HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Institute for
Internet Technologies
and Applications

frontg8

Chatsystem mit hohen Anforderungen an die Privatsphäre

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2015

Autoren: Tobias Stauber
Ueli Bosshard
Betreuer: Prof. Dr. Andreas Steffen

Projekt: frontg8

Chatsystem mit hohen Anforderungen an die Privatsphäre

Studienarbeit Herbstsemester 2015

Tobias Stauber - tstauber@hsr.ch
Ueli Bosshard - ubosshar@hsr.ch

Änderungsgeschichte

Datum	Version	Änderung	Autor
19.12.2015	1.0		Ueli Bosshard und Tobias Stauber

Inhalt

Aufgabenstellung Semesterarbeit frontg8

Studiengang	Informatik (I)
Semester	HS 2015/2016 (14.09.2015-14.02.2016)
Durchführung	Studienarbeiten
Fachrichtung	Sicherheit
Institut	ITA: Internet-Techn. und Anwend.
Betreuer	Steffen, Andreas
Gruppe	Tobias Stauber und Ueli Bosshard

Allgemeine Beschreibung

Unsere Lösung für ein betreiberunabhängiges Text-Chatsystem besteht aus einer Serverkomponente für POSIX konforme Systeme und einer Desktop-Clientkomponente.

Der Server kümmert sich ausschliesslich um den Austausch von Text-Messages und wird bewusst sehr schlank gehalten. Dabei ist uns wichtig, dass diese zentrale Stelle möglichst keine Angriffsfläche bietet, d.h. es werden keine Informationen gespeichert, die nicht absolut nötig sind. Die Nachrichten werden in einem flüchtigen Puffer für maximal 3 Tage aufbewahrt.
Der Client ist eine CLI-Applikation und kümmert sich komplett um die Ende-zu-Ende Verschlüsselung und die Identifizierung des Kommunikationspartners mittels Zertifikaten. Er verfügt über ein Adressbuch, damit die „anonymen" Kommunikationspartner identifiziert und überprüfte Zertifikate zugeordnet werden können.

Unser Zielpublikum sind geschlossene Benutzergruppen mit sehr hohen Anforderungen an die Privatsphäre.

In der SE2-Projektarbeit wurden bereits der Server und der CLI-Client realisiert.

Ziel der Semesterarbeit

Ziel der Semesterarbeit ist es, einen Android-Client zu entwickeln welcher mit dem bestehenden Server funktioniert. Der Funktionsumfang des Android-Clients soll mindestens dem des CLI-Client entsprechen.

Konkret:
- Verwalten von Kontakten
- Importieren von Private-Keys
- Auflisten von Konversationen
- Messages verschlüsseln/senden und empfangen/entschlüsseln

Optionale Aufgaben:
- Blacklisting (Sperren von Keys, erfordert eine Erweiterung des Servers)
- Gruppenchat
- Key-Rotation (Wechsel der Sessionschlüssel)

1. Abstract

Frontg8 ist ein, aus einer Server- und einer Clientkomponente bestehendes, Chatsystem. Dabei fokussiert sich frontg8 primär auf die anonymisierte Kommunikation zwischen den Anwendern. Um dabei zu gewährleisten, dass die Anonymität gewahrt bleibt muss der verwendete Kanal nicht sicher sein. Deshalb kann die Kommunikation zum Beispiel auch übers Internet erfolgen.

Um die Privatsphäre zu wahren, ist es unerlässlich, jede beteiligte Komponente (Clients und Server) jederzeit vollständig unter eigener Kontrolle zu haben. Jede im System teilnehmende Person verfügt über ein aus öffentlichem Schlüssel (Publickey) und privatem Schlüssel (Privatekey) bestehendes Schlüsselpaar, welches eingesetzt wird um die Kommunikationspartner zu identifizieren und auch dazu die gemeinsamen Geheimnisse zu erarbeiten, welche für das Verschlüsseln und Signieren der Nachrichten verwendet werden.

Um die Anonymität zu wahren, ist es unerlässlich, dass der Server nur so wenig wie möglich über die teilnehmenden Parteien in Erfahrung bringen kann. Durch diese Anforderung entsteht die Limitation, dass der initiale Schlüsselaustausch nicht über den Server durchgeführt werden kann. Die Teilnehmer müssen also ihre Publickeys auf einem anderen Weg (Out-of-Band) austauschen. Wenn zwei Partner ihre Publickeys ausgetauscht haben, erarbeitet frontg8 daraus mittels Diffie–Hellman auf elliptischen Kurven (ECDH) je einen gemeinsamen symmetrischen Schlüssel für die Kryptographie und für das Unterschreiben der Nachrichten.

In Software Engineering 2 – Projekt (SE2P) wurden von uns (Ueli Bosshard, Felix Morgner und Tobias Stauber) bereits ein Server und ein Python-Client umgesetzt. In dieser Studienarbeit (SA) war es nun das Ziel, einen vollumfänglich kompatiblen, auf dem Mobile-Betriebssystem Android lauffähigen, Client zu entwickeln. Im Zentrum der Arbeit stand es, die Privatsphäre zu wahren, sichere Verschlüsselung einzusetzen und eine einfache Bedienung zu ermöglichen. Dazu setzen wir zeitgemässe Verschlüsselungsalgorithmen und aktuelle API-Funktionen ein.

Da Android einige interessante Möglichkeiten anbietet geheime Schlüssel zu speichern und zu verwalten, setzten wir uns im Verlauf der Arbeit intensiv damit auseinander um zu ergründen, ob es die angebotene Funktionalität erlaubt, den von uns gewünschten Funktionsumfang direkt mit Android-Bordmitteln umzusetzen. Das vollendete Produkt ist eine vollumfängliche Android Applikation, welche alle unsere Anforderungen erfüllt und es erlaubt, sichere und voll anonymisierte Kommunikation mit anderen Teilnehmern zu führen.

2. Management Summary

2.1 Ausgangslage

Als Ausgangslage durften wir auf das Resultat unseres Software Engineering 2 Projekt (SE2P) zurückgreifen. Dieses ist ein Chatsystem mit einem Client und einem Server. Dabei wurde die Referenzimplementierung eines Clients, ein Server und das verwendete Protokoll erarbeitet.

Ziel dieser Semesterarbeit war es nun einen entsprechenden Android-Client zu entwickeln, damit das System auf einem Android-Smartphone eingesetzt werden kann.

2.2 Vorgehen

Zu Beginn der Arbeit wurde die Aufgabenstellung und der Umfang festgelegt. Danach wurden Arbeitspakete und Meilensteine definiert aus welchen wir einen groben Wochenplan erarbeiteten.

Da wir uns nicht mehr stark mit der Architektur von Frontg8 auseinander setzen mussten, konnten wir uns gleich von Anfang an mit den Android-spezifischen Problemstellungen beschäftigen. Dabei haben wir uns einiges neu angeeignet, konnten aber auch auf unser Android-Wissen aus dem Modul «Userinterface and Design» sowie auf das Fachwissen von Mirko Stocker und Tobias Brunner (Zwei Dozenten der Technischen Hochschule Rapperswil) zurückgreifen.

In der Planungsphase kamen den beiden Haupteigenschaften (Sicherheit und Anonymität) besondere Beachtung zu.

2.3 Technologien

Den Server konnte ohne Anpassungen direkt aus dem SE2P übernommen werden.

Für die Kommunikation zwischen Server und Client setzen wir auf eine verschlüsselte Verbindung. Die Nachrichten werden in ein selbst definiertes Protokoll verpackt und verschlüsselt. Die Schlüssel generieren wir aus dem geheimen Teil des eigenen Schlüssels und dem öffentlichen Teil des Schlüssels des Kommunikationspartners.

Auf dem Android-Client setzen wir hauptsächlich auf das Android Framework. Zusätzlich binden wir noch eine Bibliothek ein um das Protokoll zu spezifizieren (Google Protobuf) und eine weitere um die Unzulänglichkeiten der Android-internen Verschlüsselungsbibliothek zu vermeiden (Spongy Castle).

2.3.1 Google Protobuf

Protobuf ist eine von Google entwickelte Bibliothek zur Serialisierung strukturierter Daten. Google bietet Implementationen für Java, C++ und Python an. Da die Bibliothek quelloffen ist, gibt es jedoch auch für diverse andere Sprachen, wie unter anderem C, Ruby und Haskell, Implementationen. Protobuf ist unter der 3-Klausel-BSD-Lizenz veröffentlicht.

Protobuf legt den Fokus auf Einfachheit und Leistung. Protobuf ist über Sprachgrenzen hinweg konsistent gehalten und erleichtert so den Wechsel zwischen verschiedenen Programmiersprachen.

Wir setzen Protobuf ein, da es eine weit verbreitete und Plattformübergreifende Bibliothek ist. Zudem ist das Projekt lebendig und entwickelt sich kontinuierlich weiter.

2.3.2 Spongy Castle

Als „Hauseigene“ Verschlüsselungsbibliothek bietet die Android Platform leider nur eine im Funktionsumfang stark reduzierte Version von Bouncy Castle an, wir haben deshalb zusätzlich Spongy Castle in unser Andoid-Projekt eingebunden, welches eine vollwertige Implementation von Bouncy Castle beinhaltet.

2.4 Ergebnisse

Die Arbeit hat zwei Ergebnisse geliefert. Einerseits den funktionsfähigen Android-Client und andererseits das Android-Know-How das dazu nötig war. Insbesondere haben wir uns mit den Krypto-Funktionen und den Mechanismen für das Halten von Daten im Hintergrund auseinander gesetzt. In diesem Dokument sind die wesentlichen Erkenntnisse festgehalten.

3. Einleitung und Übersicht

3.1 Motivation

Auf der Suche nach einer Text-Chat-Lösung zwischen mobilen Geräten (Bsp. WhatsApp oder Threema) haben sich folgende Kriterien herauskristallisiert, die uns wichtig erscheinen:

1. Sichere Ende-zu-Ende verschlüsselte Kommunikation und identifizierbare Partner
2. Die Serverkomponente muss selber betreibbar sein
3. Quellcode offen, d.h. kann eingesehen und überprüft werden
4. Unterstützung für mehrere Geräte pro Benutzer

Unser Zielpublikum, wozu wir uns selbst zählen, sind geschlossene Benutzergruppen mit dem Bedürfnis die Privatsphäre zu wahren. Uns war keine Lösung bekannt, die diese Kriterien erfüllt. Zudem reizte es uns, ein Text-Messaging-System zu kreieren, über welches eine solche Gruppe vollständige Kontrolle besitzt. Also beschlossen wir selbst in diese Lücke zu springen.

3.2 Idee

Unsere Lösung für ein betreiberunabhängiges Text-Chatsystem besteht aus einer Serverkomponente und einer Clientkomponente.
Der Server kümmert ausschliesslich um den Austausch von Text-Messages und wird bewusst sehr schlank gehalten. Dabei ist uns wichtig, dass diese zentrale Stelle möglichst keine Angriffsfläche bietet, d.h. es werden keine Informationen gespeichert, die nicht absolut notwendig sind. Die Nachrichten werden in einem flüchtigen Puffer für maximal 3 Tage aufbewahrt.
Der Client kümmert sich komplett um die Ende-zu-Ende Verschlüsselung und die Identifizierung des Kommunikationspartners mittels Zertifikaten. Er verfügt über ein Adressbuch, damit die „anonymen“ Kommunikationspartner identifiziert und überprüften Zertifikate zugeordnet werden können.
Technologisch konzentrieren wir uns auf sichere Ende-zu-Ende Kommunikation zwischen den Teilnehmern.

3.3 Vergleich mit anderen Produkten

Alice und Bob möchten miteinander kommunizieren. Als Kommunikationsweg möchten sie gerne das Internet benutzen, da es praktisch weltweit und zu jeder Zeit verfügbar ist. Für beide steht es im Vordergrund, dass ihre Nachrichten geheim gehalten und ihre Anonymität gewahrt bleibt.

Alice und Bob erwägen verschiedene existierende Systeme zum Austausch von Nachrichten:

- WhatsApp: Dieses System verfügt (noch) nicht über Ende-zu-Ende-Verschlüsselung[1] was bereits die erste Anforderung (Geheimhaltung) verletzt. Des Weiteren ist WhatsApp dafür bekannt, Meta-Daten, wie MAC-Adressen und IMEIs, zu sammeln[2], was Anforderung zwei (Anonymität) verletzt. Erschwerend kommt hinzu, dass WhatsApp keine OpenSource-Software ist, und somit nicht überprüft werden kann, ob das System Design-Schwächen enthält bzw. es besteht keine Möglichkeit, solche Schwächen zu beheben.
- TextSecure: TextSecure verwendet starke Ende-zu-Ende-Verschlüsselung was bereits Anforderung eins erfüllt. Jedoch ist Anforderung zwei auch bei TextSecure nicht erfüllt, da es standardmässig die Übertragung von Telefonbuchkontakten[3] verwendet um eine Initiale Kontaktaufnahme durchzuführen.
- Threema: Auch Threema verwendet starke Ende-zu-Ende-Verschlüsselung[4]. Doch auch Threema verwendet Telefonbuchkontakte als Möglichkeit zum Auffinden von Kommunikationspartnern, was wiederum Anforderung zwei verletzt.

Keines dieser Systeme kann gleichzeitig die Kernanforderungen von Alice und Bob erfüllen.

Frontg8 fokussiert sich darauf oben erwähnte Anforderungen zu Lösen. Die Benutzer des Systems sind jeweils im Besitz eines Public-Private-Keypairs welches sie zur gegenseitigen Identifikation und zum Erarbeiten eines gemeinsamen Schlüssels verwenden. Momentan wird der Schlüsselaustausch nicht über den Server durchgeführt. Alice und Bob müssen ihre Schlüssel über einen anderen Weg (Out-of-Band) austauschen. Die empfohlene Methode dafür ist es, sich physisch zu treffen. Andere Möglichkeiten, wie zum Beispiel der Austausch über einen anderen sicheren Kanal (z.B. PGP-Mail), werden nicht ausgeschlossen, erfordern jedoch die Sicherstellung des Vertrauens in das verwendete Austausch-System.
Beide Clients ermitteln mittels ECDH ein gemeinsames Geheimnis und erzeugen daraus mittels PRF ein gemeinsamen symmetrischen Verschlüsselungs-Schlüssel und einen Schlüssel für den MAC.

