeugt Gerät A auch den vierstelligen RCC und merkt sich diesen temporär zusammen mit den Keys. Denn Gerät A erzeugt zwar die Schlüssel, lässt den Code aber von einem beliebigen anderen Gerät scannen und weiß somit nicht, wem es diesen Schlüssel nun zuordnen soll. Gerät A weiß zu diesem Zeitpunkt also nicht, dass Gerät B den Code gescannt hat.

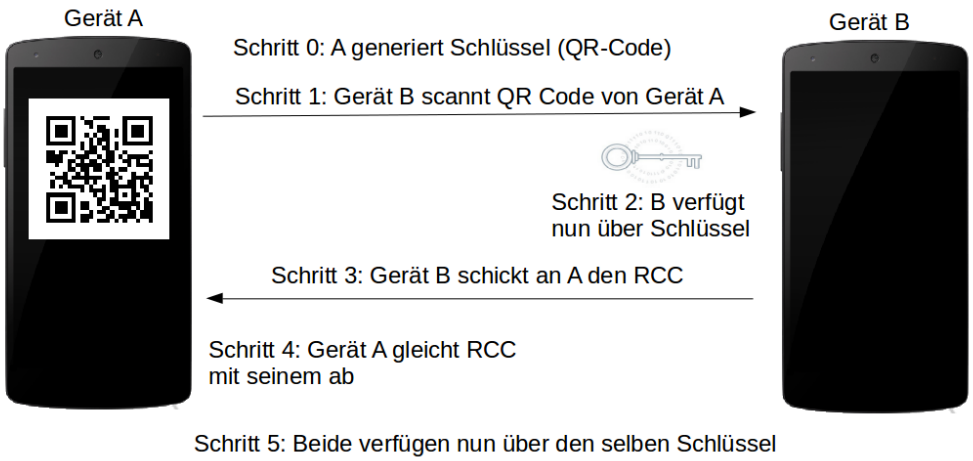

Abbildung 3.9.: Workflow QR Code

2. Nachdem Gerät B diesen Code nun gescannt hat, verfügt dieses über die Schlüssel, Metainformationen, den RCC und die Primary-ID von Gerät A. Gerät B hat nun alle benötigten Informationen, um die Schlüssel dem Partner (in diesem Fall Gerät A) zuordnen zu können. Somit werden die Keys erfolgreich in die Datenbank aufgenommen.

3. Damit Gerät A nun weiß, wem es die generierten Schlüssel zuordnen soll, sendet Gerät B komplett automatisiert im Hintergrund (somit muss der Endnutzer keine zusätzlichen Schritte durchführen) den RCC an Gerät A. Zu beachten gilt, dass nur Gerät B den richtigen RCC kennen kann, da dieser offline nur an B übertragen worden ist.

4. Gerät A empfängt nun diesen RCC über ein Medium (Beispielsweise SMS). Gerät A vergleicht diesen mit dem selbst generierten RCC. Stimmen diese überein, zieht Gerät A den Absender des RCC heran und hat somit nun eine Zuordnung zu Gerät B gefunden. Somit kann Gerät A nun erfolgreich die Schlüssel zusammen mit der Primary-ID von Gerät B in dessen Datenbank abspeichern.

5. Beide verfügen über den gleichen Schlüssel; der Austausch ist hiermit abgeschlossen.

Ein Angreifer könnte nun auf den Gedanken kommen, in diesen Prozess selbst einzugreifen. In den Schritten 3-4 wartet Gerät A auf den RCC von seinem Partner, Gerät B. Ein Angreifer, Gerät C, könnte nun versuchen, sich als Gerät B auszugeben. Aus diesem Grund haben wir den RCC so groß gewählt, dass es so gut wie unmöglich ist, den RCC zu erraten. Bei über vier Milliarden Kombinationen ist es unwahrscheinlich, dass zu einem Zeitpunkt zwei Geräte den selben RCC generieren und so eine eindeutige Zuordnung zunichte machen würden. Zudem ist ein Angreifer nicht in der Lage ist, alle vier Milliarden Möglichkeiten an sein Opfer zu schicken.

Austausch mittels NFC

NFC steht für Near-Field-Communication und ist eine Übertragungstechnologie, welche über eine kurze Distanz einen Datenaustausch ermöglicht. Hierbei ist die Rede von wenigen Zentimetern.

Zahlreiche Handelsketten rüsten mittlerweile ihre Geräte auf, um eine bargeldlose Bezahlung mit RFC-Bankomatkarten, Smartphones oder Chips zu ermöglichen. Der Endnutzer muss dabei nur seine Karte oder sein Gerät an das Lesegerät halten und kann so eine Zahlung von bis zu 25€ tätigen. Die Technologie findet sich also langsam als bequeme Zahlungsmethode in unserem Alltag ein.

Wir verwenden sie jedoch nicht für das Abfertigen von Zahlungen, sondern für den Datenaustausch. Zwar ist die Geschwindigkeit auf 424 kBit/s begrenzt, dennoch ist es für unsere Anwendung mehr als ausreichend, da wir in etwa nur 100 Byte übertragen müssen. Da mittlerweile auch fast jedes neue Smartphone über einen NFC-Chip verfügt, haben wir darin eine zweite alternative Schlüsselübertragungsmöglichkeit neben dem QR-Code gefunden.

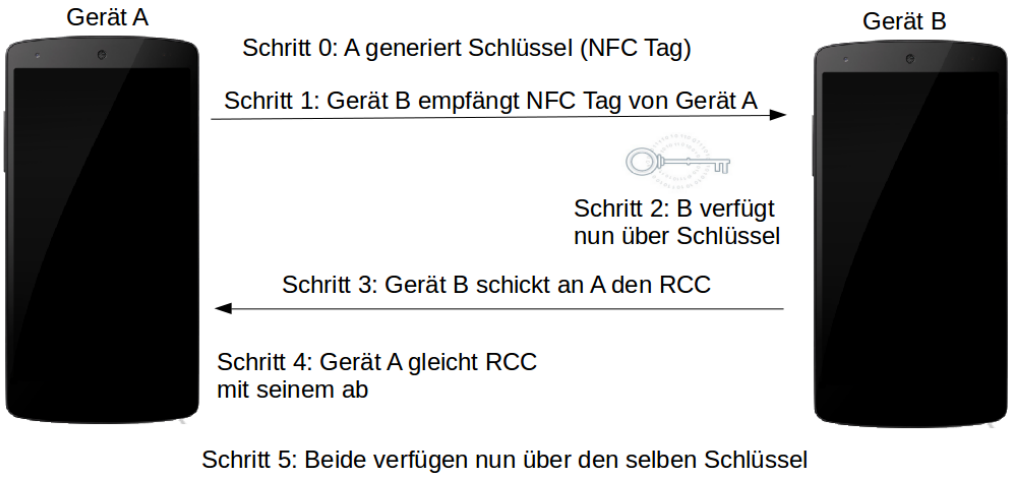

Abbildung 3.10.: Workflow NFC-Austausch

Umsetzung

Die Schritte sind vom Ablauf ident mit den Schritten des QR-Austausches. Lediglich die Umwandlung in einen QR-Code fällt weg, da wir die Base64 kodierten Bytes mit NFC direkt übertragen können.

In der Realität müssen zwei NFC-fähige Geräte nur Rückseite an Rückseite gehalten werden, um den Austausch durchzuführen.

3.1.5.4. Schlüsselspeicher

Der Schlüsselspeicher ist ein besonders kritischer Punkt in Sachen Sicherheit innerhalb der Anwendungen. Werden die Schlüssel für jede Kommunikation unzureichend gesichert, kann ein Angreifer diese auslesen und sämtliche Nachrichten wieder entschlüsseln. Aus diesem Grund haben wir sowohl starke Verschlüsselungen als auch Hashfunktionen und weitere Sicherheitsaspekte angewandt.

Anfangs ist hier die Passworteingabe beim Start der Anwendung zu erwähnen. Das sogenannte Masterpasswort muss vom Nutzer beim Startprozess eingegeben wird. Es wird beim erstmaligen Start der Anwendung vom Nutzer festgelegt und, mit einem Salt

gestärkt, verhasht. Im Anschluss wird dieser Hash als Schlüssel zum Verschlüsseln des `KeySafe`, des Schlüsselspeichers, herangezogen.

Masterpasswort erzeugen

Das vom Nutzer eingegebene Passwort wird mit einem 32-Byte-Salt kombiniert. Dieses wird zufällig durch die `SecureRandom`-Funktion in Java bei der ersten Verwendung erzeugt und gespeichert. Der Wert kann als Klartext gespeichert werden, da dieser nur benutzt wird, um Rainbowtable-Angriffe zu vereiteln.

Anschließend wird die Kombination aus Passwort und Salt mit der Hashfunktion `Scrypt` auf einen 64-Byte langen Wert gebracht. `Scrypt` ist eine Hashfunktion, die einen Wert von vordefinierter Länge erzeugt. Entwickelt und veröffentlicht wurde sie von Colin Percival im Jahre 2010. Im Gegensatz zu anderen passwortbasierten, schlüsselableitenden Hashfunktionen, wie `bcrypt`, ist die Passwortlänge nicht auf 55-Byte beschränkt. Zudem kann konfiguriert werden, wie viel Arbeitsspeicher und Prozessorzyklen beansprucht werden sollen, wodurch sich die Funktion leicht an den rasanten Fortschritt der neuen Computer anpassen lässt. [29]

Für unsere Zwecke haben wir folgende Werte für `Scrypt` gewählt:

- 4096 für den CPU- und Speicherfaktor (entspricht ~4MB Arbeitsspeicherverbrauch)
- 8 für den Speicherfaktor
- 1 für den Parallelitätsfaktor

Der Wert für den CPU- und Speicherfaktor liegt mit Absicht etwa ein Viertel unter der Standardimplementierung. Wir mussten den Wert aufgrund der schlechten Performance unter Androidgeräten herabsetzen, da selbst die neusten High-End Geräte etwa zehn Sekunden für den Hashvorgang benötigten. Mit 4096 haben wir einen Wert gefunden, der sowohl performant unter neuen als auch alten Androidgeräten läuft sowie auch die notwendige Sicherheit gewährleistet.

