Don't Spy – Sichere Kommunikation in Ihrem Team

Lukas Ruf

Albertus-Magnus-Gymnasium Rottweil

Mai Saito

Gymnasium Trossingen

Schülerforschungszentrum Südwürttemberg (SFZ)

Standort Tuttlingen

1. April 2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Die Herausforderung - Die Lösung im Überblick

Sicherheit - jeder wünscht sie sich, sei es in den eigenen vier Wänden oder bei der privaten und geschäftlichen Kommunikation. Wir alle nutzen heute täglich Smartphone, Tablet oder Computer, um mit anderen Menschen in allen Lebensbereichen zu kommunizieren. Die Sicherheit und der Schutz der Kommunikation sowie des geistigen Eigentums sind dabei ein zentrales Anliegen. Sie gewinnen im digitalen Zeitalter täglich an Bedeutung. Das Interesse an diesem hochaktuellen Themenfeld war die Motivation, uns mit diesem Thema in seiner ganzen Breite zu beschäftigen. Dabei setzten wir uns nicht nur mit der Verschlüsselung im engeren Sinne auseinander, sondern thematisierten auch das Themenfeld des geistigen Eigentums, das bei den modernen Diensten wie Facebook, Instagram und Snapchat aufgegeben wird.
Aus diesem Grund haben wir uns seit zwei Jahren mit der Entwicklung eines eigenen Verschlüsselungsverfahrens in der Theorie und später hauptsächlich mit der Realisierung der Anwendung als moderne Software beschäftigt. Als Resultat entstand unsere plattformunabhängige Messaging-App DontSpy.

DontSpy ermöglicht die sichere Kommunikation in Einzel- und Gruppenkonversationen für definierte Personenkreise in Unternehmen sowie in ähnlichen Institutionen oder Gruppen. Hierbei entwickelten wir DontSpy auf der Basis eines weiterentwickelten und symmetrischen Verschlüsselungsverfahrens. Diese Weiterentwicklung macht statistische Kryptoanalysen, wie zum Beispiel die Häufigkeitsanalyse, unbrauchbar. Der Schlüssel zur Codierung und Decodierung von Nachrichten kann komfortabel mittels eines QR-Codes ausgetauscht werden. Wir erheben durchgehend keinen Anspruch auf das geistige Eigentum, denn alle Nachrichten werden vom Server ausnahmslos, restlos und sofort nach Abruf gelöscht. Dieser kurze Zeitraum macht einen Hackerangriff nahezu unmöglich. Außerdem wird auf keinem Gerät und zu keinem Zeitpunkt der Klartext gespeichert, sodass auch hier kein unberechtigter Abgriff der Informationen möglich ist. Der Schlüssel wird dabei nicht auf dem Server gespeichert, sondern bleibt ausschließlich verschlüsselt auf dem Endgerät, sodass auch ein Verlust des Endgerätes keine große Gefahr darstellt. Als weitere Hürde für den Hacker wird für jede neue Konversation ein eigenständiger Schlüssel generiert. Wir gewährleisten darüber hinaus, dass keine kritischen systembezogenen Informationen wie Standort, Mikrophon oder Kontakte genutzt werden.

Abbildung 1 zeigt die oben beschriebenen Aspekte und insbesondere die Rolle des Servers als temporären Speicher und Verteiler der verschlüsselten Nachrichten. Zukünftig soll der zentrale Server durch ein dezentrales Verfahren auf Grundlage eines Peer-to-Peer Netzwerks ersetzt werden.

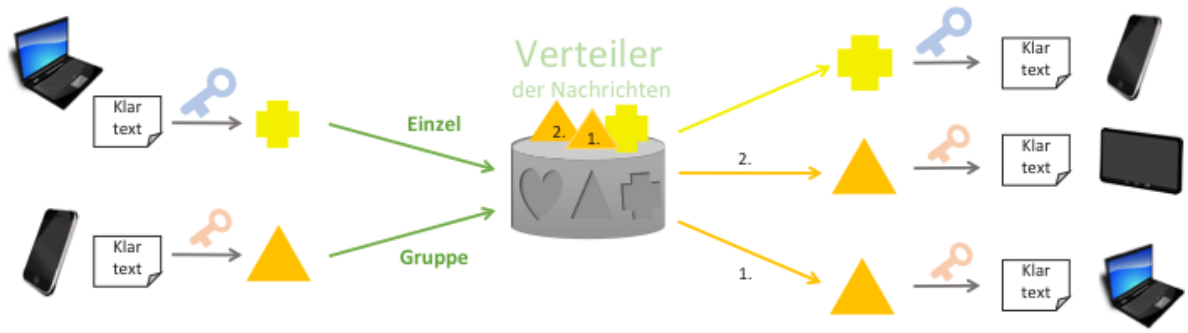

Abbildung 1: Lösung im Überblick

Obwohl sich asymmetrische Verschlüsselungsverfahren in der Praxis größerer Verbreitung erfreuen und in vielen Geschäftsprozessen asymmetrische Verschlüsselung gar notwendig ist, setzen wir auf die Weiterentwicklung eines symmetrischen Verfahrens. Wir sind der Meinung, dass asymmetrische Verschlüsselung in den von uns beschriebenen speziellen Szenarien nicht notwendig ist und wir die größere Sicherheit und höhere Schnelligkeit der symmetrischen Verschlüsselung bevorzugen und anbieten möchten. Die Nachteile der symmetrischen Verschlüsselung, wie die sichere Schlüsselübergabe und Angriffe über die Häufigkeiten, werden wir für unsere Prozesse entkräften.

2 Kryptographischer Kontext

Eine der bekanntesten symmetrischen Verschlüsselungsverfahren ist die sogenannte Vigenère-Verschlüsselung [1]. Sie baut auf der Cäsarverschlüsselung [2] auf, bei der einem Buchstaben x der Buchstabe $y = x + n \mod 26$, mit einer festen Verschiebung n zugeordnet wird. Bei der Vigenère-Verschlüsselung wird einem zu verschlüsselnden Text ein Schlüsselwort zugeordnet. Nun wird ein Quadrat mit allen 26 Verschiebungen des Alphabets genutzt, das sogenannte Vigenère-Quadrat. Es soll ein Buchstabe x, welches der Anfangsbuchstabe des Textes ist, verschlüsselt werden. So wird der Schnittbuchstabe von dem zu entschlüsselnden Buchstaben und dem n_1-verschobenen Alphabet (wobei n_1 der erste Buchstabe des Schlüsselworts ist) gesucht, welcher der verschlüsselte Buchstabe ist. Der nächste Buchstabe wird analog codiert, wobei dann der zweite Buchstabe n_2 des Schlüsselwortes als Verschiebungsparameter gilt. Wenn man beim Verschlüsseln bei n_k angekommen ist, wobei k der letzte Buchstabe des Schlüsselwortes ist, so wird wieder bei n_1 begonnen.

	Text																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
3	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
4	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
5	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
6	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
7	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
8	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
9	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
10	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
11	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
12	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
13	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
14	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
15	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
16	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
17	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
18	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
19	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
20	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
21	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
22	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
23	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
24	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
25	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
26	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Geheimtext

Abbildung 2: Vigenère-Quadrat [3]

Die Schwäche der Vigenère-Verschlüsselung ist die Rückführbarkeit auf den unverschlüsselten Text durch die Anwendung der Kryptoanalyse der Häufigkeiten [4]. Diese Methode macht sich die Tatsache zunutze, dass in fast allen Sprachen die Buchstaben unterschiedlich oft vorkommen. In der deutschen Sprache kommt beispielsweise der Buchstabe *e* mit Abstand am häufigsten vor. Um einen Text, der mit der Vigenère-Verschlüsselung codiert wurde, zu decodieren, kann man den Kasiski-Test [5] anwenden, um die Länge des Schlüsselwortes n zu ermitteln. Nun kann man jede n-te Ziffer einer Gruppe zuordnen. In diesen Gruppen kann die Häufigkeit der einzelnen Buchstaben analysiert werden und vor allem bei längeren Texten präzise bestimmt werden, welcher Buchstabe mit hoher Wahrscheinlichkeit entschlüsselt das *e* ist. Sobald der Buchstabe *e* bestimmt ist, sind alle Buchstaben entschlüsselt, da damit die Verschiebung im Alphabet bekannt ist.

2.1 Grundüberlegung unserer Verschlüsselung

Da die Kryptoanalyse der Häufigkeiten und die Mustererkennung häufig vorkommender Zeichenketten neben der Schlüsselübergabe eine große Schwachstelle der symmetrischen Verschlüsselungsmethoden ist, haben wir uns Gedanken gemacht, wie wir diese Angriffspunkte umgehen können. Dies erreichen wir unter anderem durch eine erzwungene Gleichverteilung der verwendeten Zeichen. Um die Gleichverteilung realisieren zu können, wird anstatt mit Zeichen mit sogenannten Zeichenpaaren verschlüsselt. Diese bestehen jeweils aus zwei Zeichen, die aus unserem Zeichensatz stammen. Dementsprechend gibt es bei n verwendeten Zeichen insgesamt n^2 Zeichenpaare, wobei mit unseren momentan 90 verwendeten Zeichen die Zeichenpaare mit aa, ab, ac beginnen und mit @@ enden, indem systematisch alle $90^2 = 8100$ Zeichenpaarkombinationen gebildet werden.
Je wahrscheinlicher ein Zeichen ist, umso mehr Zeichenpaare werden ihm zugeordnet. Wenn man das Zeichen verschlüsseln will, zieht man zufällig aus dem ihn zugeordneten Zeichenpaaren eines. Damit sind alle Zeichenpaare gleich wahrscheinlich. Wir verwenden Zeichenpaare deshalb, weil dies ausreicht, um genügend Elemente zu haben, um die Wahrscheinlichkeiten der einzelnen Buchstaben ausgleichen zu können, ohne dabei den verschlüsselten Text unnötig länger werden zu lassen.
Exemplarisch sei die Gleichverteilung für 90 Zeichen (siehe Tabelle 1, Spalte 2) erläutert, für die wir die sprachliche Verteilung [6] vom Institut für Deutsche Sprache (IDS) in Mannheim zur Verfügung gestellt bekommen haben.