[1] http://www.heise.de/newsticker/meldung/WhatsApp-bekommt-Ende-zu-Ende-Verschluesselung-2459903.html
[2] https://www.datenschutzbeauftragter-info.de/whatsapp-und-datenschutz-antworten-auf-die-wichtigsten-fragen/
[3] https://www.psw-group.de/blog/textsecure-die-quelloffene-verschluesselte-alternative/1223
[4] https://threema.ch/press-files/cryptography_whitepaper.pdf

3.4 Vorarbeit

Im Rahmen einer SE2-Projektarbeit wurde bereits die Serverkomponte, sowie ein Desktop-Client umgesetzt. Ausgeliefert wurden die Quelltexte zu Server und Client, Installations- und Konfigurationsanleitung, sowie eine Architekturbeschreibung und Protokollspezifikation.
Der Server, geschrieben in C++, kümmert sich ausschliesslich um den Austausch von Text-Messages und wird bewusst sehr schlank gehalten. Dabei ist uns wichtig, dass diese zentrale Stelle möglichst keine Angriffsfläche bietet, d.h. es werden keine Informationen gespeichert, die nicht absolut notwendig sind. Die Nachrichten werden in einem flüchtigen Puffer für maximal 3 Tage aufbewahrt.
Der in Python geschriebene Desktop-Client ist eine CLI-Applikation und kümmert sich komplett um die Ende-zu-Ende Verschlüsselung und das Identifizieren des Kommunikationspartners mittels Public-Key-Kryptographie. Er verfügt über ein Adressbuch, damit die „anonymen" Kommunikationspartner identifiziert und überprüften Zertifikaten zugeordnet werden können.

3.4.1 Server

Frontg8 ist ein Client-Server-System welches quelloffen ist, und somit in Eigenregie betrieben werden kann. Die Serverapplikation sammelt ausser der IP-Adresse, welche zur Etablierung der Verbindung benötigt wird, im regulären Betrieb keine Daten und auch die IP-Adresse wird nur temporär im Speicher gehalten.
Der Server bewahrt die eingereichten Nachrichten in einem Puffer-Speicher für einen einstellbaren Zeitraum (Standard ist 3 Tage) auf und entfernt sie anschliessend. Es wird dem Server-Betreiber empfohlen die mitgelieferten Skripten zur Erzeugung des Pufferspeichers zu verwenden. Die Skripte erzeugen einen nicht permanenten Speicherbereich, womit verhindert werden soll, dass die Nachrichten jemals in persistenten Speicher geschrieben werden. Diese Massnahme soll verhindern, dass die verschlüsselten Nachrichten leicht zum Zwecke einer Offline-Analyse entwendet werden.
Der Server benachrichtigt bei einer eingehenden Nachricht alle momentan mit ihm verbundenen Clients über den Eingang. Wie momentan nicht verbundene Clients die Nachrichten empfangen, ist im Abschnitt Client erläutert.

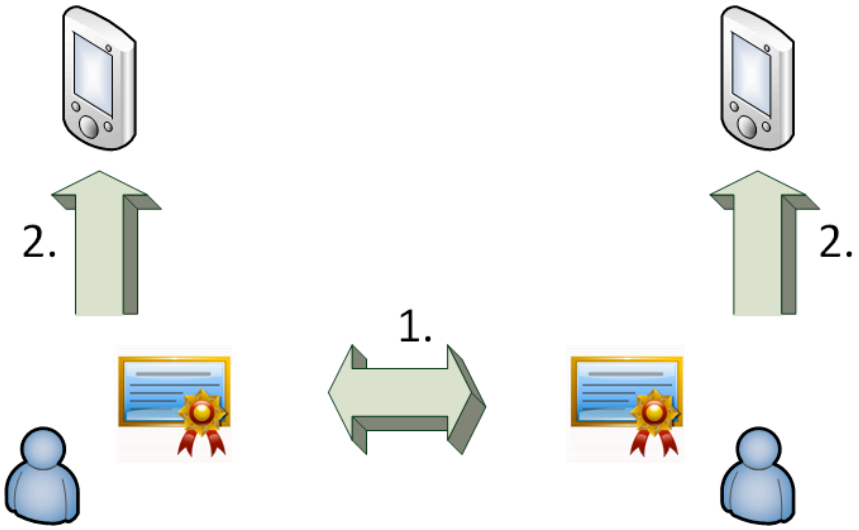
3.4.2 Client

Der Client ist für die Aufbewahrung der gemeinsamen Geheimnisse, sowie des privaten Geheimnisses verantwortlich. Des Weiteren speichert der Client die für ihn bestimmten Nachrichten für einen konfigurierbaren Zeitraum in verschlüsseltem Zustand ab, bevor er sie endgültig löscht. Da alle Clients alle Nachrichten bekommen, muss der Client feststellen können, ob die Nachricht für ihn bestimmt ist, ohne zu versuchen, die möglicherweise lange Nachricht zu entschlüsseln. Zu diesem Zweck ist jeder Nachricht ein HMAC angehängt. Dies ermöglicht es dem empfangenden Client einzig den HMAC zu prüfen. Ist der Client in der Lage die Gültigkeit des HMACs zu bestätigen, ist erstens mit grosser Sicherheit die Nachricht für diesen Client bestimmt und zweitens geklärt von welchem Absender sie stammt. Kann der empfangende Client den HMAC nicht verifizieren, verwirft er die Nachricht.

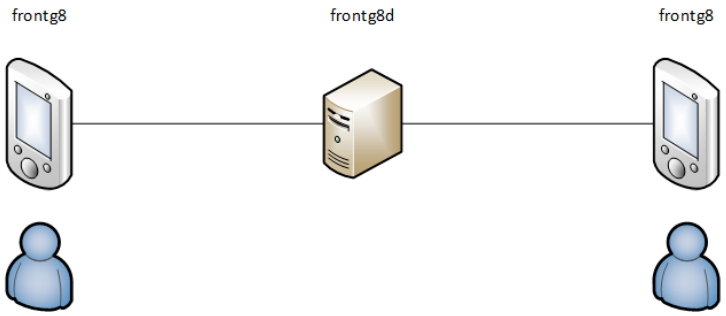
Wie zuvor beschrieben erhält der Client eine Benachrichtigung für jede eingegangene Nachricht, solange er eine aktive Netzwerkverbindung zum Server besitzt. Falls der Client momentan nicht über eine Verbindung zum Server verfügt, wird bei der nächsten Etablierung einer Verbindung vom Client eine Message-Request-Message an den Server gesendet. Diese Nachricht enthält den Hash der letzten vom Client empfangen Nachricht. Der Client erhält als Antwort vom Server eine Notification-Message, welche die Anzahl der seither eingegangen Nachrichten, sowie die Nachrichten enthalten.

3.4.3 Systemübersicht

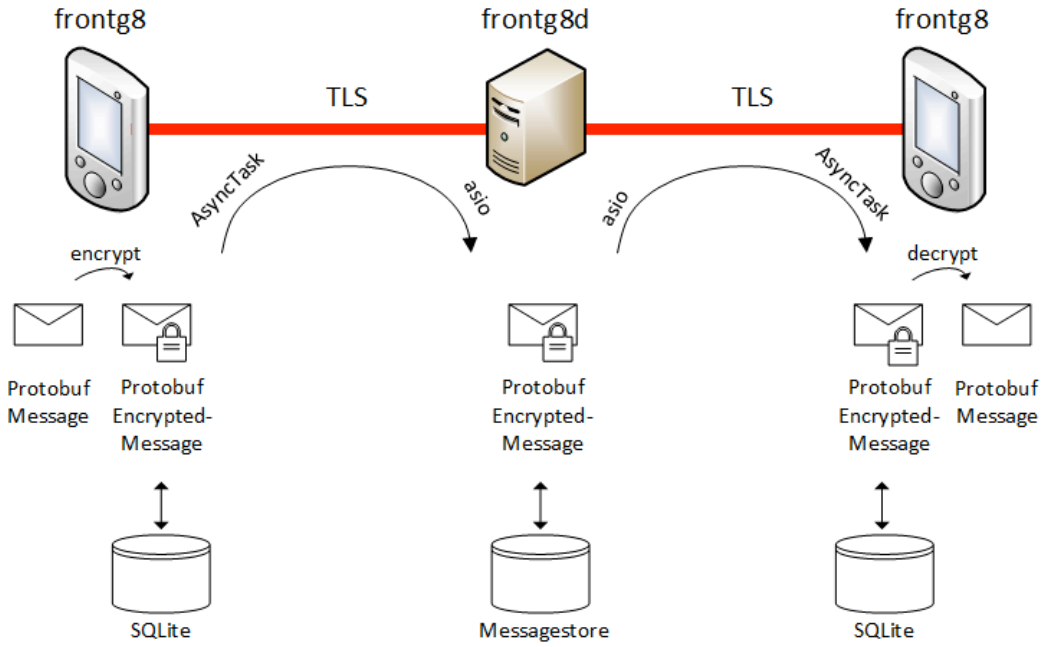
Frontg8 setzt starke Ende-zu-Ende-Verschlüsselung zwischen den Kommunikationspartnern ein. Der symmetrische Schlüssel zur Verschlüsselung der Nachrichten wird durch den Austausch von ECDH-Schlüsselteilen generiert. Zu diesem Zweck treffen sich die beiden Kommunikationspartner physisch und tauschen ihre öffentlichen Schlüssel aus.



Nachdem beide Teilnehmer ein gemeinsames Geheimnis erarbeitet haben, sind sie in der Lage mit einander zu kommunizieren. Die gesamte Kommunikation wird asynchron über einen zentralen Server durchgeführt. Die Asynchronität ermöglicht es, dass beide Teilnehmer miteinander kommunizieren können, ohne jemals gleichzeitig mit dem Server verbunden sein zu müssen.



3.4.4 Grundlegendes Konzept für den Austausch von Nachrichten



1. Die Nachricht wird in eine Protobuf Nachricht verpackt
2. Die Protobuf Nachricht wird serialisiert und anschliessend verschlüsselt.
3. Die verschlüsselte Nachricht wird wiederum in eine Protobuf-Nachricht des Typs Encrypted verpackt und für den Versand serialisiert.
4. Der TcpClient versendet die Nachricht dann über den TlsClient an den Server
5. Auf dem Server kümmert sich Asio um die TLS-Verbindung und nimmt die verschlüsselte Nachricht entgegen. Die eingegangenen Nachrichten werden im Message Store vorgehalten welcher die nach wie vor verschlüsselten Nachrichten in einer RAM-Disk speichert.
6. Verbundene Clients werden über die neue Nachricht informiert. Clients die keine aktive Verbindung offen haben, bekommen die neue Nachricht, sobald sie beim Server nach neuen Nachrichten fragen.
7. Alle Clients holen die Nachricht ab, aber nur falls die Nachricht für sie bestimmt ist, können sie die Nachricht auch entschlüsseln.

3.5 Ziele dieser Semesterarbeit

In dieser Semesterarbeit wird das bereits bestehende Frontg8-Chatsystem um einen Android-Client erweitert. Die elementaren Grundziele von Frontg8 die wir bereits in der Motivation erwähnten, mussten auch bei der mobilen Implementation eingehalten werden.

Da wir mit dem Server und dem Desktop-Client zufrieden waren, sollte nun ein Client für Android folgen, damit das System auch mobil genutzt werden kann.

4. Projektplan

4.1 Projektübersicht

4.1.1 Ziel und Zweck

Ziel der Semesterarbeit ist es, einen Android-Client zu entwickeln, der mit dem bestehenden Server (frontg8d) funktioniert. Der Funktionsumfang des Android-Clients soll mindestens dem CLI-Client entsprechen.

Konkret bedeutet dies, dass der Client zu folgendem in der Lage ist:

- Verwalten von Kontakten
- Importieren und exportieren des eigenen Schlüsselpaars
- Importieren von Public-Keys
- Auflisten der Konversationen
- Messages verschlüsseln/senden und empfangen/entschlüsseln

4.1.2 Lieferumfang

Lieferumfang ist ein lauffähiger, der Beschreibung in «Ziel und Zweck» entsprechenden, Android-Client.

4.1.3 Annahmen und Einschränkungen

Wir setzen mindestens Android 5 Lollipop (API Level 21) voraus.

Testen werden wir die Applikation ausschliesslich mit unseren eigenen Smartphones:

- Samsung Galaxy S4 (Android 5.1.1 - API Level 22)
- Motorola Moto X2 (Android 5.1 - API Level 22)

Aus zeitlichen Gründen werden wir uns stärker um Kryptographie, Sicherheit und die Funktionalität als um ein schönes Userinterface kümmern. Dementsprechend haben wir nicht den Anspruch eine massentaugliche Applikation zu entwickeln.

4.2 Projektorganisation

An diesem Projekt arbeiten die Informatikstudenten Tobias Stauber und Ueli Bosshard. Aufgrund seiner Erfahrungen aus dem SE2P wird sich Tobias Stauber um alle Themen rund um die Kryptographie kümmern. Ueli Bosshard spezialisiert sich auf das Persistieren der Daten und die Netzwerkverbindung.

4.3 Zeitplanung

Sem. W	KW	Sitzung	Tasks	Verzug
W1	KW38	15.09.	Arbeitsumgebung einrichten	-
W2	KW39	22.09.	Aufgabenstellung, Umfang definieren	-
W3	KW40	29.09.	**MS1** Grobplanung, Arbeitspakete bestimmen	-
W4	KW41	06.10.	Architektur: Domainanalyse, Risikoanalyse	3 Tage
W5	KW42	13.10.	**MS2** UI Mockups, Crypto-Abklärungen	7 Tage
W6	KW43	20.10.	Testingumgebung definieren/aufsetzen	9 Tage
W7	KW44	27.10.	**MS3** Prototyp (Alle Activities vorhandn)	13 Tage
W8	KW45	03.11.	Construction (Kontakthandling, Storage)	7 Tage
W9	KW46	10.11.	**MS4** Architektur Review	4 Tage
W10	KW47	17.11.	Construction (Protokoll, Encryption)	-
W11	KW48	24.11.	Construction, Testing	-
W12	KW49	01.12.	Dokumentation, **MS5** Feature-Freeze	1 Tag
W13	KW50	08.12.	Abschluss Dokumentation	1 Tag
W14	KW51	15.12.	Reserve, **Abgabe**	- (Mit Reservefrist eingerechnet)

Unsere Zeitplanung war eher suboptimal. Durch den Druck von Testaten in nebenläufigen Modulen kamen wir in der dritten Woche in den Verzug. Durch bessere Arbeitseinteilung konnten wir bis zur Woche zehn alles wieder aufholen. Frontg8 verursachte viel mehr Aufwand, als eine SA eigentlich verursachen sollte. Durch unser Eigeninteresse fiel es uns allerdings nicht schwer, einige Überzeit zu arbeiten. Wir investierten gemeinsam an die 450 Stunden. Und hatten gegen Ende der Arbeit keine Reserve. Vor allem wurden wir dadurch aufgehalten, dass wir uns zuerst in zwei Kerntechnologien welche von Android angeboten werden, den Keystore und den Service, einarbeiten mussten. Als Konsequenz werden wir bei zukünftigen Arbeiten für das Kennenlernen der Plattform mehr Zeit einkalkulieren.

