

Implementierung einer Smartphone Anwendung zum Austausch verschlüsselter Daten mit einer Cloud

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

eingereicht am: 10.09.2014

vorgelegt von: Tom Schubert

Studiengang: Angewandte Informatik

Fachbereich: Wirtschaftswissenschaften II

Matrikelnummer: 535279

Erstgutachter: Prof. Dr. Hermann Heßling

Zweitgutachter: Prof. Dr. Christin Schmidt

“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	1
1.1	Motivation	1
1.2	Zielsetzung	2
2	Grundlagen Android	3
2.1	Zusammenhang Kryptographie	4
3	Grundlagen Kryptologie	6
3.1	Symmetrische Verfahren	8
3.1.1	Betriebsmodi	8
3.1.2	DES, 3DES	10
3.1.3	AES	10
3.1.4	ARC4	14
3.1.5	Blowfish	14
3.2	Asymmetrische Verfahren	16
3.2.1	RSA	17
3.2.2	ElGamal	18
3.2.3	Digitale Signatur	18
3.3	Hash-Funktionen	19
3.3.1	MD5 & SHA1	19
3.3.2	SHA224, SHA256, SHA384, SHA512	19
3.3.3	Message Authentication Code	20
3.4	Schlüsselvereinbarung	21
3.4.1	Diffie Hellmann	21
3.4.2	Direkte Vereinbarung	21
3.5	Zusammenfassung	22
4	Validierung	23
4.1	Ergebnisse	24
5	Implementierung	26
5.1	Anforderungen	26
5.2	Entwurf	27
5.3	Programmierschnittstellen	28
5.3.1	KeyStore	28
5.3.2	SQLite	29
5.3.3	zxing	29

5.3.4	Bouncy-Castle	29
5.4	Programmablauf	29
5.4.1	Programmstart	30
5.4.2	Hauptmenü	31
5.4.3	Einstellungen	31
5.4.4	Server abrufen	31
5.4.5	Datei Uploaden	31
5.4.6	Datei Downloaden	32
5.4.7	Schlüsselaustausch	32
5.4.8	Datei teilen	33
6	Test	34
6.1	Performance- und Lasttest	34
6.2	Funktionstest	35
6.3	Usabilitytest	35
7	Problemfälle und Lösungen	36
7.1	Kommunikation	36
7.2	direkter Schlüsselaustausch	36
8	Zusammenfassung und Ausblick	38
8.1	Zusammenfassung	38
8.2	Ausblick	38
8.2.1	Verschlüsselung	38
8.2.2	GCM	39
8.2.3	Password-Reset	40
8.2.4	Schlüsselaustausch	40
9	Schlusswort und Dank	42
10	Anhang: Kryptoverfahren Android Version 2.3.3	I
11	Anhang: Kryptoverfahren Android Version 4.1.1	V
12	Anhang: Kryptoverfahren Android Version 4.4.2	IX
13	Anhang: Kryptoverfahren Android Version 5.0.0	XIV
14	Anhang: Rohdaten Validierung	XIX

Abbildungsverzeichnis

1.1	Smartphonennutzer in Deutschland [Stac]	1
2.1	Anteil d. Android-Versionen [Stab]	3
2.2	Code zur Anzeige von Cryptoverfahren [Staa]	4
2.3	Vorhanden Cryptoverfahren	5
3.1	AES State-Block [PPP09 , S. 100]	11
3.2	AES ShiftRow Verfahren [PPP09 , S. 100]	12
3.3	AES ShiftRow StateMatrix [PPP09 , S. 104]	12
3.4	AES MixColumns [PPP09 , S. 105]	12
3.5	AES Rundenschlüssel [PPP09 , S. 107]	13
3.6	Aufbau Blowfish [Sch96 , S. 389]	15
3.7	Funktion Blowfish [Sch96 , S. 390]	15
4.1	Zeitvergleich Asymmetrischer Verfahren	24
4.2	Auswertung Cryptoverfahren	25
5.1	Sequenzdiagramm	30
8.1	Aufbau GCM [Tam]	39