Wie man sieht, verlangt der Algorithmus einiges an Rechenleistung. Die Anwendung muss diesen Schritt jedoch auch nur ein einziges Mal beim Start durchführen, wodurch uns die Rechenleistung hier nicht stört. Ein Angreifer jedoch muss sämtliche Kombinationen durchprobieren und mit dieser rechenintensiven Hashfunktion berechnen, was den Vorgang nahezu unmöglich macht. Demnach müsste ein Angreifer unglaublich performante Hardware, welche mit enormen Kosten verbunden wäre, aufbringen, und sehr viel Zeit investieren, um nur ansatzweise einen erfolgreichen Angriff durchführen zu können.

Verschlüsselung des Schlüsselspeichers

Nachdem wir den Schlüssel durch die Eingabe des Masterpassworts erzeugt haben, wird dieser benutzt, um den Schlüsselspeicher zu öffnen. Im Schlüsselspeicher befinden sich sämtliche Schlüssel für alle Kommunikationen .

Dieser ist ähnlich wie eine Map aufgebaut. Zum einen haben wir einen Identifikationswert, im Fall von SMS-Nachrichten wäre das eine Telefonnummer. Für Facebook-Nachrichten wird die Facebook-ID zugeordnet und bei E-Mail-Kommunikation die E-Mail-Adresse. Der zweite Wert ist der Schlüssel in Form einer `KeyEntity`. Diese beinhaltet die Schlüssel, Informationen zum Verschlüsselungsverfahren und weitere Metadaten. Vereinfacht kann man sich den Schlüsselspeicher folgendermaßen vorstellen:

Identifikation	Schlüssel
0660123456789	SSB3aWxsIG5vdCBnaXZlIG15I1cmRlciBhbmQgS
manuel@ravenapp.org	SBrbm93IEknbGwgZ2V0IG5vIGSSdsbCBsZXQgdSc
fb.01234567890	GhlIEdvZHMgZGVjaWRlIG15IGZhdGUWwgYnkgY29tY

Tabelle 3.2.: Vereinfachter `KeySafe`

Der `KeySafe` wird komplett verschlüsselt im Programmordner gespeichert, wodurch auf Android keine andere Anwendung Zugriff auf diese Datei hat. Selbst wenn ein Zugriff auf die Datei erfolgen würde, wäre ein Angreifer nicht in der Lage, ohne dem Masterpasswort,

die Schlüssel für die einzelne Kommunikationen zu entnehmen.

Nach dem Start der Anwendung wird der Schlüsselspeicher mit dem Schlüssel geöffnet und temporär in den Arbeitsspeicher geladen. Solang der Schlüsselspeicher im RAM verweilt, kann die Anwendung die Schlüssel für die einzelnen Kommunikationen abfragen. Wird die Anwendung beendet, oder verweilt zu lange im Hintergrund, wird der zugewiesene Speicher gelöscht, und der Schlüsselspeicher muss erneut durch das Masterpasswort entsperrt werden. Das stellt unter anderem auch sicher, dass keine fremde Person mit Zugriff auf das Gerät die Nachrichten lesen kann.

Wenn ein neues Schlüsselpaar hinzugefügt werden soll, wird der Schlüsselspeicher erstmals geladen, entschlüsselt und das Paar eingefügt. Anschließend wird er wieder mit dem Schlüssel verschlüsselt und im Programmordner abgelegt. Somit ist es Hackern nicht möglich, mit einer schadhaften App die Schlüssel unverschlüsselt auszulesen.

3.2. Desktopanwendung

3.2.1. Darstellung des GUI

3.2.1.1. Vorwort

In den folgenden Abschnitten wird anhand von Screenshots die Desktopanwendung näher gebracht und diverse Funktionen erläutert. Verwendet wird ein Linux-PC, auf dem die Distribution Ubuntu läuft. Die Anwendung selbst ist auf Englisch eingestellt, um die fertige Lokalisierung zu veranschaulichen.

3.2.1.2. Startmenü

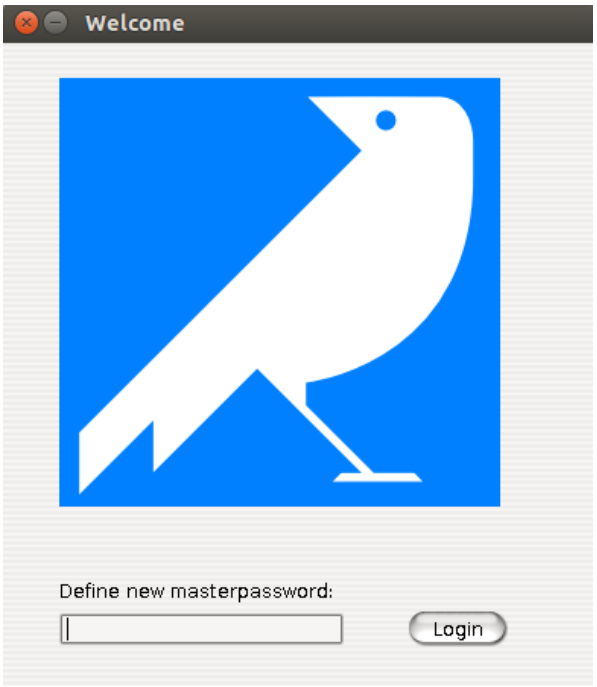

Abbildung 3.11.: Startmenü

Nach dem Starten von Raven wird man aufgefordert, ein Passwort einzugeben. Je nachdem, ob man die Anwendung zum ersten Mal startet oder bereits gestartet und ein Passwort angegeben hat, ist ein neues bzw. das zuvor definierte einzugeben. Mit

diesem werden alle benutzerspezifischen Dateien am PC verschlüsselt, sodass auch im Falle einer Gerätekompromittierung, Angreifer die privaten Daten nicht lesen kann. Nach erfolgreicher Eingabe gelangt man ins Hauptmenü der Anwendung.

3.2.1.3. Hauptmenü

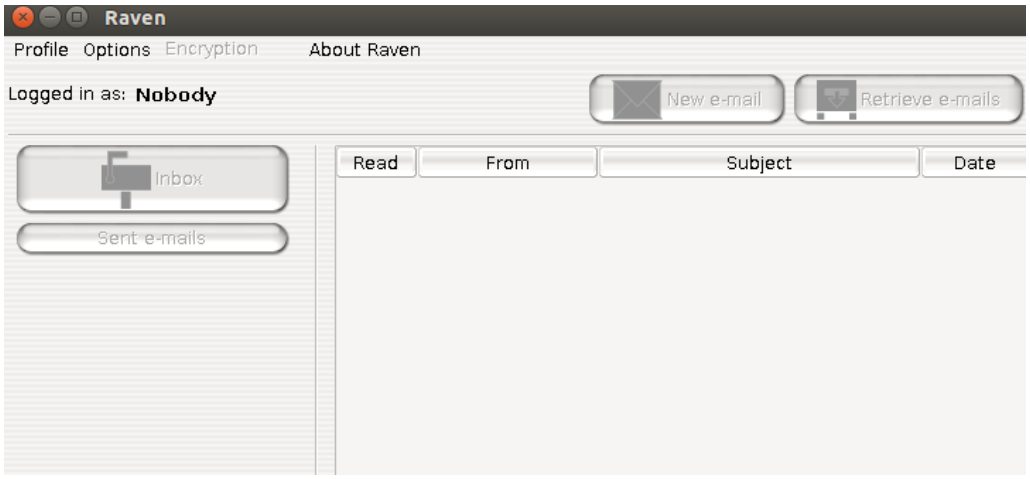

Abbildung 3.12.: Hauptmenü

Nach Eingabe des Masterpassworts sieht man das Hauptmenü der Desktopanwendung. Sollte dies zum ersten Mal geschehen oder zuvor noch kein Profil erstellt worden sein, ist man als „Nobody“ eingeloggt und dementsprechend in den Funktionen eingeschränkt.

Sämtliche Buttons sind ausgegraut, und der Benutzer muss sich für die weitere Verwendung ein Profil erstellen. Sobald das gelungen ist, können E-Mails versendet und empfangen werden. Zusätzlich ist es danach möglich, über das Menü „Encryption“ die Verschlüsselung mit einem Kommunikationspartner zu initialisieren.

In der Tabelle im unteren rechten Bereich werden jeweils die empfangenen und gesendeten E-Mails angezeigt. Sendedatum, Betreff, Sender und ob die Nachricht bereits gelesen wurde werden dabei zusätzlich angezeigt.

Selbstverständlich ist es dem Benutzer jederzeit möglich, sich mittels Klick auf „About

Raven“ näher über die Software zu informieren. Dabei wird er auf die Namen der Programmierer und die Projektwebsite aufmerksam gemacht.

