Implementierung einer Smartphone-Anwendung zum Austausch verschlüsselter Daten mit einer Cloud

26. August 2014

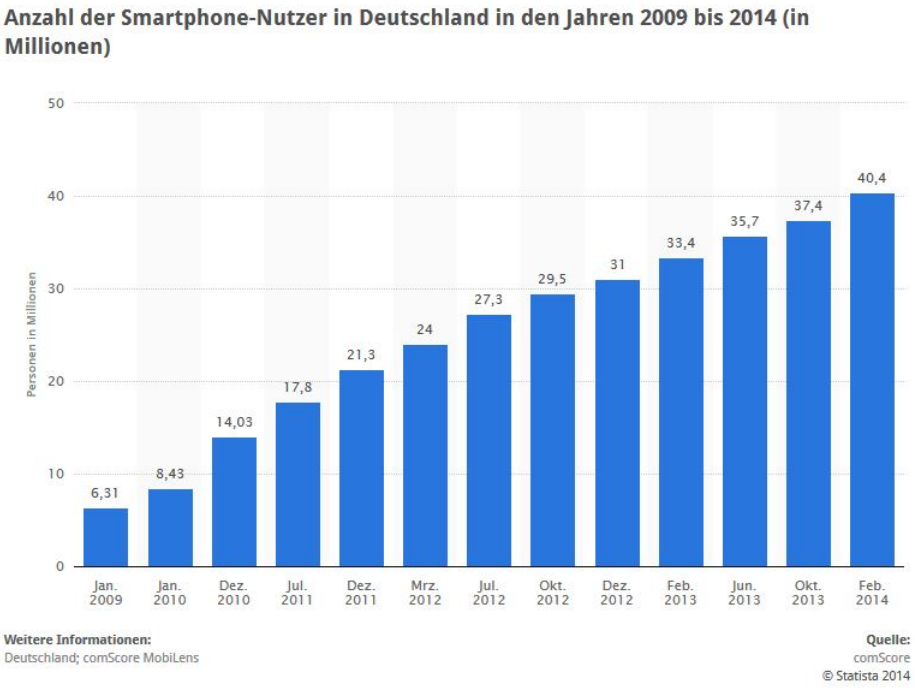
"If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology."

Bruce Schneier

Inhaltsverzeichnis

1 Einleitung

"Die Computer- und Internetnutzer in Deutschland setzen seit Bekanntwerden der geheimdienstlichen Abhöraktionen häufiger Verschlüsselungsverfahren ein.Äus der Pressemitteilung der BITKOM geht weiterhin hervor, dass von Juli 2013 auf November 2013 insgesamt 1,1 Millionen mehr Bundesbürger ihre persönlichen Dateien verschlüsseln. Besonders wichtig ist der Aspekt der Sicherheit, wenn es sich bei den Daten um relevante oder firmeninterne Informationen handelt, wie es z. B. am in Hamburg der Fall ist. Auch der Austausch von Daten von mobilen Endgeräten wie oder Tables spielen eine immer größere Rolle wie die Entwicklung der letzten Jahre zeigt (siehe Grafik).



Herkömmliche Verfahren zum Austausch von Daten reichen oftmals nicht mehr aus, wenn man den Aspekt der Sicherheit näher beleuchtet.

1.1 Motivation

Am Deutschen Elektronen Synchrotron, im folgenden DESY, werden bisher wichtige und sensible Dokumente über ein Programm Namens Dropbox gesichert und verwaltet. Dropbox bietet eine plattformunabhänginge Möglichkeit Dokumente Online abzuspeichern und von einem anderen Standort über ein internetfähiges Gerät wieder zu öffnen [https://www.dropbox.com/]. Auch wenn Dropbox nach eigenen

Angaben den Advanced Encryption Standard (AES) verwendet, bevor die Daten gespeichert werden, liegen die dafür notwendigen Schlüssel in Händen der Betreiber selbst, die somit vollen Klartextzugriff auf die Nutzerdateien haben. Dropbox begründet diesen Zugriff wie folgt: "Wie die meisten Online-Dienste verfügt auch Dropbox über einen kleinen Mitarbeiterstamm, dem aus in unserer Datenschutzrichtlinie dargelegten Gründen Zugriffsrechte auf Nutzerdaten gewährt werden muss [...]".
Da das DESY über eine eigene Cloud-Infrastruktur verfügt, sollen in Zukunft alle wichtigen Daten nicht nur in dieser Cloud gespeichert werden, sondern auch zusätzlich durch eine Verschlüsselung gesichert werden. Die Cloud am DESY stellt im Hintergrund ein Rechnernetz zum Abspeichern von Daten zur Verfügung. Durch das Programm dCache, welches das Rechnernetz im Hintergrund steuert und verwaltet, ist es dem Anwender möglich Daten in das System zu speichern, ohne dessen Struktur zu kennen. dCache sorgt dafür dass die Daten, je nach Bedarf, mehrfach abgelegt werden und bei einem Zugriff schnell zur Verfügung stehen. Die Dateien selbst werden im Hintergrund auf verschiedene Datenträger, wie z. B. SSD-Festplatten, Magnetbänder, Tapes o. ä., abgelegt. Das System sorgt dafür, dass bei reger Anfrage die Daten, sofern möglich, auf ein schnelleres Medium repliziert werden. Die genaue Struktur und Vorgehensweise des Programmes ist jedoch nicht Teil dieser Arbeit, da das hier zu entwickelnde Programm nur die Schnittstelle des dCache-Servers verwendet.

1.2 Zielsetzung

Ziel dieser Arbeit ist es aus diesem Grund einen Prototyp zu entwickeln, der einerseits mit dem Cloud-System des DESY Kommunizieren kann um dort Dateien hoch- und herunter zu laden, andererseits diese Daten auch in angemessener Form (siehe Kapitel Validierung) zu Verschlüsseln.
In der ersten Version dieser Arbeit wird ein Programm entwickelt, welches auf Android-Betriebssystemen zum Einsatz kommen kann. Darüber hinaus ist es wichtig, dass die entsprechenden Schlüssel zum entschlüsseln der Daten nicht zusammen mit den Daten abgelegt werden, sondern ausschließlich den Parteien des Datenaustauschs bekannt sein soll. Dies bedeutet, das selbst die Betreiber am DESY nicht die Möglichkeit haben die abgelegten Daten zu entschlüsseln.
Zum Ver- und Entschüsseln der Daten sollen Verfahren verwendet werden, die in der heutigen Zeit als sicher angesehen werden und Smartphones im Bezug auf Performance und Akkuverbrauch nicht zu stark belasten. Um diese Faktoren zu Validieren wird eine Testanwendung geschrieben, die mit bestimmten Faktoren die verschiedenen Verfahren untereinander überprüfen (siehe Kapitel Validierung).

1.3 Verwandte Arbeiten

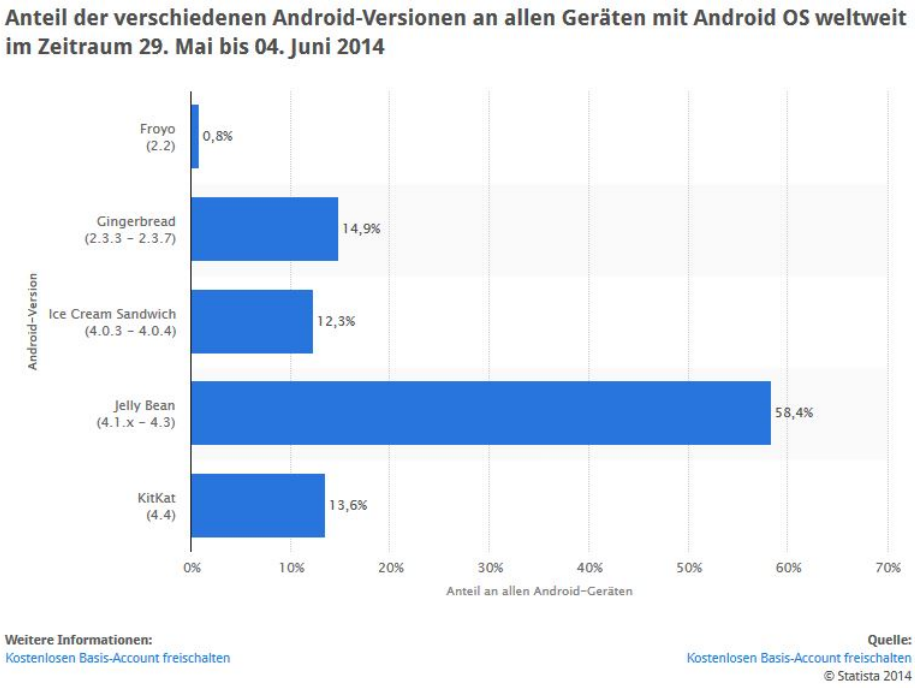
1.4 Verwandte Programme

1.5 Diese Arbeit

1.5.1 Inhaltlicher Aufbau

2 Grundlagen Android

Android ist ein Betriebssystem für Smartphones und Tablets, welches von der open handset alliance entwickelt wird. Das Konsortium besteht aktuell aus 84 Unternehmen, die an der Entwicklung des Betriebssystems arbeiten. In diesem Kapitel wird kurz darauf eingegangen, welche Kryptografischen Aspekte Android in den verschiedenen Versionen zur Verfügung stellt um diese im darauffolgenden Kapitel genauer zu Untersuchen. Aufgrund der Tatsache, dass die Android Version Gingerbread (2.3.3) im Juni 2014 noch einen Marktanteil von knapp 15% hält, ist dies auch die niedrigste vom Programm unterstützte Version.



Bei der Analyse wird darauf geachtet, dass alle Funktionalitäten die im Programm entwickelt werden, von dieser Version unterstützt werden. Die, während des Schreibens dieser Arbeit, aktuellste Version der Android API ist KitKat (4.4), bei der darauf geachtet wird, dass die eingesetzten Funktionen auch in dieser Version noch zur Verfügung stehen und nicht mit *deprecated (veraltet)* markiert sind.

2.1 Zusammenhang Kryptographie

Beim Thema Sicherheit im Zusammenhang mit Android ist erstmals der Begriff der Sandbox zu nennen. Eine Anwendung wird abgekapselt in einer eigenen Umgebung mit eigenem Prozess, eigenem Betriebssystem-

User, eigener Dalvik-VM, eigenem Heap und eigenem Dateisystem ausgeführt. Dieses abgekapselte Konstrukt wird Sandbox bezeichnet. Dadurch ist es dem Betriebssystem möglich unerlaubten Zugriff auf Ressourcen oder andere Programme zu beschränken, hierbei wird das Berechtigung- und Prozess-Management-System von Linux verwendet. Um dennoch verschiedene Zugriffe zu erlauben muss in der sogenannten Manifest-Datei der Anwendung die Berechtigung festgelegt und vom Benutzer bei der Erstinstallation bestätigt werden. Auch wenn dieses Konzept Daten zur Laufzeit innerhalb einer Anwendung schützt, ist es möglich Dateien auch auf einer SD-Karte zu speichern, in das Internet zu verschicken oder über andere Wege auszutauschen. Diese Daten sind dann außerhalb der Anwendung und gegen externe Zugriffe nicht mehr geschützt. Dennoch gibt es die Möglichkeit in Android diese Daten zusätzlich mit einer Verschlüsselung zu versehen - hierfür stellt Java, seit der Version 1.4, die *Java Cryptography Extension (JCE)* innerhalb von Android zur Verfügung. Innerhalb der Erweiterung (engl. extension) sind verschiedene Provider eingebunden, die dem Programmierer die Möglichkeit geben Kryptografische Verfahren aufzurufen, ohne die genaue Implementierung kennen zu müssen. In Java-Anwendungen gibt es diverse Implementierungen von Sun, die jedoch aus Datenschutzrechtlichen Gründen nicht in der Android Java-API vorhanden sind. der Provider Bouncy-Castle stellt eine Alternative zur Implementierung von Sun darf und wird in Android zur Verfügung gestellt. Innerhalb von Android wurde das Paket so geändert, dass es den Richtlinien der JCE entspricht. Bouncy-Castle ist einer der von Android zur Verfügung gestellten Provider - jedoch gibt es noch weitere Provider, die selbige oder andere Implementierungen zur Verfügung stellen. Mit folgendem Codeabschnitt ist es möglich die einzelnen Provider mit den unterstützten Verfahren auszulesen und untereinander zu vergleichen. Dieser Code wurde auf verschiedenen Versionen ausgeführt, um die Unterschiede der einzelnen Versionen hervorzuheben.

```
Provider[] providers = Security.getProviders();
for (Provider provider : providers) {
    Log.i("CRYPTO","provider: "+provider.getName());
    Set<Provider.Service> services = provider.getServices();
    for (Provider.Service service : services) {
        Log.i("CRYPTO","  algorithm: "+service.getAlgorithm());
    }
}
```

Im Vergleich stehen folgende Android-Versionen:

- 2.3.3 (Gingerbread) : die niedrigste vom Programm unterstützte Version
- 4.1.1 (Jelly Bean) : die Version des Entwickler-Gerätes
- 4.4.4 (KitKat) : aktuellste auf dem Markt verfügbare Android-Version
- "L" : zukünftige Version, welche bereits zu Testzwecken als Entwickler-Version freigeschaltet ist. Der Codename "L" zeigt auf, dass es sich in der Folge der Süßigkeiten (Gingerbread, HoneyComb, Ice Cream Sandwich, Jelly Bean, KitKat) vermutlich alphabetisch fortsetzen wird - die Versionsnummer ist bis dato nicht bekannt.