Nummer x	Zeichen z	p_x in %	a(x)	b(x)	Intervalllänge
1	a	4,52	1	366	366
2	b	1,37	367	477	111
3	c	2,11	478	648	171
...	...	...	...	...	...
12	l	2,91	3083	3318	236
13	m	1,94	3319	3475	157
...	...	...	...	...	...
18	r	6,14	4328	4824	497
19	s	4,57	4825	5194	370
20	t	5	5195	5599	405
...	...	...	...	...	...
90	@	0,02	8099	8100	2

Tabelle 1: Ausschnitt aus der Verschlüsselungstabelle mit den Wahrscheinlichkeiten bei 90 Zeichen

Als weitere Sicherheit benutzen wir kein verschiebungsbasiertes System, wie bei der Vigenère-Verschlüsselung, sondern für jeden einzelnen Chat eine eigene zufällige Permutation der 8100 Zeichenpaare, so dass $8100!$ Möglichkeiten entstehen.
Um dann auch die erhöhte Wahrscheinlichkeit von Zeichenketten, wie z.B. ch, st oder sch, zu verschleiern, vertauschen wir nach der ersten Verschlüsselung ein weiteres Mal mit einer algorithmischen Verschlüsselung, indem wir die verschlüsselten Zeichen in einer von uns entwickelten, geheimen Methode vertauschen. Dadurch stehen die Zeichen der Zeichenpaare dann nicht immer nebeneinander, was einen Angriff weiter erschwert.

2.2 Methodik unserer Verschlüsselung

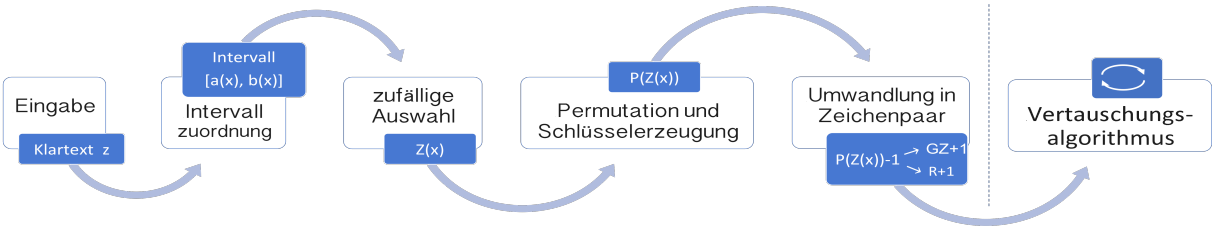

Abbildung 3: Methodik unserer Verschlüsselung

Die Verschlüsselungsmethode besteht aus fünf Schritten, um ein Zeichen zu codieren. Danach erfolgt der sechste Schritt, in dem die Zeichen des Zeichenpaares algorithmisch vertauscht werden.

Schritt 1: Die Eingabe
Das zu codierende Zeichen z, in unserem Beispiel das Zeichen *s* des Wortes *sehen*, wird eingegeben.

Schritt 2: Intervallzuordnung
Jedem Zeichen z mit der Nummer x der Tabelle 1 wird ein Intervall $[a(x); b(x)]$ und damit eine bestimmte Anzahl an Zeichenpaaren zugeordnet. Die Anzahl an Zeichenpaaren, die einem bestimmten Zeichen zugeordnet werden, hängt von der Häufigkeit dieses Zeichens in deutschen Texten ab. Auf Basis dieser Statistik erhält ein Zeichen der Wahrscheinlichkeit p_x diesen Anteil der n^2 Zeichenpaare. Im Beispiel werden dem Buchstaben s genau $90^2 \cdot 4,57\% \approx 370$ Zeichenpaare zugeordnet, wobei $a(x) = 4825$ und $b(x) = 5194$ ist. Somit ist ihm das Intervall $[4825; 5194]$ zugeordnet. Die relativen Häufigkeiten sind meist so, dass die Intervalllänge $n^2 \cdot p_x$ nicht ganzzahlig ist und das Ergebnis keine verwendbare Intervalllänge angibt. Deshalb muss das Ergebnis von $n^2 \cdot p_x$ adaptiert werden, sodass $\sum_{x=1}^{n} x^2 \cdot p_x = n^2$ bzw. alle Intervallängen in Summe n^2 ergeben, damit alle Zeichenpaare verteilt sind.

Schritt 3: Zufällige Auswahl
Nun wird aus dem Intervall $[a(x), b(x)]$ jenes Buchstabens mithilfe einer modifizierten Version von

Donald E. Knuths „Subtractive random number generator algorithm“[7], der vom C#-Compiler benutzt wird, eine Zahl $Z(x)$ ausgewählt. Je häufiger ein Zeichen statistisch gesehen in Texten vorkommt, desto mehr Zeichenpaare werden diesem zugeordnet. Durch den Algorithmus kommen alle Zeichenpaare im Durchschnitt gleich oft vor und damit sind auch alle Zeichen gleich wahrscheinlich. Aus dem vorher genannten Intervall für *s* sei die Zahl $Z = 4894$ zufällig ausgewählt.

Schritt 4: Permutation und Schlüsselerzeugung
Jener Zahl $Z(x)$ wird das Ergebnis einer Permutation zugeordnet, sodass man eine Zahl $P(Z(x))$ erhält. Die Permutation ist in unserer Verschlüsselung das Vermischen der n^2 Zeichenpaare, erneut mit Donald E. Knuths „Subtractive random number generator algorithm“[7]. So wird beispielsweise dem ersten Zeichenpaar das 1231ste Zeichenpaar zugeordnet, dem 423sten das 92ste Zeichenpaar usw. Diese Zuordnung ist unser Schlüssel. Die Anzahl der Möglichkeit wäre bei 8100 Zeichenpaaren bei $8100! \approx 1,98 \cdot 10^{28143 \cdot 10^4}$. Auf Grund dieser großen Anzahl an Möglichkeiten ist es unmöglich, bei einem Angriff alle Möglichkeiten zu probieren, beispielsweise automatisiert mit Brute-Force [8].
In dem konkret gewählten Beispiel *s* sei die permutierte Zahl $P(Z(x)) = 1009$

Schritt 5: Umwandlung von Zahl in Zeichenpaar
Die Permutation liefert ein Ergebnis in Form einer Zahl zurück. Um nun zu ermitteln, aus welchen zwei Zeichen das Zeichenpaar besteht, wird die Ganzzahldivision $\frac{P(Z(x))-1}{n} \mod n$ gerechnet. Das ganzzahlige Ergebnis GZ dieser Ganzzahldivision bezieht sich auf das erste Zeichen des Zeichenpaares; das endgültige Zeichen des Zeichenpaares ist $GZ + 1$, da bei einer Ganzzahl von 0 keine Zuordnung zu einer Zahl stattfinden kann. Dies würde bedeuten, dass die Zahl dem Zeichen Nummer 0 zugeordnet wäre, die Nummerierung der Buchstaben jedoch mit 1 beginnt. Der Rest R der Ganzzahldivision bezieht sich auf das zweite Zeichen eines Zeichenpaares. Zum Rest R muss analog 1 addiert werden, sodass das endgültige Ergebnis $R + 1$ ist. Die Additionen bei GZ und R mussten vorgenommen werden, da man zuvor 1 von $P(Z(x))$ bei der Ganzzahldivision subtrahiert hat.
Für das ursprüngliche *s* des konkreten Beispiels gilt: $\frac{1010-1}{90} = 11\ R\ 19$ und damit $GZ + 1 = 11 + 1 = 12$ und $R + 1 = 19 + 1 = 20 \rightarrow$ Zeichenpaar *lt*.
Das ursprüngliche Wort *sehen* würde dann verschlüsselt zum Beispiel '*lt,Dr3s.@r*' lauten.

Schritt 6: Vertauschung der Zeichen des verschlüsselten Textes
Die Vertauschung ist eine algorithmische Methode. Die verschlüsselte Nachricht wird in 16er-Blöcke zerteilt mit einer 16er-Permutation P vertauscht, die wie folgt ermittelt wird:

1. Jede verschlüsselte Nachricht besitzt einen UNIX-Zeitstempel T. Außerdem werden 5 zufällige, geheime Primzahlen p_i $(i = 0 - 4)$ und eine Zeichenkette $J = 0123456789ABCDEF$ benutzt.
2. Multiplikation des Zeitstempel mit den Primzahlen: $T_i = T \cdot p_i \mod 16^{10}$.
3. Umwandlung ins Hexadezimalsystem: $H_i = T_i$. (Die $\mod 10$-Rechnung von T_i in Schritt 2 erfolgte deswegen, um alle Zahlen im Hexadezimalsystem höchstens zehn stellig zu haben, damit die Zeichenketten nicht zu lang werden.)
4. Erzeugen der Zeichenkette $S = H_1H_2H_3H_4H_5J$.
5. Erzeugen der Zeichenkette P, indem jedes erste Auftreten eines Zeichens in S in P festgehalten wird und alle Dopplungen herausgestrichen werden. (Um sicher alle 16 Zeichen zu haben wurde J an den Schluss gehängt.)

Die verschlüsselte Nachricht wird nun so lange in 16er-Blöcke zerteilt und mit P vertauscht, bis sie noch 18 oder weniger Zeichen enthält. Ab dort wird mit 10er-, 6er- und 4er-Permutationen weitervertauscht, um auch sicher alle Zeichen vertauscht zu haben. Die 10er-Permutation entsteht aus den Ziffern von P. Die 6er-Permutation entsteht aus den Buchstaben von P. Die 4er-Permutation erhält man durch eine Modulo 4 Rechnung bei der 16er-Permutation und einem erneuten streichen der doppelten Zeichen.
Im Beispiel wäre '*lt,Dr3s.@r*' mit einer 10er-Permutation vertauscht '*,@l3Drst.r*'.