3.2.1.4. Erstellen eines Profils

Abbildung 3.13.: Profilerstellung

Um nun E-Mails erhalten oder versenden zu können, ist die Erstellung eines Profils notwendig. Es sind alle angeführten Felder auszufüllen, wobei der Profilname beliebig gewählt werden kann. Mit einem eindeutigen und aussagekräftigen Namen kann man bei mehreren Profilen leichten zwischen diesen unterscheiden.

Die restlichen Felder sind gemäß den Informationen auszufüllen, die einem vom E-Mail-Adressanbieter bereitgestellt werden. Wichtig ist noch zu erwähnen, dass man beim

„Eingehenden Mail Server (Incoming mail server)“ wählen kann, ob man seine Nachrichten mittels POP3 oder IMAP abrufen möchte.

3.2.1.5. Im Hauptmenü eingeloggt

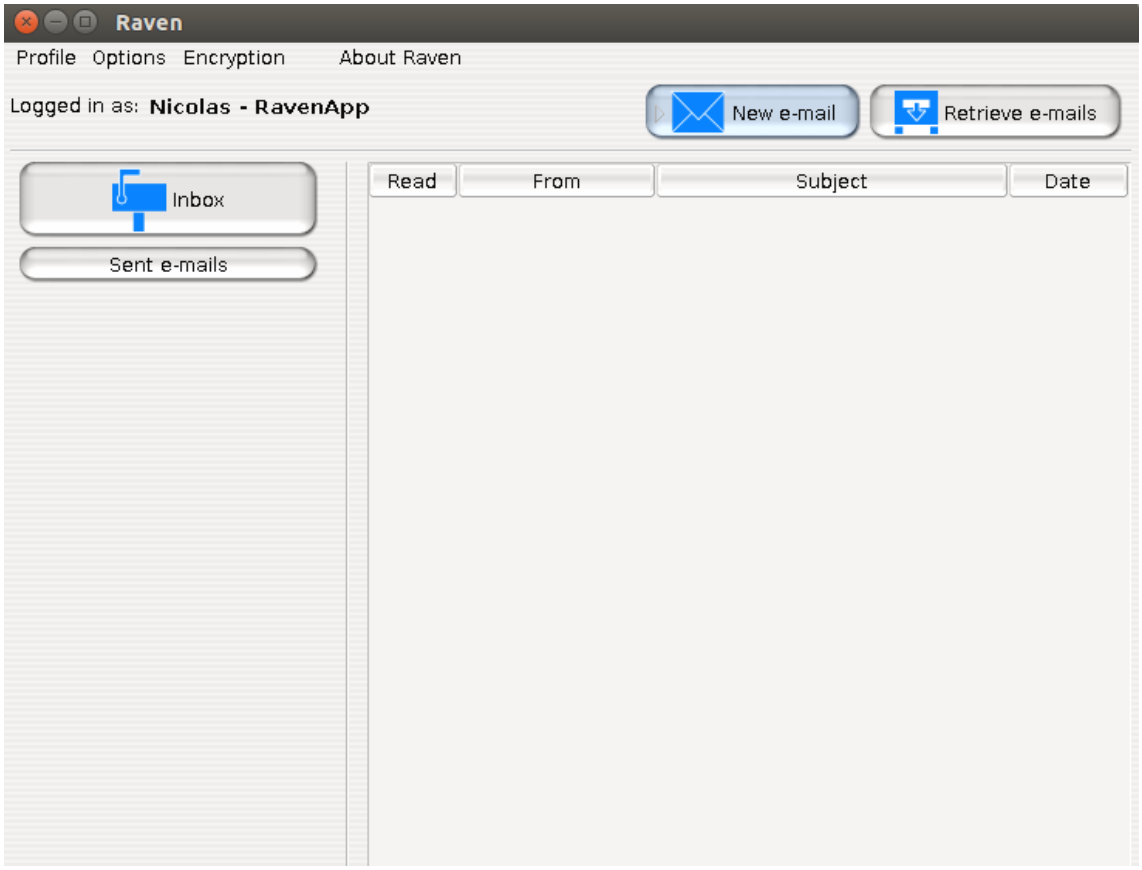

Abbildung 3.14.: Im Hauptmenü eingeloggt

Nach dem Erstellen eines Profils gelangt man wieder zum Hauptmenü, wobei sich hier nun die „Logged in as“-Anzeige ändert. Sie zeigt den eben definierten Profilnamen an und diverse Funktionen, die zuvor ausgegraut waren, sind nun aktiv. Beispielsweise sind das Schlüsselaushandlungen und das Abrufen oder Versenden von E-Mails.

3.2.1.6. Schlüsselaustauschmethoden im Hauptmenü

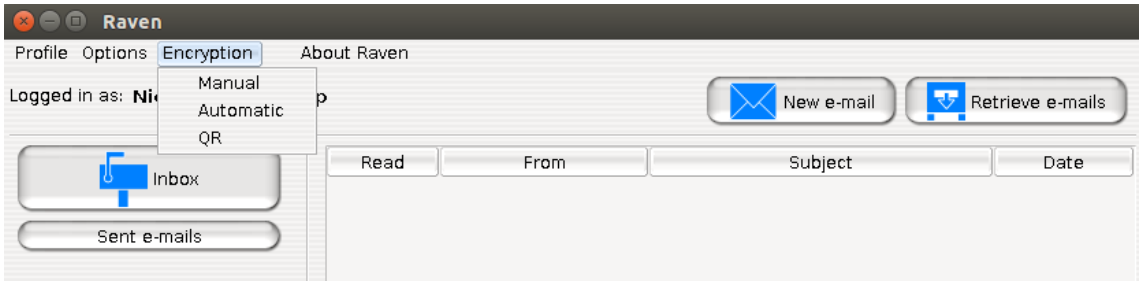

Abbildung 3.15.: Schlüsselaustauschmethoden

Nach dem Anlegen eines Profils stehen dem Beutzer weitere Auswahlmöglichkeiten zur Verfügung. So sieht man durch das Klicken auf den Reiter „Encryption“ die drei möglichen Arten des Schlüsselaustausches. „Manual“ ist ein statischer offline-Eintrag, bei dem auf beiden Seiten für das gegenüberliegende Ziel derselbe Schlüssel angegeben wird. „Automatic“ vollzieht automatisch den Schlüsselaustausch, lediglich die Partnerdresse ist anzugeben. Die letzte Auswahl, „QR“, dient zum Erstellen eines QR-Codes, der mittels Androidanwendung eingescannt werden kann.

3.2.1.7. Schlüsselaustauschmethode „Manual“

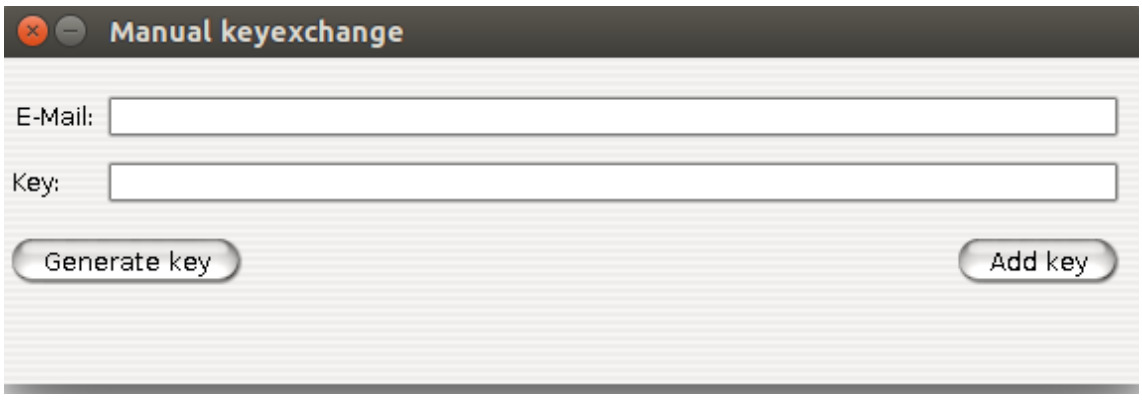

Abbildung 3.16.: Schlüsselaustauschmethode „Manual“

Bei Auswahl der „Manual-Methode“ sieht man zwei Eingabefelder. Im Feld „E-Mail“

ist die E-Mail-Adresse des Kommunikationspartners einzutragen, im Feld „Key“ wiederum der symmetrische Schlüssel. Dieser muss natürlich auf beiden Geräten gleich sein. Durch das Klicken auf den Button „Generate Key“ kann man sich auch vom Programm einen Schlüssel erzeugen lassen. Wichtig hierbei ist jedoch zu wissen, dass der erzeugte Schlüssel dann dem Kommunikationspartner (am besten über ein anderes, sicheres Medium) überreicht werden muss.

Im Endeffekt muss also derselbe Schlüssel bei beiden Gesprächspartnern eingetragen sein. Nach dem Befüllen der zwei Felder kann man mittels Klick auf „Add Key“ den Schlüssel hinzufügen.