Im Vergleich der Ausgabe eines Gerätes mit Android 2.3.3 und eines mit 4.1.1 bzw. 4.4.4 und "L" liegt der Hauptunterschied in der Unterstützung von Elliptischen Kurven für das Diffie-Hellmann-Verfahren

(ECDH) und den Digital-Signature-Algorithm (ECDSA), welche in der Version 2.3.3 nicht unterstützt werden. Des Weiteren ist ab der Version 4.4.x der Provider OpenSSL und deren Algorithmen spezifischer dargestellt. Folgende Verschlüsselungsverfahren werden sowohl von der Version 2.3.3 als auch von der Version 4.1.1 , 4.4.4 und L unterstützt und werden im nachfolgenden Kapitel näher erläutert:

Verschluesselung	**Hash-Funktion**
AES	MD5
ARC4	SHA1
Blowfish	SHA256
DES	SHA384
3DES	SHA512
RSA	
ElGamal	

Des Weiteren wird das Key-Wrap-Verfahren (AES und 3DES), der Authentifizierungsalgorithmus HMAC und der Standard X509 beschrieben. Die Vollständige Liste aller Unterstützen Algorithmen mit dessen Providern befindet sich im Anhang. Welcher Provider für welchen Algorithmus besser geeignet ist, kann man nicht pauschalisieren und auch nicht auf einen spezifischen Anwendungsfall verallgemeinern. Im Kapitel Validierung werden Geschwindigkeitsaspekte beider großen Provider (OpenSSL und Bouncy-Castle), sofern möglich, gegenübergestellt um für jeden Algorithmus den geeigneten Provider zu wählen. Falls es innerhalb eines Providers zu größeren Sicherheitslücken von einem verwendeten Algorithmus kommt, ist es durch das JCE möglich diesen ohne weitere Code-Änderungen zu wechseln.

3 Grundlagen Kryptologie

Das Wort Kryptologie stammt aus dem Griechischen *kryptós* für verstecken und *lógos* für die Lehre.Dieser Zweig umfasst die Kryptographie - die Wissenschaft die sich mit der Absicherung von Daten beschäftigt, die Kryptoanalyse - welche für das Aufbrechen von Geheimnachrichten zuständig ist sowie der Mathematik. Im Bereich der Kryptologie ist es das Ziel eine Nachricht, welche aus lesbaren Zeichen (Klartext) besteht unverständlich zu machen (Verschlüsseln) und daraus einen Geheimtext (Chiffretext) zu erzeugen. Dieses Verfahren wird mit mathematischen Funktionen und einem Schlüssel (Key) durchgeführt. Die Umkehrung von Chiffretext in Klartext (Entschlüsselung) wird ebenfalls durch eine mathematische Funktion und einen Schlüssel durchgeführt. Ziel dieser Ver- und Entschlüsselung ist es Nachrichten zwischen einem Sender und Empfänger so auszutauschen, dass ein Angreifer diese nicht mitlesen, oder im verschärftem Sinne nicht verändern kann. Hierbei besteht eine Nachricht in der Informatik immer aus binären Daten und kann eine Textdatei, ein Bild, ein Video oder vieles mehr darstellen. Ver- und Entschlüsselung sind mathematische Funktionen, die auf den Klartext, bzw. auf den Chiffretext angewendet werden.

Terminologie

Um die Lesbarkeit zu erhöhen wird Klartext im folgenden mit M (engl. Message), Chiffretext mit C (engl. Chiffre), die Verschlüsselungsfunktion mit E (engl. Encoding), die Entschlüsselungsfunktion mit D (engl. Decoding) und dem Schlüssel K (engl. Key) beschrieben. Zum Verschlüsseln kommt also folgende Funktion zum Einsatz:

$\mathrm{E}_K(\mathrm{M}) = \mathrm{C}$

Um den Chiffretext wieder zu Entschlüsseln wird die umgekehrte Richtung angewandt:

$\mathrm{D}_K(\mathrm{C}) = \mathrm{M}$

Zusammengefasst muss also gelten: das Verschlüsseln einer Nachricht und das darauffolgende Entschlüsseln des erzeugten Chiffretextes, mit der dazugehörigen Funktion und korrektem Schlüssel, muss wieder den Klartext ergeben. Mathematisch beschrieben ist das wie folgt:

$\mathrm{D}_K(\mathrm{E}_K(\mathrm{M})) = \mathrm{M}$

Um einen sicheren Kanal zwischen Sender und Empfänger zu gewährleistet, reicht es nicht allein die Nachricht zu verschlüsseln. Authentifizierung, Integrität und Verbindlichkeit müssen darüber hinaus gewährleistet sein um sicher zu Kommunizieren.

Authentifizierung beschreibt hierbei das Verfahren indem sich die Identität einer Person beweisen lässt. Im Umkehrschluss bedeutet das, dass sich ein Angreifer nicht als eine andere Person ausgeben kann. Aus der Authentifizierung folgt dann die **Autorisierung**, also das Prüfen, ob der Benutzer die Rechte hat, die er fordert.

Integrität bedeutet, dass sichergestellt werden kann, dass eine Nachricht bei der Übermittlung zwischen Sender und Empfänger nicht durch einen Angreifer verändert wurden ist.

Verbindlichkeit beschreibt dass der Sender nicht leugnen kann, dass eine Nachricht gesendet wurde. Dies ist eine Steigerung der Authentifizierung, denn durch Verbindlichkeit ist es außerdem gewährleistet, dass der Empfänger einer Nachricht gegenüber einer dritten Person den Absender der Nachricht glaubwürdig machen kann.

Kerhoff's Maxime

Ein Aspekt in der Kryptographie sind die Kerkhoffs' Maxime, die folgendes Aussagen: "the security of the encryption scheme must depend only on the secrecy of the Key K_e, and not on the secrecy of the algorithms." [Schneier, 1997] Übersetzt bedeutet es, dass die Sicherheit eines Kryptographischen Verfahrens auf der Geheimhaltung des Schlüssels beruhen muss und nicht auf derer des Verschlüsselungsalgorithmus.

Verfahren

Prinzipiell unterteilt man Kryptographie in zwei Verschiedene Verfahren. Die symmetrischen Verfahren und die asymmetrischen, auch public key infrastructure genannt. Generell lässt sich über das "bessere Verfahren" keine Aussage treffen, da es für beide Verfahren Vor- und Nachteile gibt. Bruce Schneier fasste es wie folgt zusammen:
"Symmetrische Kryptographie eignet sich am besten zur Verschlüsselung von Daten. Sie ist um Größenordnungen schneller und nicht anfällig für chosen-ciphertext-Angriffe. Public-Key-Kryptographie schafft Dinge, die außerhalb des Einsatzbereichs symmetrischer Kryptographie liegen und eignen sich am besten für die Schlüsselverwaltung und eine Vielzahl der Protokolle [...]."
Der im Zitat verwendete Ausdruck, chosen-ciphertext-Angriff beschreibt einen Angriff auf ein Kryptosystem, bei dem der Kryptoanalytiker verschiedene Chiffretexte zur Entschlüsselung auswählen kann und entsprechend Zugriff auf den dazugehörigen Klartext besitzt. Die Aufgabe bei dieser Art des Angriffes besteht darin, den entsprechenden Schlüssel herauszufinden. Neben der chosen-ciphertext-Angriffe gibt es weitere Angriffsszenarien auf Kryptosysteme, wie z. B. ciphertext-only, known-plaintext, chosen-plaintext, chosen-key und weitere. Da es sich bei dieser Arbeit nicht um eine Kryptoanalyse eines Systems handelt, werden diese Szenarien nicht näher erläutert. Es wird davon ausgegangen, dass wenn eines dieser Szenarien zum knacken des Systems führt, dieses kryptographische Verfahren bereits heute als unsicher angesehen wird.

3.1 Symmetrische Verfahren

Bei symmetrischen Verschlüsselungsverfahren existiert ein Schlüssel, der jeweils für Ver- und Entschlüsselung verwendet wird. Dieser Schlüssel muss bereits beiden Parteien bekannt sein, bevor ein verschlüsselter Kanal aufgebaut werden kann. Eines der Probleme bei symmetrischen Verfahren ist der Austausch des Schlüssels, den man von Sender zu Empfänger, bereits vor der sicheren Kommunikation, übertragen muss (siehe Kapitel Schlüsselvereinbarung). Symmetrische Verfahren unterteilt man in zwei Grundtypen, die Block- und Stromchiffrierung. Bei der Blockchiffrierung wird der Klartext in Blöcke, mit fester Größe, aufgeteilt und innerhalb des Blockes werden die mathematischen Funktionen angewandt. Bei der Stromchiffrierung werden die Daten nicht in Blöcken zusammengefasst, sondern jedes einzelne Klartextbit wird in ein Chiffrebit überführt. [Schneier 1996, Seite 223]

3.1.1 Betriebsmodi

Betriebsmodi sind verfahren bei der das eigentliche Kryptografische Verfahren mit einer Rückkopplung und einigen einfachen Operationen verknüpft wird. Ist eine Nachricht länger als die für das Verfahren angegebene Blocklänge, so muss die Nachricht mit einem Betriebsmodi angepasst werden. Wichtig bei den verschiedenen Betriebsmodis ist, dass sie die Sicherheit des Cryptoverfahrens nicht beeinträchtigen.

Padding

Bei der Blockchiffrierung ist eine fest Blocklänge vorgegeben, in der die Daten vorhanden sein müssen. Ist dies nicht der Fall, müssen diese so modifiziert werden, dass das Verfahren damit umgehen kann. Um z. B. den letzten Block einer Nachricht, der nicht der Blocklänge entspricht, zu vervollständigen wird er mit einem regelmäßigen Muster aufgefüllt. Das Muster und die Art dieser Auffüllen, bzw. das Kennzeichnen hängt von den Verschiedenen Padding-Verfahren ab. (z. B. PKCS5, PKCS7 o. ä.)

ECB

ECB (*electronic codebook mode*) ist ein Betriebsmodi, bei der der Klartext in verschiedene Blöcke, der benötigten Länge, aufgeteilt wird und jeder dieser Blöcke einzeln verschlüsselt werden. Das Konkatinieren dieser Blöcke ergibt dann den neuen Chiffretext. Problem bei diesem Verfahren ist, dass 2 gleiche Klartextblöcke auf den identischen Chiffreblock ergeben, was wiederum für den Angreifer sichtbar und Nutzbar sein kann. Aus diesem Grund wird ECB als unsicher angesehen und sollte deshalb nicht verwendet werden ("Do not ever use ECB for anything" [Ferguson 2003, Seite 69]).