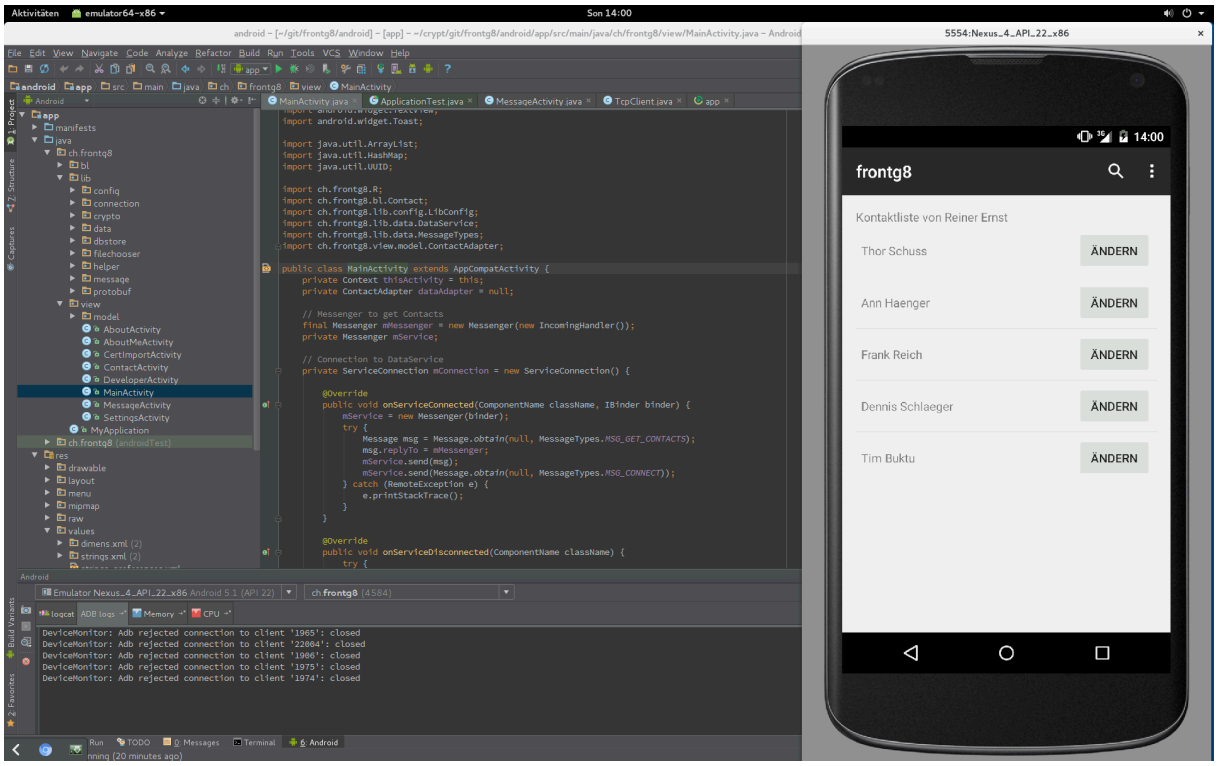
4.4 Risikomanagement

Risikoanalyse:

Risiko	Beschreibung	Gewichtung	Folgen
Keystorage	Schlüssel können nicht im Android-Keystore gespeichert werden	Mittel	Mehr eigener Code muss geschrieben werden um dieselben Funktionalitäten zu erhalten.
Android	Das Einarbeiten in Android ist aufwändiger und dauert länger als erwartet	Mittel	Die App müsste notdürftig fertiggestellt und anschliessend überarbeitet werden.
Bibliotheken	Das Finden von Bibliotheken, welche unseren Anforderungen genügen ist schwer	Schwer	Sehr viel eigener Code müsste geschrieben werden.
Kompatibilitäts probleme	Wir können nicht mit dem Server kompatibel sein und müssen diesen anpassen	Sehr Schwer	Die Funktionalitäten, welche diese Änderungen erfordern müssen weggelassen werden
Testing	Wir können das Projekt nicht komplett automatisiert Testen	Gering	Ein Teil des Testings müsste manuell gemacht werden.

4.5 Infrastruktur

Die Studenten welche für Studienarbeiten eingeteilt sind, verfügen über einen Arbeitsplatz mit PC. Auf dem PC installiert ist Arch-Linux, Android Studio 1.3 (JetBrains) und git.



Daneben entwickeln die Studenten auf den eigenen PC und Testen mit ihren Android-Smartphones.

Für das Projektmanagement steht ein Linux-Server mit installiertem Redmine zur Verfügung.

5. Use Cases

5.1 Wichtige Konzepte

Einführend möchten wir kurz auf die Konzepte eingehen, die wir für unsere Arbeit verwenden.

5.1.1 Messages

Unser Protokoll spezifiziert, dass zwischen Server und Client verschiedene Arten von Messages ausgetauscht werden. Wir differenzieren dabei in folgende Typen:

5.1.1.1 EncryptedMessage

Alle Nachrichten vom Typ Encrypted-Messages werden vom Server nur zwischengespeichert. Der Server selbst kann die Nachrichten nicht entschlüsseln. Zudem gibt es keine Zieladresse, so können ausschliesslich die beteiligten Clients wissen, wer mit wem was austauscht.

5.1.1.2 MessageReqMessage

Die MessageRequest-Message dient dazu, den Server nach allen Nachrichten seit dem letzten Abruf zu fragen. Dazu wird in der Nachricht der SHA256 Hash der letzten Nachricht, welche man empfangen hat, mitgesendet. So kann der Server entscheiden: Wenn er eine Nachricht mit dem entsprechenden Hash im Journal hat, wird er einem eine NotifyMessage mit allen neueren Nachrichten senden. Ansonsten sendet der Server eine NotifyMessage mit allen Nachrichten, welche er im Journal hat.

5.1.1.3 NotifyMessage

Die Notify Message enthält die Information, wie viele Nachrichten gesendet wurden und enthält jene Nachrichten auch gleich.

5.1.1.4 DataMessage

Die Data-Message schliesslich ist für die Übertragung der Text-Nachricht zuständig. Die Daten werden dabei mit dem symmetrischen Session-Key verschlüsselt.

5.1.2 Kontakte, Session, PublicKey

Der Client besitzt ein Adressbuch mit seinen Kontakten. Zu jedem seiner Kontakte speichert der Client den Public-Key des jeweiligen Kontakts, aus dem er dann den Session Key ableiten kann.

5.2 Übersicht Usecases

Folgende Usecases sind durch die Android-Applikation abzudecken:

- Initiale Konfiguration
- Applikation starten
- Nachricht lesen
- Nachricht verschicken
- Nachrichten löschen
- Kontakte verwalten

Nachfolgend werden die Usecases beschrieben.

5.3 Usecase: Initiale Konfiguration

Die initiale Konfiguration sorgt dafür, dass die Applikation nach dem ersten Start in einen benutzbaren Zustand gebracht wird. Dazu ist es nötig, sinnvolle Standardwerte zu setzen und ein persönliches Schlüsselpaar zu generieren.

5.4 Usecase: Applikation starten

Dieser Usecase deckt die Aktionen ab, die bei jedem Start der Applikation durchlaufen werden müssen. Als erstes wird die Konfiguration eingelesen, dann werden die persistierten Daten geladen und schliesslich wird eine Verbindung zum Server aufgebaut und überprüft, ob neue Nachrichten vorliegen.

5.5 Usecase: Nachricht lesen

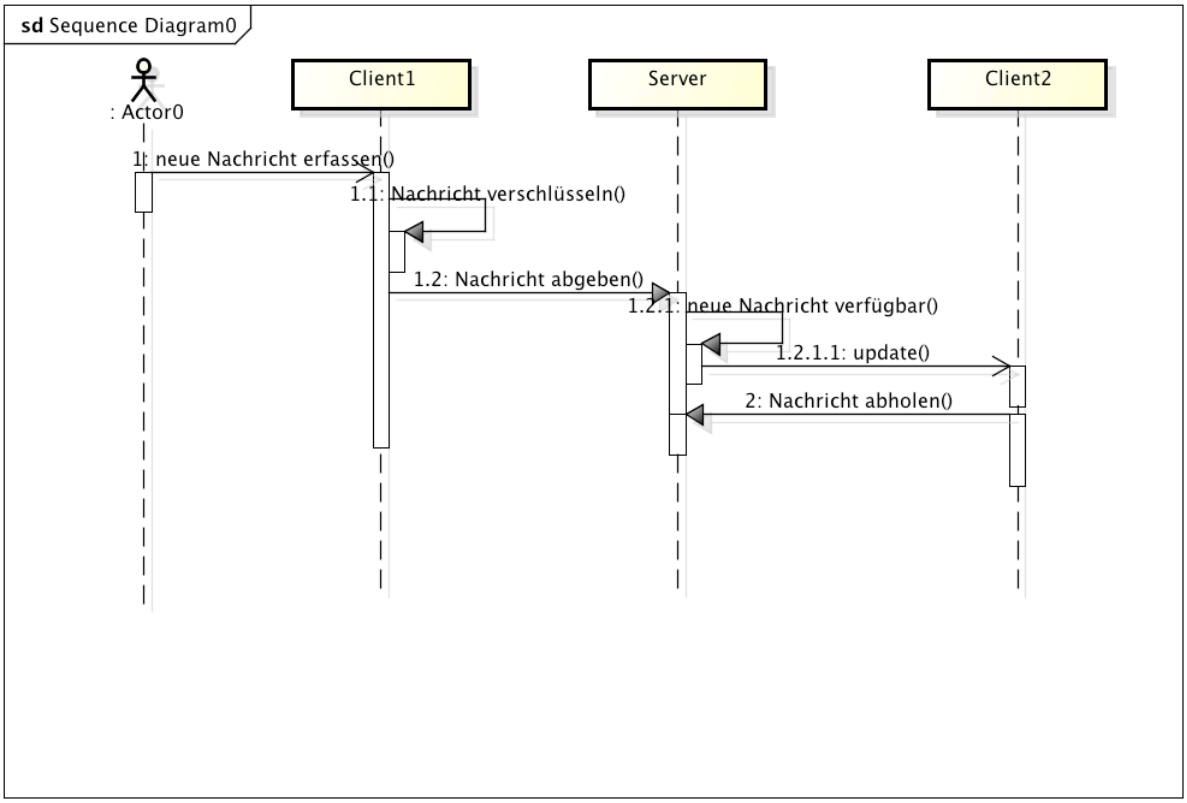
In der Kontaktübersicht zeigt eine Zahl hinter dem Kontaktnamen, ob eine neue Nachricht vorliegt. Dies ist der Fall wenn:

- Persistierte Nachrichten noch nicht gelesen wurden
- Nach dem Start neue Nachrichten vorliegen
- Der Server in einer Notification eine neue Nachricht für den betreffenden Kontakt geschickt hat.

Nach dem Antippen des Kontaktnamens wird die Nachrichtenliste angezeigt.

5.6 Usecase: Nachricht verschicken

Nachdem der Benutzer einen Text eingegeben hat und auf "senden" drückt wird die Nachricht verschlüsselt und an den Server geschickt. Der Server informiert darauf hin alle verbundenen Clients, inklusive den Clients von dem aus die Nachricht verschickt wurde. Der Client bekommt also die Nachricht wieder zurück, worauf die Nachricht lokal gespeichert wird. Ausserdem wird erst dann die Nachricht wieder entschlüsselt und im Nachrichtenverlauf angezeigt.



5.7 Usecase: Nachrichten löschen

In der Nachrichtenübersicht können im Menu alle Nachrichten gelöscht werden, worauf hin die Nachrichten aus dem persistenten Speicher gelöscht werden.

5.8 Usecase: Kontakte verwalten

Im Wesentlichen kann der Usecase "Kontakte verwalten" als simpler CRUD-Usecase beschrieben werden (CRUD: Create, Read, Update, Delete). Hierzu kann ein neuer Kontakt angelegt werden und in der Detailansicht bearbeitet oder gelöscht werden.

Bei diesem Usecase ist noch zu überlegen, ob es möglich ist, den Publickey des Kontaktes komfortabel zu erfassen.

5.9 Abusecases

Da wir bei unserer Arbeit einen Schwerpunkt auf die Sicherheit legen, möchten wir uns an dieser Stelle auch Gedanken über möglichen Missbrauch unserer Chatlösung machen.

5.9.1 Angriff auf Vertraulichkeit

Das zentrales Kriterium für unsere Anwendung ist die Vertraulichkeit der Nachrichten. Ein Angriffspunkt stellt also die eingesetzte Kryptographie dar. Aus dieser Überlegung werden wir uns intensiv mit dem Thema befassen und auch in dieser Dokumentation noch speziell behandeln.

5.9.2 Angriff auf Verfügbarkeit

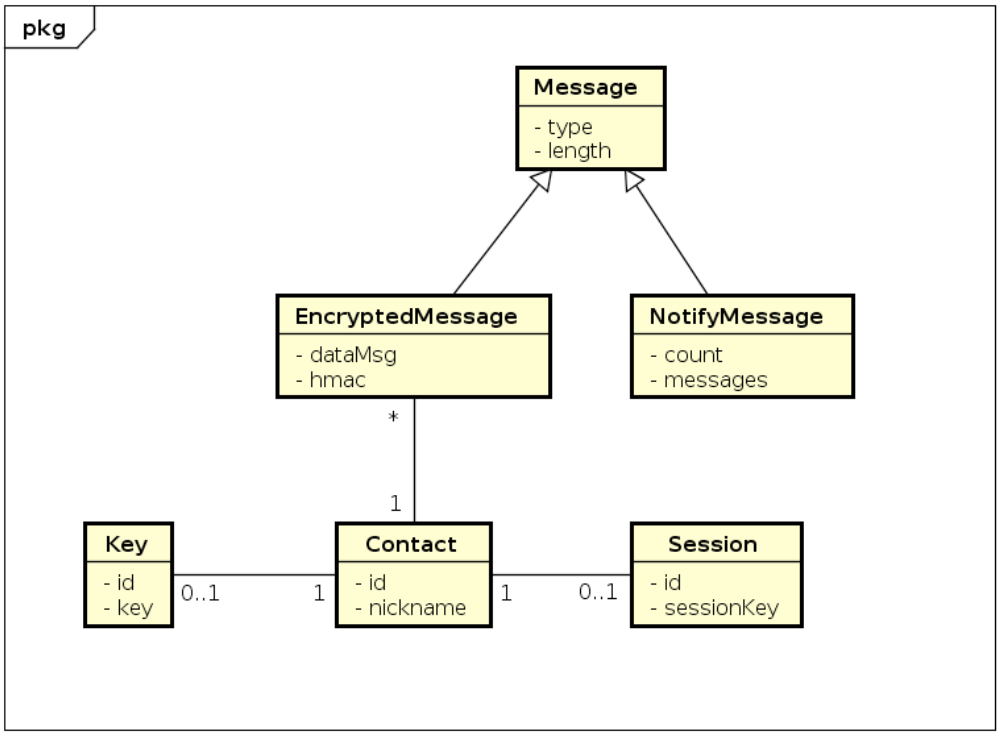
Ein weiterer typischer Angriffspunkt stellt die Verfügbarkeit des Dienstes dar. Hierzu braucht ein Angreifer nur genügend Ressourcen um das anzugreifende System übermässig auszulasten. Dazu haben wir uns bei unserer Lösung folgende Überlegungen gemacht: Unser Ziel war es von Anfang an, dass unsere Lösung von unabhängigen Benutzergruppen jeweils für sich selbst betrieben wird. Weiter legen wir Wert darauf, dass wir möglichst effizient prüfen, ob eine eingehende Nachricht von einem bekannten Kontakt kommt. Mit diesen Massnahmen können wir einen Denial-of-Service-Angriff nicht ausschliessen, zumindest jedoch die Anfälligkeit dafür reduzieren.