3.2.1.8. Schlüsselaustauschmethode „Automatic“

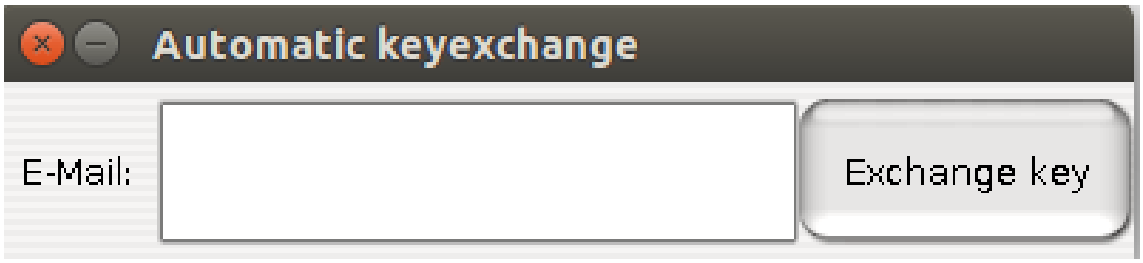

Abbildung 3.17.: Schlüsselaustauschmethode „Automatic“

Bei Auswahl der zweiten Methode, „Automatic“, wird der Schlüsselaustausch komplett automatisch durchgeführt. Es muss lediglich die E-Mail-Adresse des Gesprächspartners angegeben werden und anschließend auf „Exchange Key“ geklickt werden, siehe 3.1.5.3 auf Seite 35.

3.2.1.9. Schlüsselaustauschmethode „QR“

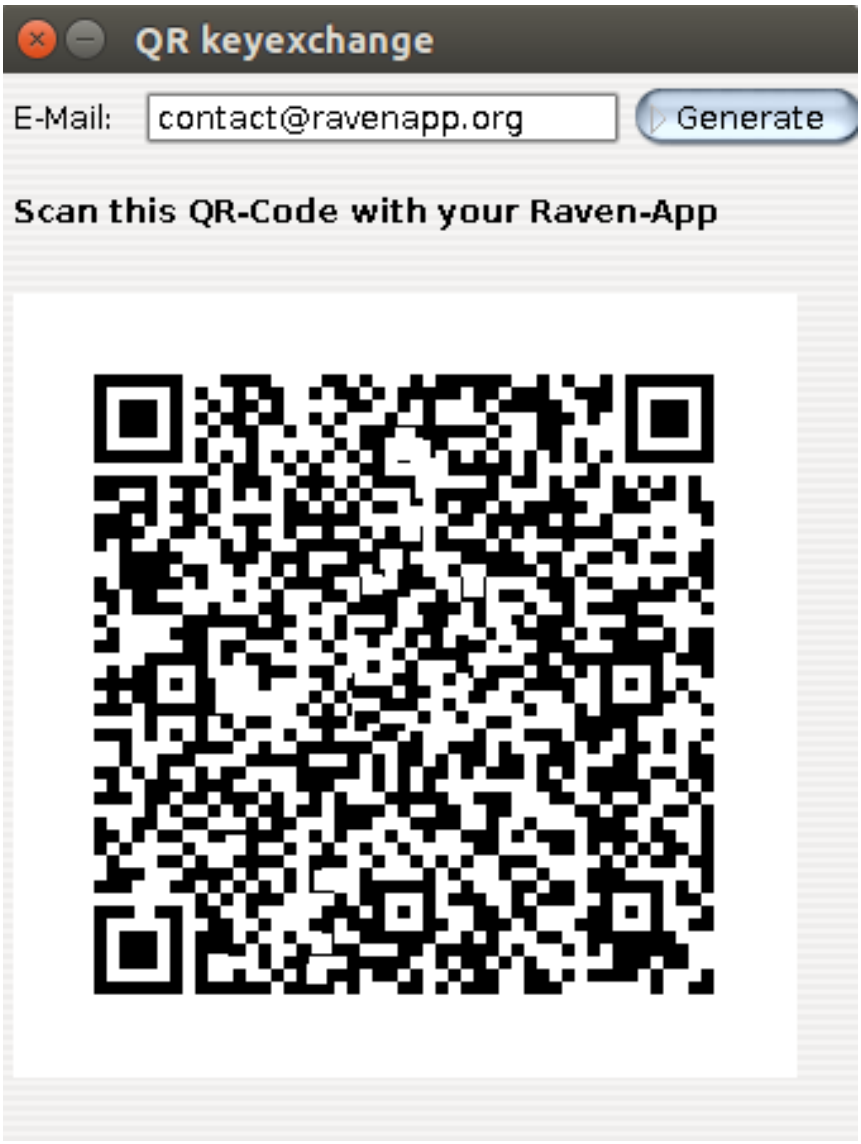

Abbildung 3.18.: Schlüsselaustauschmethode „QR-Key Exchange“

Die dritte und letzte Methode „QR-Key Exchange“ erstellt für die eingegebene E-Mail-Adresse einen QR-Code, der dann mit der Raven-App für Android eingescannt werden kann. Für nähere technische Informationen zum Ablauf des Schlüsselaustauschs siehe 3.1.5.3 auf Seite 41.

3.2.1.10. Verfassen einer E-Mail

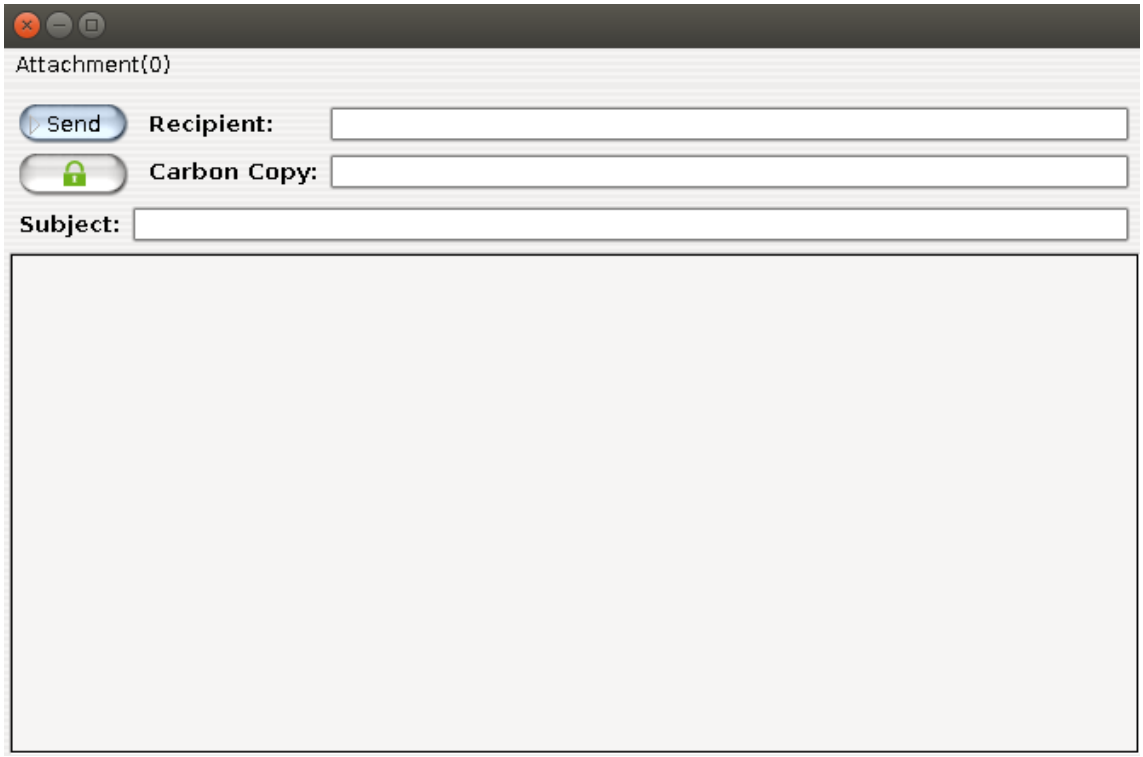

Abbildung 3.19.: E-Mail verfassen

Durch das Klicken auf „New e-mail“ im Hauptmenü öffnet sich ein neues Fenster, in dem man eine Nachricht verfassen kann. Im Feld „Recipient“ ist der Empfänger einzutragen, in „Carbon Copy“ gegebenenfalls weitere Personen, die die E-Mail erhalten sollen. Ansonsten ist nur noch das „Subject“, also Betreff, und im unteren Bereich der Text selbst auszufüllen.

Möchte man Attachments anhängen, so ist dies durch das Klicken auf „Attachment“ in der linken oberen Ecke möglich. Es öffnet sich ein Dateibrowser, in dem man eine Datei zum Anhängen auswählen kann. Für das Anhängen von mehreren Dateien muss dieser Vorgang wiederholt werden.

Der wohl wichtigste Teil ist das Schloss links von „Carbon Copy“. Ist es grün und verschlossen, so wird die E-Mail verschlüsselt versendet. Möchte man die E-Mail jedoch

unverschlüsselt senden, so kann man dies durch das Klicken auf das Schloss bewerkstelligen, wobei hier dann das Schloss grau und „geöffnet“ angezeigt wird.

3.2.2. Aufbau der Anwendung

3.2.2.1. Programmstart

Nach dem Starten der Anwendung wird man dazu aufgefordert, ein Masterpasswort einzugeben. Ist es das erste Einloggen, so muss ein neues Passwort festgelegt werden, ansonsten ist das zuvor definierte anzugeben. Dieses Passwort dient dazu, die Datei „`profiles.raven`“, die alle Profile inklusive Schlüssel und Nachrichten enthält, zu verschlüsseln und entschlüsseln. Dadurch wird der Benutzer vor Ausspionieren durch Diebstahl bzw. Gerätekompromittierung geschützt.