CBC

Beim *cipher block chaining mode* (CBC) wird das Problem von ECB umgangen, indem man jeden Klartextblock mit den vorherigen Chiffretextblock mit einer XOR-Verknüpfung durchführt:

$$C_i = E(K, P_i \oplus C_{i-1})$$

Dadurch werden alle Bits eines Klartextblockes mit einer bereits verschlüsselten Nachricht verknüpft. Gleiche Blöcke werden so mit unterschiedlichen Cryptotexten verknüpft und ergeben so unterschiedliche

Ausgaben. Da die obenstehende Formel erst angewandt werden kann, wenn ein Chiffretextblock vorliegt muss die Möglichkeit geschaffen werden, den ersten Klartextblock auch zu verknüpfen (also C_0). Der Initiale Block der für diese Verknüpfung angewandt wird heißt *initialization vector* (IV). Es gibt verschiedene Möglichkeiten diesen initialization vector zu bestimmen. Zum einen kann man einen festen IV für alle Nachrichten wählen, das hätte wiederum zur Folge, dass 2 gleiche Klartextblöcke zu einem identischen Chiffreblock verschlüsselt werden (siehe ECB). Als weitere Möglichkeit besteht den IV zu iterieren - jedoch ist auch diese Möglichkeit nicht zu nutzen, da sich der IV bei einer Iteration zu Beginn nur um 1 Bit unterscheidet und dies sich im Chiffretext dann auch lediglich auf 1 Bit auswirkt. Um alle Bits des ersten Klartextblocks zu verändert, besteht die Möglichkeit des sogenannten *random IV*. Der Initialisierungsvektor wird komplett zufällig gewählt und entsprechend der Nachricht angehangen, oder vorangestellt, damit beim Entschlüsseln diese Verknüpfung wieder rückgängig gemacht werden kann.

OFB

Beim OFB (*Output feedback mode*) wird nicht die Klartextnachricht zur Verschlüsselung benutzt, sondern eine Zufallsreihe von Bytes. Die daraus resultierende verschlüsselte Nachricht kann dann mit der Klartextnachricht verknüpft werden. Der Vorteil besteht darin, dass die Berechnung des Chiffretextes aus der Zufallsreihe bereits durchgeführt werden kann, bevor der eigentliche Klartext zur Verfügung steht. Dieses Verfahren ist außerdem in der Lage, Klartexte zu verschlüsseln die kleiner als die eigentliche Blocklänge sind (Wichtig für Byteweise Anwendungen, z. B. Terminalanwendungen).

$K_0 := IV$
$K_i := E(K, K_{i-1})$
$C_i := P_i \oplus K_i$

CFB

Der *cipher feedback mode* ähnelt dem oben beschriebenen OFB Modus, mit dem Unterschied, dass die verschlüsselte Nachricht (nach Verknüpfung des Klartextes) als neuer Block für die nächste Verschlüsselung angesehen wird. Das bedeutet, dass eine Verschlüsselung erst stattfinden kann, sobald der erste Klartextblock vorliegt (Dieser kann, wie auch bei OFB, kleiner sein als die Blockgröße des zu Grunde liegenden Kryptoverfahrens).

$C_0 = P_0 \oplus E_K(IV)$
$C_i = P_i \oplus E_K(C_{i-1})$
$P_i = C_i \oplus E_K(C_{i-1})$

CTR

Im Counter-Modus kann die Berechnung des Schlüssels, wie auch beim OFB, vor dem ersten Klartextblock erfolgen. Innerhalb des Modi gibt es einen internen Zähler, der immer konstant erhöht wird. Aus diesem

Zähler und dem Schlüssel wird dann ein neuer Schlüssel erzeugt, mit dem der Klartext verschlüsselt werden kann. Der Vorteil dieses Verfahrens besteht darin, dass man nicht die komplette Nachricht entschlüsseln muss, sondern einzelne Teile des Chiffretextes entschlüsseln kann. Hierfür setzt man den internen Zähler auf die gewünscht Stelle, erzeugt den Schlüssel und entschlüsselt den Chiffreblock. Wichtig ist, dass der selbe Zähler mit den selben Schlüssel nicht doppelt verwendet werden soll (Problem: zwei identische Klartextblöcke ergeben identischen Chiffretext).

3.1.2 DES, 3DES

"Its restricted key size of 56 bits and small block size of 64 bits make it unsuitable for today's fast computers and large amounts of data. It survives in the form of 3DES, which is a block cipher built from three DES encryptions in sequence. This solves the most immediate problem of the small key size, but there is no known fix for the small block size. [...] we do not recommend using either DES oder 3DES in new designs." [Ferguson 2003, Seite 51] Niels Ferguson weißt darauf hin, dass DES aufgrund seiner Schlüsselänge von 56 bits und der Blockgröße von 64 bits ungeeignet für heutige Systeme ist. Weiterhin beschreibt er, dass auch durch 3DES das Problem der geringen Blockgröße nicht behoben wird und er schlussfolgert dass man in heutigen neuen Systemen beide Verfahren nicht verwenden sollte.
Da das System als unsicher angesehen ist, wird auf eine nähere Untersuchung und Erläuterung der mathematischen Funktionen verzichtet. DES und 3DES wird in der zu entwickelnden Anwendung nicht implementiert.

3.1.3 AES

Der Advanced Encryption Standard, im folgenden AES genannt, ist ein Verschlüsselungsverfahren welches auf Blockchiffrierung beruht und seit 2002 ein offizieller Standard ist. Entwickelt wurde der neue Standard mit dem Namen Rijndael bei einer Ausschreibung für einen neuen Sicherheitsstandard durch J. Daemen und V. Rijmen, die sich gegen 14 andere Konkurrenten durchsetzen konnten. Auch unter den besten 5 dieser Ausschreibung waren die Verfahren MARS, RC6, Serpent und Twofish, wobei keiner dieser fünf eine Sicherheitsschwäche aufwies. Rijndael konnte letztendlich durch seine einfache Struktur und gute Performance im Software-, sowie im Hardwarebereich überzeugen. Darüber hinaus hat die US National Security Agency (NSA) AES für interne Dokumente bis zum Sicherheitsstatus TOP SECRET, mit einer Schlüssellänge von 192 oder 256 und für den Sicherheitsstatus SECRET mit einer Schlüsselänge von 128 Bit erlaubt, was verdeutlicht, dass selbst Kryptographen von Geheimdiensten diesen Standard als sicher ansehen.

Funktionsweise

AES arbeitet mit einer Blockgröße von 128 Bit, also 16 Byte welcher intern als 2-Dimensionale Matrix (4x4) abgespeichert wird und auf der mathematische Funktionen angewandt werden. Alle Funktionen innerhalb von AES werden Byteweise ausgeführt (8 Bit-Blöcke). Die interne Nachricht bezeichnet man als *state*, also den aktuellen Status des Blockes.

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

Die Schlüssellänge des Verfahren ist entweder 128, 192 oder 256 Bit, wovon auch die auszuführende Rundenzahl abhängt (10 Runden bei 128 Bit, 12 bei 192 und 14 bei 256 Bit Schlüssellänge). In jeder dieser Runden werden Verfahren angewandt um den Klartext weiter zu verschlüsseln. Bei AES sind das *Byte-Substitution, ShiftRow, MixColumns* und *KeyAddition*.

Ausgenommen von dieser Regel ist die letzte Runde, in der MixColumns ausgelassen wird. Zusätzlich wird vor der ersten Runde die Funktion KeyAddition angewendet.

ByteSubstitution

In der Funktion ByteSubstitution wird eines der beiden Verfahren zum Verbergen von Redundanz angewendet - die Konfusion. Die Konfusion sorgt dafür, dass der Zusammenhang zwischen Klartext und Chiffrat verschleiert wird und möglichst aus einer kleiner Änderung im Klartext eine große Änderung im Chiffrat erzeugt wird. Hierzu wird jedes Byte in eine sogenannte S-Box eingegeben, wobei diese wiederum ein Byte aus Ausgabewert hat (Dieses Verfahren wird für alle 16 Byte eines Blockes angewandt). Die S-Box selbst ist eine 16x16 Matrix, mit der zu jeder eingegebenen Bit-Reihenfolge eine neue Ausgabe-Reihenfolge erzeugt wird. Ziel ist es, durch minimale Veränderung des Eingabewertes eine maximale Veränderung des Ausgabewertes zu erzeugen. Darüber hinaus ist die in AES verwendete S-Box nicht linear - das bedeutet, dass die Addition zweier einzelner Ausgabewerte nicht das selbe Ergebnis liefert wie die Addition zweier Eingabewerte:

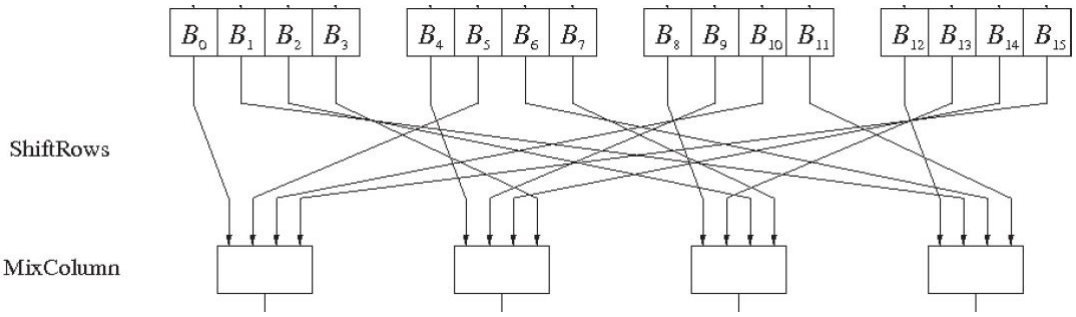
$$S(A) + S(B) \neq S(A + B)$$

Zusätzlich ist die S-Box bijektiv, es existiert also zu jeder Bitreihenfolge genaue eine eindeutige Zuweisung - Bitreihenfolgen die zwei Ausgabewerte erzeugen können existieren nicht. Im Umkehrschluss bedeutet das, dass jedes Ausgabebyte der S-Box wieder durch eine inverse S-Box zurück transformiert werden kann (Wird bei der Entschlüsselung verwendet). Die S-Boxen innerhalb von AES sind alle identische, sodass 16x pro Runde immer die selbe Matrix verwendet wird. Das hat zur Folge, dass sie in den den meisten Softwareimplementierungen durch fixe Tabellen realisiert werden, anstatt sie jedes mal neu zu Berechnen. Die Berechnung einer S-Box erfolgt durch Endliche Körper und eine fixe Multiplikation und Addition um die Logik der endlichen Körper zu verwischen. [c. Paar, Seite 102

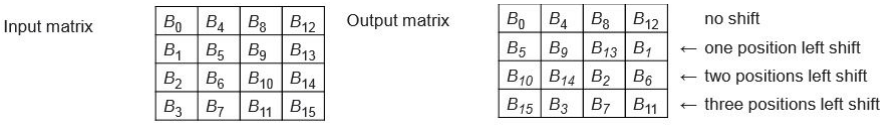
ShiftRow

Das zweite Verfahren zum Vergeben von Redundanz ist die Diffusion, bei der die Redundanz verteilt wird (Einfachster Anwendungsfall ist das Vertauschen der Klartextbuchstaben in eine neue Reihenfolge). Eine Funktion die innerhalb von AES die für Diffusion sorgt, ist das ShiftRow-Verfahren. Hierbei werden die Bytes innerhalb einer Spalte des state-Blockes auf alle anderen Spalten aufgeteilt. Eine Änderung innerhalb einer Spalte (A_0 bis A_3 der state-Matrix) hat somit Auswirkung auf alle anderen Spalten (Auswirkung auf komplette State-Matrix). Folgende Grafik soll das verdeutlichen, wobei B_0, B_1, ..., B_{15} jeweils

die Bytes A_0, A_1, ..., A_{15} nach der Transformation durch die S-Box sind. In der Grafik ist die interne 4x4 Matrix (state) hier Spaltenweise nebeneinander abgebildet. (Vergleiche state-Matrix)



Die in der Grafik gezeigten Linien, die die Verschiebung darstellen sollen, ist innerhalb der state-Matrix durch einfaches Shifting realisiert. Hierbei wird in der ersten Zeile keine Verschiebung durchgeführt, in der zweiten Zeile wird jedes Byte um 1 nach Links rotiert, in der dritten 2 nach Links und in der vierten Zeile 3 nach Links.