5.9.3 Weiter Überlegungen

Für Abusecases, die den Server betreffen, verweisen wir auf die entsprechenden Dokumente aus dem SE2-Projekt.

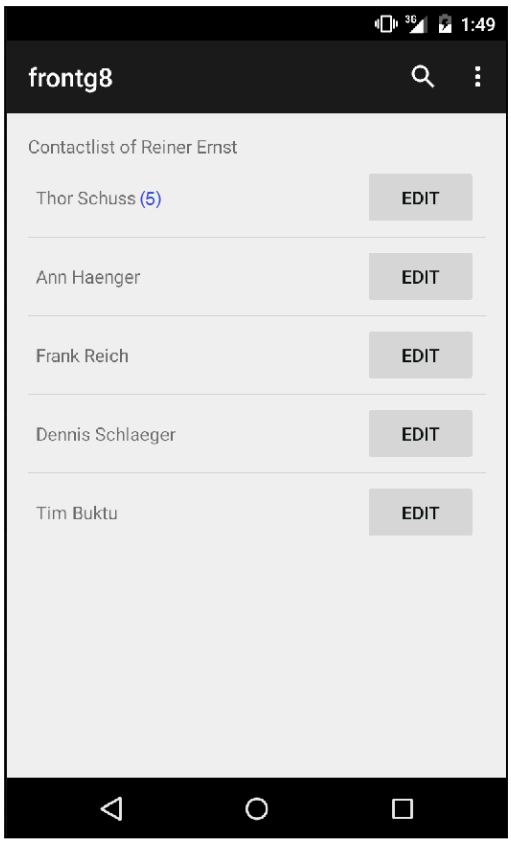
6. Analyse und Design Spezifikationen

6.1 Domainmodell



6.2 Userinterface

Das Userinterface beschränkt sich im Wesentlichen auf einige wenige, simple Darstellungen der Kontakte, Nachrichten und Einstellungen.

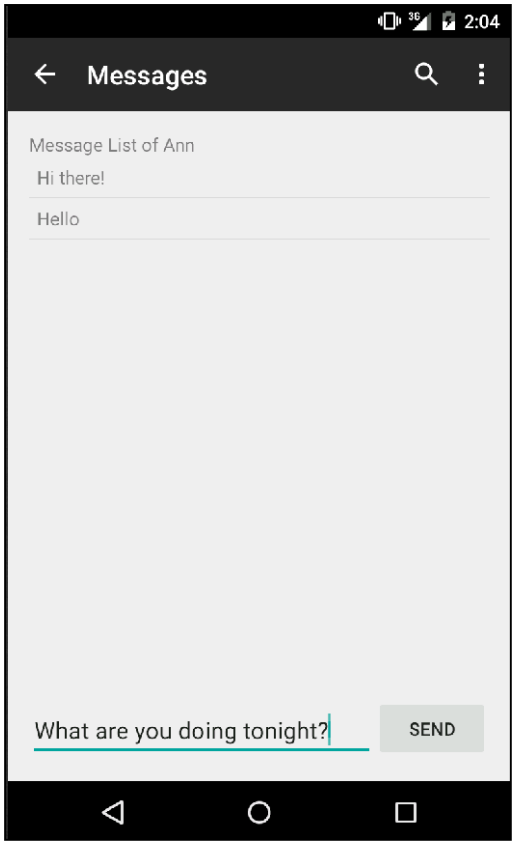


6.2.1 Kontaktliste

Bei der Kontaktliste handelt es sich um die Hauptansicht, welche auch sogleich nach dem Start von frontg8 angezeigt wird.

Folgende Funktionalität wird angeboten:

- Ein simples Anklicken des Kontaktnamens, erlaubt es, zu dessen Nachrichtenliste zu wechseln.
- Durch Antippen des EDIT-Button wird die Kontaktanzeige aufgerufen werden.
- Hinzufügen von neuen Kontakten mittels Menüeintrag.
- Die kleine Zahl hinter dem Kontaktnamen dient als Indikator, wie viele ungelesene Nachrichten für diesem Kontakt vorhanden sind.



6.2.2 Nachrichtenliste

Die Nachrichtenliste zeigt die Konversation mit einem einzelnen Kontakt.

Folgende Funktionalität wird angeboten:

- Eingeben von neuen Nachrichten
- Senden der Nachricht mittels SEND-Button
- Sofortiges Anzeigen von neu eingegangenen Nachrichten
- Löschen der Nachrichtenliste im Menü

Was nach der SA noch zu implementieren ist:

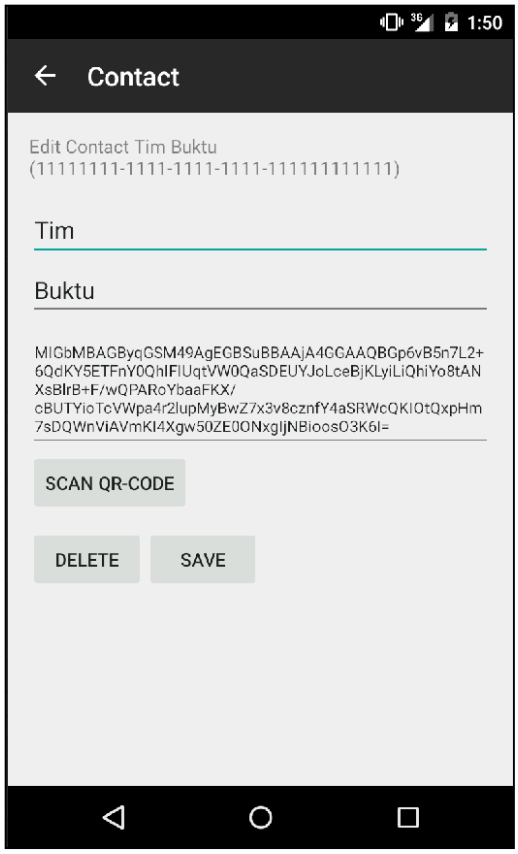
- Farbliche Kennzeichnung des Senders der Nachricht.

6.2.3 Kontaktanzeige

Die Kontaktanzeige wird sowohl benutzt um einen Kontakt zu ändern, als auch um einen neuen Kontakt anzulegen. Hier finden sich die zentralen Elemente, welche für den Kontakt benötigt werden: Ein Anzeigename, eine eindeutige Identifikationsnummer und der Public-Key.

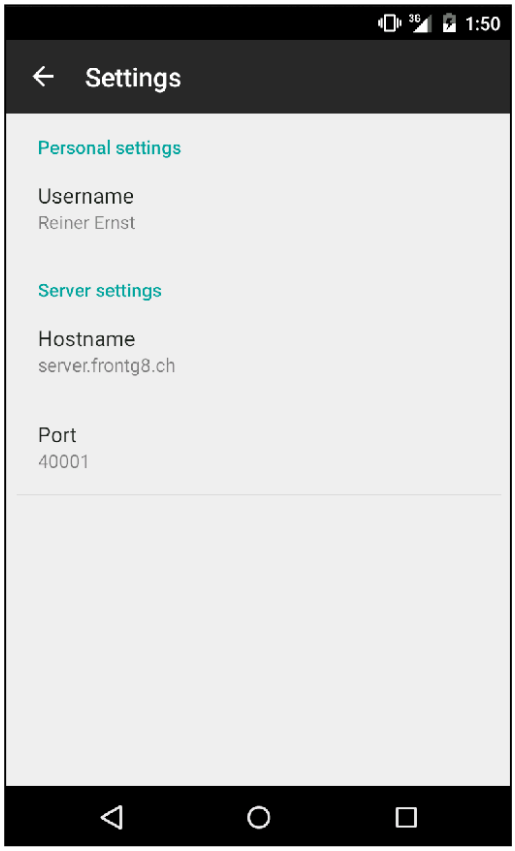
Folgende Funktionalität wird angeboten:

- Anlegen eines neuen Kontaktes
- Bestehenden Kontakt ändern
- Kontakt löschen
- Durch den "SCAN QR-CODE" Button wird eine externe Applikation geöffnet, welche es erlaubt einen Public-Key als QR-CODE zu scannen.



6.2.4 Einstellungen

In den Einstellungen kann der verwendete Server geändert werden. Ausserdem kann hier der eigene Benutzername gesetzt werden, welcher dann an diversen Stellen angezeigt wird.



6.2.5 Eigener Public-Key

Um den Austausch des Public-Keys zu erleichtern, zeigen wir diesen als QR-Code an. Dieser Code kann in der Kontaktansicht gescannt werden.

7. Protokollspezifikation

7.1 Äusseres Nachrichtenformat

Alle Nachrichten verfügen über die gleiche Struktur:

Length	Type	Res
MSGData		

7.1.1 Beschreibung der Felder

Feld	Typ	Länge in Byte	Bedeutung
Length	Integer	2	Länge der Nachricht
Type	Integer	1	Typ der Nachricht
Res	-	1	Reserviert
Data	Protobuf	Length - 4	Protobuf Nachrichtendaten

Das Feld **Res** ist für spätere Erweiterungen reserviert und wird momentan immer auf **0xFF** gesetzt. Im **Data** Feld werden die spezifischen Protobuf Daten der jeweiligen Nachricht transportiert. Im Falle der Encrypted-Message besteht das **Data** Feld aus den verschlüsselten Protobuf Daten gefolgt von einem MAC.

7.2 Nachrichten-Typen

7.2.1 MSGTypeEncrypted - Verschlüsselte Nachricht.

Feld	Typ	Länge in Byte	Bedeutung
SubType	Byte[]	1	Subtyp der Nachricht
Session	Byte[]	16	Session ID
MSGData	Byte[]	512	Nachrichtendaten

Das **Session** Feld beinhaltet eine 128-Bit Random-UUID um die assoziierte Session zu identifizieren. Es wird im Rahmen des SE2-Projekts nicht verwendet, ist jedoch für zukünftige Erweiterungen reserviert.

7.2.1.1 Subtypen:

Subtyp	Wert	Bedeutung	Verwendung
Data	0	Daten-Nachricht	Text-Nachrichten
Control	1	Kontroll-Nachricht	Session-Control

Der Subtyp **Control** wird im Rahmen des SE2-Projekts nicht verwendet, ist jedoch für zukünftige Erweiterungen reserviert.

7.2.2 MSGTypeBlacklist - Blacklist Nachricht

Feld	Typ	Länge in Byte	Bedeutung	Optional
SubType	Byte[]	1	Subtyp der Nachricht	
Count	Integer	2	Anzahl Einträge	X
Blacklist	Byte[]	Count * 64	Blacklist Einträge	X
Hash	Byte[]	64	Hash der Blacklist	X
MAC	Byte[]	64	MAC	X

7.2.2.1 Subtypen:

Subtyp	Wert	Bedeutung	Verwendung
Hash	0	Blacklist Hash	Benachrichtigung durch den Server
Request	1	Blacklist Request	Anforderung durch den Client
Blacklist	2	Blacklist	Austausch der Blacklist

Der Subtyp **Hash** erzwingt das optionale Feld **Hash**. Eine Nachricht dieses Subtyps wird nur vom Server beim Verbindungsaufbau versendet.
Der Subtyp **Request** erzwingt das Fehlen der optionalen Felder und wird nur vom Client gesendet.
Der Subtyp **Blacklist** erzwing die Felder **Count**, **Blacklist** und **MAC** und wird nur vom Server versendet.

7.2.3 MSGTypeRevocation - Revozierungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
PubKey	Byte[]	65	Public Key
Signatur	Byte[]	???	Signatur über den PubKey

Das Feld **PubKey** beinhaltet den revozierten Public Key. Im Feld Signatur wird die **Signatur** des revozierenden Clients über den Public Key übertragen.

7.2.4 MSGTypeNotification - Benachrichtigungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
Count	Byte[]	1	Anzahl Nachrichten im Bundle
Bundle	Byte[]	-	Nachrichten Bundle

7.2.5 MSGTypeMessageRequest - Nachrichten Anforderungs Nachricht

Feld	Typ	Länge in Byte	Bedeutung
Hash	Byte[]	64	Hash der letzten Nachricht

8. Kryptographie

8.1 Datenhaltung

Die Daten sollen nur zur Laufzeit entschlüsselt im Speicher gehalten werden. Da die Nachrichten bereits in einem verschlüsselten Zustand versendet werden, können sie auch gleich in diesem Zustand auf die Datenbank gespeichert werden. Jedes Mal, wenn der Datenhaltungsservice der Applikation gestartet wird, muss dieser alle Nachrichten von der Datenbank lesen, entschlüsseln und den entschlüsselten Kontakten anhängen.

8.1.1 Auf dem Server

Der Server muss die Nachrichten zu keinem Zeitpunkt entschlüsseln können, er braucht sie lediglich während einer einstellbaren Zeit (standardmässig drei Tage) vorzuhalten.

Die Metadaten werden in einem Journal vorgehalten, das für die Speicherung auf der Festplatte verschlüsselt wird. Beim Starten des Servers wird nach einem Passwort gefragt, das mittels PBKDF2-SHA256 zu einem symmetrischen Schlüssel der Länge 128Bit verarbeitet wird. Dieser Schlüssel wird eingesetzt um die Journaleinträge mittels AES128-CBC zu verschlüsseln.

8.1.2 Auf dem Client

Der Client muss die Nachrichten für den Kommunikationspartner verschlüsseln und Nachrichten vom entsprechenden Kommunikationspartner entschlüsseln. Damit der Kommunikationspartner erkennen kann, ob eine Nachricht für ihn verschlüsselt wurde, versieht der Absender das Paket noch mit einem Keyed-Hash Message Authentication Code SHA256 (HMAC SHA 256) welcher er mit dem Session-Schlüssel für Singnaturen erstellt. Dieser, kryptographisch sichere, Nachrichten-Authentifikationscode wird beim Empfänger von der Nachricht gelöst. Danach generiert der Empfänger über der verbleibenden Nachricht ebenfalls den HMAC. Wenn der HMAC mit dem richtigen Schlüssel generiert wurde, ist er gleich wie der HMAC, welcher der Nachricht angehängt wurde. Trifft dies zu, wird die LibCrypto die Nachricht entschlüsseln und der Nachrichtenliste des Kontakts anhängen, mit dessen Schlüssel der richtige HMAC generiert wurde.