1 Einleitung

„Die Computer- und Internetnutzer in Deutschland setzen seit Bekanntwerden der geheimdienstlichen Abhöraktionen häufiger Verschlüsselungsverfahren ein.“[[BIT13](#)] Aus dieser Pressemitteilung der BITKOM geht weiterhin hervor, dass von Juli bis 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 Deutschen Elektronen-Synchrotron (DESY) in Hamburg der Fall ist. Auch der Austausch von Daten über mobilen Endgeräten wie Smartphones oder Tables spielt eine immer größere Rolle, wie die Entwicklung der letzten Jahre zeigt (siehe Grafik).

Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2014 (in Millionen)

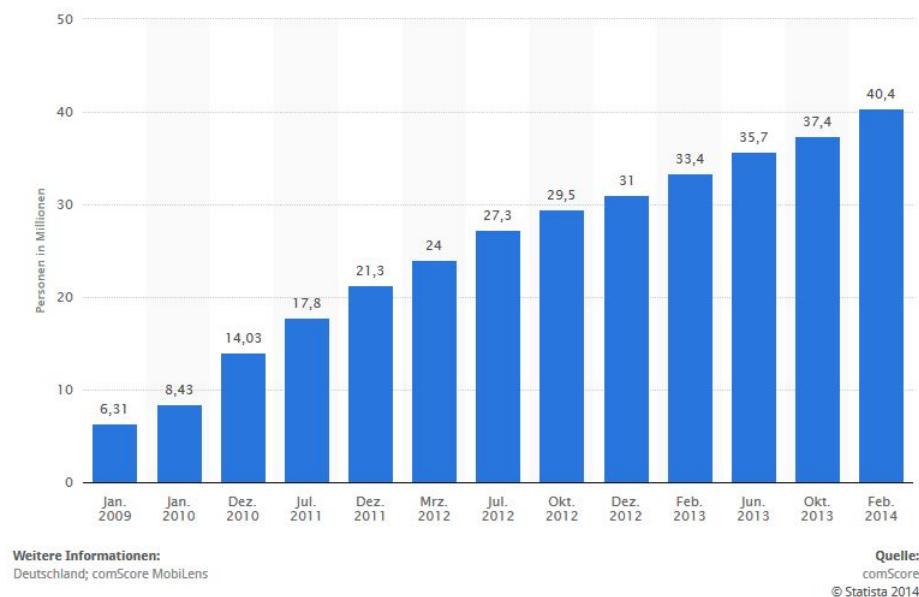


Abbildung 1.1: Smartphonenuutzer in Deutschland [[Stac](#)]

Herkömmliche Verfahren zum Austausch von Daten reichen unter Berücksichtigung des Sicherheitsaspekts oftmals nicht mehr aus.

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 plattformunabhängige Möglichkeit, Dokumente online abzuspeichern und von einem anderen Standort über ein internetfähiges Gerät wieder zu öffnen [Droa]. Auch wenn Dropbox nach eigenen Angaben den Advanced Encryption Standard (AES) verwendet, bevor die Daten gespeichert werden, liegen die dafür notwendigen Schlüssel in den 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 [...]“. [Drob]

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 steuert und verwaltet, ist es dem Anwender möglich, Daten im 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 auf verschiedenen Datenträgern, 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, der andererseits diese Daten auch in angemessener Form (siehe Kapitel Validierung) verschlüsselt.