3.2.2.2. Benutzerinterface (GUI)

Für das Erstellen des Benutzerinterfaces kam die API und Grafik-Library Swing zum Einsatz. Durch den praktischen Architekturaufbau von Swing war es mittels „Look and Feel“ möglich, das GUI automatisch an das Betriebssystem anzupassen. Das bedeutet, dass die Komponenten des Interfaces sich selbstständig an das systemtypische Aussehen anpassen. Des Weiteren unterstützt die Anwendung nicht nur Deutsch, sondern auch Englisch als Interface-Sprache und der Wechsel zwischen den beiden benötigt keinen Neustart.

Zudem wurde das frei zugängliche Look and Feel „LiquidLookAndFeel“ [10] verwendet, um das Benutzerinterface optisch aufzuwerten. Hauptgrund dafür war, dass das standardmäßige Java Look and Feel „Metallic“ dem Team optisch nicht zusagte.

3.2.2.3. Dateien „profiles.raven“ und „salty.dat“

Um überhaupt einmal Nachrichten erhalten oder versenden, geschweige denn Schlüsselaushandlungen initialisieren zu können, muss ein Profil erstellt werden. Raven erlaubt dem Benutzer, mehrere Profile anzulegen und während der Laufzeit problemlos zwischen diesen umzuschalten.

Damit die Profile nicht bei jeder Programmbeendung neu angelegt und alle ausgehandelten Schlüssel nochmals ausgetauscht werden müssen, aber auch alle gesendeten und erhaltenen Nachrichten nicht verloren gehen, müssen alle eben genannten Inhalte noch gespeichert werden. Dies geschieht durch das automatische Speichern beim Beenden der Anwendung in die Datei „raven.profiles“.

Zusätzlich wird noch eine Datei namens „salty.dat“ angelegt, die den Salt enthält, mit dem die „raven.profiles“ Datei verschlüsselt wurde. Sie wird ebenfalls benötigt, um die Datei auch wieder entschlüsseln zu können. Beide Dateien werden im Verzeichnis der Anwendung angelegt. Das bedeutet nun, dass bei einem Verzeichniswechsel oder Systemwechsel beide Dateien mitgenommen werden müssen, da sonst die Inhalte nicht entschlüsselt werden können.

3.3. Androidanwendung

3.3.1. Allgemein

Die Anwendung wurde mit der Entwicklungsumgebung Eclipse in Java geschrieben. Dabei kamen die notwendigen Erweiterungen für die Android-Programmierung in Eclipse zum Einsatz. Für die Umsetzung wurden auch zahlreiche Libraries, wie BouncyCastle für Kryptographie und AppCompat für eine bessere Kompatibilität unter verschiedenen Androidversionen verwendet.

3.3.2. Oberfläche und Design

Uns war es ein besonderes Anliegen, die Anwendung so simpel und benutzerfreundlich wie möglich zu gestalten. Der Nutzer soll in möglichst wenigen und verständlichen Schritten verschlüsselte Nachrichten senden und empfangen können. Zudem soll er sich in diversen Diensten, wie Facebook, anmelden und leicht zwischen diesen wechseln können. Dabei haben wir uns an die neusten Android-Design-Richtlinien gehalten und unsere Oberfläche für die neue „Android-Lollipop-Darstellung“ optimiert, dem sogenannte „Material Design“.

Das „Material Design“ folgt der einfachen Philosophie, dass alles materiell in Ebenen aufgebaut wird, ähnlich wie Papier, das übereinander gelegt wird.

Bei einer Anwendung legt man sich meist auf drei verschiedene Farben fest. Zum einen die primäre Hauptfarbe, in welcher die Titelleiste dargestellt wird. Dazu kommt eine etwas dunklere Variante der ersten Farbe, die primäre dunkle Farbe, die in der Statusleiste angewandt wird. Schlussendlich folgt eine Art Kontrastfarbe die dann Textfelder, Buttons, Ladeansichten, Floating Buttons und weitere Elemente einfärbt.

Bei unserer Anwendung haben wir uns als Hauptfarbe für den Blauton `#175ea2` entschieden. Die Statusleiste wurde im Blauton `#204f84` gestaltet.

3.3.3. Umsetzung

Umgesetzt haben wir das „Material Design“ mit der Google-Library „AppCompat“. Diese ermöglicht es uns, das „Material Design“ und neue Funktionen von „Android-Lollipop“ auch unter älteren Android-Versionen nutzen zu können. Somit können wir für jede Version die selbe Oberfläche bereitstellen und eine Vielzahl an Geräten damit unterstützen.

3.3.3.1. Anmeldebildschirm

Grafisch gesehen ist der Anmeldebildschirm recht simpel gehalten. Im oberen Bildschirmbereich befindet sich zentral unser Logo. Darunter wurde das Eingabefeld für die Passworteingabe platziert und automatisch fokussiert, wodurch sich die Tastatureingabe für das Passwort automatisch öffnet. Man erspart dem Anwender demnach einen weiteren Klick und somit auch Zeit. Nach einer erfolgreichen Eingabe kann man nun einerseits durch das Drücken der Enter-Taste auf der Tastatur, aber auch durch den Pfeil direkt neben der Eingabe fortfahren.

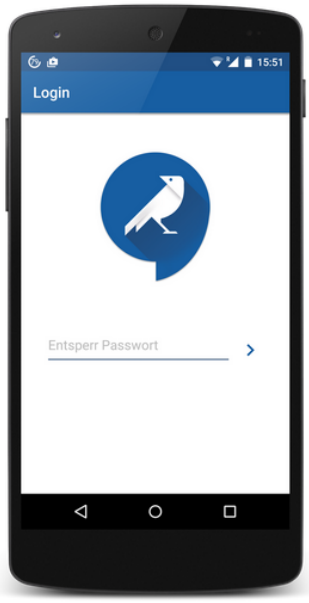

Abbildung 3.20.: Anmeldebildschirm

3.3.3.2. SMS-Ansicht

Daraufhin findet man sich im Hauptbildschirm wieder, im Falle dieser Beispielgrafik in der SMS-Ansicht. Hier werden alle Kontakte, mit denen zuletzt mittels SMS geschrieben wurde, aufgelistet. Um die Darstellung persönlicher wirken zu lassen, wird auf der linken Seite das Profilbild der Person angezeigt. Direkt im Anschluss folgt rechts der Name in etwas größerer und kräftigerer Schrift. Handelt es sich um eine verschlüsselte Kommunikation, so wird der Name der Person in Grün, gefolgt von einem Schloss dargestellt. Somit ist für den Nutzer auf einem Blick erkenntlich, ob er mit der ausgewählten Person verschlüsselt oder unverschlüsselt kommuniziert.

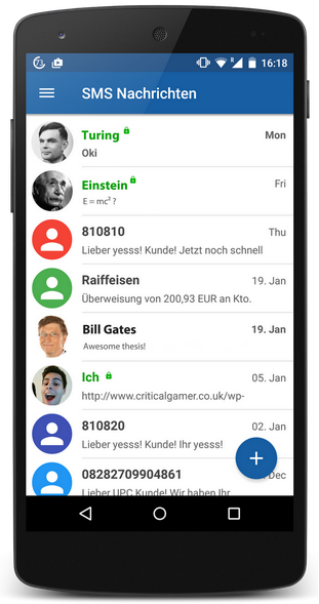

Abbildung 3.21.: SMS-Ansicht

Direkt unter dem Namen wird ein Teil der letzten Nachricht dieser Person angezeigt. Schlussendlich folgt am rechten Rand das Datum an dem die letzte Nachricht empfangen wurde. Umgesetzt wurde diese Ansicht mit einem `ListView`, also einer Listenansicht von mehreren Elementen. Diese hat den Vorteil, dass sie gleich standardmäßig scrollbar ist und sich an die Bildschirmgröße anpasst. Die Gestaltung der Elemente innerhalb der Liste wurde von uns ebenfalls nach den bereits erwähnten Richtlinien entwickelt.

3.3.3.3. `NavigationDrawer`

Inspiriert durch die Google-Design-Richtlinien haben wir die Titelleiste so frei wie möglich gehalten. Die Option, eine neue Nachricht zu verfassen, befindet sich in Folge nun am unteren rechten Bildschirmrand in Form eines sogenannten Floating-Buttons. Dieser agiert dynamisch nach Nutzerverhalten. Scrollt der Nutzer beispielsweise durch die Kontaktliste, wird der Button automatisch ausgeblendet, um den Lesefluss des Nutzers nicht zu stören. Unterbricht der Nutzer nun das scrollen, wird der Button automatisch wieder am unteren Rand eingeblendet.

Um nun bequem zwischen SMS, Facebook Nachrichten und E-Mail-Ein-/Ausgang wechseln zu können, haben wir uns für einen sogenannten `NavigationDrawers` entschieden. Dieser kann zum einen durch einen Klick auf das Symbol in der Titelleiste ausgefahren werden, aber auch durch eine Wischgeste vom linken zum rechten Bildschirmrand.

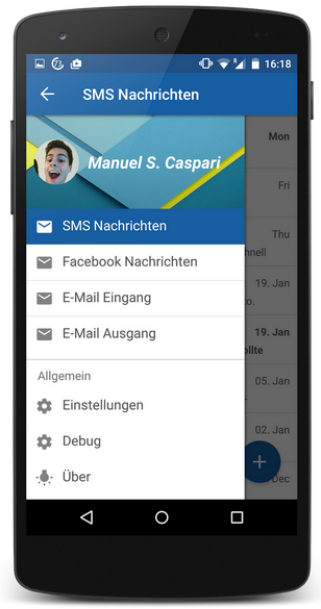

Abbildung 3.22.: NavigationDrawer

Der `NavigationDrawer` dient zum leichten Umschalten zwischen den einzelnen Kommunikationskanälen. Mit einem einfachen Klick ist es nun möglich, im Bruchteil einer

Sekunde zwischen SMS.Nachrichten, Facebook.Nachrichten oder E-Mails zu wechseln. Weiters gelangt man von dieser Ansicht in die Raven-Einstellungen oder kann sich über die Entwicklung der Anwendung näher informieren.