8.2 Transportverschlüsselung

Wir verwenden Transport Layer Security (TLS 1.2) um die Nachrichten während dem Transport zu verschlüsseln. Dies ist notwendig, damit auch all jene Nachrichten die sich an den Server wenden, wie z.B. Steuer- und Kontrollnachrichten, verschlüsselt sind. Da der Server diese Nachrichten verstehen muss, dürfen sie nicht, wie die Datamessages, an einen Kontakt verschlüsselt werden. Da ein weiteres spezifisches Verschlüsseln keine nennenswerte Verbesserung der Sicherheit einbringen würde, beschlossen wir diese nur durch TLS zu verschlüsseln. Dazu verwenden wir folgende Cipher Suite: ECDHE – ECDSA – AES128 – GCM – SHA256

Zweck	Algorithmus
Verschlüsselung	*AES Galois/Counter Mode (GCM) 128 Bit*
Digitale Signatur	*DSA auf elliptischen Kurven (ECDSA) 256 Bit*
Schlüsselaustausch	*Flüchtiger Diffie Hellman auf elliptischen Kurven (ECDHE) 256 Bit*
Hashfunktion	*SHA-256*

8.3 Nachrichtenverschlüsselung

Jeder Teilnehmer besitzt einen asymmetrischen ECDH Schlüssel. Der öffentliche Teil wird mit dem gewünschten Kommunikationspartner ausgetauscht. Aus dem eigenen Privatekey und dem Publickey

des jeweiligen Kommunikationspartners werden zwei symmetrische Schlüssel abgeleitet. Je einer für das Verschlüsseln und einen um zu Signieren.

8.3.1 ECDH-Schlüssel erstellen

Die Public- und Privatekeys werden auf der Kurve "secp521r1" gewählt. Initial wählten wir diese Kurve um die Kompatibilität zum Python-Client zu gewährleisten. Allerdings mussten wir feststellen, dass diese durch die leicht unterschiedlichen Versionen der Schlüsselableitungsfunktion und der unterschiedliche Grösse der Initialisierungsvektoren nicht realisiert werden kann.

8.3.2 Symmetrische Schlüssel aushandeln

Der Schlüssel, welcher aus dem ECDH-Handshake hervorgeht, ist ein symmetrischer Schlüssel mit einer Länge von 512 Bit. Aus diesem werden dann mit kdf2 zwei 256 Bit Schlüssel abgeleitet.

8.3.3 Verschlüsseln

Die LibCrypto erhält vom DataService eine Data-Message, die sie mit dem Session Schlüssel für Kryptographie (SK.C) verschlüsselt. Für die Verschlüsselung verwenden wir Advanced Encryption Standard im Galois/Counter Mode mit einer Schlüssellänge von 256 Bit (AES256-GCM). Den Counter initialisieren wir jeweils mit einem 16 Byte Zufallswert, welchen wir vorne an die Nachricht anhängen. Über die verschlüsselte Daten-Nachricht wird ein HMAC(SHA-256) gebildet. Als Schlüssel für die HMAC-Funktion dient der SK.S. Der HMAC wird an die verschlüsselte Daten-Nachricht angehängt. Danach wird die verschlüsselte Nachricht an den DataService.

8.3.4 Entschlüsseln

Der Kontakt erhält jegliche Nachrichten als grosse binäre Objekte (BLOBs). Zuerst wird das BLOB geteilt, indem die sechzehn header-Counter-Bytes und der HMAC-trailer abgeschnitten werden. Über dem verbleibenden Teil des BLOBs (der Teil, welcher die verschlüsselte Data-Message enthält) wird der HMAC SHA256 gebildet. Wenn dann beide HMACs übereinstimmen wird die verschlüsselte Nachricht in der Datenbank abgelegt, der verschlüsselte Cryptoteil entschlüsselt und die daraus entstandene Daten-Nachricht zurückgegeben.

8.3.5 Entscheidungen

- Wir verwenden elliptische Kurven, da die notwendige Rechenleistung geringer ist als bei RSA.[5]
- Wir verwenden HMAC, da dies uns sowohl das Prüfen der Integrität, wie auch der Authentizität ermöglicht.
- Wir verwenden SHA-256, da die Sicherheit ausreicht und die Geschwindigkeit sowohl auf 32Bit wie auch auf 64Bit Systemen annehmbar ist.
- Wir haben uns für AES-256/CTR entschieden, da die Laufzeit nicht markant von AES-128 abweicht. Countermode wählten wir, damit wir nicht an 128Bit Blöcke gebunden sind.
- Da identische Nachrichten nach dem Verschlüsseln bei identischem Counter Initialwert immer gleich aussehen würden, initialisieren wir den Counter mit einem 128Bit langen für Kryptographie tauglichen Zufallswert. Den Initialwert transportieren wir zum Empfänger, indem wir ihn vor die Nachricht setzen.
- Als HASH-Funktion für die Identifizierung der Nachrichten verwenden wir SHA256, aus denselben Gründen wie oben genannt.
- Die Session Schlüssel (SK.S und SK.C) werden mit der Schlüsselableitungsfunktion 2 mit SHA 256 Digest (KDF2 SHA256) abgeleitet.

[5] http://dl.acm.org/citation.cfm?id=570691

- Wir wollten zuerst die Software so umsetzen, dass sie die Schlüssel nach dem Beenden automatisch aus dem Speicher entfernt (Memory-wipe), allerdings ist dies durch die beschränkten Möglichkeiten von Java nicht umsetzbar ohne mit Java Native Interface auf C-Funktionalität zurückzugreifen.[6]
- Da der Android interne Keystore zwar die sicherste Möglichkeit der Schlüsselverwaltung ist, wollten wir ihn einsetzen, allerdings konnten wir unsere Anforderungen damit nicht umsetzen, da der Keystore bis Android 6.0 kein ECDH unterstützt.[7]

8.4 Kryptographische Verfahren

8.4.1 Verwendung von Diffie Hellman, Key-Negotiation

Um einen sicheren symmetrischen Schlüssel auszutauschen, ohne den Schlüssel persönlich zu übergeben oder über einen unsicheren Kanal zu senden, setzen wir ECDH ein. Jeder Teilnehmer hat ein Schlüsselpaar auf der Curve 'secp512r1'.

8.4.2 Symmetrische Schlüssel, Session-Key

Session-Key (Der Schüssel, welcher aus dem ECDH Handshake hervorgeht)	Länge: 512Bit
Session-Key.Crypto (SK.C)	Länge: 256Bit Abgeleitet von SK mit KDF2 SHA256
Session-Key.Signing (SK.S)	Länge: 256Bit Abgeleitet von SK mit KDF2 SHA256

8.4.3 Cipher für symmetrische Ver- und Entschlüsselung

AES-256 im CTR-Mode	Schlüssel: Session-Key.Crypto Counter-Initial-value: Cryptographical random 16 byte int

8.4.4 HMAC für das Signieren der Nachrichten

HMAC(SHA-256)	Schlüssel: Session-Key.Signing

8.4.5 Verschlüsselung der Daten

In der Android-Implementation unseres Clients beschlossen wir auf eine weitere Verschlüsselung der abgelegten Daten zu verzichten. Die Nachrichten werden ja verschlüsselt abgelegt, und die Schlüssel sind im Keystore verschlüsselt, welcher mit einem Passwort geschützt ist.

[6] http://developer.android.com/training/articles/memory.html
[7] http://developer.android.com/training/articles/keystore.html

8.5 Android Krypto-Bibliothek

Unsere LibCrypto erlaubt es, alle Kryptographischen Operationen mit nur einem Methodenaufruf zu erledigen. Es können Nachrichten verschlüsselt und entschlüsselt, Schlüssel ausgehandelt und Hashs generiert werden. Um Methoden, die ein Schlüssel benötigen, auszuführen, muss ein KeystoreHandler mitgegeben werden. In diesem werden dann die kryptographischen Operationen ausgeführt.

8.6 Android Keystore-Handler

8.6.1 Der Android Keystore

Android bietet eine Implementation des Java Keystores an, mit dem alle Operationen die einen Schlüssel benötigen direkt in einem sicheren System-Prozess erledigt werden. Unter einem System, welches dies anbietet, kann sogar das TPM verwendet werden um Schlüssel zu speichern und Operationen damit auszuführen.
Dies bedeutet eine gute Verbesserung der Sicherheit. Leider bietet der Android-Keystore bis zu Android 6.0 (API Version 23) keine Unterstützung für ECDH. Aus diesem Grund sind wir gezwungen in frontg8 vorläufig den gewöhnlichen Java Keystore zu verwenden.

8.6.2 Spongy Castle / Bouncy Castle

Android wird mit einer abgespeckten Version des Kryptographie Bibliothek Bouncy Castle ausgeliefert. Da darin auch der Support von ECDH gestrichen wurde, mussten wir auf Spongy Castle, eine vollständige Implementation der Bibliothek, zurückgreifen. Spongy Castle ist nichts weiter, als die vollständige Bouncy Castle Bibliothek, welche in das Paket SC verschoben wurde, um Namenskollisionen mit der abgespeckten Bouncy Castle Version zu verhindern. Spongy Castle wird dann als Security Provider im System registriert. Wenn der Keystore mit SpongyCastle als Security Provider erstellt wurde, können auch Operationen mit elliptischen Kurven auf dem Keystore ausgeführt werden.

8.6.3 Die Aufgaben des Keystore-Handlers

Der KeystoreHandler übernimmt die Verwaltung der Schlüssel. Das heisst, er weiss, wie die Schlüssel heissen, zu wessen UUID sie gehören und welchem Zweck sie dienen. Wenn der DataService den KeystoreHandler initialisiert, lädt er den serialisierten Keystore vom Speicher und initialisiert damit den neuen Keystore. Da alle Operationen, welche vertrauliche Schlüssel benötigen, direkt vom KeystoreHandler durchgeführt werden, reicht die LibCrypto solche Operationen direkt an den KeystoreHandler weiter. Auch das Erstellen, Updaten und Löschen von Schlüsseln wird vom KeystoreHandler erledigt.

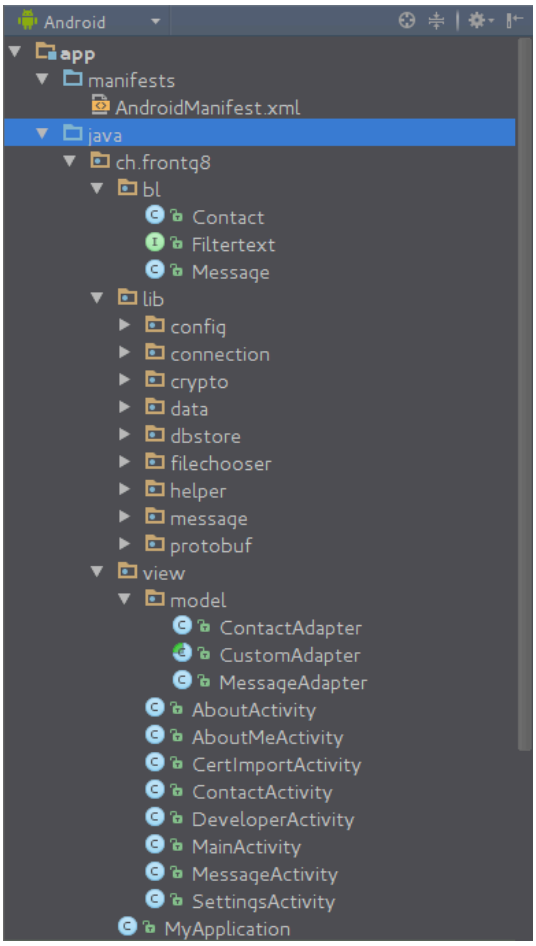
9. Ergebnisse

9.1 Strukturierung der Software

Wie schon bei dem SE2 Projekt, wurde die Software in sinnvolle Teilbereiche unterteilt, wodurch die Abhängigkeiten übersichtlich bleiben. Dadurch kann das Projekt leichter angepasst oder erweitert werden.

9.1.1 Business-Logik (bl)

In der Business-Logik finden sich die Klassen "Contact" und "Message" welche direkt dem Domänen-Modell entnommen sind.



9.1.2 View

Im Package "view" finden sich alle Views, in der Android-Terminologie Activites genannt, wieder. Im Package "model" befinden sich die View-Adapter, welche für das Befüllen und Aktualisieren der Listen zuständig sind.

9.1.3 lib/config

Die LibConfig (unsere Konfigurationsbibliothek) ist die Schnittstelle um Einstellungen im Android-Spezifischen Speicher für Einstellungen simpel und effizient zu speichern. Wir speichern hier alle Werte, welche in den Einstellungen geändert werden können. Darüber hinaus merken wir uns hier, welche Nachricht zuletzt vom Server abgerufen wurde, damit wir nicht jedes Mal alle Nachrichten abrufen müssen.

9.1.4 lib/connection

Im diesem Paket befinden sich all diese Bestandteile, welche dafür sorgen, dass die Verbindung zum Server hergestellt und gehalten werden kann. Dabei wird die Netzwerkkommunikation asynchron geführt, wie es die Android-Guidelines fordern [8]. Der ConnectionService, welcher den TcpClient hält, detektiert, wenn die Verbindung zum Server unterbrochen wird. Darauf notifiziert er den Nutzer. Dieser hat nun die Möglichkeit, die Anwendung anzuweisen, die Verbindung wiederaufzubauen. Zudem wird bei jedem Sendeversuch des Nutzers versucht die Verbindung wiederherzustellen, falls diese unterbrochen sein sollte.

[8] http://developer.android.com/training/basics/network-ops/connecting.html

9.1.4.1 Nachrichten empfangen

Der TcpClient unterhält einen Run Loop, welcher dazu dient regelmässig zu prüfen, ob dem TlsClient ungelesene Bytes im Buffer vorliegen. Wenn dieser eine Nachricht empfängt, liest der TcpClient diese aus und leitet sie an den ConnectionService weiter. Dieser wiederum reicht die Nachricht an den DataService weiter.

9.1.4.2 Nachrichten senden

Wenn der ConnectionService vom DataService den Auftrag bekommt eine Nachricht zu versenden, dann leitet dieser die Nachricht an den TcpClient weiter, welcher den Thread notifiziert, der fürs Senden der Nachrichten zuständig ist. Dieser Thread schreibt dann die Nachrichtenbytes in den OutputStream des SslSockets des TlsClients.

9.1.5 lib/crypto

Die Kryptographie Bibliothek bietet die notwendige Funktionalität zum Ver- und Entschlüsseln von Nachrichten an. Mehr Informationen zur Verschlüsselung finden sich im Kapitel "Crypto"

9.1.6 lib/data

Das Kernstück der Datenbibliothek ist, wie jenes der Verbindung, als Service implementiert und hält alle flüchtigen Daten die zur Laufzeit entschlüsselt vorhanden sein müssen, wie etwa Kontakte und deren Nachrichten. Dieser Service ist zentraler Bestandteil unserer Software und nimmt via dem Android Messaging-System Befehle vom Userinterface entgegen und führt entsprechende Aktionen auf den vorgehaltenen Daten, auf dem persistenten Speicher oder auf der Netzwerkverbindung aus.
Mehr zum DatenService findet sich im Kapitel "Daten Service"

9.1.7 lib/dbstore

Die Library Dbstore nimmt Kontakte oder Nachrichten entgegen und speichert sie in einer SQLite Datenbank. Beim Programmstart oder bei Bedarf liest die Library die Daten wieder aus der Datenbank und erstellt die benötigten Objekte.