Entschlüsselung
Soll ein codiertes Wort entschlüsselt werden, müssen alle Schritte rückwärts angewendet werden.

3 Softwaretechnischer Kontext

Auf Basis der in Kapitel 2 analysierten theoretischen Konzepte realisierten wir DontSpy auf informationstechnischer Seite als Softwareanwendung. Die nachfolgenden Kapiteln erläutern die Konzepte aus informationstechnischer Sicht im Detail.

3.1 Technischer Kontext

Als Laufzeitumgebung der als Client fungierenden Endgeräte wird die *Xamarin Platform* [9] und der *.NET-Standard* [10] in der Entwicklungsumgebung *Visual Studio* [11] verwendet, um die Plattformunabhängigkeit (Android, iOS, UWP) auf Basis eines einheitlichen Quellcodes in der Programmiersprache *C#* [12] zu gewährleisten. *Xamarin.Forms* [13] ermöglicht zusätzlich die Definition zu nativ transpilierbaren grafischen Benutzeroberflächen.
Für die Client-Server-Kommunikation wird das *RESTful-Paradigma* [14] verwendet. In *JSON* [15] werden die hierbei kommunizierten Daten (de-)serialisiert. Die serverseitige Programmierung wird mit der Programmierspache *PHP* [16] in der Entwicklungsumgebung *Atom* [17] realisiert. Der Fokus des in PHP erzeugten Quellcodes liegt zwischen Client und *MySQL-Datenbank* [18], wobei hierzu die Schnittstelle *PHP Data Objects* [19] genutzt wird. Der *Apache-Webserver* [20] interpretiert serverseitig den PHP-Code. Der *Cross-Secure-Storage* [21] erzeugt und verwaltet einen lokalen, verschlüsselten Schlüssel-Wert-Speicher zur Persistierung von beispielsweise unseren Chat-eigenen Schlüsseln. Weitere Daten werden in einer relationalen *SQLite-Datenbank* [22] lokal gespeichert. Weitere Frameworks [23, 24] ermöglichen u.a. die Codierung der QR-Codes und das anschließende Auslesen durch einen plattformübergreifenden Kamerazugriff für die Schlüsselübertragung.

3.2 Anwendungsfalldiagramm

Die nachfolgenden Abschnitte beschreiben die in der Abbildung 4 angegebenen Anwendungsfälle:

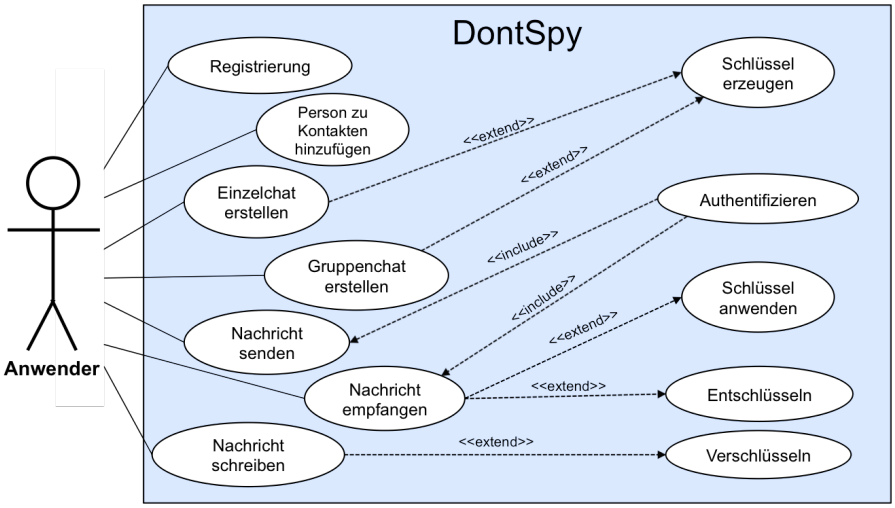

Abbildung 4: Anwendungsfalldiagramm

Registrierung: Dieser Anwendungsfall wird nur beim erstmaligen Start und vor der erfolgreichen Registrierung angewendet. Der Anwender wählt einen Benutzernamen, der ein Pseudonym oder sein echter Name sein kann. Über diesen kann man ihn innerhalb der Anwendung über eine Suche finden und zu den eigenen Kontakten hinzufügen.

Authentifizieren: Bei jeder Kommunikation mit dem Server erfolgt eine HTTP-Basic Authentifizierung des Anwenders durch Überprüfung der Identität, sodass er nur Nachrichten für diesen Anwender empfangen kann und Nachrichten im Namen dieses Anwenders gesendet werden können.

Person zu Kontakten hinzufügen: Die Person kann über eine Suchleiste anhand des Benutzernamens aufgefunden und den eigenen Kontakten hinzugefügt werden. Personen, die den Kontakten hinzugefügt wurden, kann man eine Nachricht schreiben oder zu einem Gruppenchat hinzufügen.

Einzelchat erstellen: Auf der Chatübersichtsseite befinden sich unter anderem Einzelchats, welche man erstellen kann, indem man eine Person aus den Kontakten auswählt. Hierbei wird ein Schlüssel generiert (siehe Schlüssel erzeugen). Im Einzelchat kann der Anwender dann eine Nachricht schreiben.

Gruppenchat erstellen: Auf der Chatübersichtsseite befinden sich neben Einzelchats auch Gruppenchats. Nach Anwählen dieses Buttons gelangt der Anwender zu der Übersicht seiner Kontakte und kann wählen, welche Kontakte er dem Gruppenchat hinzufügen will. Hierbei wird ein Schlüssel generiert (siehe Schlüssel erzeugen). Im Gruppenchat kann der Anwender dann eine Nachricht schreiben.

Schlüssel erzeugen: Bei der Erzeugung eines Einzel- oder Gruppenchats wird jedes Mal ein Schlüssel generiert. Diesen Schlüssel muss der Anwender dann der Person oder den Personen im jeweiligen Chat zukommen lassen, die diese Nachricht entschlüsseln können sollen (siehe auch Schlüssel anwenden).

Schlüssel anwenden: Falls der Anwender noch keinen Chat mit dem Sender der Nachricht hat, hat der Anwender auch noch keinen Schlüssel, um die Nachricht zu entschlüsseln. Der Anwender muss zuerst den Schlüssel im verschlüsselten Speicher seines Geräts speichern. Dafür muss der Schlüssel übertragen werden (siehe Kapitel 3.5.2). Wenn der Schlüssel vorhanden ist, kann dieser für die Ver- und Entschlüsselung angewendet werden.

Nachrichten empfangen: Sobald der Anwender die Anwendung geöffnet hat, wird automatisch und kontinuierlich geprüft, ob Nachrichten für ihn auf dem Server bereitstehen. Falls dies der Fall ist, werden diese auf das Endgerät geladen und entschlüsselt. Dabei werden sie, falls der Chat ein Einzelchat ist, sofort auf dem Server gelöscht und bei Gruppenchats, sobald der letzte Anwender der Gruppe die Nachricht empfangen hat, sodass die Nachricht nur sehr kurz öffentlich ist.

Nachricht schreiben: Sowohl in Gruppen- als auch in Einzelchats kann der Anwender Nachrichten schreiben. Dazu befindet sich im Chatfenster eine Eingabeleiste, wobei die Systemtastatur zum Eintippen der Nachricht erscheint. Nach Betätigen des Senden-Knopfs wird die Nachricht verschlüsselt.

Nachricht senden: Die verschlüsselte Nachricht wird mit einer einmaligen Identifikationsnummer (ID) versehen, damit jede Nachricht eindeutig identifizierbar wird und so nicht verwechselt werden kann, und einem Zeitstempel, damit sowohl der Empfänger als auch der Sender weiß, wann die Nachricht gesendet wurde. Dann wird sie zum Server gesendet. Außerdem wird die Nachricht dem Sender im eigenen Chatfenster angezeigt.

Verschlüsseln: Die Nachricht wird verschlüsselt (siehe Kapitel 2.2).

Entschlüsseln: Die Nachricht wird entschlüsselt (siehe Kapitel 2.2).

3.3 Systemaufbau

Unsere Anwendung ist als Client-Server-Architektur strukturiert, die über RESTful-Endpunkte miteinander kommuniziert.

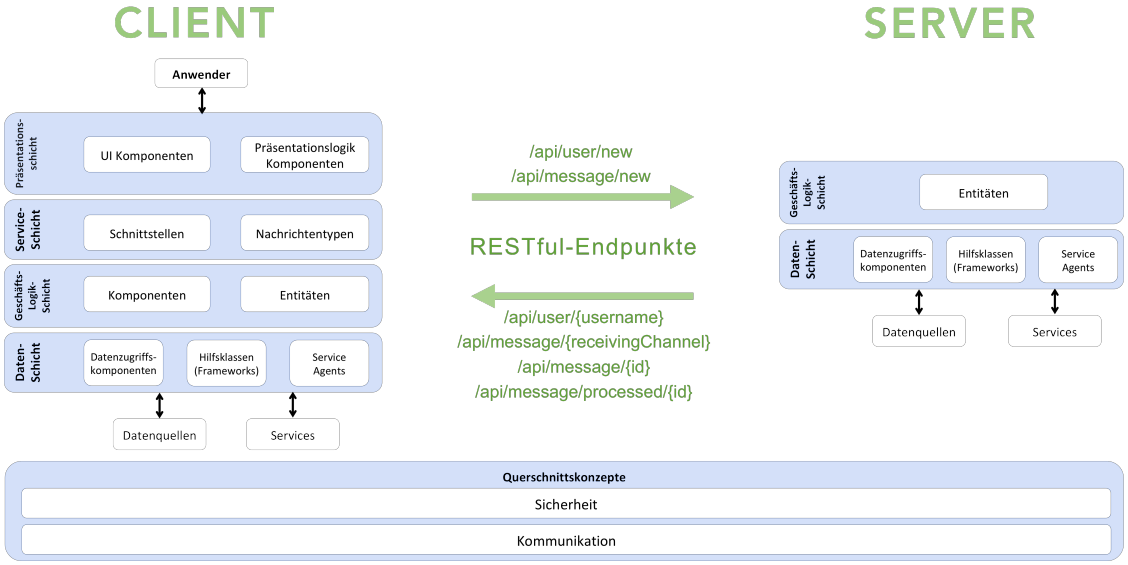

Abbildung 5: Struktur und deren Kommunikation in unserer Software [25]

Der Schwerpunkt liegt bei uns auf dem Client. Der Server ist funktional reduziert, da wir ihn langfristig eliminieren möchten. Der Grund hierfür ist die größere Angriffsmöglichkeit auf zentralisierte Komponenten.