Aufgebaut ist dieses Menü mit einer Liste. Im oberen Bereich haben wie einen personalisierten Banner mit Name und dem Kontaktbild des Benutzers, welches entweder lokal vom Gerät oder von Facebook, geladen werden kann. Darunter sind nun die eigentlichen Elemente zum Umschalten. Man beachte zudem die Aufteilung in Kategorien, so sind die eigentlichen Dienste von den Einstellungen durch einen Trennstrich optisch getrennt.

3.3.3.4. Nachrichten-Ansicht

Die Nachrichten-Ansicht wird mittels einem Klick auf den eigentlichen Kontakt geöffnet. Der Nutzer erhält so einen Überblick über alle bisher gesendeten Nachrichten. Zudem ist das Senden neuer Nachrichten und das Aushandeln eines Schlüssels möglich. Das Verfassen einer neuen Nachricht erfolgt, wie bereits aus ähnlichen Messenger-Diensten bekannt, durch eine Eingabemöglichkeit am unteren Bildschirmrand. Zudem können Emoticons mit einem Klick auf das Smileysymbol links neben der Eingabemöglichkeit hinzugefügt werden. Bilder werden automatisch verarbeitet, wenn in der Nachricht ein Link zu einer Grafik gefunden wird. Diese wird dann in der Nachrichtenübersicht auch tatsächlich als Grafik dargestellt.

Abbildung 3.23.: Nachrichten-Ansicht

Das Schlüsselsymbol in der Titelleiste gibt an, dass es sich um eine unverschlüsselte Kommunikation handelt und man einen neuen Schlüsselaustausch initiieren kann. Dies

erfolgt durch einen Klick auf den Schlüssel, woraufhin ein weiteres Menü geöffnet wird. In diesem hat der Nutzer die Möglichkeit, zwischen den Verfahren zum Austausch zu wählen, wobei die einfachste Lösung in grüner Farbe am oberen Rand angezeigt wird.

3.3.3.5. Schlüsselaustausch-Varianten

Für erweiterte Austauschoptionen werden unter der Bezeichnung „Fortgeschritten“ die Methoden „QR-Code“ und „NFC“ gelistet. Anzumerken ist, dass ein Austausch mittels NFC nur möglich ist, wenn das Gerät auch über einen NFC-Chip verfügt. Dies wird in unserer Anwendung durch einen NFC-Banner am unteren Rand des Bildschirms signalisiert. Ist dieses nicht vorhanden, so ist auf diesem Gerät kein Austausch mittels NFC möglich. Falls ein NFC-Chip vorhanden ist, kann ein Austausch ganz einfach mittels Kontakt der Geräte erfolgen.

Beim Austausch des QR-Codes gibt es zwei verschiedene Rollen: Der Ersteller und der Empfänger. Der Ersteller kreiert einen neuen QR Code spezifisch für den Empfänger. Der Code sollte auf keinen Fall für mehrere Kommunikationen verwendet werden, da sonst jede Kommunikation über den selben Schlüssel verfügt. Der Empfänger hingegen kann den QR-Code mittels einer geeigneten QR-Reader-App, einlesen und verarbeiten.

Die einfachste Methode ist jedoch der automatische Austausch. Dabei muss sich der Nutzer um nichts weiter Gedanken machen, da die Anwendung sämtliche Aufgaben im Hintergrund ausführt. Zudem kann hier ein Schlüsselaustausch über das Internet sicher erfolgen, die Geräte müssen also nicht in ein und dem selben Raum sein. Der Partner, welcher ebenfalls die Androidanwendung installiert haben muss, antwortet dann vollkommen automatisiert im Hintergrund dem Gegenüber. Anschließend ist der Austausch abgeschlossen und beide Geräte verfügen über den selben Schlüssel.

3.3.4. Android Beam

Der Austausch mit NFC verwendet die Google-Schnittstelle „Android Beam“. Diese Schnittstelle ermöglicht den einfachen Austausch von Informationen, wie Kontakt-Informationen, Links oder aber auch andere kurze Bytefolgen.

Aktiviert wird Android-Beam indem man zwei NFC-fähige Geräte Rücken an Rücken hält. Anschließend wird auf beide Geräte die Option „Drücken um zu beamen“ gezeigt, wobei derjenige Nutzer den Druck tätigt, der die Information auf das andere Gerät übertragen möchte. Wir machen uns dieses Verfahren für den Schlüsselaustausch zunutze.

Im Grunde übertragen wir den selben Byte-Strom wie auch bei QR. Derjenige, der den Druck tätigt, überträgt dabei die `ExchangeInformation` auf das andere Gerät.

4. Ergebnisse

4.1. Desktopanwendung

4.1.1. GUI

Wie geplant ist es uns gelungen, ein intuitives Benutzerinterface für die Desktopanwendung zu erzeugen. Dieses ist nicht nur benutzerfreundlich ist, sondern passt sein Aussehen ebenfalls an das Betriebssystem an und erlaubt einen reibungslosen Sprachwechsel zwischen Deutsch und Englisch während der Laufzeit.

4.1.2. Funktionen

Die Anwendung erlaubt das verschlüsselte und unverschlüsselte Senden und Empfangen von E-Mails. Alle erhaltenen und gesendeten Nachrichten, ausgehandelte Schlüssel und erstellte Profile werden automatisch gespeichert und beim Starten nach erfolgreicher Eingabe des Masterpassworts geladen.

4.2. Androidanwendung

Im Bereich der Androidanwendung wurden folgende Ergebnisse erzielt:

- Mittels der Androidanwendung ist es dem User möglich, direkt auf seine SMS- und Facebook-Kontakte zuzugreifen. Dadurch wird ein umständliches Exportieren und Importieren der Kontakte überflüssig. Dies erhöht somit die Komfortabilität der Anwendung.
- Es ist möglich, sich mit seinem Facebook- und Emailkonto bei den jeweiligen Diensten anzumelden und diese zu verwenden. Dies erfolgt durch ein simples Loginfenster, sodass der Benutzer durch einfaches Eingeben der Kontoinformationen Zugriff auf seinen Account hat.
- Der User kann in der Anwendung Nachrichten über Email, SMS und Facebook senden und empfangen. Vorteilhaft ist hierbei, dass die Schnittstellen zu den darunterliegenden Diensten sehr transparent gehalten wurde. Aufgrund dessen geht die Verwendung dieser Funktionen ausgesprochen leicht von der Hand, und man merkt praktisch nichts von den von uns benutzten bzw. geschaffenen Interfaces.
- Die Anwendung kann Schlüssel generieren und diese mit anderen Geräten austauschen. Dabei wird der Schlüssel automatisch erzeugt. Der Austausch kann auf drei verschiedene Arten erfolgen:

    1. Automatisch - Hierbei muss sich der Benutzer, nach dem Drücken des entsprechenden Buttons, um nichts weiter kümmern und im Hintergrund wird ein Diffie-Hellman-Austausch durchgeführt (siehe 3.1.5.3 auf Seite 35).

2. QR-Code - Bei dieser Methode generiert ein Benutzer einen QR-Code, der anschließend vom Partner gescannt wird. Die Schlüsselübertragung erfolgt somit offline (siehe 3.1.5.3 auf Seite 41) .
3. NFC - Mittels dieser Variante müssen die Verschlüsselungspartner nur ihre beiden NFC-fähigen Geräte aneinander halten. Im Zuge dessen wird der Schlüssel wieder offline vereinbart (siehe 3.1.5.3 auf Seite 46).

- Dem User ist es möglich, Nachrichten durch einen vorher initiierten Schlüsselaustausch zu verschlüsseln. Dabei wird dem Benutzer offen gelassen, ob er die Nachricht verschlüsselt oder im Klartext senden möchte. Selbstverständlich merkt der User nichts von der im Hintergrund durchgeführten Ver- und Entschlüsselung.
- Die Androidanwendung unterstützt in der Facebook- und SMS-Ansicht das Senden und Darstellen von Emoticons. Zudem wird ein gesendeter Link zu einem Bild automatisch aufgelöst und das Bild wird dem Benutzer angezeigt.
- Der Benutzer kann zwischen deutscher und englischer Sprache wählen. Die Funktionen der Anwendung werden dadurch selbstverständlich nicht beeinträchtigt.

Somit ist zu sagen, dass die Androidanwendung sowohl sämtliche primären, als auch sekundären Ziele erfüllt.