9.1.8 lib/filechooser

Aufgrund Design-Entscheidungen von Android wird kein File-Chooser zur Verfügung gestellt[9], deshalb haben wir in der Library File-Chooser einen einfachen File-Chooser umgesetzt, damit wir Dateien bzw. Verzeichnisse im User-Interface darstellen können.

9.1.9 lib/helper

Dieses Paket enthält nur eine Tuple-Klasse, welche wir als Rückgabetyp verwenden, da Java keine integrierte Klasse bereitstellt, die es erlaubt Wertepaare abzubilden.

9.1.10 lib/message

Die Library Message stellt alle nötigen Funktionen für den Umgang mit Frontg8-Profobuf-Nachrichten zur Verfügung. Dies umfasst eine Klasse, welche die Integerwerte welche für den Typ der Nachricht steht, als Felder besitzt, damit keine Integerwerte im Code verwendet werden müssen. Des Weiteren ist da die Klasse MessageHelper. Sie bietet diverse Methoden an, um mit geringem Aufwand Frontg8-Protobuf-Nachrichten zusammenzubauen oder empfangenen Nachrichten ihr Inhalt zu entnehmen.

[9] http://developer.android.com/guide/topics/providers/document-provider.html

Diese Bibliothek greift auf die LibCrypto zurück, falls Entschlüsselungs- oder Verschlüsselungsarbeit ansteht.

9.1.11 lib/protobuf

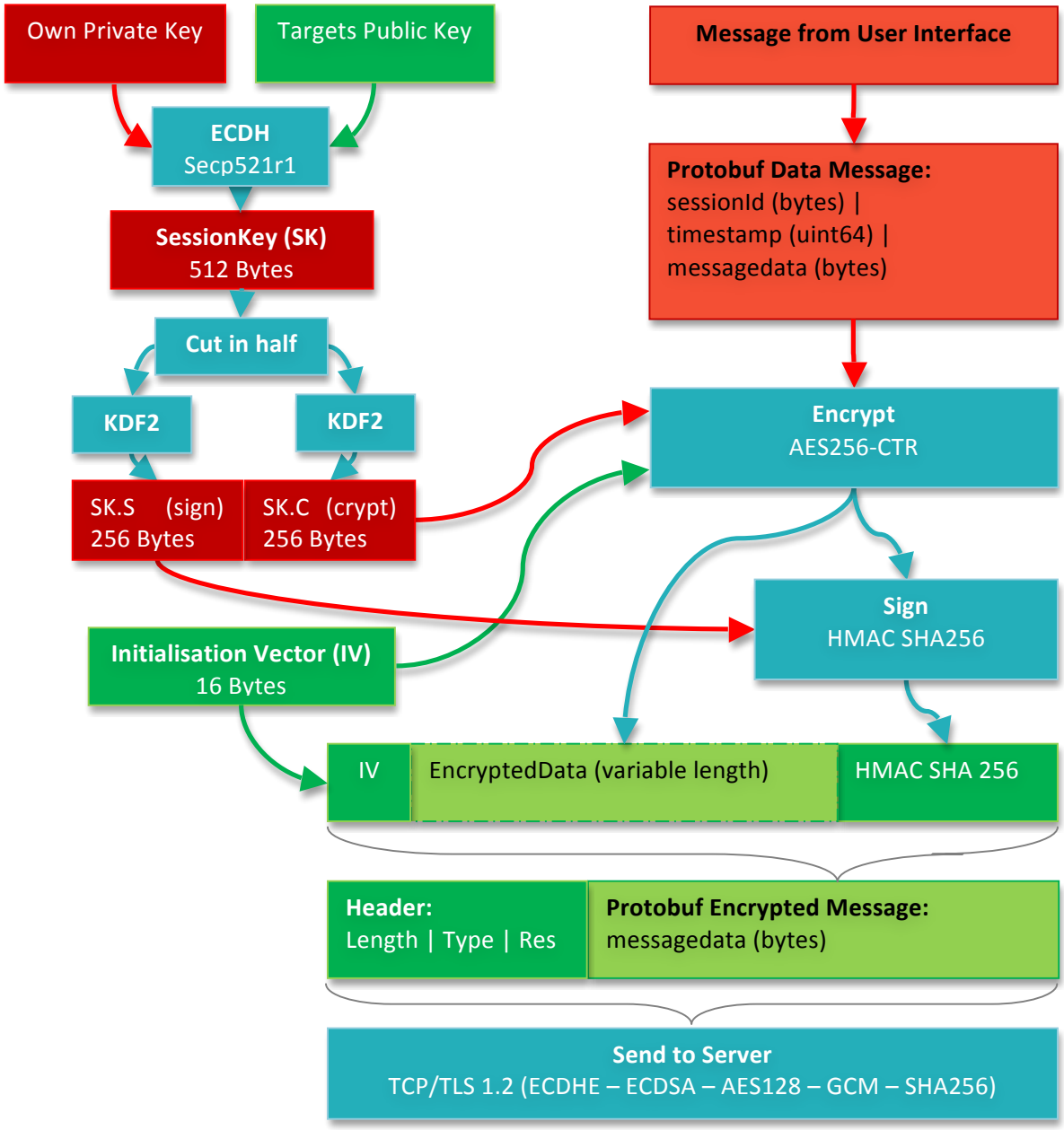
Die Library Protobuf enthält den durch Protobuf generierten Code, welcher unsere Protokollimplementierung für Frontg8 darstellt. Da der Code vom Protobuf-Compiler erstellt wird, wird dringend geraten ihn nicht zu modifizieren.

9.1.12 Kryptographie Flussdiagramm

Die kryptographischen Abläufe lassen mit dieser Abbildung aufzeigen. Dabei gilt folgender Farbcode.

Rote Felder	Bilden geheime Daten ab. Vor allem die geheimen Schlüssel.
Grüne Felder	Bilden öffentliche Daten wie Publickey und Initialisation-Vector ab.
Hellrote Felder	Sind unverschlüsselte Nachrichten die noch geheim sind.
Hellgrüne Felder	Sind verschlüsselte und somit öffentliche Nachrichten.
Hellblaue Felder	Bilden Prozesse wie Verschlüsseln, Signieren usw. ab.

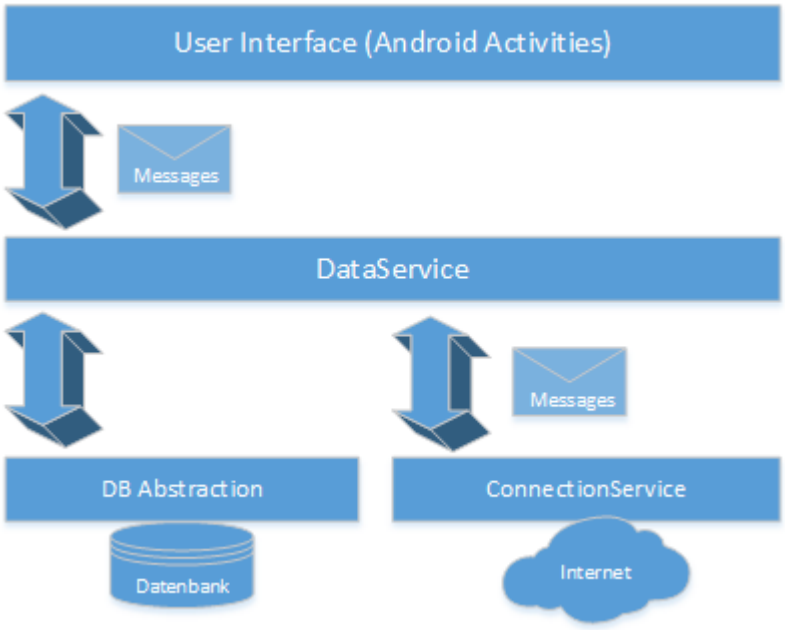
Details zu den kryptographischen Vorgängen sind dem Kapitel Kryptographie zu entnehmen.



9.2 Datenhaltung

9.2.1 Herausforderung

Da wir die Daten verschlüsselt persistieren, mussten wir uns mit dem zusätzlichen Zeitaufwand für das Ver- und Entschlüsseln auseinandersetzen. Darüber hinaus baut Android seine Applikationen immer wieder neu auf, z.B. nachdem das Gerät rotiert wurde. Damit wir nun nicht bei jeder Bildschrimaktualisierung die Daten neu aufbereiten müssen, ist eine stabile Datenhaltung anzustreben. Um dies umzusetzen, erstellten wir einen Service, der diese Aufgabe übernimmt. Damit nun dieser Service nicht dauernd neu gestartet wird, beschlossen wir ein angepasstes ApplicationObject zu erstellen. Dieses ApplicationObject ersetzt die Basisklasse, auf der jede Android-Anwendung aufsetzt. Die Erweiterung erlaubt es, den DataService beim Start von frontg8 mit zu starten. Darauf können alle anderen Teile einfach zum Service verbinden. Dies macht den DataService zum Kernstück der Businesslogik unserer Applikation.



9.2.2 Hauptaufgaben

Die Hauptaufgabe des DataService ist es, alle relevanten Daten vorzuhalten und bei Bedarf so weit zu bearbeiten, dass sie an andere Teile der Applikation weitergegeben werden können. Zu diesem Zweck werden folgende Komponenten vom DataService vorgehalten:

Keystore Handler	Mit diesem können die Schlüssel verwaltet und verwendet werden. Mit seiner Hilfe können Nachrichten verschlüsselt oder entschlüsselt werden.
Nachrichten Abonnenten	Wenn eine Nachricht eingeht, oder wenn eine beim Start entschlüsselt wird, werden die hier registrierten Abonnenten (Activities) informiert.
Kontakt Abonnenten	Wenn sich bei einem Kontakt etwas ändert, z.B. wenn eine neue ungelesene Nachricht vorhanden ist, werden die hier registrierten Abonnenten informiert.

Datenbank Verbindung	Alles was persistiert werden muss, oder von der Datenbank gelesen werden soll, erfolgt hierüber.
Connection Service	Um nach aussen zu kommunizieren wird dieser Service verwendet.
Notification Manager	Wenn neue Nachrichten eintreffen und frontg8 nicht geöffnet ist, wird dem Nutzer eine Notifikation angezeigt.

9.2.3 Initialisierung

Sobald der DataService vom Android System erstellt wurde, beginnt das Initialisieren des DataService. Dazu startetet dieser seinen ConnectionService und fragt den Server, ob neue Nachrichten vorhanden sind. Nun wird ein asynchroner Task gestartet, der zuerst alle Kontakte und deren Nachrichten von der Datenbank liest, dann die Nachrichten sogleich entschlüsselt und anschliessend dem Kontakt anhängt.

9.2.4 Kommunikation

9.2.4.1 Android Messenger

In Android basiert die Kommunikation zwischen den einzelnen Teilen der Applikation auf einem System von Messengern[10]. Wenn eine Activity oder ein Service auf einen anderen Service verbindet, dann erhält diese vom Service ein Messenger, der es der Activity erlaubt mit ihm zu kommunizieren.

9.2.4.2 Nachrichtentypen

Der Messenger unterscheidet die Nachrichten anhand von Integerwerten. Damit im Code keine Integerwerte verwendet werden mussten, erstellten wir eine Klasse MessageTypes, die es erlaubt Werte als Felder mitzugeben und so Fehler zu reduzieren. Dieses Vorgehen ist auch als Refactoring "Replace Magic Number with Symbolic Constant"[11] bekannt. Die Nachrichtentypen sollten nicht mit den Nachrichtentypen für Frontg8-Protobuf verwechselt werden. Der ConnectionService und der DataService haben jeweils ein eigenes Set von MessageTypes.

9.2.4.3 Handler

Der DataService hält einen Handler für die Kommunikation mit dem ConnectionService und einen für die Kommunikation mit dem User Interface.

9.2.4.3.1 Connection-Incomming-Handler

Wenn Nachrichten vom ConnectionService hereinkommen, werden diese geparst, entpackt, entschlüsselt und abgespeichert. Wenn die Verbindung abbricht, wird der DataService darüber informiert. Mögliche Anfragen sind:

MSG	Eine eingehende Nachricht vom ConnectionService. Es wird versucht aus den Bytes eine Frontg8-Protobuf-Notification Nachricht zu parsen. Wenn dies erfolgreich war, wird mittels einer MessageHelper-Funktion eine ArrayList aller Frontg8-Protobuf-Encrypted extrahiert. All diese Encrypted Nachrichten werden nun an einen asynchronen Task weitergegeben, der für jede Nachricht den verschlüsselten Inhalt an die LibCrypto weitergereicht, die ihn entschlüsselt. Aus den entschlüsselten Bytes wird eine Frontg8-

[10] http://developer.android.com/guide/components/bound-services.html#Messenger
[11] http://refactoring.com/catalog/replaceMagicNumberWithSymbolicConstant.html

	Protobuf-Data Nachricht geparst und sogleich dem Kontakt angehängt. Nun wird die verschlüsselte Nachricht auf die Datenbank gespeichert und alle Nachrichten-Abonnenten dieses Kontakts informiert.
Connection_Lost	Der ConnectionService hat die Verbindung verloren. Nun wird über den Notification Manager des DataServices eine entsprechende Notification angezeigt.

9.2.4.3.2 Data-Incomming-Handler

Alle Anfragen des User Interfaces werden hier empfangen und behandelt. Mögliche Anfragen sind:

Get_Contacts	Der Sender der Nachricht wird den Kontakt-Abonnenten hinzugefügt und es werden ihm alle Kontakte zugesandt.
Unregister_Contacts	Das Kontakt-Abo des Senders wird wiederrufen.
Get_Contact_Details	Dem Sender wird der Kontakt mit der entsprechenden UUID zugesandt.
Update_Contact	Der Kontakt wird aktualisiert im DataService und in der Datenbank.
Remove_Contact	Der Kontakt wird aus dem DataService und der Datenbank entfernt und die Schlüssel werden gelöscht.
Contains_SK	Dem Sender wird mitgeteilt, ob Session Schlüssel für die im Parameter angegebene UUID existieren.
Register_For_Messages	Der Sender wird als Nachrichten-Abonnent für die mitgesandte UUID registriert.
Unregister_For_Messages	Der Sender wird aus der Liste der Nachrichten-Abonnenten für die mitgesandte UUID ausgetragen.
Send_MSG	Der DataService verschlüsselt die mitgesandte Nachricht für die spezifizierte UUID und reicht sie dem ConnectionService weiter um sie zu versenden.
Request_MSG	Veranlasst den DataService dazu, den Server nach neuen Nachrichten zu fragen.
Get_Key	Dem Sender wird der eigene PublicKey zugesandt.
Change_PW	Ändert das Passwort des KeyStores.
Gen_New_Keys	Generiert ein neues Schlüsselpaar und handelt mit dem neuen Privatekey die Session Schlüssel neu aus.
Del_All_MSG	Löscht alle Nachrichten des mittels UUID angegebenen Kontaktes im DataService und in der Datenbank.
Reset	Setzt die Applikation zurück, löscht alle Session Schlüssel und alle Nachrichten.
Reset_Unread	Setzt für den Kontakt mit der angegebenen UUID den Zähler für ungelesene Nachrichten zurück.
Export_Key	Exportiert das eigene Schlüsselpaar als .pem Datei an den angegeben Pfad.
Import_Key	Importiert ein Schlüsselpaar aus einer .pem Datei vom angegeben Pfad.
Import_Cacert	Importiert ein Zertifikat vom angegebenen Pfad.
Connect	Weist den ConnectionService an ein Verbindungsversuch zu starten, falls keine Verbindung besteht.
Reconnnect	Weist den ConnectionService an die Verbindung abzubauen und eine Neue aufzubauen.