Anwender: Der Anwender kann nach seiner Registrierung einzelne Chats erstellen und Nachrichten versenden und empfangen.

Präsentationsschicht: Die Präsentationsschicht wird nach dem MVVM-Prinzip aufgebaut, die die Präsentationslogik von der Geschäftslogik trennt: Das, was man als Nutzer sieht, ist die View und beinhaltet die Definition der UI-Komponenten; die Präsentationslogik wird in den ViewModels verarbeitet. Im ViewModel werden ebenso Daten aufbereitet (z.B. wird der UNIX-Zeitstempel zum Datum formatiert), Daten werden aus den Modellen angereichert oder Daten werden bewusst nicht angezeigt (Content Filter). Ersteres wird in den XAML-Dateien mittels Xamarin.Forms der Views (Benutzeroberflächen-Klassen) realisiert; letzteres in klassischen C#-Klassen.

Serviceschicht: Wir nutzen verschiedene Nachrichtenobjekte, um bestimmte Daten der Präsentationsschicht zu den unteren Schichten (unterhalb der Serviceschicht) zu transferieren. Die Struktur dieser Nachrichtenobjekte ist dabei in der Serviceschicht definiert und bezeichnet man als Nachrichtentypen. Ebenfalls nutzen wir Schnittstellen (Interfaces), um zum einen unseren Klassen, die ein solches Interface implementieren, eine klare Struktur (Vertrag) zu geben. Der andere Grund zur Nutzung von Interfaces in DontSpy ist die schnelle Austauschbarkeit der Klassen, die ein solches Interface implementieren. Somit ist unsere Anwendung für Weiterentwicklungen und Wartungen vorbereitet.

Geschäftslogik-Schicht: Unsere Geschäftslogik besteht im Wesentlichen aus der Kryptographie-Komponente unserer Verschlüsselungsmethode (siehe Kapitel 2.2). Ein Teil derer besteht aus der Verschlüsselungslogik, mit der die Nachrichten auf dem Endgerät vor dem Senden zum Server verschlüsselt werden. Analog werden auf dem Gerät des Empfängers die Nachrichten mit der Entschlüsselungslogik entschlüsselt. Die Datenbasis, sprich welche Zeichen verwendet und welchen Intervallen sie zuzuweisen sind, und Helfer der Kryptographie-Logik sind in eigenen Klassen definiert. Für die Generierung der Schlüssel existiert ebenso eine Klasse. Die verarbeiteten Daten, wie beispielsweise die ID oder die Nachricht, werden als Entitäten gespeichert, sodass jene Daten zwischengespeichert sind.

Datenschicht: Als Anwender (Client) werden sowohl die gespeicherten Nachrichten, die bereits auf diesem Endgerät verschlüsselt wurden, als auch ein Verweis auf den verwendeten Schlüssel der Chats in einer lokalen SQLite-Datenbank gespeichert. Da diese Informationen und der Schlüssel besonders geschützt werden müssen, werden sie im sogenannten Cross-Secure-Storage gespeichert. Ferner werden Nachrichten, die noch nicht vom Empfänger abgerufen wurden (da er noch nicht online war), in einer MySQL-Datenbank auf dem Server zwischengespeichert. Wenn die Nachrichten vom Empfänger abgerufen wurden, nachdem sich die Anwendung mit dem Server verbunden hat, werden die Nachrichten vom Server gelöscht.

Querschnittskonzepte: Die Sicherheit ist der Kernaspekt unserer Anwendung (siehe Kapitel 3.7.2). Die asynchrone Kommunikation wird mittels des RESTful-Paradigmas gewährleistet.

Services: Für die Kommunikation zwischen dem Client und dem Server wird das RESTful-Paradigma genutzt. Die Daten, die zwischen Server und Client gesendet werden, werden in sogenannten JSON-Dokumenten serialisiert bzw. deserialisiert.

Datenquellen: Wir verwenden SQLite, den Cross-Secure-Storage und MySQL, wobei MySQL über unsere RESTful-API (Services) angesteuert wird.

3.4 Softwareaufbau

Im Folgenden soll ein exemplarischer Ausschnitt aus dem Klassendiagramm unserer Software einen Eindruck vermitteln. Danach folgt eine vollständige Auflistung aller Klassen mit Kurzbeschreibung.

Ausschnitt des Sendens im Klassendiagramm

Wenn der Sender eine Nachricht verschickt, so tut er dies auf der ChannelPage, wo er seine Nachricht an den Empfänger eingeben kann. Dies ist die View, die mit dem ChannelPageViewModel auf Basis des MVVM-Architekturstils zusammenhängt.

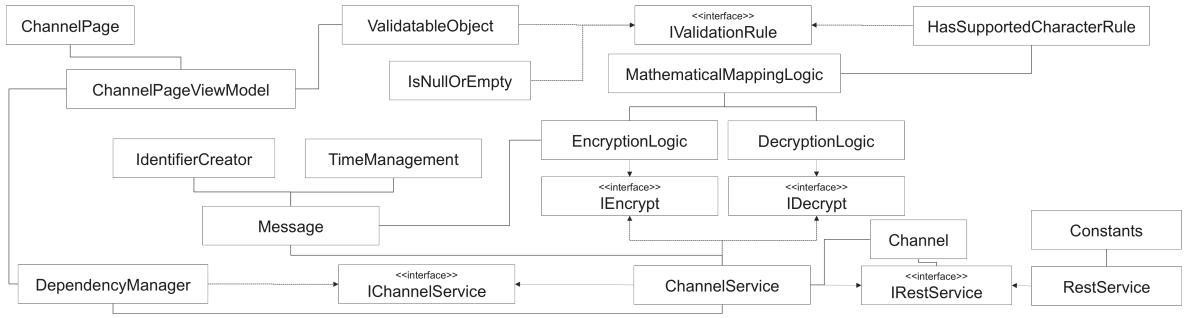

Abbildung 6: Ausschnitt des Sendens im Klassendiagramm

Über dieses ViewModel wird die eingegebene Nachricht über das ValidatableObject, das entsprechende Validierungsregeln über ein Interface koppelt, validiert. In diesem Szenario ist die benötigte Validierungsregel HasSupportedCharacterRule, die die Nachricht auf ungültige Zeichen (den Zeichen, die nicht in der MathematicalMappingLogic enthalten sind) prüft. Falls die Nachricht ungültige Zeichen besitzt, wird auf der ChannelPage eine Fehlermeldung unterhalb des Eingabefelds angezeigt und die Nachricht kann dementsprechend nicht verschickt werden. Die validierte Nachricht wird dem ChannelService übergeben und in der EncryptionLogic mit unserem Verschlüsselungsverfahren verschlüsselt, wobei in der MathematicalMappingLogic die mathematischen Grundlagen hierfür implementiert sind. Die verschlüsselte Nachricht wird der Message übergeben bzw. in ein Message-Objekt gepackt. Das Message-Objekt wird mit folgenden Metadaten versehen: Der Nachricht wird zum einen eine eindeutige ID zugewiesen, die mit dem GUID-Verfahren generiert wird und somit weltweit einzigartig ist, um die eindeutige Zuweisung zu gewährleisten. Zum anderen wird die Nachricht mit dem sogenannten UNIX-Timestamp versehen, der die Sekundenanzahl vom 01.01.1970 bis zum Zeitpunkt des Sendens zählt. Dieses Message-Objekt kann dem RestService übergeben werden, die die Nachricht mit ihren Metadaten zum Server mittels der in Constants definierter Endpunkte schickt.

Klassenübersicht

	Klassenname	Funktion
DontSpy		
	App	Einstiegspunkt, verwaltet Abhängigkeiten, allokiert und deallokiert Ressourcen
	Constants	Definiert Konstanten
	DependencyManager	Verwaltet Abhängigkeiten
	LocalDatabaseOptions	Verwaltet Lokale Datenbank
	DebuggingOptions	Bietet Funktionalitäten für das Debugging
Model	Channel	Enthält Daten der Channels
	ChannelUser	Beziehungstabelle für relationales Model, ist Verbindung zwischen Channel und User
	DecryptedMessage	Enthält Daten von entschlüsselten Nachrichten
	Interval	Enthält Daten der Intervalle
	Message	Enthält Daten von verschlüsselten Nachrichten
	SelectableData	Wrapperklasse für Präsentationslogik, um spezifische Daten zu selektieren
	User	Enthält Daten der User
Interfaces	IChannelService	Schnittstelle für ChannelService
	IDecrypt	Schnittstelle für DecryptionLogic
	IEncrypt	Schnittstelle für EncryptionLogic
	IEntity	Schnittstelle für die Entitäten unser lokalen Datenbank
	IKeyHandling	Schnittstelle für KeyHandling
	ILocalize	Schnittstelle um die Sprache zu setzen
	IPullService	Schnittstelle für PullService
	IRestService	Schnittstelle für RestService
	IStorage	Schnittstelle für Verbindung mit lokaler Datenbank
	IUserService	Schnittstelle für UserService
	IValidationRule	Schnittstelle für die Validierungsregeln