4.3. Lizenzierung

Desktop- und Androidanwendung sind unter der GPLv3 veröffentlicht [30] . Dadurch ist es jedem gestattet, den Quellcode einzusehen, ihn zu modifizieren oder selbst zu verwenden. Zur Klastellung wurde jeder Quellcode-Datei folgende Präambel angefügt:

```
/** Copyright 2015
        by Philipp Adam, Manuel Caspari, Nicolas Lukaschek
    contact@ravenapp.org

    This file is part of Raven.
    Raven is free software:
        you can redistribute it and/or modify
    it under the terms of the
        GNU General Public License as published by
    the Free Software Foundation,
        either version 3 of the License, or
    (at your option) any later version.

    Raven is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY;
        without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

        See the GNU General Public License for more details.
    You should have received
        a copy of the GNU General Public License
    along with Raven.  If not, see <http://www.gnu.org/licenses/>.
**/
```

Listing 4.1: Code-Präambel

5. Evaluation und Resümee

5.1. Resümee

Rückblickend ist das Projekt als ein Erfolg zu werten. Da wir bereits aus dem Vorjahr einen Prototypen aus dem NTPR-Unterricht übernommen haben, sind wir folglich mit einem erheblichen zeitlichen und technischen Vorsprung in die Diplomarbeit gestartet. Dieser Prototyp war der Vorgänger unserer momentanen Androidanwendung und war bereits in der Lage, verschlüsselte SMS zu versenden. Zudem war bereits der Schlüsselaustausch über QR-Code und NFC realisiert.

Besagten Vorsprung haben wir ausgebaut, indem wir uns bereits in den Sommerferien um die Planung und technische Realisierung des Projekts gekümmert haben. So war beispielsweise die Grobplanung der Diplomarbeit bereits im September, also noch vor offiziellem Start des Projekts, abgeschlossen. Zudem haben wir uns intern mittels des SCRUM-Frameworks organisiert und somit den Fokus der Arbeitsenergie eher auf die technische Umsetzung, als auf das theoretische Planen gelenkt.

Folglich waren wir von Beginn an sehr flexibel aufgestellt, und unser zeitliches Budget war großzügig bemessen. Der Hauptgrund für diese zeitliche Einteilung war die Tatsache, dass wir neben der Arbeit an unserer Diplomarbeit noch dem Regelunterricht in der Schule folgen mussten. Auf diese Art und Weise konnten wir uns vor einer Schularbeit oder einem Test auch fast ausschließlich auf das Lernen konzentrieren und hatte keinen zeitlichen Druck, an der Diplomarbeit weiterzuarbeiten.

Die Koordination innerhalb des Teams und mit den Betreuern erfolgte ebenfalls außerordentlich gut. Ein hohes Maß an Zuverlässigkeit und Leistungsbereitschaft auf

beiden Seiten garantierte einen reibungslosen Projektablauf. Beispielsweise waren bei allen Meetings auch immer alle angekündigten Teilnehmer vorhanden, die Meetings haben plangemäß begonnen und wurden folglich schnell und effizient abgewickelt.

Ein weiterer Erfolgsgrund für das Projekt war sicherlich der hohe Level an Selbstständigkeit im Projektteam. Auftretende technische Probleme wurde meist mittels Selbstrecherche gelöst, und die Betreuer wurden nur bei gröberen Entscheidungen (beispielsweise wie und welche Verschlüsselungs-Libraries eingebunden werden soll) hinzugezogen. Folglich wurde der administrative Anteil am Projekt so klein wie möglich gehalten, und in den Monaten Oktober und November ist fast die Gänze an Arbeitsenergie in die technische Umsetzung geflossen.

Da wir bereits in den Monaten davor einen Großteil der „handwerklichen Programmierarbeit“ erledigt hatten und im Dezember selbst viele Tests angesetzt waren, haben wir uns hauptsächlich auf das Erstellen und Proben der vorgeschriebenen internen Zwischenpräsentation fokussiert. Aufgrund des erarbeiteten zeitlichen Spielraums konnten wir uns das erlauben und sämtliche ausstehende primären Ziele im Rahmen der Weihnachtsferien erfüllen. Bis Mitte Februar beanspruchte vor allem die öffentliche Zwischenpräsentation den Großteil unserer Zeit.

Die Folgemonate haben wir uns fast ausschließlich mit dem Schreiben dieses Buches, dem Erreichen optionaler Ziele und dem Vorbereiten auf die Abschlusspräsentation befasst.

5.2. Planung vs. Realisierung

Dass die zu Beginn eines Projekts geplanten Ziele am Ende nur teilweise erfüllt werden, oder sich im Laufe des Semesters ändern, haben wir in den letzten drei Jahren PPM miterleben können. Oftmals hatten die betreuenden Lehrer neue Ideen, Wünsche oder Abänderungsvorschläge, die es als Projektteam umzusetzen galt. Die Folge war meist Verwirrung und Stagnation im Projektfortschritt, und im Endeffekt wurden weder die alten noch die neuen Projektziele vollständig umgesetzt.

Aus dieser Erfahrung haben wir bei der Durchführung unserer Diplomarbeit geschöpft. Zu Beginn wurden die Ziele sehr vorsichtig und minimalistisch formuliert, da wir nicht zu 100% wussten, worauf wir uns einlassen. Beispielsweise aufgrund der fehlenden dauerhaften Facebook-API-Unterstützung von offizieller Seite haben wir uns das Versenden von verschlüsselten Facebook-Nachrichten als optionales Ziel gesetzt.

Es stellte sich heraus, dass diese vorsichtige Zielsetzung im Projekt genau die richtige Entscheidung war. Wir haben uns weder mit vielen, komplizierten Zielen übernommen, noch eine Vielzahl an neuen Anforderungen an das Projekt gestellt. Folglich sind wir strukturiert und ohne zeitlichen Druck an unsere selbst auferlegten Aufgaben herangegangen und haben jedes primäre Ziel im Projektverlauf umsetzen können. Glücklicherweise hat unser zeitliches Budget noch die Implementierung einiger optionaler Ziele, wie etwa das Versenden von Facebook-Nachrichten oder Email-Anhängen, zugelassen.

5.3. Persönliche Eindrücke, Lessons Learned

5.3.1. Philipp Adam

Ich persönliche habe dieses Projekt als eine Möglichkeit gesehen, das erworbene, theoretische Grundlagenwissen der letzten fünf Jahre sinnvoll und durch die Arbeit an einem konkreten Produkt einzusetzen. Im Zuge dessen habe ich mich vor allem mit der E-Mail-Library in Java beschäftigt und auf diesem Gebiet dazugelernt. Das Ergebnis war eine, im Vergleich zu einer im Regelunterricht erstellten, relativ umfangreiche Anwendung. Die mit der wachsenden Größe der Software einhergehenden Probleme waren für mich ebenfalls eine neue Erfahrung.

Weiters ist mir bewusst geworden, dass die Basis eines erfolgreichen Projekts die Planung und das Management sind. Leider ist, aufgrund der Größe der Projekte, die Planung im PPM Regelunterricht nie wirklich zum Tragen gekommen und wurde eher zur Pflichtübung degradiert.

Bei unserer Diplomarbeit hat uns die umsichtige und flexible Planung allerdings sehr viele Vorteile gebracht. Dabei habe ich auch festgestellt, dass das Management zwar wichtig ist, es allerdings nur als Mittel zum Zweck eingesetzt werden sollte. Bei einem zu großen Fokus auf die Administration kann das Projekt, besonders bei einem kleinen Team von zwei bis vier Personen, durchaus „zu Tode geplant“ werden. Dies äußerst sich meist darin, dass die Teammitglieder länger mit der Definition von Arbeitspaketen oder dem Schreiben von Berichten beschäftigt sind als mit der eigentlichen technischen Umsetzung.

5.3.2. Nicolas Lukaschek

Die Diplomarbeit stellte für mich das erste wirkliche Programmierprojekt dar, das komplett bis zum Ende durchgezogen und erfolgreich abgeschlossen wurde. Dieses bot nicht nur die Möglichkeit, das gesamte Wissen der letzten Jahre umzusetzen, sondern ebenfalls

einige neue Erfahrungen zu sammeln.

Dazu kam noch die positive Erfahrung der Interaktion innerhalb des Teams. Dank guter Planung und Einsatz und Motivation war es uns möglich, die Android- und PC-Anwendung früh fertigzustellen, sodass uns genug Zeit für das Diplomarbeitsbuch blieb. Die Meetings in der Schule wurden stets sinnvoll genutzt, von allen Teilnehmern ernst genommen und erhaltenes Feedback immer verarbeitet.

Interne Konflikte gab es selten bis kaum und wenn, konnten diese schnell ohne Blutvergießen gelöst werden. Rückblickend war also ein konstantes und konzentriertes Arbeiten sehr von Vorteil, da in den entscheidenden letzten Monaten nur wenig Leistung für die Diplomarbeit erbracht werden musste. Da der Großteil der Arbeit bereits zuvor fertig gestellt wurde, konnten wir uns im „Endspurt“ voll und ganz auf die finalen Tests und Überprüfungen vorbereiten.

5.3.3. Manuel Caspari

Im Zuge dieses Projektes standen wir vor vielen Herausforderungen, die wir aber alle aufgrund der guten Planung und unseres Ehrgeizes bewältigen konnten. Jede neue Herausforderung sah ich als Möglichkeit an, mein Wissen zu stärken und meine Fähigkeiten im Bereich der Programmierung zu verbessern. Ohne diesen Ehrgeiz und die Motivation unseres Teams wäre das Projekt vermutlich nie ein solcher Erfolg geworden.

Ich habe mich im Zuge dieses Projekts vor allem intensiv mit Kryptographie, NFC, Android und der Facebook-Schnittstelle befasst und diese auch umgesetzt. Gerade die Kryptographie hat mich schon immer sehr interessiert, doch hatte ich vor dem Projekt nie die Chance, das Wissen im großen Stil anwenden zu können. Das gab mir auch die Möglichkeit, mich mit zahlreichen neuen Verschlüsselungsverfahren und Hashfunktionen zu befassen, welche mir vorher noch nicht einmal bekannt waren.