MixColumns

Die MixColumns-Funktion ist die zweite Funktion in AES die für die Diffusion sorgt - sie bewirkt, dass die Änderung eines einzigen Eingabebytes in die Funktion alle Ausgabebytes verändert. Hierbei wird jede Spalte (als Vector dargestellt) mit einer festen 4x4 Matrix multipliziert.

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

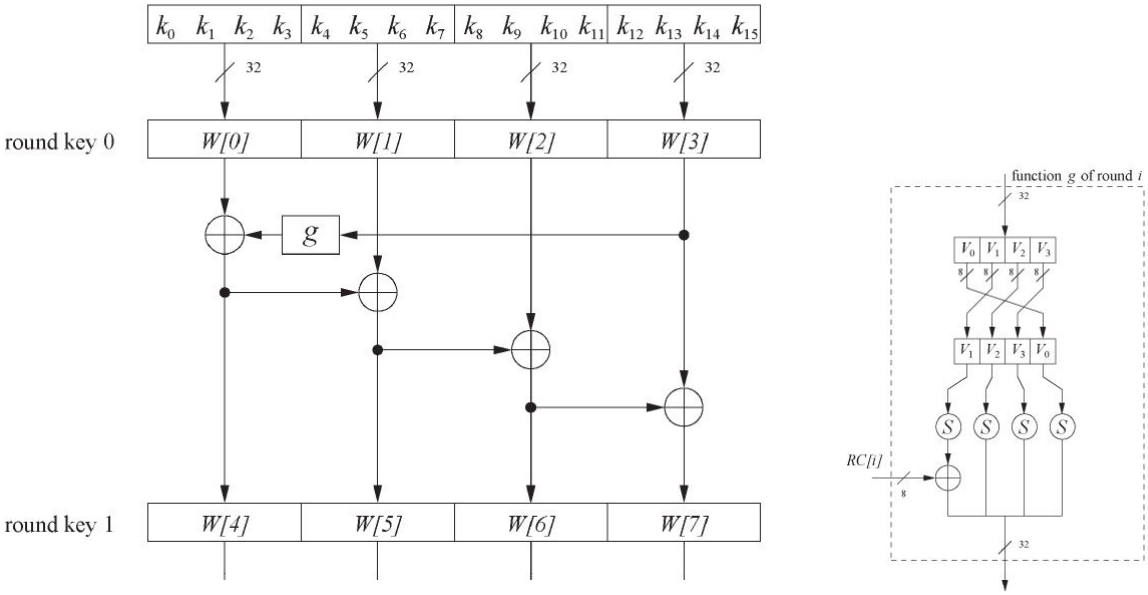
In der Grafik beschreibt der Vector B_0, B_5, B_{10}, B_{15} genau die erste Spalte nach der Verschiebung durch ShiftRow. Durch die starke Diffusion die durch die Verteilung der Bytes von einer Spalte auf alle Spalten in der Funktion ShiftRow und die Vermischung aller Bytes durch die MixColumns-Funktion erreicht wird, ist es dem Verfahren AES mögilch in 3 Runden jedes Byte des Klartextes von allen 16Byte der state-Matrix abhängig zu machen. Wenn also die zu verschlüsselnde Nachricht aus einer 1 und restlichen Nullen besteht wird diese 1 in nur 3 Runden auf alle anderen Nullen Auswirkung zeigen.

KeyAddition

Beim KeyAddition wird jeweils der aktuelle Block (4x4 state-Matrix) mit dem aktuellen Rundenschlüssel (16 byte) via XOR Bitweise verknüpft.

Rundenschlüssel

AES erzeugt für die verschiedenen Runden, die bei der Ver- und Entschlüsselung durchlaufen werden Rundenschlüssel (1 Schlüssel mehr als Runden die durchlaufen werden), welche in 4x 32-Bit große Blöcke (Word-oriented) abgespeichert werden. In der ersten Runde entspricht der Rundenschlüssel dem Original AES-Schlüssel (W[0] bis W[3] = 4x32Bit = 128Bit Schlüssellänge). Der letzte Word-Block einer Runde, wird dann durch eine Funktion gegeben und mit den anderen Blöcken XOR-Verknüpft.



Die Funktion g rotiert hierbei jeweils die Eingabebytes und führt sie durch eine nichtlineare S-Box. Am Ende dieses Verfahrens wird noch die Rundennummer mittels XOR dem linken Teilblock zugefügt. Das Ergebnis dieser Durchführung (W[4] bis W[7]) ist dann der Rundenschlüssel für die erste Runde. Dieses Verfahren wird wiederholt bis alle Rundenschlüssel entsprechend berechnet wurden. Die Anzahl der benötigten Rundenschlüssel und damit verbunden mit den benötigten Word-Blöcke erhöht sich mit der Erhöhung der Schlüssellänge von AES.

Entschlüsselung

Für die Entschlüsselung eines AES-Chiffretextes müssen alle Funktionen in umgekehrter Reihenfolge und umgekehrter Logik (Inverse Funktionen) ausgeführt werden. So hat man z. B. in der letzten Runde der Verschlüsselung die Funktion MixColumns nicht ausgeführt - so wird man in der ersten Runde der Entschlüsselung diese Funktion ebenfalls nicht ausführen. Darüber hinaus muss man für alle fixen Matrizen, die verwendet wurden eine inverse Matrix erstellen (S-Boxen, MixColumns-Matrix). Das Shifting in der Funktion ShiftRow erfolgt bei der Entschlüsselung dann entsprechend nach Rechts, anstatt nach Links wie bei der Entschlüsselung. Ausgenommen von der Umgekehrten Logik ist die Berechnung der Rundenschlüssel - da man in der ersten Runde der Entschlüsselung den letzten Rundenschlüssel benötigt, der bei der Verschlüsselung eingesetzt wurde, müssen zu Beginn der Entschlüsselung erstmals alle Rundenschlüssel berechnet werden um diese dann zu verwenden. Die Berechnung der Rundenschlüssel selbst ist identisch.

3.1.4 ARC4

RC4, oder auch ARC4 (Arcfour) genannt ist eine Stromverschlüsselung, wird also Bitweise ent- und verschlüsselt. Nach dem Aufdecken geheimer Informationen der NSA durch den Whistleblower Edward Snowden, hat der Kryptograph Jacob Appelbaum (Mitentwickler des Sicherheitsnetzwerkes Tor und Unterstützer von WikiLeaks) auf Twitter einen Post geteilt in dem er sagt, dass mit RC4 verschlüsselte Daten von der NSA in Echtzeit entschlüsselt werden können: "RC4 is broken in real time by the #NSA - stop using it." Diese Behauptung wird auch von Bruce Schneier (Experte für Kryptographie, Entwickler der Verfahren Blowfish und Twofish, Mitglied in mehreren Verbänden) in seinem offiziellen Blog als plausibel bestätigt: "Someone somewhere commented that the NSA's groundbreaking cryptanalytic capabilities could include a practical attack on RC4. I don't know one way or the other, but that's a good speculation." Darüber hinaus warnen verschiedene Seiten, wie Golem und Heise, die sich mit Informatik beschäftigen vor der Verwendung von RC4. Selbst das Bundesamt für Sicherheit in der Informationstechnik (BSI) schreibt in einer technischen Richtlinie für Kryptographische Verfahren Anfang 2014: "Der Verschlüsselungsalgorithmus RC4 in TLS weist erhebliche Sicherheitsschwächen auf und darf nicht mehr eingesetzt werden." Das BSI gibt zusätzlich an, dass das Verschlüsselungsverfahren AES verwendet werden soll. Durch diese gezeigten Publikationen wird das Verfahren RC4 als unsicher angesehen und in dieser Arbeit nicht verwendet.

3.1.5 Blowfish

Blowfish ist eine Blockchiffrierung mit einer Blockgröße von 64 Bit. Wie auch bei DES und Triple-DES beschrieben ist diese Blockgröße für die heutigen Computer ungeeignet. Um diese Problem zu beheben hat der Erfinder des Verfahrens Bruce Schneier das Verfahren Twofish entwickelt, welches bei der Ausschreibung von AES auch unter den 5 Finalisten nominiert war. Es arbeitet auf einer Blockgröße von 128 Bit. Auch wenn es keine Beweisbaren belege für die Unsicherheit von Blowfish gibt, merkt Bruce Schneier in einem Interview an, dass man Twofish verwenden solle: "If people ask, I recommend Twofish instead." Aus der Analyse der Verfügbaren Kryptoverfahren, die in den verwendeten Android-Versionen angeboten werden ist Twofish noch nicht standardisiert enthalten. Durch die Validierung im späteren Kapitel bleibt offen, ob das Verfahren Blowfish in der Implementierung Einzug finden wird.

Funktionsweise

Die Vorgehensweise der Datenverschlüsselung von Blowfish beruht auf dem Feistel-Netzwerk. Hierbei wie der Block in 2 Hälften unterteilt, wobei die eine Teilhälfte immer durch eine Funktion verändert wird und die andere Teilhälfte in die nächste Runde weitergeben wird. Dabei wird bei jeder Runde beide Hälften miteinander vertauscht, sodass jede Teilhälfte alle 2 Runden der Funktion unterzogen wird.

$L_i = R_{i-1}$
$R_i = L_{i-1} \oplus f(R_{i-1}, PK_i)$

In folgender Grafik wird das Verfahren von Blowfish und der Feistel-Struktur verdeutlicht dargestellt:

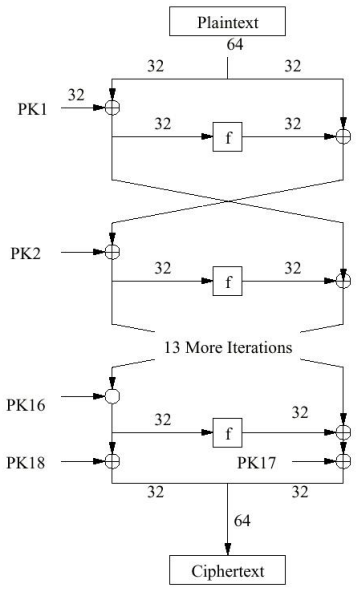
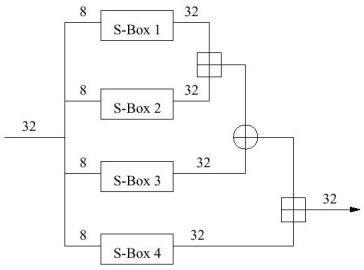


Abbildung 3.1: Bild1

Hierbei beschreibt PK_i den Teilschlüssel der aktuellen Runde. Innerhalb der Funkion von Blowfish wird die 32 Bit große Teilhälfte in 4x8 Bit unterteile gesplittet und jeweils einer Substitution durch S-Boxen unterzogen. Im Gegensatz zu AES sind alle 4 S-Boxen in Blowfish verschieden. (Berechnung: siehe Teilschlüssel) Folgende Vorgehensweise wird innerhalb der Funktion durchgeführt:



Das Ergebnis von S-Box 1 und S-Box 2 wird Addiert und Modulo 2^{32} berechnet. Anschließend wird das Ergebnis mit dem Ergebnis von S-Box 3 XOR-Verknüpft und zum Schluss dessen Ergebnis mit dem Ergebnis aus S-Box 4 erneut addiert und Module 2^{32} errechnet. Zusammenfassend ergibt das folgende Formel:

$$F(x_L = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

Teilschlüssel

Für die Generierung der Teilschlüsse, sowie der Erzeugung der S-Boxen existiert ein Algorithmus, der wie folgt vorgeht:

1. Das P-Array (Array der Größe 18, welches die Teilschlüssel mit je 32 Bit hält) und die vier S-Boxen werden der Reihe nach mit den Hexadezimalstellen von π befüllt.
2. P_1 wird mit den ersten 32 Bit des Schlüssels XOR-Verknüpft, P_2 mit den zweiten. Ist das Schlüsselende erreicht, wird von vorne begonnen. Dieses Verfahren wird solange ausgeführt, bis alle 18 Felder des P-Arrays mit den Schlüsselbits XOR-Verknüpft sind.
3. Eine Zeichenkette bestehend aus Nullen wird in den Blowfish Algorithmus gegeben (Wichtig ist, dass hierbei bereits das geänderte P-Array verwendet wird)
4. Das Ergebnis der Verschlüsselung aus Punkt 3 wird als P_1 und P_2 verwendet.
5. Die Ausgabe von Punkt 3 wird erneut Verschlüsselt (Wieder mit dem geänderten P-Array)
6. Die Ausgabe von Punkt 5 wird zu P_2 und P_4.
7. Dieses Verfahren wird solange fortgesetzt, bis alle Elemente des P-Array, sowie der Reihe nach alle vier S-Boxen mit der Ausgabe des wechselnden Blowfish-Algorithmus ersetzt wurden.