10. Schussfolgerungen

10.1 Was wurde erreicht

10.1.1 Produkt

Das ursprüngliche Ziel den Funktionsumfang des Python-Clients in Android umzusetzen wurde vollumfänglich erreicht. Darüber hinaus haben wir das Scannen des Public-Keys mittels QR-Code umgesetzt, was es deutlich erleichtert, den Client im Alltag zu benutzen.

10.1.2 Know-How

Wir haben uns während der Semesterarbeit intensiv mit Android befasst und wissen inzwischen, wie auch etwas grössere Android-Applikationen umgesetzt werden können.

Zunächst haben wir uns mit der Krypto-Infrastruktur auseinander gesetzt. Dabei ist es auch wichtig, sich mit den API-Levels zu beschäftigen. Hier gilt es für jede Applikation festzulegen, ob man auf einen älteren API-Level setzt um alte Geräte zu unterstützen, oder ob man auf die neuen Funktionen des höheren API-Levels nicht verzichten kann. Wir haben uns für unser Projekt dazu entschieden einen möglichst hohen API-Level zu benutzen, damit wir von den neuen Sicherheits-Features möglichst viel profitieren können.

Anschliessend haben wir uns mit dem Konzept der Activities und Services befasst. Hier mussten wir unsere Konzepte an die Android-Architekur anpassen, da sich das Neu-Laden der Activities stark auf den Programmablauf auswirkt.

10.2 Beurteilung Ergebnis

Nach unserem Ermessen konnten wir die anfänglich festgelegten Ziele erfüllen, die wir in keinem Konkurrenz-Produkt gefunden haben. Wir haben damit eine benutzbare Android-Applikation, die genau unseren Bedürfnissen entspricht. Einzig was die Inbetriebnahme und das UI-Design angeht, können wir nicht mit den am Markt etablierten Lösungen mithalten. Für uns ist das Projekt ein voller Erfolg. Wir schufen einen funktionsfähigen frontg8 Client für Android und konnten dabei auch viel Wissen gewinnen, welches uns für weitere Arbeiten an frontg8 oder auch anderen Android-Projekten zugute kommt.

10.3 Weiteres Vorgehen

Die Android-Applikation ist inzwischen gut benutzbar, wir haben uns aber aus Zeitgründen nicht intensiv mit dem Userinterface beschäftigt. Google gibt hier mit dem Material-Design Vorgaben[12], die dem Entwickler helfen, eine typische Android-Applikation zu bauen. Soll die Applikation produktiv verwendet werden, so ist ein Umbau des Userinterface mit den Vorgaben aus dem Material-Design sicher einer der nächsten Schritte.

[12] http://developer.android.com/design/material/index.html

10.4 Verbesserungsvorschläge

10.4.1 Technisch

An einigen Stellen lässt sich die Architektur noch etwas vereinfachen. Beispielsweise haben wir zwei Hintergrund-Services implementiert, wobei wir nach dem heutigen Wissensstand mit Einem auskommen würden. Allgemein sind wir aber sehr zufrieden mit der Implementation und sehen keinen Grund dafür, dass man etwas ändern müsste.

10.4.2 Projektvorgehen

Eine Technologie einzusetzen, die man nicht so gut kennt ist ein Risiko. Setzt man auf Technologien, die man schon kennt, so lässt sich das Projekt viel besser planen und abschätzen. In unserem Fall sind wir das Risiko aber sehr gerne eingegangen, da wir bei dem Projekt dadurch sehr viel lernen konnten.

11. Literaturverzeichnis

Bei unserer Arbeit haben die anschliessend aufgeführten Kapitel aus unserem SE2-Projekt übernommen und für diese Arbeit entsprechend überarbeitet. Die Dokumente aus dem SE2-Projekt sind öffentlich zugänglich auf der Homepage von frontg8: http://www.frontg8.ch/

- Kapitel "Systemübersicht": SE2-Projekt, SoftwareArchitektur_frontg8_v1-3.pdf, Kapitel "Systemübersicht"
- Kapitel "Kryptographie": SE2-Projekt, Crypto_frontg8_v1-3.pdf
- Kapitel "Protokollspezifikation": SE2-Projekt, Domainanalyse_frontg8_v1-4.pdf, Kapitel "Protokoll"
- Kapitel "Usecases": SE2-Projekt, Domainanalyse_frontg8_v1-4.pdf, Kapitel "Usecases"

An einigen Stellen in diesem Dokument verweisen wir mittels Fussnoten auf die Android-Dokumentation (http://developer.android.com/), welche für uns die erste Anlaufstelle für Fragen zu der Entwicklung unter dieser Plattform darstellt.

12. Anhänge

- Persönlicher Bericht von Tobias Stauber
- Persönlicher Bericht von Ueli Bosshard
- Glossar
- Eigenständigkeitserklärung
- Sitzungsprotokolle

12.1 Persönlicher Bericht von Tobias Stauber

Auf die Semesterarbeit zurückblickend kann ich sagen, dass ich in diesem Semester viel lernte. Meine Kenntnisse über Android wurden vertieft und gefestigt. Da unsere SA ein Projekt war, das wir aus Eigeninitiative vorgeschlagen haben, bin ich tief zufrieden mit dem erzeugten Resultat und werde das Projekt frontg8 gerne in meiner Freizeit weiterführen.
Es gibt am Android-Client noch viel zu verbessern. Das User Interface sollte überarbeitet werden. Ich bin motiviert meine Erfahrungen mit der Netzwerkkommunikation in eine überarbeitete Version einfliessen zu lassen. Ich habe viel gelernt und könnte inzwischen vieles wesentlich eleganter umsetzen. Zudem muss die Applikation, vor dem Release im Play-Store, noch auf das Material-Design angepasst werden.
Ich habe die Kryptographie unter Java und Android besser zu verstehen gelernt. Ich kann nun für die Richtige der diversen Technologien entscheiden und sie zielgerichtet anwenden. Obwohl ich nur wenig mit den Android-Userinterfaces zu tun hatte, konnte ich mit Unterstützung von Ueli viel verstehen und fühle mich bereit eigene Projekte anzugehen.
Mein grösster Fehler in dieser Arbeit war es, zu glauben, dass es einfacher sein würde auf den Word-Dokumenten aus dem SE2 Projekt aufzubauen. Wie sich herausstellen sollte traf genau das Gegenteil ein. Heute würde ich alles in LaTeX schreiben, auch wenn dies bedeutet, dass alles in Handarbeit aus den Word-Dokumenten übernommen werden muss.
Ich bin dankbar für die Unterstützung von Professor Steffen, der uns in Kryptografie basierten Themen immer kompetent beriet und unterstützte. Und dank den Android-Tipps von Mirko Stocker und Tobias Brunner konnten wir einige Verständnisprobleme schnell und effizient aus dem Weg räumen.
Punkte, die ich für die Bachelor Arbeit verbessern werde sind natürlich auch vorhanden. Wenn’s ums Zeiterfassen ging, liess meine Disziplin stark zu wünschen übrig. Auch das Aktualisieren der Tickets in Redmine ging bei meinem Arbeitseifer unter.
Schlussendlich wurde das Projekt erfolgreich und glücklich abgeschlossen. Es war eine interessante Erfahrung so lange so eng mit einem Kollegen zusammenzuarbeiten und es entstanden hin und wieder kleinere Reibereien, welche zum Glück jeweils schnell beigelegt werden konnten. Ich bin Professor Steffen dankbar für sein Angebot frontg8 als SA weiterzuführen.

12.2 Persönlicher Bericht von Ueli Bosshard

Diese Studienarbeit war für mich eine ganz spezielle Erfahrung. Da wir den Auftrag für das Projekt selbst formuliert haben, konnten wir uns ganz nach den eigenen Interessen einsetzen. Ausserdem konnten wir von der Analyse aus dem vorangegangenen Projekt profitieren und mussten so nicht ganz von vorne anfangen. Dadurch konnten wir uns ganz auf die Android-spezifische Umsetzung kümmern und haben sehr viel über die Android-Plattform gelernt.

Wie zu erwarten funktioniert nicht alles immer auf Anhieb, wenn man sich mit etwas Neuem auseinander setzt. Zum Teil vergingen Tage, bis wir auch nur wieder einen kleinen Fortschritt vorweisen konnten. Es gab aber auch Momente, bei denen es überraschend schnell voran ging. Beispielsweise die Generierung des QR-Codes konnte ich innerhalb von zwei Stunden umsetzen. Nach zwei weiteren Stunden konnten wir dann den QR-Code auch wieder einlesen. - Solche Kleinigkeiten haben mich sehr motiviert.

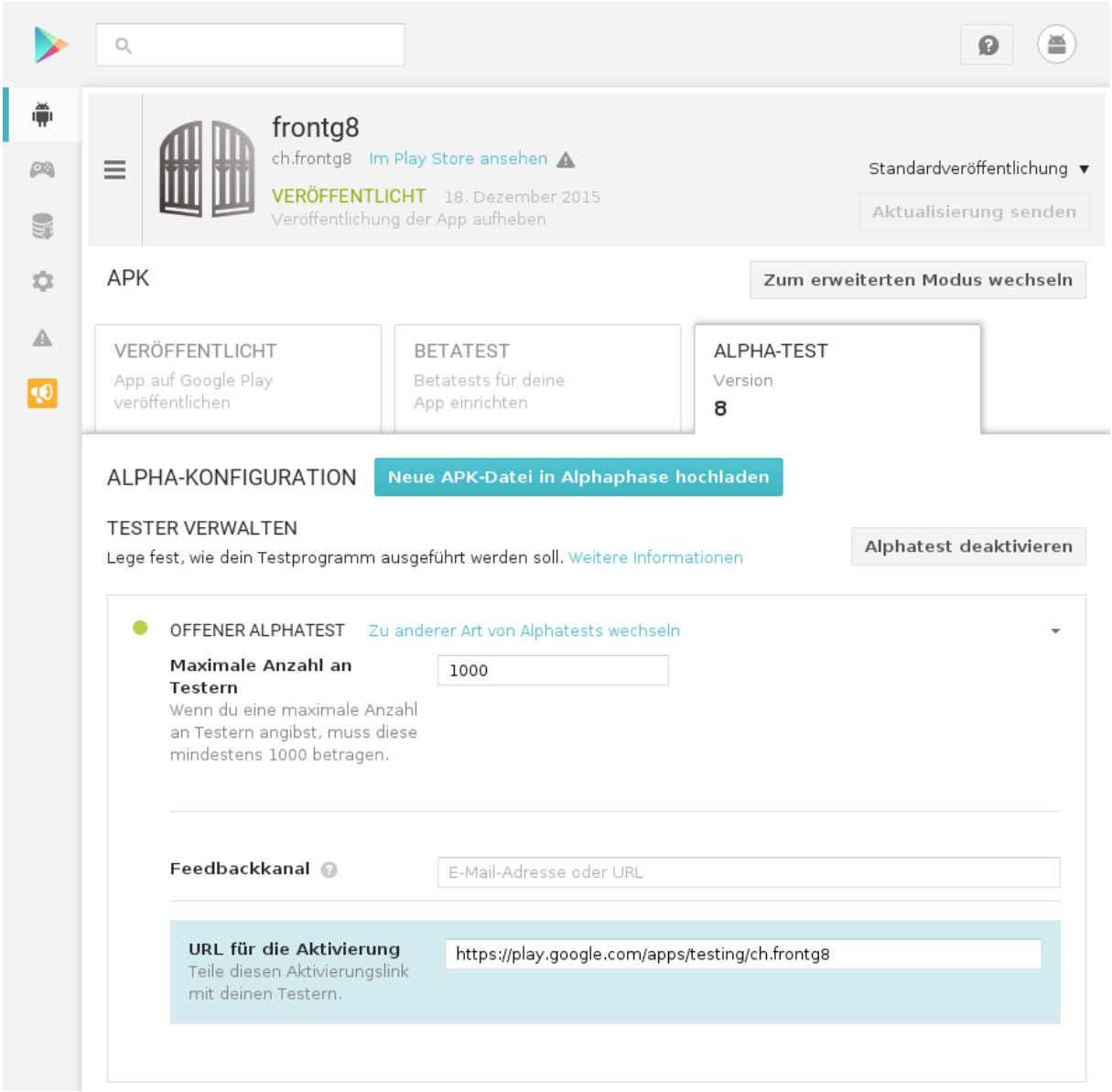
In einem Punkte habe ich mich komplett verschätzt, ich habe mich in zu viele Module eingeschrieben und musste mir deshalb die Zeit sehr gut einteilen. Mitte Semester musste ich mich dazu durchringen ein Modul zugunsten der Studienarbeit fallen zu lassen.

Wenn ich die Applikation genau betrachte, muss ich schon zugeben, dass sie etwas hübscher aussehen könnte, aber was die Funktionalität angeht, bin ich sehr zufrieden. Für eine Veröffentlichung würde ich mir wirklich nur noch eine Umsetzung mit Material-Design wünschen, ansonsten sind wir wirklich schon sehr weit gekommen.

Mir hat die Arbeit sehr viel Spass bereitet und diskutiere jeweils gerne mit Personen, die sich für unsere Arbeit interessieren. Ich finde vor allem spannend, dass wir uns bewusst dazu entschieden haben, einige Dinge anders als die etablierten Chatsysteme zu handhaben, auch wenn es auf Kosten der Usability geht.

Ich denke wir haben in dieser Arbeit vor allem auch sehr viel davon profitiert, dass wir uns mit anderen Personen ausgetauscht haben. Die Gespräche mit Prof. Dr. A. Steffen, Mirko Stocker und Tobias Brunner haben uns zu einer praxistauglichen Lösung geführt. Als sich gegen Ende der Arbeit ein Fehler beim inkrementellen Abrufen der Nachrichten vom Server bemerkbar gemacht hat, konnten wir auf die Hilfe von Felix Morgner zurück greifen. Er hat sich für uns nochmal an den Server-Code gesetzt, hat den Fehler repariert und hat uns somit das Feature gerettet. An dieser Stelle vielen Dank an alle, die uns unterstützt haben!

Der folgende Screenshot zeigt unsere Applikation im Play Store. Er symbolisiert das erreichen unseres persönlichen Ziels: Eine funktionierende Android-Applikation anbieten zu können.