Obwohl ich zuvor schon an zahlreichen kleineren Apps und Spielen gearbeitet habe, war

5. Evaluation und Resümee

Raven mein erstes großes Projekt unter Android. Ich konnte einen komplett neuen Einblick in die Planung und dessen Umfang erlangen und daraus neues Wissen für kommende größere Projekte sammeln.

6. Ausblick

Das Team möchte die beiden Anwendungen kostenlos der Öffentlichkeit zur Verfügung stellen. Die Androidanwendung wird über den Google Playstore, die Desktopanwendung über die Projektwebsite (http://www.ravenapp.org) gratis erhältlich sein. Es ist geplant, dass wir die Software noch einige Zeit (ca. sechs Monate) nach Abschluss der Diplomarbeit supporten und einige Bugs fixen werden. Allerdings werden wir keinen mehrjährigen, langfristigen Support für die Anwendungen anbieten. Ebenfalls ist nicht geplant, dass neue Funktionen nach Abschluss der Diplomarbeit in die Software implementiert werden.

A. Anhang

TOFT-Plakat

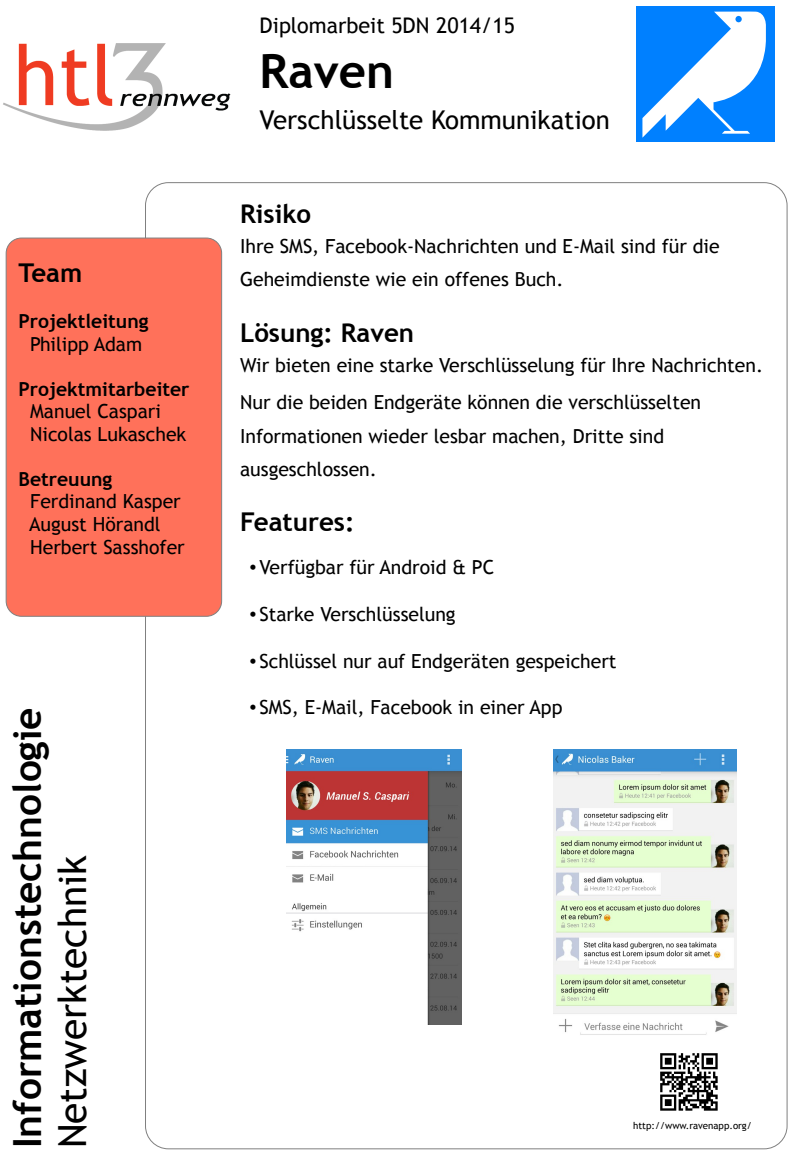

Abbildung A.1.: Plakat für den TOFT

Autorenvermerk

Person	Erstellte Abschnitte inklusive Unterabschnitte
Philipp Adam	— Kurzfassung — Abstract — Vorwort — 1. Einleitung — 2. Planung — 3.1.4 Verwendete Libraries — 4.2 Androidanwendung — 4.3 Lizensierung — 5.1 Resümee — 5.2 Planung vs. Realisierung — 5.3.1 Persönlicher Eindruck — 6 Ausblick
Manuel Caspari	— 3.1.3 Facebook-Nachrichten — 3.1.4 Verwendete Libraries — 3.1.5. Core-Library — 3.3.Androidanwendung — 5.3.3 Persönlicher Eindruck
Nicolas Lukaschek	— 3.1.1 E-Mail — 3.1.2 SMS — 3.2 Desktopanwendung — 4.1 Desktopanwendung — 5.3.2 Persönlicher Eindruck

Tabelle A.1.: Autorenvermerk

B. Quellenverzeichnis

[1]

Wikipedia: BLAKE (2015)
https://en.wikipedia.org/wiki/BLAKE_(hash_function) , [5.2.2015]

[2]

Barrel, Richard: How does a cryptographically secure random number generator work?(2010)
http://stackoverflow.com/questions/2449594
/how-does-a-cryptographically-secure-random-number-generator-work , [3.2.2015]

[3]

Wikipedia: AES (2015)
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard , [5.2.2015]

[4]

Wikipedia: Threefish (2015)
https://en.wikipedia.org/wiki/Threefish , [6.2.2015]

[5]

Schneier, Bruce: Will Keccak = SHA-3? (2013)
https://www.schneier.com/blog/archives/2013/10/will_keccak_sha-3.html , [7.2.2015]

[6]

Wikipedia: Diffie Hellman key exchange (2015)
https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange, [24.2.2015]

[7]

Wikipedia: Primitive root modulo n (2015)
https://en.wikipedia.org/wiki/Primitive_root_modulo_n , [27.2.2015]

[8]

Bitcoin Wiki: Secp256k1 (2014)
https://en.bitcoin.it/wiki/Secp256k1 , [2.3.2015]

[9]

Google: Material design (2015)
https://www.google.com/design/spec/material-design
/introduction.html#introduction-goals, [10.03.2015]

[10]

Sourceforge: Liquid Look and Feel (2015)
http://sourceforge.net/projects/liquidlnf/http://sourceforge.net/projects/liquidlnf/, [5.11.2014]

[11]

Crispin, Mark: Internet Message Access Protocol (2003)
https://tools.ietf.org/html/rfc3501, [26.3.2015]

[12]

Klensin, John: Simple Mail Transfer Protocol (2001)
https://www.ietf.org/rfc/rfc2821.txt, [26.3.2015]

[13]

Meyers, Mellon, Rose: Post Office Protocol(1996)
https://www.ietf.org/rfc/rfc1939.txt, [26.3.2015]

[14]

Ho Chi Minh: Emojicon(2015)
https://github.com/rockerhieu/emojicon, [6.4.2015]

[15]

Melnykov, Oleksandr: Floating Action Button(2015)
https://github.com/makovkastar/FloatingActionButton, [6.4.2015]

[16]

Navas, Ivan: Material Design Library (2015)
https://github.com/navasmdc/MaterialDesignLibrary, [6.4.2015]

[17]

Hedley, Jonathan: Jsoup (2015)
http://jsoup.org/, [6.4.2015]

[18]

Humphrey, Charlton, Hewitt: Remote-Stacktrace (2015)
https://code.google.com/p/android-remote-stacktrace/, [6.4.2015]

[19]

Thiel,Alexandre: Java Mail for Android (2015)
https://code.google.com/p/javamail-android/, [6.4.2015]

[20]

Google: Appcompat (2015)
https://developer.android.com/tools/support-library/features.html, [6.4.2015]

[21]

Google: Google Play Services (2015)
https://developer.android.com/google/play-services/index.html, [6.4.2015]

[22]

Smith, James: Android Async HTTP(2015)
http://loopj.com/android-async-http/, [6.4.2015]

[23]

mikeai, xendren: Liquid Look and Feel(2015)
http://sourceforge.net/projects/liquidlnf/, [6.4.2015]

[24]

Tyley, Roberto: Spongeycastle(2015)
https://rtyley.github.io/spongycastle/, [6.4.2015]

[25]

Owen, Sean: ZXING(2015)
https://github.com/zxing/zxing, [6.4.2015]

[26]

Apache: Apache Commons Validator(2015)
https://commons.apache.org/proper/commons-validator/, [6.4.2015]

[27]

Hamburg, Michael: Understanding Intel's Ivy Bridge Random Number Generator(2012)
http://electronicdesign.com/learning-resources
/understanding-intels-ivy-bridge-random-number-generator, [16.3.2015]

[28]

goodreads: Benjamin Franklin (2015)
http://www.goodreads.com/quotes
/140634-those-who-surrender-freedom-for-security-will-not-have-nor, [18.4.2015]

[29]

Wikipedia: Scyrpt (2015)
http://de.wikipedia.org/wiki/Scrypt , [18.4.2015]

[30]

GNU: GNU-Lizenzen: Tipps (2015)
http://www.gnu.org/licenses/gpl-howto.html,[19.4.2015]