Bleibt der Schlüssel identisch, so kann die Anwendung die in 521 Iterationen errechneten Teilschlüssel speichern und muss diese nicht bei jeder Verschlüsselung neu errechnen. Das hier gezeigte Verfahren zum

Erstellen der Teilschlüssel wird auch Schlüsselexpansion genannt, da der Algorithmus nun auf insgesamt 4168 Bit Teilschlüsseln besteht.

3.2 Asymetrische Verfahren

Im Gegensatz zu den bisher gezeigten Verschlüsselungsverfahren, ist der Ansatz bei Asymmetrischen Verfahren ein anderer. Bisher gab es einen geheimen Schlüssel, den man sowohl für die Verschlüsselung, als auch für die Entschlüsselung verwendete. Die Hauptaufgabe bei diesen Verfahren lag darin, den geheimen Schlüssel sicher von A nach B zur transportieren, ohne das ein eventueller Angreifer diesen mitlesen kann. Bei Asymmetrischen Verfahren ist der Grundgedanke derer, dass man jeweils zum Verschlüsseln einen Schlüssel besitzt als auch zum Entschlüsseln einen separaten Schlüssel. Der Schlüssel, welcher für die Verschlüsselung zuständig ist, wird Public Key genannt, da man ihn den Partnern öffentlich zuteilen kann. Das Pardon dazu ist der Private Schlüssel, der im eigenen Besitzt bleibt und nur dafür ist, die Verschlüsselte Nachricht wieder zu Dechiffrieren. Einer der Hauptaufgaben von Asymmetrischen Verfahren, oder auch Public-Key Verfahren genannnt, besteht darin 2 Schlüssel zu finden, zwar jeweils für die Verschlüsselung und Entschlüsselung zu gebrauchtn sind, aus denen man aber den jeweils anderen Schlüsselteil nicht errechnen kann. Der Grundgedanke dieses Verfahrens stammt von Whitfield Diffie und Martin Hellman, die hierzu 1976 ein Konzept auf der *National Computer Conference* vorstellen.

3.2.1 RSA

RSA, nach den Erfindern Rivest, Shamir und Adleman benannt ist, ist eines der oben beschriebenen Asymmetrischen Kryptoverfahren. Die Sicherheit von RSA beruht auf dem Problem der Primfaktorzerlegung - d. h. es ist schwierig, aus einem gegebenem Faktor 2er Primzahlen, diese zurück zu rechnen.
Das Erstellen der Schlüssel wird wie folgt ausgeführt:

1. wähle 2 große Primzahlen p und q
2. berechne n = p * q
3. berechne phi(n) = (p-1) * (q-1)
4. wähle ein e für das gilt e $\in$ {1,2,...,phi(n)-1} und ggT(e, phi(n)) = 1
5. berechne d, sodass gilt d * e $\equiv$ 1 mod phi(n)
6. Öffentlicher Schlüssel = (n,e) , Privater Schlüssel = d

Nach der Berechnung kann der öffentliche Schlüssel (n,e) frei zugänglich gemacht werden. Dieser Schlüssel wird verwendet um Nachrichten zu Verschlüsseln, oder eine Nachricht zu signieren (siehe Digitale Signaturen). Zum Entschlüsseln des entsprechenden Chiffretextes wird der private Schlüssel d benötigt. Die Werte p, q, phi(n) werden nicht mehr benötigt, dürfen jedoch auch nicht frei zugänglich gemcht werden, da mit diesen die Berechnung des privaten Schlüssels erfolgen kann. Zum Chiffrieren und Dechiffrieren einer Nachricht werden folgende mathematische Funktionen angewandt, wobei M = Message und C =

Chiffrat bedeutet:

- $C = M^e \bmod n$
- $M = C^d \bmod n$

Die Schlüssellänge des Verfahrens RSA ist variabel zwischen 512 und 2048 zu wählen, wobei sie die Bitlänge der berechneten Zahl n beschreibt. Zu beachten ist außerdem, dass es lediglich möglich ist Werte zu Verschlüsseln die im Bereich M $\in \{0, \ldots, n-1\}$ liegen. Um dennoch größere Nachrichten zu verschlüsseln, teilt man den Klartext in Blöcke auf und verschlüsselt diese einzeln mit dem Verfahren. Zur Entschlüsselung werden nach der Dechiffrierung die entsprechenden Blöcke wieder aneinandergehängt.
Die Geschwindigkeit des Verfahrens hängt stark von der Wahl der Exponenten ab. Wählt man den öffentlichen Exponent e relativ klein, so wird die Verschlüsselung entsprechend schneller, hingegen ein großer privater Schlüssel d die Entschlüsselung entsprechend längere Zeit in Anspruch nimmt. Darüber hinaus ist es wichtig um die Performance von RSA zu steigern eine Möglichkeit zu finden, große Exponenten schnell zu errechnen. Hierbei kommt das Square- and Multiply Verfahren zum Einsatz.

3.2.2 ElGamal

ElGamal ist, wie auch RSA, ein Asymmetrisches Verfahren - d. h. es gibt einen öffentlichen- und einen privaten Schlüssel. Die Sicherheit des Verfahrens besteht in der Schwierigkeit diskrete Logarithmen über einen endlichen Körper zu berechnen.
Um ein Schlüsselpaar zu erzeugen wählt man eine Primzahl p und zwei Zufallszahlen g und x, welche kleiner als p sind und berechnet:

$y = g^x \bmod p$

Der öffentliche Schlüssel ist y, g und p - der private Schlüssel ist x. Um eine Nachricht zu verschlüsseln wählt man ein zufälliges k, welches relativ prim zu p-1 ist (1 = ggT(k,p-1)) und berechnet die Verschlüsselung wie folgt:

$a = g^k \bmod p$
$b = y^k\ M \bmod p$

wobei a und b den Chiffretext bildet, welcher doppelt so lang ist wie der Klartext. Die Entschlüsselung des Chiffretextes erfolgt dann durch:

$M = b/a^x \bmod p$

3.2.3 Digitale Signatur

Digitalte Signaturen, oder auch elektronische Signaturen sind in der Lage die, im Grundlagen Kryptologie Kapitel, gezeigten Anforderungen (Authentizität, Integrität und Verbindlichkeit) zu erfüllen. Zur Durchführung digitaler Signatur werden asymmetrische Verschlüsselungsverfahren verwendet (z. B. RSA oder ElGamal). Im folgenden Beispiel soll erläutert werden, wie das Verfahren der elektronischen Signatur anzuwenden ist. Angenommen es gibt 2 Teilnehmer die sich Daten senden wollen (in dem Fall Alice und Bob) und beide Parteien verfügen über den jeweils öffentlichen Schlüssel des Gegenübers.
Alice signiert das Dokument mit ihrem privaten Schlüssel K_{PrivA} (aus Perfomancegründen wird lediglich der Hash-Wert (siehe Kapitel Hash-Funktionen) der Nachricht signiert) und verschlüsselt dann die Nachricht und Signatur mit dem öffentlichen Schlüssel von Bob K_{PubB}.

sig = E(K_{privA}, Hash(M))
C = E(K_{PubB}, M + sig))

Der Chiffretext C wird dann zu Bob übertragen, der im ersten Schritt die Nachricht mit seinem privaten Schlüssel K_{privB} entschlüsselt. Anschließend entschlüsselt er die digitale Signatur mit dem öffentlichen Schlüssel von Alice K_{PubA}. Bob errechnet nun aus der bereits entschlüsselten Nachricht den Hash-Wert und prüft den mit den Hash-Wert aus der Signatur - stimmen beide Werte überein ist die Nachricht verifiziert.

M, sig = D(K_{PrivB}, C)
AliceHash(M) = D(K_{PubA}, sig)
AliceHash(M) ?= Hash(M)

Unter der Voraussetzung, dass beide Parteien sicher sind den öffentlichen Schlüssel des gewünschten Partners zu besitzen sorgt dieses Verfahren dafür, dass Bob nach dem Verifizieren der Nachricht zum einen sicher sein kann, dass nur Alice (als alleinige Besitzerin des privaten Schlüssels) die Nachricht unterschrieben hat (Authentifizierung und Verbindlichkeit), zum anderen kann er aufgrund des Hash-Wertes der Nachricht sicher sein, dass die Nachricht bei der Übertragung nicht verfälscht wurde (Integrität).

3.3 Hash-Funktionen

Hash-Funktionen sind mathematische Einweg-Funktionen - das bedeutet, dass ein Wert h = H(M) leicht erzeugt werden kann, jedoch nicht aus h wieder M - es existiert keine Umkehrfunktion. Darüber hinaus ist es praktisch nicht möglich verschiedene Eingabewerte M1, M2, ... zu finden, die den selben Ausgabewert erzeugen H(M) = H(M1). Da Hash-Funktionen eine Nachricht beliebiger Länge auf einer Nachricht fester Länge abbilden ist eine Abbildung auf einen identischen Hash-Wert nicht auszuschließen. Es muss also eine Hash-Größe gewählt werden, sodass es praktisch unmöglich ist alle möglichen Hash-Werte zu berechnen und zu speichern. Ist die Hash-Größe lediglich 64 Bit lang, so gibt es 2^{64} Möglichkeiten einen Hash-Wert. Dem Angreifer reichen jedoch 2^{32} Nachrichten M und den dazugehörigen Hash-Wert h = H(M), sodass die Wahrscheinlichkeit für eine Kollision größer als 0,5 ist. Diese Erkenntnis beruht auf dem

Geburtstags-Paradoxon, welches besagt, dass lediglich 23 Personen in einem Raum genügen, um mit einer Wahrscheinlichkeit größer als 0,5, 2 davon zu finden, die am selben Tag Geburtstag haben. Diese Logik lässt sich auch auf Hash-Funktionen abbilden und kann somit die Komplexität, eine Kollision mit über 50% Wahrscheinlichkeit zu finden, von 2^k auf k * $2^{k/2}$ reduzieren.

3.3.1 MD5 & SHA1

Wie beschrieben sollen Hash-Funktionen eine Größenordnung besitzen, die es heutigen Systemen schwierig macht Kollisionen zu errechnen und zu speichern. MD5 arbeitet mit einer Größe von 128bit und es ist möglich mit nur 2^{64} Schritten eine Kollision zu entdecken (Geburtstags-Paradoxon). Ähnliches Problem zeigt sich bei der Verwendung von SHA1, welches eine Hashwert-Größe von 160 bit hat, also 2^{80} Schritte für eine Kollision. Beide Größenordnung reichen für heutige Systeme nicht aus und sollten deshalb nicht verwendet werden.