12.3 Glossar

Abkürzung	Erklärung
0xFF	Eine Nummer im hexadezimalen Format
Activity	Unter Android eine Ansicht des User Interfaces
AES	**Advanced Encryption Standard** ein Algorithmus um Daten mit einem symmetrischen Schlüssel zu verschlüsseln
AES128-CBC	**AES** mit **128 Bit** Länge und **CBC** als Betriebsmodus
AES256-CTR	**AES** mit **256 Bit** Länge und **CTR** als Betriebsmodus
Alice und Bob	Zwei metasyntaktische Personennamen, welche zur Erläuterung von Abläufen verwendet werden.
Android	Ein Betriebssystem für Mobilgeräte wie z.B. Smartphones
Android BC	Android wird mit einer abgespeckten Version des Kryptographie Bibliothek Bouncy Castle ausgeliefert. Da darin auch der Support von ECDH gestrichen wurde, mussten wir auf Spong Castle, eine vollständige Implementation der Bibliothek, zurückgreifen.
API	**Application Programming Interface**, eine Schnittstelle über die man mit einer Applikation oder einem Betriebssystem wie z.B. Android kommunizieren kann.
Asio	Eine Bibliothek für C++, die **asynchronen Input und Output** handhaben kann.
AsyncTask	Eine von Android angebotene Möglichkeit um Aufgaben asynchron zu erledigen.
BC	Siehe **Bouncy Castle**
BLOB	**Binary Large Object** (Grosses Binäres Objekt). Diese Art von Objekt verfügt über eine nicht-textbasierte Struktur.
Bouncy Castle	Eine unter Java verbreitete Kryptographie Bibliothek. Sie unterstützt alle verbreiteten Algorithmen.
Byte[]	Die [] zeigen, dass es sich um ein Array von Bytes handelt.
CBC	**Cipher Block Chaining** ist ein Betriebsmodus für symmetrische Blockciphers in der Kryptographie. Dies ist notwendig, da sonst gleiche Blöcke immer gleich verschlüsselt würden, was die Sicherheit schwächt.
Cipher	Eine Funktion, die Eingangsdaten verschlüsselt. Sie wird z.B. mit Algorithmen wie **AES** verwendet.
Ciphertext	Der verschlüsselte Plaintext
CLI	**Command Line Interface** / Kommandozeile
CRUD	**Create Read Update Delete** (Erstellen Lesen Editieren Löschen)
CTR	**Counter Mode** ist ein Betriebsmodus für symmetrische Blockciphers in der Kryptographie. Dabei wird ein Zähler nach jeden n Bits inkrementiert, wodurch sich in einer Nachricht wiederholende "Blöcke" nicht zu gleichem Ciphertext führen. Dies ist notwendig, da sonst die Sicherheit geschwächt würde.
Datenbank	Die **Datenbank** dient dazu Daten dauerhaft mit System abzuspeichern.
DB	Siehe **Datenbank**
Decrypted	Entschlüsselt
Digest	Die "Verdauungsfunktion" einer Hashfunktion. Dadurch wird aus den Eingangsdaten den eigentlichen Hash gebildet.
EC	Elliptic Curve (Elliptische Kurven). Sie werden genutzt um ähnlich harte Probleme wie das Problem des diskreten Logarithmus, zu schaffen.

ECDH	**Elliptic Curve Diffie-Hellman Key Exchange** ist ein Schlüsselaustauschprotokoll. Es dient dazu, aus dem eigenen Geheimnis und dem öffentlichen Schlüssel des Partners einen starken symmetrischen Schlüssel abzuleiten.
Encrypted	Verschlüsselt
End-zu-End Verschlüsslung	Eine Verschlüsslung, die direkt zwischen zweien Kommunikationspartnern stattfindet. Der Server, über den die Kommunikation fliesst, kann nichts entschlüsseln oder unbemerkt verändern.
frontg8d	Die Serverkomponente unseres Chatsystems.
GCM	**Galois/Counter Mode** ist eine effiziente Implementation des Counter Mode (**CTR**) für Blockciphers.
Geheimnis	In diesem Dokument ist damit oftmals ein geheimer Schlüssel oder der geheime Teil eines Schlüsselpaars gemeint.
git	Ein Versionsverwaltung System
Hash	Eine Prüfsumme über einem Eingabewert. Für gleichbleibenden Input ist der Output immer gleich.
HMAC	Der **Keyed-hash Message Authentication Code** dient dazu, Gewissheit über den Ursprung von Daten oder Nachrichten zu erhalten und ihre Integrität zu überprüfen. Dies ist möglich, da beide Parteien den HMAC berechnen können ohne die Nachricht zu entschlüsseln. Wenn der HMAC nicht übereinstimmt, ist auch die Nachricht nicht zu entschlüsseln.
HSR	**Hochschule für Technik Rapperswil**
IMEI	Eine **International Mobile Station Equipment Identity** Number dient dazu, ein mobilnetzfähiges Gerät eindeutig zu identifizieren.
initial	Ursprünglich, von Anfang an vorhanden.
Initialisation Vector	Eine Zahl, die den Startwert für den Counter beim Counter Mode (**CTR**) vorgibt. Diese Zahl wird mittels einer **PRF** gewählt. Dies ist wichtig, damit gleicher Plaintext nicht in gleichem Ciphertext resultiert.
IV	Siehe **Initialisation Vector**
JNI	**Java Native Interface**, eine **API** um aus der JVM auf Systemfunktionen des Hostsystems zuzugreifen.
Journal	Die "Liste" der Nachrichten, die der Server momentan im Buffer hält.
JVM	**Java Virtual Machine**, die virtuelle Maschine, in der der Java-Code ausgeführt wird.
KDF2	Die **Key Derivation Function 2** wird verwendet um aus einem symmetrischen Schlüssel, mit einem weiteren Wert, einen neuen Schlüssel abzuleiten. Dies verwenden wir, damit wir die Schlüssel nicht zu lange verwenden müssen.
Keystore	Eine Möglichkeit, unter Java, geheime Schlüssel sicher zu speichern.
Keystore Handler	Eine Bibliothek um den **Keystore** einfacher zu bedienen.
KH	Siehe **Keystore Handler**
MAC	Eine Kurzform von **Message Authentication Hash** (siehe HMAC)
MAC-Adresse	Die Adresse einer Netzwerkkarte (**Media Access Control**)
Meta-Daten	Nicht den Inhalt der Kommunikation, sondern Verbindungsinformationen betreffende Daten wie z.B. wer mit wem, wann kommuniziert hat.
MS	Ein **Meilenstein** im Projektfortschritt.
MSG	Kurzform für Message (Nachricht)
Notify Message	Eine Nachricht, die der Server verwendet, um Clients mit neu eingegangen Nachrichten zu versorgen.

PBKDF2-SHA256	**Password Based KDF 2** ist eine Funktion um aus einem Passwort einen Schlüssel abzuleiten. Dabei wird die **SHA256** Hashfunktion verwendet. Dies führt zu einem 256Bit langem symmetrischen Schlüssel.
persistieren	Dauerhaft speichern so, dass auch nach einem Neustart des Geräts alles noch vorhanden ist.
PGP	**Pretty Good Privacy** eine Möglichkeit seinen Mailverkehr zu verschlüsseln. Dabei ist das Vertrauen dezentral aufgebaut. Es wird also dem Nutzer und anderen Nutzern vertraut, die bestätigen, dass der Nutzer der ist, der er zu sein vorgibt.
Plaintext	Unverschlüsselter Text
POSIX	**Portable Operating System Interface** eine unter Unix-Systemen verbreitete Schnittstelle zwischen Betriebssystem und Applikationen.
PRF	Die **Pseudo-Random Function** ist eine Familie von effizient berechenbaren Funktionen, die von einem Zufallsorakel praktisch ununterscheidbar sind.
Privatekey	Der private Teil eines asynchronen Schlüssels (siehe auch Geheimnis)
Protobuf	**Protobuf** ist eine von Google entwickelte Bibliothek zur Serialisierung strukturierter Daten. Google bietet Implementationen für Java, C++ und Python an. Da die Bibliothek quelloffen ist, gibt es jedoch auch für diverse andere Sprachen, wie unter anderem C, Ruby und Haskell, Implementationen. Protobuf ist unter der 3-Klausel-BSD-Lizenz veröffentlicht.
Publickey	Der öffentliche Teil eines asynchronen Schlüssels. Dieser wird dem Kommunikationspartner mitgeteilt.
QR-Code	Ein optisch leicht einlesbares Muster, ähnlich einem Strichcode
RAM-Disk	Ein Speicher, der nicht persistent ist. Es wird ein Teil des RAM des Rechners verwendet um eine sehr performante Speichermöglichkeit zu schaffen. Allerdings verschwinden alle Daten, falls die Stromversorgung ausfällt. (Dies ist bei frontg8d gewünscht und dient der Sicherheit)
Redmine	Eine Plattform für Software-Projektmanagement
SA	**Studien Arbeit** (HSR Modul)
SC	Siehe **Spongy Castle**
SE2P	**Software Engineering 2 Projekt** (HSR Modul)
secp521r1	Eine, laut NIST, für starke Kryptographie taugliche Elliptische Kurve.
serialisieren	Ein Objekt in einem Programm in einen Zustand versetzen, in dem es versandt oder abgespeichert werden kann.
Session Key	Der Session Key (**Session Schlüssel**) ist ein symmetrischer Schlüssel und wird durch Anwenden des ECDH aus dem eigenen Privatekey und dem Publickey des Partners erzeugt. Er hat eine Länge von 512 Bit. Pro Partner wird ein eigener Session Key erzeugt.
Session Key Crypto	Der Teil des SK, der verwendet wird um die Nachrichten zu verschlüsseln oder entschlüsseln. Der SK.C wird erzeugt, indem die Hälfte des SK mit KDF2 abgeleitet wird.
Session Key Sign	Der Teil des SK, der verwendet wird um den HMAC über der verschlüsselten Nachricht zu bilden. Der SK.S wird erzeugt, indem die Hälfte des SK mit KDF2 abgeleitet wird.
SHA256	**Secure Hash Algorithm 2** mit einer Hashlänge von 256 Bit.
Signatur	Eine digitale Unterschrift, die beweist ob die unterschrieben Daten wirklich vom angeblichen Absender stammen.
SK	Siehe **Session Key**
SK.C	Siehe **Session Key Crypto**

SK.S	Siehe **Session Key Sign**
Spongy Castle	Spongy Castle ist nichts weiter, als die vollständige Bouncy Castle Bibliothek, welche in das Paket SC verschoben wurde, um Namenskollisionen mit der abgespeckten Bouncy Castle Version, die mit Android geliefert wird, zu verhindern.
SQLite	Eine Datenbank Implementation
SSL Socket	Eine direkte Netzwerkverbindung mit **TLS.**
TCP	Das **Transmission Control Protocol** ist ein Protokoll um Daten übers Netz zu senden, bei dem die Übertragung garantiert ist.
TLS	**Transport Layer Security** bietet an, eine Verschlüsselung über TCP zu legen so, dass die Kommunikation, falls abgefangen, nicht einfach mitgelesen werden kann.
UseCases	Anwendungsfälle für unsere Software. Sie werden verwendet um zu spezifizieren, was man alles erreichen will (z.B. eine Nachricht schreiben und senden).
UUID	**Universally Unique Identifier** (Eine einmalige Nummer, mit der ein Kontakt eindeutig identifiziert werden kann)
Zufallsorakel	Ein **Zufallsorakel** ist ein Algorithmus, der für jede Eingabe aus X eine gleichverteilt zufällig gezogene Ausgabe aus Y zurückgibt, mit der Einschränkung, dass ein einmal bestimmter Funktionswert festliegt, also bei gleicher Anfrage auch die gleiche Antwort gegeben wird.

12.4 Eigenständigkeitserklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:

Name, Unterschrift:

Name, Unterschrift:

12.5 Sitzungsprotokolle

Kickoff
Sitzungsprotokoll Kickoff
Termin: Mi, 16.09.2015, 15:00
Teilnehmer: A.Steffen, T.Stauber, U.Bosshard
Thema: Kickoff

Sitzungen
Sitzungen finden ein Mal pro Woche statt, jeweils am Dienstag um 14:00 Uhr. Dokumente bitte bereits am Vorabend an Hr. Steffen schicken.

Dokumente

Aufgabenstellung
Zu erarbeiten:

5. Was wird umgesetzt (vgl. Beschreibung in AVT)?
6. Was ist Pflichtteil?
7. Was ist Optional?
8. Umfang: Eine A4-Seite

Grober Projektplan

- 3-4 Meilensteine
- Reserve für Dokumentation

Weitere Dokumente

- Bericht: Checkliste und Vorlagen auf HSR Intern
- Beschreibung Protokoll (aus SE-Projekt übernehmen und aktualisieren)
- Risikoanalyse
- User Inferface Mockups, Usability
- Umsetzung Security und Storage
- *Bsp. Schlüsselverwaltung auf Android? (Keystore)*
- *Kryptobibliotheken (NaCl)*

vServer
Der bestehende Server kann weiter verwendet werden.

Woche 2
Termin: Di, 22.09.2015, 14:00
Teilnehmer: A.Steffen, U.Bosshard
Thema: Aufgabenstellung

Aufgabenstellung
Aufgabenstellung ist so in Ordnung. (Ein Satz entfernt)

Grobe Zeitplanung
Bis zum nächsten Mal: Grobe Zeitplanung, was in welcher Woche? Reserve am Schluss berücksichtigen.

Woche 3
Termin: Di, 29.09.2015, 14:00
Teilnehmer: A.Steffen, T.Stauber, U.Bosshard
Thema: Wochenplan

Grobe Zeitplanung
Wochenplan ist so in Ordnung.

Nächste Schritte
Bis zum nächsten Mal: Domainmodel und Risikoanalyse

Woche 4
Termin: Di, 06.10.2015, 14:00
Teilnehmer: A.Steffen, T.Stauber, U.Bosshard
Thema: Domainmodell, Risiken

Themen
9. Domainmodell
10. Risiken

Nächste Tasks
- Abklärung Crypto-Risiken

Nächste Sitzung
Dienstag, 13.10.2015, ausnahmsweise um 13:00 Uhr.

Woche 5
Termin: Di, 13.10.2015, 13:00
Teilnehmer: A.Steffen, T.Stauber, U.Bosshard
Thema: Crypto-Abklärung, UI

Themen

11. Crypto-Abklärung, Keystore in Android
12. Umsetzung UI -> Usecases definieren

Nächste Tasks

- Use-Cases zu UI definieren
- Testumgebung, Prototyp

Woche 6
Termin: Di, 20.10.2015, 14:00
Teilnehmer: A.Steffen, T.Stauber, U.Bosshard
Thema: Aktueller Stand Prototyp

Themen

13. Aktuelle Arbeiten an Crypto-Library
14. Bisher umgesetztes im Prototyp

Hinweise

- Im Bericht Funktionalität von Keystore beschreiben, Erkentnisse festhalten