	Klassenname	Funktion
Business Logic	DecryptionLogic	Entschlüsselt
	EncryptionLogic	Verschlüsselt
	KeyHandling	Generiert Schlüssel
	MathematicalMappingLogic	Verwaltet Intervalltabelle und Tabelle in denen die Zeichen festgehalten sind
	QrCodeLogic	Codiert und decodiert QR-Code verwaltet Zugriff auf Kamera und Dateisystem
Presentation		
.Behaviors	BehaviorBase	Basisklasse um spezifische Verhalten in der Präsentationslogik zu simulieren
	EventToCommandBehavior	Mapping von UI-Events auf Kommandos
	LineColorBehavior	Simuliert Validierungsverhalten anhand von Farbauswahl der Umrandung von UI-Elementen
.Converter	FirstValidationErrorConverter	Prüft das Fehlverhalten bei der Validierung
	ItemTappedEventArgsToItemTappedConverter	Mappt ein TappedEvent zu einem TappedConverter
	MessageSnippetConverter	Erzeugt einen Nachrichtenausschnitt der letzten Nachricht eines Chats und zeigt diesen auf der Chatübersichtsseite
	MessagesTimestampsConverter	Wandelt Unix Zeitstempel in ortsspezifische Uhrzeit und Datum um
	TimestampConverter	Justiert den Zeitstempel der Chats auf der Chatübersichtseite
.Effects	EntryLineColorEffects	Basisklasse um für die Validierung farbige Linien anzuzeigen

	Klassenname	Funktion
.Validation.Rules	EqualsRule	Überprüft ob zwei Objekte gleich sind
	HasSupportedCharacterRule	Überprüft ob das eingegebene Zeichen von unser App verschlüsselt werden kann
	IsNullOrEmptyRule	Überprüft ob ein Wert 0 ist
	StringLengthRule	Schaut ob es für ein String ein Minimum oder Maximum gibt und überprüft ob er diese erfüllt
	WithinNumberRangeRule	Schaut ob es für einen Integer ein Minimum oder Maximum gibt und überprüft ob dieser diese erfüllt
.Validation	ValidatableObject	Validiert definierte Daten
	ValidationBase	Abstrakte Basisklasse zum Hinzufügen von Validierungen und deren Ausführung
View	AnchorPage	Startseite des UI, enthält die ChannelsPage und die ContactsPage zwischen denen man hin und her wischen kann
	ChannelPage	Enthält Design und Daten der Seite eines Chats
	ChannelsPage	Enthält Design und Daten der Chatübersichtsseite
	ContactsPage	Enthält Design und Daten der Seite auf der man eine Überblick über seine Kontakte hat und Kontakte hinzufügen kann
	RegistrationPage	Enthält Design und Daten der Registrierungsseite
	TranslateExtension	Hilfsklasse um die richtige Sprache des Benutzers zu erkennen und ihm die von uns bereitgestellte korrekte Sprachvariante anzuzeigen
View Model	ChannelPageViewModel	Enthält Funktion einer Chatseite
	ChannelsPageViewModel	Enthält Funktion der Chatübersichtseite
	ContactsPageViewModel	Enthält Funktion der Seite mit den Kontakten
	RegistrationPageViewModel	Enthält Funktion der Registrierungsseite
Service	ChannelService	Enthält Funktionen um einen Channel zu erstellen oder von der Datenbank zu laden, sowie Nachrichten zu senden
	PullService	Holt Nachrichten vom Server

	Klassenname	Funktion
	RestService	Enthält RESTful-Endpunkte
	StorageService	Verwaltet SQLite-Datenbank und Cross-Secure-Storage
	UserService	Enthält Funktionen, um einen eigenen Anwender zu erstellen und Kontakte hinzuzufügen
Utils	IdentifierCreator	Erzeugt eindeutige Identifikationsnummer für Nachrichten, bevor sie gesendet werden
	ImageResourceExtension	Hilfsklasse, um Bilder in UWP Releases korrekt anzuzeigen
	PlatformCulture	Definiert Sprache sowie Zeit des Anwenders abhängig vom Ort
	TimeManagement	Erstellt aktuellen Zeitstempel
Translations	AppResources.de	Deutsche Sprachdatei
	AppResources	Englische Sprachdatei
DontSpy.UWP		
	App	Einstiegsklasse für UWP-Anwendungen
	LocalDatabaseUwp	Richtet Lokale Datenbank auf einem UWP Gerät ein
	MainPage	UI-Wrapper
.Effects	EntryLineColorEffects	Für UWP angepasste Version von der allgemeingültigen EntryLineColorEffects Klasse
DontSpy.Android		
	MainActivity	Einstiegsklasse für Android-Anwendungen und UI-Wrapper
	StorageDroid	Verwaltet SQLite-Datenbank und Cross-Secure-Storage
	LocalizeDroid	Definiert Sprache sowie Zeit des Anwenders abhängig vom Ort
DontSpy.iOS		
	Main	Einstiegsklasse für iOS-Anwendungen
	AppDelegate	UI-Wrapper
	StorageIos	Verwaltet SQLite-Datenbank und Cross-Secure-Storage
	LocalizeIos	Definiert Sprache sowie Zeit des Anwenders abhängig vom Ort

	Klassenname	Funktion
Server		
api	Index	Einstiegsklasse
classes	Mapper	Basisklasse, um Datenbankverbindung zu erhalten
	MessageEntity	Enthält Daten von Nachrichten
	MessageMapper	Enthält Funktionen, um Nachrichten vom Server abzurufen in der Datenbank zu sichern sowie zu löschen
	UserEntity	Enthält Daten von Usern
	UserMapper	Enthält Funktionen, um einen User aus der Datenbank zu erhalten und zu speichern

	Klassenname	Funktion
Server		
src	Dependencies	Abhängigkeit zur MySQL-Datenbank
	Middleware	Authentifizierung + Einstellungen von zugelassenen HTTP-Verben
	Routes	Verwaltet alle Endpunkte und was passiert, wenn diese aufgerufen werden
	Settings	Enthält die Verbindungsdaten der Datenbank und allgemeine Einstellungen

Tabelle 2: Klassenauflistung von DontSpy

3.5 Laufzeitsichten

Einige primäre Laufzeitsichten unserer Anwendung werden im Folgenden dargestellt.

3.5.1 Channelerzeugung

Die Einzel- und Gruppenchats können im technischen Kontext mit dem Begriff des Channels zusammengefasst werden. Einen Channel kann man auf zweierlei Wegen beitreten: Entweder erstellt der Nutzer selbst den Channel oder er wird einem Channel durch einen anderen Nutzer hinzugefügt.

Channelerzeugung durch Nutzeraktion Hierbei wird die Channelerzeugung für Gruppenchats beschrieben, selbiges funktioniert ebenso bei Einzelchats mit dem Unterschied, dass nicht mehrere Personen hinzugefügt werden müssen.

1. **Button**: Zunächst muss der Button „Gruppe erstellen“ betätigt werden.
2. **Channelerzeugung**: Es wird ein neuer, virtueller Channel (abstraktes Objekt) erzeugt.
3. **Channel-ID-Zuweisung**: Diesem erzeugten Channel wird eine eindeutige Channel-ID zugewiesen, sodass keine Dopplungen auftreten und keine fremden Channels angezeigt werden können.

4. **Teilnehmer hinzufügen**: Für den Nutzer wird nun die Kontaktseite, auf der alle gespeicherten Kontakte aufgelistet werden, sichtbar. Auf dieser werden die Personen, die der Gruppe hinzugefügt werden sollen, markiert und anschließend bestätigt. Bei diesem Vorgang werden die Teilnehmer (die Personen, die vom Nutzer markiert wurden) dem Channel hinzugefügt.
5. **Schlüsselgenerierung**: Nun wird ein neuer Schlüssel erzeugt (siehe Kapitel 2.2).
6. **Speicherung des Schlüssels im Cross-Secure-Storage**: Der Schlüssel wird im Cross-Secure-Storage gespeichert. Dieser ist ein Schlüssel/Wert Speicher und ist mit plattformeigenen Verschlüsselungsmechanismen geschützt. iOS benutzt KeyChain, Android einen passwortgeschützten Schlüsselspeicher und UWP speichert den Schlüssel mit Hilfe von Datenschutzmechanismen ab. Dabei ist unser Schlüssel mittels eines Identifikators eindeutig zuweisbar[21].
7. **Anzeigen der Channel-Seite**: Der Channel wird auf der Channelübersichtsseite angezeigt. Zu jedem Channel ist zusätzlich zum Channel-Namen (bei Einzelchannels ist dies der Benutzername des Channel-Partners, in Gruppenchats die Teilnehmer) ein Teil der Nachricht und der Zeitstempel als Vorschau einzusehen.
8. **Lokale Datenbank-Speicherung**: Der Channel wird mit seinen Daten in der lokalen Datenbank gespeichert. Dabei ist jedoch wichtig, dass der Schlüssel nicht in der MySQL-Datenbank gespeichert wird, sondern ausschließlich lokal im Cross-Secure-Storage auf dem Endgerät. Ein Verweis im Channel gibt an, welcher Schlüssel für diesen Channel genutzt wird, sodass man bei Bedarf (wenn beispielsweise die Anwendung neu gestartet wird) auf den Verweis zurückgreifen und den korrekten Schlüssel verwenden kann.
9. **On-boarding-Nachricht wird versendet**: Den anderen Teilnehmern der Gruppe wird eine Textnachricht geschickt, nachdem auch beim Empfänger dieser Channel erstellt wurde und auch in der Channelübersicht angezeigt wird. Diese Nachricht signalisiert dem Empfänger, dass man jener neuen Gruppe angehört, und signalisiert dem System, dass dieser neue Channel auf seinem lokalen Gerät erstellt werden soll. Somit wurde der Empfänger in den Channel integriert.