3.3.2 SHA224, SHA256, SHA384, SHA512

Alle 4 Hash-Funktionen gehören zur SHA2-Familie und sind von der Vorgehensweise zur Berechnung des Hash-Wertes identisch. Unterschieden sind zwischen SHA256 und SHA512 die Blockgröße (512 Bit, 1024 Bit), Die Anzahl der Wörter (16x32Bit Wörter, 16x64Bit Wörter), sowie die Anzahl der Konstanten (64 Konsanten, 80 Konstanten). Bei den Verfahren SHA224, sowie SHA384 wird jeweils das größere Hash-Verfahren komplett berechnet und die letzten entsprechenden Bits weggelassen. Da sich die Verfahren von Ihrer Funktionsweise nicht unterscheiden, wird hier lediglich SHA256 näher erklärt.

Zu Beginn werden 8 Blöcke je 32 Bit (256 Bit = Hash-Größe) initialisiert (Nachkommastellen der Wurzeln der ersten 8 Primzahlen), sie erhalten die Bezeichnungen a - h. Auf ihnen finden mathematische Funktionen statt. Ausserdem werden 64 Blöcke je 32 Bit mit Rundenkonstanten (Bestimmt aus den Kubikwurzeln der ersten 64 Primzahlen), sie erhalten die Bezeichnung k[i]. Der Klartext wird in 512 Bit große Blöcke unterteilt und, falls erforderlich, am Ende aufgefüllt. Jeder Block wird nun in 16 x 32 Bit Worte aufgesplittet . Diese 16 Worte werden anschließend auf 64 Worte expandiert. Für jedes dieser 64 Worte (im folgenden w[i]) finden nun folgende mathematischen Funktionen statt: (i ist hierbei die Zählervariable der Wörter)

S1 := (e $\gg$ 6) $\oplus$ (e $\gg$ 11) $\oplus$ (e $\gg$ 25)
ch := (e $\wedge$ f) $\oplus$ ($\neg$ e $\wedge$ g)
temp1 := h + S1 + ch + k[i] + w[i]
S0 := (a $\gg$ 2) $\oplus$ (a $\gg$ 13) $\oplus$ (a $\gg$ 22)
maj := (a $\wedge$ b) $\oplus$ (a $\wedge$ c) $\oplus$ (b $\wedge$ c)
temp2 := S0 + maj

Nach dieser Berechnung findet eine Verschiebung der Variablen statt (h=g ; g=f ; f=e ; e=d+temp1 ; d=c ; c=b ; b=a ; a=temp1+temp2).
Nachdem die oben beschriebene Berechnung für alle 64 Runden durchgeführt wurde, werden die entsprechenden Werte (a-h) miteinander konkateniert und ergeben somit den Hash-Wert.

3.3.3 Message Authentification Code

Hash-Funktionen an sich bieten lediglich die Sicherheit der Integrität der Daten, also dass die Daten beim Empfänger unverändert angekommen sind. Über den Ursprung der Daten, also die Authentizität, kann eine Hash-Funktion keine Sicherheit gewährleisten. Um dieses Problem zu beheben gibt es das MAC-Verfahren (Message Authentification Code). MAC-Funktionen verwenden zum errechnen eines Hash-Wertes zusätzlich einen geheimen Schlüssel der beiden Parteien vor der Kommunikation bekannt sein muss. Wird dem Dokument entsprechend ein MAC-Wert angefügt so muss der Empfänger die selbe MAC-Funktion auf das Dokument anwenden und prüfen ob beide MAC-Werte (der selbst errechnete und der zugesandte) übereinstimmen - ist dies der Fall, so ist die Authentizität gewährleistet (sofern sichergestellt ist, dass der Schlüssel geheimgehalten wurde). Dieses Verfahren ist jedoch nicht in der Lage Verbindlichkeit zu gewährleisten (einem dritten glaubwürdig zu machen, wer der Absender ist), da auch der Empfänger in der Lage ist, den entsprechenden MAC zu berechnen.

3.4 Schlüsselvereinbarung

Das Hauptproblem für alle Kryptografischen Ver- und Entschlüsselungsverfahren ist die Vereinbarung eines gemeinsamen Schlüssels. Selbst bei Asymmetrischen Verfahren ist nicht sichergestellt, dass durch einen Man-in-the-middle Angriff ein potentieller Angreifer, seinen eigenen Public-Key in das System schleust und somit über den dazugehörigen Privaten Schlüssel verfügt. Verschiedene Verfahren sollen es ermöglich einen Schlüssel auszutauschen, ohne das ein Angreifer diesen auch erhält.

3.4.1 Diffie Hellmann

Dieses Verfahren beruht auf einfacher mathematischer Potenzierung und Modulo-Rechnung, für den Angreifer besteht jedoch das Problem der diskreten Logarithmen in endlichen Körpern (siehe RSA-Verfahren). Das Problem dieses Verfahrens ist, dass es gegen Man in the middle Agriffe nicht geschützt ist, da ein potentieller Angreifer seine eigenen Potenzen und Modulo-Werte in das System schleusen kann und aufgrund dessen beide Parteien den privaten Schlüssel berechnen. [Schneier 587, Ferguson 211]

3.4.2 Direkte Vereinbarung

QR-Code

PGP-Server

3.5 Zusammenfassung

4 Validierung

Bei der Validierung soll gezeigt werden ob die im vorigen Kapitel als sicher angesehen Verfahren auch auf heutigen Android-Geräten zum Einsatz kommen können. Hierbei sollen die Verfahren im Bezug auf ihre Schlüssellänge, sowie die Dateigröße validiert werden. Die Validierung soll in Bezug auf Geschwindigkeit bzw. Dauer des Verfahrens, Akkuverbrauch und Wärmeentwicklung durchgeführt werden.

Für die Validierung wird ein Alcatel One Touch 997D mit der Android Version 4.1.1 (Jinger Bread) verwendet. Das Gerät verfügt über ein 1GHz Dual Core Prozessor (ARM Corext A9) und 512 MByte RAM, als Stromversorgung ist ein Lithium-Ionen-Akkumulator mit 1800 mAh verbaut. Zu beachten ist, dass die aufgenommen Messwerte stark von der verwendeten Hardware und installierten Software abhängig sind. Ein Vergleich zu anderen Geräten oder dem gleichen Gerät mit verschiedener Software kann nicht durchgeführt werden. Darüber hinaus kann jedoch angenommen werden, da alle Messwerte nahezu selbige Startbedingungen hatten, dass die Differenzen der Verfahren untereinander um einen gewissen Grad der Verschiebung ähnlich sind.

Aus dem Kapitel Grundlagen geht hervor, dass im Symmetrischen Bereich die Verfahren Advanced Encryption Standard (AES) mit den Schlüssellängen 128bit, 192bit und 256bit und Blowfish mit den Schlüssellängen 128bit, 256bit und 446bit Validiert werden muss. Auf beide Verfahren kann man die aufgezählten Betriebsmodi (CBC, OFB, CFB, CTR) verwenden, wodurch Dateien die größer als die Blockgröße verschlüsselt werden knnen. Die Asymmetrischen Verfahren müssen nicht validiert werden, da der Hauptaufwand bei diesen Verfahren auf der Schlüsselgenerierung beruht die lediglich beim ersten Start der Anwendung ausgeführt wird. Als Hash-Funktionen können laut Grundlagen Kapitel lediglich die Verfahren der SHA2-Familie zum Einsatz kommen, dessen Hash-Länge auf die Anwendung angepasst werden kann.

Um die beiden Symmetrischen Verfahren zu Validieren werden jeweils Dateien der Größe 1MB, 5MB und 20MB in zehnfacher Ausführung verschlüsselt und anschließend entschlüsselt. Diese Vorgehensweise wird für alle Schlüssellängen und alle Modi ausgeführt.
Pro Modi fallen so 180 Messungen an (10 Messungen x3 Dateien x3 Schlüssellängen x2 für Ver- und Entschlüsselung). Jede dieser Messung enthält den aktuellen Akkustand, die Temperatur und die Dauer die diese Messung benötigt hat. Eine genaue Aufgliederung aller einzelnen Messergebnisse befindet sich im Anhang. Nach einer erfolgreichen Messreihe eines Modi muss das Smartphone auf den Urzustand gebracht werden, d. h. dass der Akku aufgeladen und die Temperatur normalisiert werden muss. Insgesamt fallen so 1440 Messungen an (180 Messungen x4 Modi x2 Verfahren)

Um die Verfahren mathematisch zu validieren müssen vorerst die Gewichtungen der einzelnen Messfak-

toren dargelegt werden. Der Hauptaspekt für den reibungslosen Ablauf der zu entwickelnden Anwendung ist die Geschwindigkeit, in der die Ver- und Entschlüsselungen durchgeführt werden. Anschließend folgt der Akkuverbrauch und zum Schluss die Temperatur. So ergibt sich nach eigenem Ermessen ein Maßstab von 3 : 2 : 1 für Geschwindigkeit : Akkuverbrauch : Temperatur.

In der Nachfolgenden Tabelle werden alle Verfahren mit Punkten von 1-10, wobei 1 das beste und 10 das schlechteste Ergebnis liefert, versehen und mit dessen Gewichtung multipliziert. Das Ergebnis soll eine Übersicht über alle Faktoren und deren Gewichtung geben und daraus folgern welches Verfahren für die zu entwickelnde Anwendung am besten geeignet ist.

Wie aus den Tabellen zu sehen ist, ist der Counter-Modus bei allen Messungen stets führend. Aus diesem Grund wird dieser Modus mit dem AES Verfahren angewandt. Als Schlüssellänge wird die maximale Länge von 256bit verwendet, da sie sich bei der Messung nur um einigen Sekunden von den geringeren Schlüssellängen unterscheiden.

5 Implementierung

In diesem Kapitel soll auf den internen Aufbau der Anwendung eingegangen werden. Es wird gezeigt, warum welche Verfahren in welcher Form angewendet wurden und welche Strukturen wie zusammen arbeiten.
Die Implementierung der Anwendung erfolgte für Android-Geräte und wurde mit dem Android-Development-Tool (ADT) und der IDE eclipse in der Programmiersprache Java entwickelt.

5.1 Anforderungen

An die zu entwickelnde Anwendung sind verschiedene Anforderungen gestellt, die sowohl beim Entwurf als auch letztendlich bei der Implementierung beachtet werden müssen.
Die Anwendung soll in der ersten Version auf Android-Geräten mit Versionen neuer als 2.3.3 (siehe Kapitel Einleitung) entwickelt werden. Darüber hinaus ist die strikte Vorgabe gegeben, dass die Schlüssel, die für die Ver- und Entschlüsselungen der Dateien zuständig sind, nicht zusammen mit den Daten selbst auf den Server abgelegt werden. Darüber hinaus ist es außerdem vorgegeben, dass die Schlüssel nicht alle auf einer Server-Struktur abgelegt werden und nach Bedarf abgefragt werden - die Schlüssel sollen lediglich den Endgeräten bekannt sein, die nötige Dateien entweder verschlüsseln oder entschlüsseln wollen. Dem Besitzer der Datei ist der Schlüssel bis zu deren Löschung bekannt. Andere Parteien können den Schlüssel halten, müssen es jedoch nicht zwingend.
Als weitere Anforderung ist vorgegeben, dass Verschlüsselungsverfahren oder andere Kryptographische Konzepte, wie Hash-Funktionen oder Schlüsselspeicherung, nicht selbständig implementiert werden sollen. Das Problem bei selbstständiger Implementierung geht ein auf die Probleme von kryptographischen Angriffsmethoden ein. So muss u. a. sichergestellt werden das Nebenhören oder Timing-Angriffe auf z. B. Vergleichsfunktionen oder Angriffe auf Zufallszahlengeneratoren nicht durchführbar sind. Um alle Angriffsszenarien zu untersuchen und deren Lösung zu implementieren erfordert hohe Kenntnisse und ist in dieser Arbeit nicht Erfordert. Um Methoden der Kryptographie einzusetzen soll deshalb in der Arbeit auf standardisierte Bibliotheken, wie derer von Android, zurückgegriffen werden in der Hoffnung, dass die meisten Angriffsszenarien von diesen Bibliotheken bereits abgedeckt werden. Da die Anwendung später für wissenschaftliche Mitarbeiter am DESY eingesetzt werden soll, die nicht unbedingt über Fachkenntnisse der Bereiche Informatik und Kryptographie besitzen, sollen komplexe Strukturen oder komplizierte Vorgehensweisen möglichst einfach und transparent in der Anwendung durchgeführt werden. Es sollen keine Abfragen zur Wahl der Verfahren oder ähnliche fachspezifische Fragen dem Benutzer gestellt werden. Die Anwendung soll intuitiv und einfach zu Bedienen sein. Zusammengefasst lassen sich daraus folgende Muss- und Soll-Kriterien ableiten:

Muss-Kriterien:

- Dateien müssen verschlüsselt werden
- Schlüssel dürfen nur den Endgeräten bekannt sein
- Schlüssel müssen innerhalb der Anwendung sicher abgespeichert werden
- Schlüssel müssen auf sicherem Wege mit anderen Parteien teilbar sein.
- Krypto-Verfahren dürfen nicht selbstständig implementiert werden.