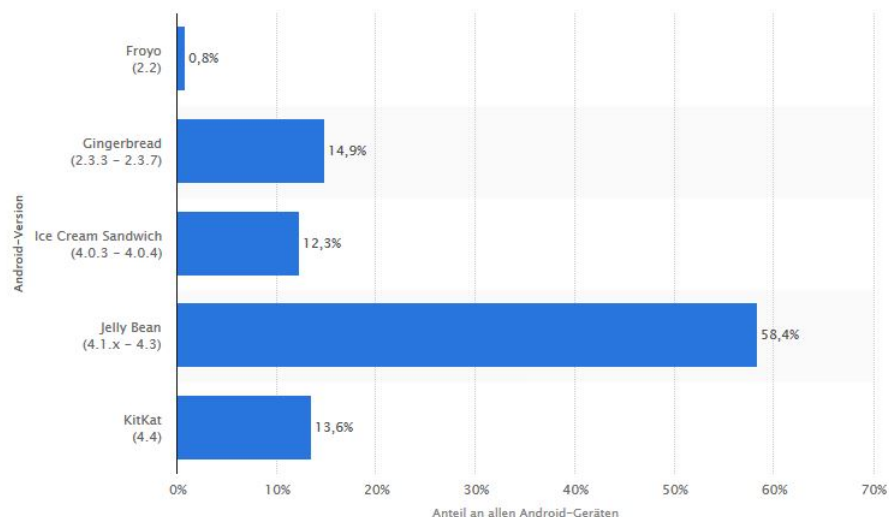
In dieser Arbeit wird die erste Version eines Programmes entwickelt, welches auf Androidbetriebssystemen zum Einsatz kommt. Darüber hinaus ist es wichtig, dass die entsprechenden Schlüssel zum Entschlüsseln der Daten nicht zusammen mit den Daten abgelegt werden, sondern ausschließlich den Parteien des Datenaustauschs bekannt sein sollen. Dies bedeutet, dass selbst die Betreiber am DESY nicht die Möglichkeit haben, die abgelegten Daten zu entschlüsseln.

Zum Ver- und Entschlüsseln 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 überprüft (siehe Kapitel 4 Validierung).

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 [ope]. In diesem Kapitel wird kurz darauf eingegangen, welche kryptographischen 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.

Anteil der verschiedenen Android-Versionen an allen Geräten mit Android OS weltweit im Zeitraum 29. Mai bis 04. Juni 2014



Weitere Informationen:
[Kostenlosen Basis-Account freischalten](#)

Quelle:
[Kostenlosen Basis-Account freischalten](#)
© Statista 2014

Abbildung 2.1: Anteil d. Android-Versionen [Stab]