Channelerzeugung, indem man selbst durch einen anderen Benutzer hinzugefügt wird

1. **Nachrichten vom Server werden geladen**: Alle Nachrichten mit der jeweiligen Empfänger-ID werden vom Server zum Endgerät gepullt, d.h. alle Nachrichten, die dem jeweiligen Nutzer zugeordnet werden, werden auf das lokale Gerät geladen. Der Pull-Vorgang wird alle fünf Sekunden wiederholt, bedeutet, dass alle fünf Sekunden geprüft wird, ob eine (verschlüsselte) Nachricht für jenen Nutzer vorhanden ist. Falls vorhanden, werden sie auf das lokale Gerät übertragen. Die Daten, die übertragen werden, werden im JSON-Dokument serialisiert.
2. **Schlüsseleingabe**: Der Schlüssel muss mittels eines QR-Codes eingelesen werden.
3. **Anlegen des Channels**: Ein neuer Channel wird angelegt.
4. siehe Schritte 6.-8. bei Channelerzeugung durch Nutzeraktion (siehe oben)

3.5.2 Schlüsselübergabe

Einer der Angriffspunkte der symmetrischen Verschlüsselung, auf welchem unsere Anwendung basiert, ist die sichere Schlüsselübergabe. Da bei unserer Lösung die Sicherheit höchste Priorität besitzt, wurde hierfür ein eigenes Konzept entwickelt, bei dem der Schlüssel in QR-Codes codiert wird. Aus diesem Grund ist unsere Anwendung für definierte Gruppen konzipiert, bei dem möglichst der Schlüssel während einem Treffen in persona ausgetauscht wird.

1. **Schlüsselgenerierung**: Wenn der Chat erstellt wird, nachdem der/die Teilnehmer ausgewählt wurde/n, wird der chateigene Schlüssel generiert. Der Schlüssel umfasst zunächst aufgrund der Auflistung der Zahlen von 1 bis 8100 (bei einem Textsatz von 90 Zeichen) ungefähr 40000 Zeichen.

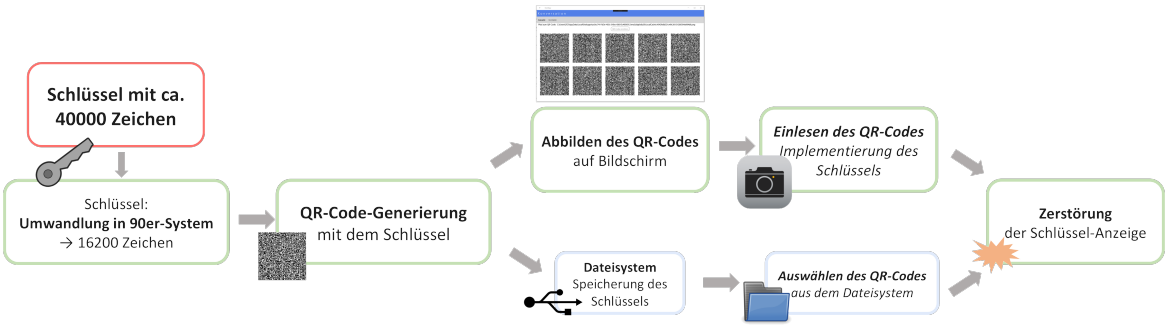

Abbildung 7: Schlüsselübertragung

2. **Reduktion des Schlüssels**: Da diese Zeichenanzahl jedoch nicht zuverlässig von dem Framework [24], das für die QR-Codes zuständig ist, verarbeitet werden kann, wird der Schlüssel deutlich reduziert. Hierzu wird ein eigenes 90er-System, das auf unserem Textsatz von 90 Zeichen basiert, auf den Schlüssel angewendet. Nach der Reduktion umfasst der Schlüssel nun 16200 Zeichen, da im 90er-System alle 8100 Zahlen zweistellig sind.
3. **Zerteilung des Schlüssels:** Dieser Schlüssel wird nun in mehrere Teile aufgeteilt, da in einen QR-Code, nach unseren ausführlichen Tests, lediglich 2000 Zeichen zuverlässig codiert werden können. Aus den einzelnen Teilen werden die jeweiligen QR-Codes generiert.
4. **Anzeige**: Die einzelnen QR-Codes werden auf einer Schlüssel-Seite angezeigt sowie ein Pfad, der angibt, wo der Schlüssel temporär gespeichert ist.
5. **Übertragung**: Für die Übertragung ergeben sich zwei Möglichkeiten: Zum einen kann der QR-Code von dem Ersteller-Gerät, wo er auf jener Seite angezeigt wird, von den anderen Teilnehmern mittels ihrer Gerätekamera eingelesen werden. Zum anderen kann der Schlüssel über das Dateisystem übertragen werden, wobei beispielsweise die generierte Bilddatei über ein sicheres Firmennetz oder per Post (wie auch bei den TAN-Briefen im Finanzwesen) übertragen wird.
6. **Zerstören**: Nachdem der Schlüssel erfolgreich implementiert werden konnte, können die QR-Codes, die den Schlüssel enthalten, gelöscht werden. Das Zerstören ist aus Gründen der Sicherheit sinnvoll und möglich, da er sich im sicheren Speicher des Endgeräts befindet (Cross-Secure-Storage).

3.5.3 Senden einer Nachricht

1. **Button**: Der Button „Senden“, der sich beim Eingabefeld befindet, wird betätigt.
2. **Validierung der Nachricht**: Die Nachricht wird validiert, d.h. es wird zunächst geprüft, ob eine Nachricht in das Eingabefeld eingegeben wurde und ob nur erlaubte Zeichen verwendet wurden. Falls beispielsweise Sonderzeichen eingegeben wurden, die nicht in unserem Zeichensatz enthalten sind, wird eine Fehlermeldung ausgegeben.
3. **Verschlüsseln der Nachricht**: Die Nachricht wird verschlüsselt. Nachdem der Schlüssel beim Channel-Erzeuger für den jeweiligen Channel generiert wurde, wird er im Cross-Secure-Storage gespeichert. Falls man jedoch nicht der Channel-Erzeuger ist, muss der Schlüssel für den jeweiligen Channel zuvor über einen QR-Code ausgetauscht werden.
4. **Speichern der Nachricht**: Die verschlüsselte Nachricht und die dazugehörigen Informationen in der Nachrichten-Entität (ID, messageHeader, Empfangschannel, Zeitstempel, Nachricht, processingCounter) werden in der SQLite-Datenbank auf dem Gerät des Senders gespeichert.
5. **Generierung der JSON-Datei**: Aus der Nachricht und allen Daten der Nachrichten-Entität wird ein JSON-Dokument generiert, in der die Informationen zusammengefasst werden.
6. **Senden der JSON-Datei**: Diese JSON-Datei wird über den RESTful-Endpunkt „/message/new“ zum Server gesendet.

7. **Ankommen am Server**: Der Server wartet am RESTful-Endpunkt „/message/new“ darauf, dass Nachrichten ankommen.
8. **JSON deserialisieren**: Sobald das JSON-Dokument beim Server ankommt, wird es deserialisiert, sodass die darin gespeicherten Daten durch unsere Algorithmen verarbeitet werden.
9. **Speichern in Datenbank**: Die verschlüsselte Nachricht wird zusammen mit ihrer ID, dem Zeitstempel, dem Absender und der Channel-ID in einer Message-Tabelle, der Message-Relation in der MySQL-Datenbank, temporär gespeichert.

3.5.4 Empfangen einer Nachricht

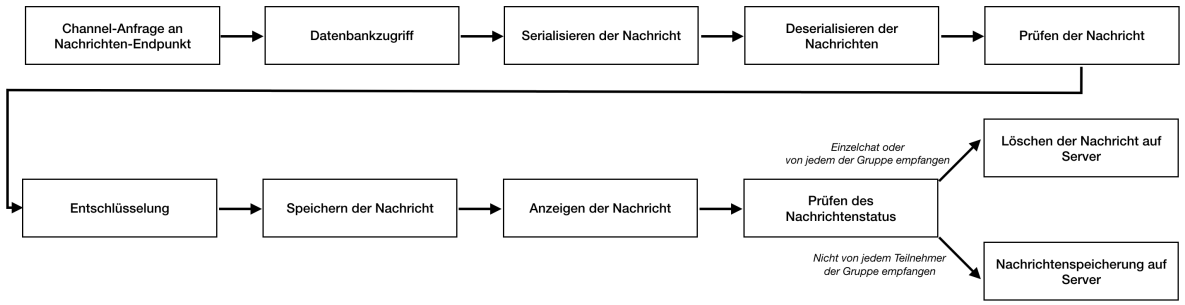

Abbildung 8: Laufzeitsicht des Empfangens einer Nachricht

1. **Channel-Anfrage an den Server**: Der zu Empfangende einer Nachricht sendet kontinuierlich für jeden seiner lokal existierenden Channels und für die eigene Benutzer-ID eine Anfrage auf neue Nachrichten zum Server (Pull-Prinzip).
2. **Datenbankzugriff**: Bei der MySQL-Datenbank werden über den Endpunkt „/message/{receivingChannel}“ alle Nachrichten ausgewählt, die den vorherigen Bedingungen entsprechen.
3. **Serialisieren der Nachricht**: Alle Nachrichten, die für einen Benutzer bestimmt sind, werden in ein JSON-Dokument aggregiert, bedeutet, dass alle Daten jener Nachrichten im JSON-Dokument serialisiert werden und anschließend an das Endgerät verschickt werden.
4. **Deserialisieren der Nachricht**: Auf dem Endgerät wird das JSON-Dokument deserialisiert, d.h., dass die einzelnen Eigenschaften des JSON-Dokuments, die zuvor aggregiert wurden, zu einzelnen Daten, wie beispielsweise Zeitstempel oder Absender extrahiert werden.
5. **Prüfen der Nachricht**: Es wird geprüft, ob die Nachricht bereits auf dem Gerät des Empfängers vorhanden ist, was beispielsweise bei einem Serverfehler passieren könnte. Ein Fehler könnte sein, dass während des Empfangsprozesses abgebrochen werden musste, sodass zwar die Nachricht und die dazugehörigen Informationen auf dem Endgerät gespeichert werden konnten, ohne dass der Server darüber benachrichtigt werden konnte.
6. **Entschlüsselung**: Die Nachricht wird entschlüsselt (siehe Kapitel 2.2).
7. **Speichern der Nachricht**: Die verschlüsselte Nachricht wird mit ihren zusätzlichen Informationen der Entität auf dem Gerät des Empfängers, d.h. lokal, in der SQLite-Datenbank gespeichert.
8. **Anzeigen der Nachricht**: Die entschlüsselte Nachricht wird dem Empfänger im dazugehörigen Channel angezeigt. Somit kann dieser, wenn er auf den Channel klickt, die Nachricht in entschlüsselter Form sehen.
9. **Prüfen des Nachrichtenstatus**: Es wird geprüft, wie der Nachrichtenstatus der jeweiligen Nachricht ist: Ist die Nachricht für eine Einzelperson bestimmt, wird die Nachricht auf dem Server gelöscht. Ebenso wird die Nachricht gelöscht, wenn in einer Gruppe alle Personen die Nachricht erhalten haben (Fall 1, siehe oberer Pfeil der Abbildung 7). Wenn man jedoch eine Gruppenchat-Nachricht löschen würde, obwohl sie noch nicht alle Personen der Gruppe erreichte, wäre das