Soll-Kriterien:

- Anwendung soll Komplexität transparent behandeln
- Anwendung soll intuitiv zu bedienen sein

5.2 Entwurf

Da die Anwendung in der Objektorientierten Sprache Java entwickelt wird, werden die einzelnen Klassen die für die Anwendung von nöten sind in verschiedene Pakete unterteilt, die die oberste Struktur der Anwendung widerspiegeln. Das Paket (engl.: package) Activities enthält alle Klassen die in Android zum Anzeigen von Nutzerinhalten von nöten sind. Eine Activity in Android hat einen Lebenszyklus von Create bis Destroy in der Inhalte, die in einer XML-Datei definiert werden, dem Benutzer auf dem Smartphone angezeigt werden.
Darüber hinaus soll es ein package Helper geben, welches Klassen beinhaltet die gewissen Abläufe in einzelne Funktionen zusammenfassen und so die internen komplexen Zusammenhänge zu verschleiern. Des Weiteren sollen die Klassen dafür sorge tragen, dass interne Anbindungen verdeckt werden und für den Nutzer nicht sichtbar sind. So soll z. B. die Klasse DatabaseHelper für das Arbeiten mit der Datenbank helfen, jedoch die Struktur der Datenbank selbst (also deren Tabellen und Spalten) verschleichern. Die Klasse stellt dann nur jene Funktionen zur Verfügung die für den Ablauf der Anwendung von nöten sind.
BroadcastReceiver nennt man in Android Klassen welche die Möglichkeit haben gewisse Aktionen des Gerätes abzufangen und anschließend eigenen Code auszuführen. So kann es z. B. von Interesse sein zu erfahren, dass das Gerät sich mit einem WLAN verbundet hat, oder ein neues Foto geschossen wurde. Um diese Klassen zusammenzufassen wird das Paket BCReceiver erstellt, in dem alle diese Klassen eingebunden sind.
Als letztes Paket wird das Paket External erstellt um hier externe open-source und frei zugängliche Inhalte einzufügen.
Das Zusammenspiel der Pakete soll dafür sorge tragen, dass die einzelnen Activities, wie die MainActivity (das Hauptbild der Anwendung), die nötigen Informatinoen aus den Helper-Klasse abrufen in diese anzeigen. Aktionen durch den Anwender werden entweder von den BroadcastReceivern oder der Activity abgefangen und entsprechend ausgeführt oder an Helper-Klassen weitergegeben. Diese flache Struktur soll das zukünftige Arbeiten erleichtern und einen übersichtlichen und schnellen Einstieg in die Anwendung bieten.

Folgende kurze Übersicht der Pakete und Klassen soll erläutern welche Klassen für welche Funktionalitäten eingesetzt werden sollen:

Activities:

- MainActivity: Das Hauptfenster der Anwendung. Hier sollen alle wichtigen Funktionen zur Verfügung stehen und dem Anwender die möglichkeit bieten auf weitere Activites zu navigieren.
- ServerActivity: Diese Activity soll den aktuellen Inhalt des Servers anzeigen und dem Benutzer die Möglichkeit bieten diesen Inhalt herunterzuladen oder entsprechend mit anderen Benutzern zu teilen
- ProfileActivity: Dieses Fenster soll Informationen zum eigenen Profil enthalten
- ShareActivity: Hier soll dem Benutzer die Möglichkeit gegeben werden, die gewählte Datei mit einem Benutzer zu teilen.

Helper:

- CryptoHelper: Diese Klasse abstrahiert die Methoden zum ver- und entschlüsseln der Daten sowie den Umgang mit Hash-Funktionen oder weiteren Verfahren die im Zusammenhang mit Kryptographie stehen
- DatabaseHelper: Diese Klasse beinhaltet die Struktur der Datenbank und bietet Funktionen zum einfachen arbeiten an
- KeyStoreHelper: Alle Funktionen zum Arbeiten mit einem KeyStore (siehe Bibliotheken) sind hierin enthalten.

Für vorher nicht absehbare Szenarien ist es über die oben angegebene Klassen hinaus möglich weitere Implementierungen durchzuführen. Die oben angegeben Klassen und Pakete stellen lediglich einen geplanten Aufbau dar und sind erweiterbar.

5.3 Programmierschnittstellen

Programmierschnittstellen (engl.: application programming interface, kurz: API) sind Funktionen anderer Anwendungen die zur Verfügung gestellt werden. So ist es z. B. möglich mit eigenem Programm Funktionen einer Datenbankanwendung z. B. SQLite zu nutzen.

5.3.1 KeyStore

KeyStore ist eine von Java zur Verfügung gestellte Schnittstelle zum sicheren sichern von Schlüsseln (engl.: key). Der KeyStore ist eine Datei die auf dem Gerät selbst abgelegt wird. Diese Datei wird dann durch die Schnittstelle mit einem Passwort verschlüsselt. Des Weiteren sorgt das Betriebssystem dafür, dass die Datei nur durch den Anwender, der den KeyStore erstellt hat, und den root-Benutzer zugreifbar ist. Der Aufbau des KeyStores ist eine Key-Value-Struktur. Jeder Schlüssel der zu speichern ist, wird mit einem eindeutigen Key abgelegt, unter dem man später den Schlüssel wieder abrufen kann. Darüber hinaus sorgt die Struktur dafür, dass ohne die nötigen Keys auch die Values verdeckt bleiben - eine Auflistung aller Key-Values ist durch den KeyStore nicht möglich.

5.3.2 SQLite

SQLite ist eine Datenbank, welche in Android standardmäßig zur Verfügung gestellt werden. Mit ihr ist es möglich Daten in einer relationalen Datenbank mit Tabellen und Spalten abzuspeichern. Darüber hinaus können Daten mit sogenannten SQL-Anweisungen (Structured Query Language) wieder abgefragt werden. Android sorgt dafür, dass entsprechende Tabellen die für die Anwendung benötigt werden beim errsten Start der App eingerichtet werden.

5.3.3 zxing

zxing ist eine von Google bereitgestellt API zum Erstellen und Lesen von 1D / 2D Barcodes, u. a. auch für den in der Anwendung genutzte QR-Code.

5.3.4 Bouncy-Castle

Mit dieser Schnittstelle ist es möglich Kryptographische Verfahren zu verwenden. Zwar ist Bouncy-Castle bereits in angepasste Version in Android standardisiert integriert, bietet jedoch in der Version nicht alle durch die eigentliche Schnittstelle angebotenen Funktionen. Mit Boncy-Castle ist es mögliche Funktionen wie das Ver- und Entschlüsseln mit AES, RSA oder anderen Verschlüsselungsverfahren durchzufürehn. Ausserdem sind verschiedene Codefragmente für den Umgang mit Zertifikaten und Schlüsseln bereits vorhanden.

5.4 Programmablauf

In diesem Kapitel soll erläutert werden, welche Abläufe das Programm ausführt, wenn gewisse Interaktionen mit der Anwendung durch den Benutzer ausgeführt werden.

5.4.1 Programmstart

Der Programmstart ist generell zu unterscheiden in zwei Fälle. Zum einen gibt es den intialen Start, also den ersten Start der Anwendung überhaupt und zum anderen gibt es den normalen Start, bei der dir App bereits vorher initialisiert wurde.
In beiden Fällen erscheint zu Beginn der Anwendung eine Meldung, bei der man sein eigenes Master-Passwort eingeben muss. Dieses Passwort schützt die auf dem Smartphone gespeicherten sensiblen Daten indem sie mit diesem Passwort verschlüsselt werden. Beim initialen Start wird dieses Passwort verwendet um einen KeyStore zu erstellen, in dem das eigene RSA KeyPair gespeichert wird. Das KeyPair der größe 1024Bit wird ebenfalls beim ersten Appstart erzeugt und direkt in dem angelegten KeyStore abgespeichert. Darüber hinaus wird eine Datenbank angelegt in der die öffentlichen Schlüssel von Freunden inkl. deren HashWert und einem frei wählbaren Namen abgelegt wird. Diese Informationen werden aus zwei Gründen nicht im KeyStore gespeichert:

- der Zugriff auf den KeyStore dauert länger, als der Zugriff auf die Datenbank, da die Daten im KeyStore stets verschlüsselt gehalten werden.

- im KeyStore ist es lediglich möglich KeyValue-Paare abzulegen, weitere Informationen wie Hash-Werte oder Namen sind nicht speicherbar.

Sind beide Speichermöglichkeiten (KeyStore, Datenbank) erzeugt so ist die Initialisierung der Anwendung abgeschlossen und die Anwendung wechselt in die *MainActivity* - also das Hauptmenü.

Bei erneutem Start der Anwendung sollten die initialen Vorgänge abgeschlossen sein, dies wird überprüft indem zunächst das Master-Passwort abgefragt wird. Mit diesem Passwort wird versucht die vorhandene KeyStore-Datei zu lesen. Erfolgt das Lesen, so ist das Passwort korrekt - ist ein Lesen nicht möglich, so ist das Passwort falsch, in dem Fall wird der Benutzer darauf hingewiesen und muss es erneut versuchen. Erst nach erfolgreichem verbinden mit dem KeyStore wechselt die Anwendung in das Hauptmenü.

5.4.2 Hauptmenü

Von diesem Fenster aus, welches nach dem Initialisierungsvorgang der Anwendung erscheint hat der Benutzer die Möglichkeit die verschiedenen Funktionen innerhalb der Anwendung zu wählen. In einer Liste hat er die Funktionen Server, Einstellungen, Profil und Import - wobei Import nur eine vorübergehende Lösung bietet (siehe Datei teilen). Darüber hinaus findet er in der oberen Titelleiste noch zwei Symbole welche Datei hochladen und Nutzer hinzufügen beschreiben. Durch die Wahl der gewünschten Funktion wird in die entsprechende Activity gewechselt.

5.4.3 Einstellungen

In diesem Fenster werden Grundlegende Einstellung für die Verbindung mit dem Server hinterlegt. Zum einen ist hier die Server-Adresse sichtbar, welche jedoch durch den Benutzer nicht änderbar ist. Darüber hinaus ist es zwingend notwendig, dass der Benutzer in diesem Fenster die für den Serverzugriff nötigen Anmeldeinformationen einträgt.

5.4.4 Server abrufen

Beim Wahl der Funktion Server abrufen wird zunächst geprüft, ob die in den Einstellungen relevanten Benutzername und Passwort-Informationen für den Serverzugriff vorhanden ist - ist dies nicht der Fall erfolgt eine Fehlermeldung und der User wird darauf hingewiesen diese einzutragen. Sind die Daten vorhanden wird eine sichere Verbindung mittels TLS (Transport Layer Security) mit dem Server aufgebaut. Der Server antwortet mit einer Response und liefert eine Liste mit den auf dem Server abgelegten Dateien. Diese Antwort des Servers wird entsprechend der Bedürfnisse angepasst und dem Benutzer in einer Liste dargestellt.
Im weiteren hat der User in diesem Fenster die Möglichkeit durch einen Klick auf eine Datei zu entscheiden ob er diese herunterladen oder mit einem Freund teilen möchte. Durch das Auswählen einer der beiden Möglichkeiten werden die entsprechende Funktionen innerhalb des Programms aufgerufen.