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 zunächst der Begriff der Sandbox zu nennen. Eine Anwendung wird abgekapselt in jeweils einer eigenen Umgebung mit Prozess, Betriebssystem-User, Dalvik-VM, Heap und eigenem Dateisystem ausgeführt. Dieses abgekapselte Konstrukt wird als Sandbox bezeichnet. Dadurch ist es dem Betriebssystem möglich, Zugriff auf Ressourcen oder andere Programme zu beschränken; hierbei wird das Berechtigung- und Prozess-Management-System von Linux verwendet [BP13, S. 33]. 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, via 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 kryptographische 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 dar und wird in Android zur Verfügung gestellt. Bei 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());
    }
}
```

Abbildung 2.2: Code zur Anzeige von Cryptoverfahren [Staa]

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" liegen die Hauptunterschiede 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:

Verschlüsselung	Hash-Funktionen
AES	MD5
ARC4	SHA1
Blowfish	SHA256
DES	SHA384
3DES	SHA512
RSA	
ElGamal	

Abbildung 2.3: Vorhanden Cryptoverfahren

Die vollständige Liste aller unterstützten 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. Durch die Abkapselung der Verfahren durch das JCE ist es möglich, im Quellcode die Angabe des Providers nicht spezifisch angeben zu müssen, um so das Betriebssystem einen geeigneten Provider wählen zu lassen. Falls es innerhalb eines Providers zu größeren Sicherheitslücken von einem verwendeten Algorithmus kommt, ist es möglich, diesen ohne schwerwiegende Code-Änderungen zu wechseln.

3 Grundlagen Kryptologie

Das Wort Kryptologie stammt aus dem Griechischen *kryptós* für verstecken und *lógos* für die Lehre [Duda, Dudb]. 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 die 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 in verschärfem Sinne 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:

$$E_K(M) = C$$

Um den Chiffretext wieder zu entschlüsseln, wird die umgekehrte Richtung angewandt:

$$D_K(C) = M$$

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

$$D_K(E_K(M)) = M$$

Um einen sicheren Kanal zwischen Sender und Empfänger zu gewährleisten, reicht es nicht, 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. Auf die Authentifizierung folgt 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 wurde.

Verbindlichkeit beschreibt, dass der Sender nicht leugnen kann, dass eine Nachricht gesendet wurde. Dies ist eine Steigerung der Authentifizierung, denn durch Verbindlichkeit ist außerdem gewährleistet, dass der Empfänger einer Nachricht gegenüber einer dritten Person den Absender der Nachricht glaubwürdig machen kann [Sch96, S. 1f].

Kerckhoffs' Maxime

Ein Aspekt in der Kryptographie ist das Kerckhoffs'sche Prinzip, das folgendes aussagt: "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." [FS03, S. 23] 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 nicht bestimmen, welches dieser Verfahren das "bessere" ist, da beider Verfahren Vor- und Nachteile haben. 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 [...]." [Sch96, S. 254f]

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. [Sch96, S. 7] Neben dem chosen-ciphertext-Angriff gibt es weitere Angriffsszenarien auf Kryptosysteme, wie z. B. ciphertext-only, known-plaintext, chosen-plaintext, chosen-key etc. 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 3.4 Schlüsselvereinbarung). Symmetrische Verfahren unterteilt man in zwei Grundtypen, die Block- und die 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 [Sch96, S. 223].

3.1.1 Betriebsmodi

Betriebsmodi sind Verfahren, bei denen 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 Betriebsmodi ist, dass sie die Sicherheit des Kryptoverfahrens nicht beeinträchtigen.

Padding

Bei der Blockchiffrierung ist eine feste 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 dieses Auffüllens, bzw. das Kennzeichen hängt von den verschiedenen Padding-Verfahren ab (z. B. PKCS5, PKCS7 o. ä.).

ECB