fatal, weil sonst nicht jeder die Informationen der Nachricht erhielte. Deshalb wird die Nachricht auf dem Server zwischengespeichert. Der Status der Nachricht wird überprüft, indem jede Gruppe einen sogenannten ProcessingCounter hat, der der Anzahl der Personen in der Gruppe entspricht. Beim Abrufen einer Nachricht von einer Gruppe wird, sofern man nicht der letzte ist, der diese Nachricht abrufen muss, ein Befehl zum Server gesendet, der den ProcessingCounter um 1 erhöht. Erst nachdem jeder die Nachricht erhalten hat und somit der ProcessingCounter bei der Teilnehmerzahl $tz - 1$ angelangt ist, wird sie dann vom Server gelöscht (Fall 2, siehe unterer Pfeil des Schemas).

3.6 Verteilungsdiagramm

Die Anwendung DontSpy nutzt einen klassischen Client-Server Architekturstil. Das Endgerät des Anwenders, worauf sich eine Client Anwendung in der Ausführungsumgebung .NET-Standard (siehe Kapitel 3.1) befindet, kommuniziert über das TCP/IP Protokoll mit dem Applikationsserver, auf welchem sich ein Apache-Webserver mit der Serverlogik und eine MySQL Datenbank mit den Relationen *user* und *messages* befinden. Die Relation *user* speichert alle registrierten Anwender der DontSpy Anwendung, während die *messages* Relation alle Nachrichten temporär speichert, bis diese vom Client abgerufen wurden. Dies bedeutet folglich, dass der Server lediglich die Funktion des temporären Zwischenspeichers der Nachrichten einnimmt.

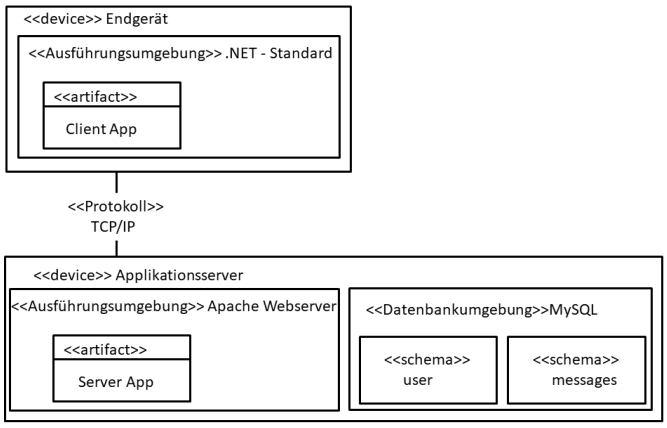

Abbildung 9: Verteilungsdiagramm

3.7 Querschnittskonzepte

Querschnittskonzepte sind jene Entwürfe, die nicht nur einen Bereich der Anwendungsgestaltung, sondern mehrere oder gar alle Ebenen ansprechen. Hierbei ist das Design ein übergreifendes Themenfeld, das beispielsweise die Programmierung als auch die Ideenfindung selber tangiert. Die Sicherheit, die ebenso mehrere Bereiche der Umsetzung anspricht, ist besonders ausschlaggebend für die Konzeption von DontSpy, da unsere Anwendung darauf den Hauptfokus legt.

3.7.1 UI/UX-Konzepte

Unter UI/UX (User Interface/User Experience) versteht man in der Mensch-Computer-Interaktion die Bereiche, die sowohl vom Benutzer als auch von der mathematisch-informationstechnischen Umsetzung der Anwendung abhängen. Zum einen muss das Design programmiert werden, zum anderen muss es vom Programm her so gestaltet sein, dass es den Benutzer anspricht. Somit müssen bei der Konzeption beide Akteure gleichermaßen betrachtet werden.
Bei DontSpy ist es aufgrund des Qualitätsanspruches von großer Bedeutung, dass unsere Anwendung benutzerfreundlich ist, ohne dabei den Aspekt der Sicherheit zu vernachlässigen. Realisiert wurde die Benutzerfreundlichkeit durch ein bewusst schlicht gehaltenes Design, das sich nur auf die nötigsten Elemente reduziert. Vom Client werden keinerlei persönliche Daten benötigt, da beispielsweise eine E-Mail den Anspruch der Sicherheit nicht gewährleisten kann (da diese beispielsweise gehackt werden kann). Stattdessen verwenden wir das Konzept des Benutzernamens. Er dient der System-internen Verwaltung der verschiedenen Benutzer als auch als Anzeigenamen für weitere Benutzer. Jedem Benutzer steht frei, wie anonym sein Benutzername ausfallen soll.
Wir orientierten uns an den Benutzererfahrungskonzepten bereits etablierter Messaging-Anwendungen. Somit ist für den Benutzer die intuitive, bereits gewohnte Bedienung möglich, ohne sich in ein neues

Konzept einarbeiten zu müssen. Auf der Hauptseite, die bei DontSpy die Chatübersichtsseite ist, findet die Verwaltung statt. Auf dieser kann der Client sowohl neue Einzel- als auch Gruppenchats erstellen, kann aber auch seine Chats erreichen, die er bereits erstellt hatte.
Eine gewohnte Bedienung sollte ebenso durch die Konsistenz innerhalb der Anwendung gewährleistet werden. Die Verwendung von der Primärfarbe CornflowerBlue und einem neutralen Grau ist konsistent; dieses monochromatische Farbschema findet auf jeder Seite Anwendung und jeder Button derselben Funktion wird konsequent gleich repräsentiert. Ebenso findet sich die Vertrautheit in der Typographie wieder: Genutzt wird Arial, sodass eine neutrale, serifenlose Schrift Klarheit in der Darstellung schafft.

3.7.2 Sicherheitskonzepte

Die Sicherheit ist bei DontSpy der Kernaspekt, der unter allen Umständen gewahrt werden soll.
Unter den drei Aspekten der Vertraulichkeit, Integrität und Verfügbarkeit, die auch als CIA-Triade (Confidentiality, Integrity, Availability) bekannt ist, und dem Zusatz der Privatsphäre, wird die Sicherheit beleuchtet [26].

Vertraulichkeit: Beim Aspekt der Vertraulichkeit wird die Verhinderung unautorisierter Informationsgewinnung thematisiert:

1. **Authentifizierung**: Bei der Authentifizierung wird gegenüber dem Server die Echtheit des Nutzers sichergestellt. Beispielsweise findet eine Autorisierung statt, wenn Nutzerdaten nach der Registrierung zum Server gesendet oder Nachrichten abgerufen werden. Desweiteren finden Authentifizierungen gegenüber Komponenten statt, die wir auf dem lokalen Endgerät des Anwenders ablegen, beispielsweise gegenüber der lokalen Datenbank (SQLite) oder dem Cross-Secure-Storage.
2. **Verschlüsselung**: Alle Nachrichten, die das Endgerät verlassen, sind ausnahmslos verschlüsselt; insbesondere bedeutet dies, dass auch die Nachrichten auf dem Server stets verschlüsselt sind. Ebenso bleibt unser symmetrisches Verschlüsselungsverfahren (siehe Kapitel 2.2) unzugänglich, was den Algorithmus, die Tabellen und Intervalle betrifft. Im Sinne der Sicherheit ist es umso wichtiger, dass diese von uns geschützt und unzugänglich bleiben. Die Nachrichten werden erst beim Empfänger mithilfe des jeweiligen Schlüssels entschlüsselt, wenn ihm die Nachricht angezeigt werden soll. Eine Klarnachricht wird dementsprechend nie unverschlüsselt gespeichert.
3. **Schlüsselübergabe**: Jeder Chat bekommt seinen eigenen Schlüssel, sodass die Sicherheit innerhalb von DontSpy sichergestellt ist. Das bedeutet eben auch, dass ein zufällig gehackter Schlüssel nur einen einzigen Chat offen legt, aber die komplette weitere Kommunikation bleibt sicher. Der Schlüssel muss sowohl dem Sender als auch dem Empfänger bekannt sein (siehe Kapitel 3.5.2).
4. **Cross-Secure-Storage**: Zur Speicherung hochsensibler Metadaten auf dem lokalen Endgerät verwendet DontSpy einen verschlüsselten Schlüssel-Wert-Speicher. Dies bedeutet, dass ein Wert bzw. die Daten verschlüsselt werden, sodass einem Wert ein bestimmter Schlüssel zugewiesen ist, ohne den die Daten lediglich in verschlüsselter Form da liegen.
5. **Weitgehende Eliminierung des Servers**: Gearbeitet wurde mit der Intention, dass die Anwendung möglichst wenig mit dem Server interagieren muss, um eine für Angriffe anfällige Zentralinstanz zu vermeiden. Deshalb werden die verschlüsselten Nachrichten nur so lange auf dem Server zwischengespeichert, bis sie vom Empfänger empfangen werden können; danach wird die Nachricht vom Server gelöscht.
6. **Nachrichtenaustausch**: Die verschlüsselte Nachricht, die vom Gerät zum Empfänger über den Server gesendet wird, wird nur so lange gespeichert, bis der Empfänger online ist und die Nachricht empfangen hat - anschließend wird diese gelöscht. Nachdem die noch verschlüsselte Nachricht beim Empfänger angekommen ist, wird diese mit dem jeweiligen Schlüssel des Chats entschlüsselt, aber auch nur dann, wenn diese angezeigt wird; ansonsten wird sie auch lokal verschlüsselt gespeichert.