5.4.5 Datei Uploaden

Um eine Datei auf dem Server abzulegen hat der Benutzer 2 Möglichkeiten. Zum einen kann er im Hauptmenü in der Titelleiste das entsprechende Symbol zum hochladen anklicken und wird dann aufgefordert

über einen Dateiexplorer zu der Datei zu navigieren. Die dort angewählte Datei wird entsprechend auf dem Server hochgeladen.
Die andere Variante ist direkt aus dem Context einer Datei selbst, z. B. einem Bild. Hier wählt der Anwender den von Android zur Vergügung gestellten Knopf zum teilen der Datei wählt die Option dCache-Cloud.
In beiden Fällen wird der Pfad zur Datei vom System abgegriffen und an den *CryptoHelper* weitergegeben, der die Datei verschlüsselt. Die Verschlüsselung erfolgt indem zuerst ein Symmetrischer AES-Schlüssel der Länge 256Bit erzeugt wird und mit diesem die Datei im Counter-Modus (CTR) verschlüsselt wird und innerhalb der SD-Karte im Unterordner dCache-Cloud/.enc/ abgelegt wird. Um den entsprechenden Schlüssel zu sichern wird über den Dateinamen ein Hash-Wert (SHA-256) gebildet und beides an den *KeyStoreHelper* weitergegeben. Dieser speichert den Hash-Wert als Key und den Schlüssel als Value in seiner internen Struktur und sichert das ganze mit dem eingehend gewählten Master-Passwort. Der symmetrische Schlüssel kann nun anhand des Hash-Wertes des Dateinamens wieder gefunden werden.

5.4.6 Datei Downloaden

Um eine Datei vom Server zu laden muss der Anwender in der *ServerViewActivity*, das Fenster welches den Inhalt des Servers zeigt, die entsprechende Datei anwählen und im auftretenden Kontextmenü die Option *Download* auswählen.
Die Datei wird dann vom Server heruntergeladen und auf der SD-Karte im Ordner dCachec-Cloud/.enc/ gesichert. Nachdem die Datei erfolgreich heruntergeladen wurde wird über den *KeyStoreHelper* der entsprechende Schlüssel herausgesucht. Dies geschieht indem über den Dateinamen ein Hash-Wert erzeugt wird und dieser im KeysStoreHelper abgefragt wird. Der daraus erlangte SecretKey wird an den *CryptoHelper* weitergeben, der die Datei von der SD-Karte liest und mit dem entsprechenden Schlüssel wieder entschülsselt. Das Ergbnis der Entschlüsselung wird auf der SD-Karte im Ordner dCache-Cloud/ abgelegt.

5.4.7 Schlüsselaustausch

Die generelle Bezeichnung Schlüsselaustausch ist innerhalb der Anwendung differenziert zu betrachten. Auf der einen Seite existieren zu jeder Person privater sowie öffentlicher Schlüssel, wobei der öffentliche Schlüssel mit anderen Personen ausgetauscht werden muss. Zum anderen existiert zu jeder verschlüsselten auf dem Server abgelegten Datei ein symmetrischer AES-Schlüssel, der mit den Personen geteilt werden muss, die Zugang zu der Datei erhalten sollen. (siehe Datei teilen).
Der Austausch des öffentlichen Schlüssels wird über einen Quick-Response-Code (QR-Code) ausgeführt. Hierbei wird der eigene öffentliche Schlüssel kodiert und in Base64-Format (lesbare ASCII-Form) überführt. Dieser String-Text wird dann entsprechend durch den *QRCodeGenerator* in einen 2Dimensionalen QR-Code überführt. Um Fehler oder Manipulation zu vermeiden wird zusätzlich über den erzeugten Schlüssel ein Hash-Wert gebildet und als Fingerprint ausgewiesen.
Die Gegenseite, also der Anwender der den öffentlichen Schlüssel erlangen möchte, kann im Hauptmenü in der Titelleiste den Knopf für *Person hinzufügen* wählen und gelangt dann in den QR-Code-Scanner. Der eingesetzte QR-Code Scanner ist eine externe Anwendung die aufgerufen wird, sofern sie auf dem Smartphone installiert ist. Sollte die Anwendung nicht installiert sein, wird eine Fehlermeldung erzeugt und der Anwender darauf hingewiesen diese Anwendung zu installieren. Das auftretende Fehlerfenster ist

automatisch mit dem Google Play Store verbunden und schlägt bereits eine kostenfreie Anwendung vor. Scannt der Benutzer nun mit der geöffneten Anwendung den Barcode seines gegenübers wird der gescannte Code automatisch in das SecretKey-Format für Android zurückgeführt und der entsprechende Fingerprint errechnet. Ein direkter Abgleich beider Fingerprints stellt sicher, dass eine Manipulation nicht stattgefunden hat. Der Anwender hat nun noch die Möglichkeit seinem jetzt neu hinzugefügten Freund einen Namen zu geben um ihn im späteren Verlauf der Anwendung wieder einwandfrei identifizieren zu können. Um Fehlerquellen zu vermeiden muss dieser gewählte Namen eindeutig sein (doppelte Namensvergabe ist nicht gestattet).

5.4.8 Datei teilen

Die Funktion Datei teilen erlaubt es dem Anwender verschlüsselte auf dem Server abgelegte Dateien mit anderen Personen zu teilen. Um dies durchzuführen ist es notwendig, dass die Person die Zugriff auf die Datei erlangen soll den nötigen AES-Schlüssel, der bei der Generierung erstellt wurde, auf sicherem Wege erhält. Da eine Übertragung über das Internet als unsicherer Kanal gilt muss die Übertragung entsprechend mit digitalen Signaturen und Verschlüsselung geschützt werden. Hierfür wird zuerst die zu übertragende Nachricht erzeugt, indem ein JSON-Objekt erstellt und folgende KeyValue-Paare darin abgelegt werden:

- Dateiname: der Dateinamen der verschlüsselten Datei
- AES-Key: der notwendige SecretKey zum entschlüsseln der Datei
- PublicKey-Hash: Der Hash-Wert des eigenen PublicKeys (= Fingerprint aus Schlüsselaustausch)

Dieses Objekt wird in eine Nachricht umgewandelt, auf die die Hash-Funktion SHA-256 angewandt wird. Der erzeugte Hash-Wert wird mit dem eigenen privaten Schlüssel verschlüsselt und stellt die digitale Signatur dar. Anschließend wird die ursprüngliche Nachricht und die erzeugte Signatur mit dem öffentlichen Schlüssel des Gegenübers verschlüsselt und erzeugt somit die endgültig zu versendende Nachricht. Diese erzeugte Nachricht kann nun über unsichere Kanäle dem Partner übertragen werden. Um zu verifizieren, dass die Nachricht an der Empfängerseite korrekt übertragen wurde wird sie zuerst mit dem eigenen privaten Schlüssel entschlüsselt und erzeugt somit die ursprüngliche Nachricht und die digitale Signatur. Durch das entschlüsseln der digitalen Signatur mit dem öffentlichen Schlüssel des Gegenübers und dem Vergleich des darin enthaltenen Hash-Wertes und dem Hash-Wert der aus der enthaltenen Nachricht selbst erzeugt werden kann ist zum einen sichergestellt, dass die Nachricht nicht verändert wurde und das sie wirklich von der Person stammt, die sie vorgibt zu sein. Ist die Verifizierung abgeschlossen wird der in der Nachricht erhaltende AES-Schlüssel mit Dateinahmen und Hash-Wert des Dateinamens in der internen auf dem Gerät abgelegten Datenbank gespeichert. Beim Download kann der entsprechende Schlüssel dann aus der Datenbank gelesen werden um die Datei zu entschlüsseln.

6 Test

6.1 Performance- und Lasttest

Um eine Anwendung auf ein Smartphone zu bringen, die komplexe Strukturen wie Verschlüsselung oder größere Berechnungen durchführt ist es wichtig, dass diese Anwendung nicht die Fähigkeiten des Gerätes selbst überfordern. So ist es z. B. einem Anwender nicht zumutbar, wenn die Anwendung selbst eine so hohe Prozessorlast erzeugt, dass weitere Aktionen mit dem Smartphone unbrauchbar sind. Außerdem ist eine nicht flüssige Anwendung oder ein extremer Temperaturanstieg durch Rechenleistung nicht zu gebrauchen. Um diese Faktoren bereits vor der Entwicklung der Anwendung sicherzustellen wurde eine Validierung auf rechenintensive Inhalte der Anwendung ausgeführt. Im Kapitel Validierung wurde gezeigt welche Auswirkungen Verschlüsselungsverfahren, die innerhalb der Anwendung verwendet werden, auf einem gewählten Gerät haben. Hierbei wurde der verbrauchte Akkustand, sowie der Temperaturanstieg und die verbrauchte Zeit gemessen. Eines der Hauptaspekte der Anwendung ist die Verschlüsselung selbst, die durch diese Testreihe validiert wurde. Als weiteren Zeitaspekt der Anwendung ist die Dauer die benötigt wird um eine Datei auf den Server zu laden. Da diese Geschwindigkeit jedoch von der Verfügbaren Netzabdeckung, sowie dem vom Benutzer gewählten Vertrag oder der Verbindung zum WLAN abhängt, müssen die Faktoren in der Anwendung nicht getestet werden.
Durch mehrfaches Ausführen vieler Verschlüsselungsverfahren aneinander wurde der Prozessor dauerhaften Berechnungen ausgesetzt und zeigt so die Auswirkung von Verbrauch und Temperatur im Lastzustand. Die im Kapitel Validierung gezeigten Werte geben an, dass bei ungefähr 180 Ver- und Entschlüsselungen von 1-20MB Dateien ein Akkuverbauch von ca. 25% stattfindet und sich die Temperatur auf ein Maximum von 38°C erhöht. Die Zeiten, die für die Verschlüsselung benötigt wurden, liegen innerhalb weniger Sekunden bei kleinen Dateien und bis zu einer halben Minute bei 20MB.

6.2 Funktionstest

6.3 Usabilitytest

Um die im Kapitel Anforderung gestellten Kriterien der leichten Bedienbarkeit und der Intuitivität zu beweisen bedarf es eigener Testszenarien. Diese Eigenschaften einer Anwendung sind jedoch objektiver Natur und können nur von Menschenhand selbst bewertet werden. Da es aufgrund des zeitlichen Rahmens dieser Arbeit nicht möglich war eine Testlauf mit ausgewählten Personen durchzuführen, so soll nach Abschluss der erzeugte Prototyp erstmals Studenten der HTW und ausgewählten Mitarbeitern des DESY an die Hand gegeben werden. Differenziert wird dabei in zwei Personengruppen: Jene die die Anwendung kennen und derer Funktion wissen und jene die ohne jegliche Hintergrundinformationen die Anwendung bedienen sollen. Durch diese Tests soll zum einen gezeigt werden, dass selbst für unwissende Anwender

die App intuitiv zu bedienen ist, als auch durch Wissen über die Funktionsweise der Anwendung dessen Struktur klar und eindeutig darauf verweist welche Fenster in welcher Reihenfolge geöffnet werden müssen um das gewünschte Ziel zu erreichen. Diese Usabilitytests werden dann als Grundlage für weitere Diskussionen oder Änderungen herangezogen um ggf. aufgetretene Mängel zu beseitigen.

6.4 UI-Test

6.5 weitere

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

7.2 Ausblick

7.2.1 Verschlüsselung

7.2.2 GCM

7.2.3 Password-Reset

7.2.4 Schlüsselaustausch