Integrität: Die Integrität thematisiert die korrekte Funktionsweise des Systems als auch die Korrektheit der Daten, um unautorisierte Veränderungen im System zu vermeiden:

1. **Lokale Datenbank**: In der lokalen Datenbank werden keine sensiblen Daten, wie zum Beispiel die verschlüsselten Nachrichten, gespeichert. Sensible Daten, wie zum Beispiel die Schlüssel die gespeichert sind, sind stets durch den Cross-Secure-Storage hochverschlüsselt.
2. **MySQL-Datenbank**: Die MySQL-Datenbank auf dem Server, die die Nutzer auflistet und Nachrichten mit ihren Zusatzinformationen vorübergehend speichert, ist Passwort-geschützt.
3. **Cross-Secure-Storage**: Der Cross-Secure-Storage ist durch seine interne Verschlüsselung unzugänglich, sodass darin hochsensible Daten auch lokal gespeichert werden können.

Verfügbarkeit: Die Verfügbarkeit thematisiert die Verlässlichkeit des Systems:

1. **Verfügbarkeit**: Es wird seitens des Systems sichergestellt, dass DontSpy für den Benutzer verfügbar ist. Hierbei sind bestimmte Mechanismen eingebaut, die beispielsweise sicherstellen, dass Nachrichten auch ohne Internetverbindung nicht verlorengehen. Diese werden lokal zwischengespeichert und werden erst bei verfügbarer Internetverbindung an den Server geschickt.

Privatsphäre: Die Privatsphäre thematisiert den Datenschutz, der den Nutzer betrifft:

1. **Pseudonymität**: Jeder Nutzer besitzt einen Benutzernamen, der zum einen für alle Personen sichtbar ist und andererseits für die anwendungsinterne Verwaltung genutzt wird, um ihn beispielsweise in der Anwendung ausfindig zu machen. Hierbei kann von jedem selbst entschieden werden, wie hoch der Grad der Anonymität in seinem Benutzernamen sein soll.
2. **Wenige Befugnisse nötig**: DontSpy kommt mit sehr wenigen Systemberechtigungen aus, sodass keine Daten des Nutzers, wie beispielsweise die Kontakte, von Drittanbietern weitergenutzt werden können. Die Ausnahme bildet die Gerätekamera. Bei der Schlüsselübergabe kann optional die Kamera zum Einlesen der QR-Codes genutzt werden. Alternativ kann man den Schlüssel über das Dateisystem austauschen.

4 Ausblick

Der Server ist als Zentralinstanz aufgrund seiner dauernden Verfügbarkeit und eindeutigen IP Nummer angreifbar. Daher wollen wir langfristig diese Zentralinstanz vollends eliminieren. Realisiert werden soll es durch ein Peer-to-Peer-Netzwerk, in der jedem Teilnehmer (Peer) ein Dienst zuteil wird. Dabei interagiert jeder gleichberechtigt im Netzwerk der Teilnehmer, anstatt dass eine höhergestellte Zentralinstanz wie ein Server vorhanden ist [27]. Die Teilnehmer sind nicht permanent verfügbar und wechseln dadurch auch Ihre IP Nummern, was sie schwerer angreifbar macht.
Unser Konzept beruht auf der eigenen Schlüsselgenerierung pro Chat und der Schlüsselübergabe bei einem persönlichen Treffen. Falls dies nicht möglich ist, haben wir zusätzlich die Möglichkeit des Schlüsselaustausches über ein sicheres Firmennetzwerk implementiert. Eine interessante Erweiterung für die Zukunft hat sich in der Diskussion mit den Juroren beim Landeswettbewerb ergeben. Der Gedanke ist, dass wir eine weitere Möglichkeit implementieren, bei dem sich der Anwender dafür entscheiden kann, den Schlüssel über ein anerkanntes asymmetrisches Verfahren auszutauschen. Der Anwender entscheidet dann selber ob er dieses Sicherheitsrisiko eingehen möchte. Da ja aber pro Chat ein eigener Schlüssel generiert wird, ist das Gesamtrisiko überschaubar.
Für den internationalen Einsatz muss die Anwendung auf weitere Sprachen erweitert werden, wobei die Häufigkeiten auf die jeweilige Sprache adaptiert werden müssen. Wir haben bereits begonnen die Anwendung zusätzlich in Englisch zu konzipieren, so dass bei einer englischsprachigen Umgebung automatisch alle Bildschirmbilder englischsprachig sind. Um auch Häufigkeiten der englischen Sprache zu erhalten, sind wir auf der Suche nach einem Institut ähnlich dem IDS in Mannheim.

5 Danksagung

Wir möchten uns an dieser Stelle ganz herzlich bei unseren Betreuern Tobias Straub (Hochschule Furtwangen University, Universität Stuttgart) für die intensive Betreuung auf informationstechnischer Seite, bei David Ploß (Universität Konstanz) für die mathematische Unterstützung und bei Helmut Ruf für die Koordination und fachliche Beratung bedanken.
Zusätzlich haben uns noch die Juroren des Regional- und Landeswettbewerbs, und hier speziell Herr Prof. Reich von der Hochschule Furtwangen und Wolfgang Killinger von der Robert Bosch GmbH, mit interessanten Anregungen versorgt, die wir umgehend einfließen ließen.
Ebenso gilt unser Dank einer Arbeitsgruppe des Instituts für Deutsche Sprache in Mannheim unter der Leitung von Cyril Belica und unserem Kontakt Rainer Perkuhn, die auf unsere Anfrage hin eine ausführliche Analyse der Zeichenhäufigkeiten in der deutschen Sprache durchführte und somit eine zuverlässige Datenbasis geschaffen werden konnte.
Außerdem möchten wir uns beim Schülerforschungszentrum Südwürttemberg (Standort Tuttlingen) für die Vermittlung von Betreuern, Forschungsräumen und Mitteln, um Geräte zu kaufen, bedanken.

6 Quellenverzeichnis

Literatur

[1] Funktonsweise der Vigenereverschlüsselung, http://www.kryptowissen.de/vigenere-verschluesselung.html, 02.01.2018

[2] Funktonsweise der Cäsarverschlüsselung, https://de.serlo.org/informatik/verschluesselung/caesar-verschluesselung-48121, 02.01.2018

[3] Vigenere-Quadrat, https://cybersicherheit.cispa.saarland/osa/images/vigenere-quadrat.jpg, 23.11.16

[4] Funktonsweise einer Häufigkeitsanalyse, http://www.matheprisma.uni-wuppertal.de/Module/Enigma/index.htm?4, 02.01.2018

[5] Funktonsweise des Kasiski-Tests, http://kryptografie.de/kryptografie/kryptoanalyse/kasiski-test.htm, 02.01.2018

[6] Häufigkeiten der verwendeten Zeichen http://www1.ids-mannheim.de/fileadmin/kl/derewo/DeReChar-v-uni-204-a-c-2018-02-28-1.0.html, 28.02.2018

[7] Random Methode des .net Frameworks, https://msdn.microsoft.com/en-us/library/system.random(v=vs.110).aspx, 02.01.2018

[8] Brute-Force-Methode, https://www.sciencedirect.com/science/article/pii/S0898122112001393, 27.03.2018

[9] Xamarin, https://www.xamarin.com, 27.02.18

[10] .NET, https://www.microsoft.com/net/, 27.02.18

[11] Visual Studio, https://www.visualstudio.com/de/, 27.02.18

[12] C#, https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/, 10.01.2018

[13] Xamarin.Forms, https://www.xamarin.com/forms, 27.02.18

[14] RESTful-Paradigma, http://www.gregbulla.com/TechStuff/Docs/ws-restful-pdf.pdf, 27.02.18

[15] JSON (Newtonsoft.Json), https://www.newtonsoft.com/json, 27.02.18

[16] PHP, http://php.net, 27.02.18

[17] Atom, https://atom.io, 27.02.18

[18] MySQL-Datenbank, https://www.mysql.com/de/, 10.01.2018

[19] PHP Data Objects, http://de.php.net/manual/de/book.pdo.php, 27.02.18

[20] Apache, https://httpd.apache.org, 27.02.18

[21] Cross Secure Storage (sameerIOTApps.Plugin.SecureStorage), https://sameer.blog/2016/02/01/secure-storage-plugin-for-xamarin/, 27.02.18

[22] SQLite (SQLiteNetExtensions.Async), https://www.nuget.org/packages/SQLiteNetExtensions.Async/, 27.02.18

[23] Xam.Plugin.Media, https://github.com/jamesmontemagno/MediaPlugin, 27.02.18

[24] ZXingNet, https://archive.codeplex.com/?p=zxingnet, 27.02.18

[25] Systemaufbau, https://msdn.microsoft.com/en-us/library/ee658090.aspx, 09.01.2018

[26] CIA-Triade, http://www.kryptowissen.de/schutzziele.php, 10.01.2018

[27] Peer-to-Peer-Kommunikation, https://de.wikipedia.org/wiki/Peer-to-Peer, 12.01.2018