ECB (*electronic codebook mode*) ist ein Betriebsmodus, bei dem der Klartext in verschiedene Blöcke der benötigten Länge aufgeteilt wird und jeder dieser Blöcke einzeln verschlüsselt wird. Das Konkatenieren dieser Blöcke ergibt den neuen Chiffretext. Problem bei diesem Verfahren ist, dass 2 gleiche Klartextblöcke auf dem identischen Chiffreblock abgebildet werden, was wiederum für den Angreifer sichtbar und von Nutzen sein kann [Sch96, S. 223ff]. Aus diesem Grund wird ECB als unsicher angesehen und sollte deshalb nicht verwendet werden ("Do not ever use ECB for anything" [FS03, S. 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 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). Weiter besteht die Möglichkeit 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 verändern, besteht die Möglichkeit des sogenannten *random IV*. Der Initialisierungsvektor wird komplett zufällig gewählt und entsprechend der Nachricht angehängen oder vorangestellt, damit beim Entschlüsseln diese Verknüpfung wieder rückgängig gemacht werden kann. [Sch96, S. 227ff]

OFB

Beim OFB (*Output feedback mode*) wird nicht die Klartextnachricht zur Verschlüsselung benutzt, sondern eine Zufallsreihe von Bytes. Die daraus resultierende verschlüsselte Nachricht kann anschließend 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) [Sch96, S. 240ff].

$$K_0 := IV$$

$$K_i := E(K, K_{i-1})$$

$$C_i := P_i \oplus K_i$$

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) [Sch96, S. 235ff].

$$C_0 = P_0 \oplus E_K(IV)$$

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

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

CTR

Im Counter-Modus kann die Berechnung des Schlüssels, wie auch beim OFB, vor dem ersten Klartextblock erfolgen. Innerhalb des Modus gibt es einen internen Zähler, der immer konstant erhöht wird. Aus diesem Zähler und dem Schlüssel wird 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 gewünschte Stelle, erzeugt den Schlüssel und entschlüsselt den Chiffreblock. Wichtig ist, dass der selbe Zähler mit demselben Schlüssel nicht doppelt verwendet werden soll (Problem: zwei identische Klartextblöcke ergeben identischen Chiffretext) [Sch96, S. 243].

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." [FS03, S. 51]

Niels Ferguson weist 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 wird, 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 keines dieser fünf eine Sicherheitsschwäche aufwies. Rijndael konnte letztendlich durch seine einfache Struktur und gute Performance im Software-, sowie im Hardwarebereich überzeugen. [Eck11, S. 343f] Darüber hinaus erlaubt die NSA für die AES-Verschlüsselung von Dokumenten mit dem Sicherheitsstatus TOP SECRET eine Schlüssellänge von 192 bis 256 Bit, für solche mit dem Status SECRET 128 Bit. Dies verdeutlicht, dass selbst Kryptographen von Geheimdiensten diesen Standard als sicher ansehen [PPP09, S. 89].

Funktionsweise

AES arbeitet mit einer Blockgröße von 128 Bit, also 16 Byte, welche intern als 2-zweidimensionale Matrix (4x4) abgespeichert 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}

Abbildung 3.1: AES State-Block [PPP09, S. 100]

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 *ByteSubstitution*, *ShiftRow*, *MixColumns* und *KeyAddition*.

Ausgenommen von dieser Regel ist die letzte Runde, in der *MixColumns* ausgelassen werden. Zusätzlich wird vor der ersten Runde die Funktion *KeyAddition* angewendet [PPP09, S. 89ff].

ByteSubstitution

In der Funktion *ByteSubstitution* wird eines der beiden Verfahren zum Verbergen von Redundanz angewendet - die Konfusion. Sie sorgt dafür, dass der Zusammenhang zwischen Klartext und Chiffre verschleiert und möglichst aus einer kleinen Änderung im Klartext eine große Änderung im Chiffre erzeugt wird. Hierzu wird jedes Byte in eine sogenannte S-Box eingegeben, wobei diese wiederum ein Byte als Ausgabewert hat (dieses Verfahren wird für alle 16 Bytes 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 genau 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 identisch, sodass 16x pro Runde immer die selbe Matrix verwendet wird. Das hat zur Folge, dass sie in den meisten Softwareimplementierungen durch fixe Tabellen realisiert werden, anstatt sie jedes Mal neu zu berechnen [PPP09, S. 101ff].

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, \dots, B_{15} jeweils die Bytes A_0, A_1, \dots, 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).

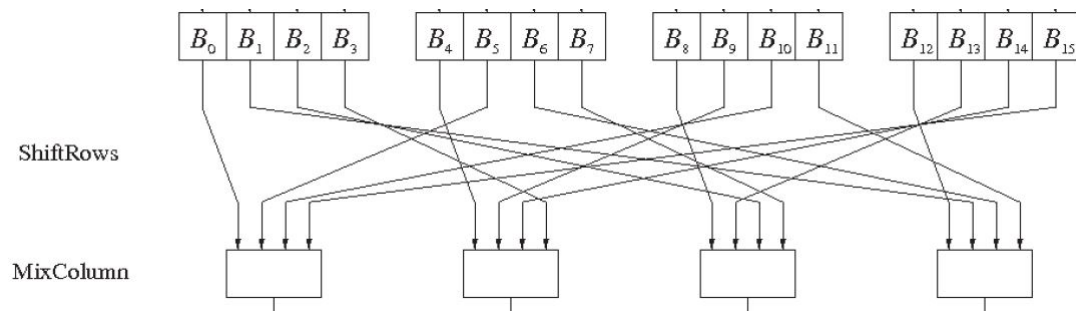


Abbildung 3.2: AES ShiftRow-Verfahren [PPP09, S. 100]

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.

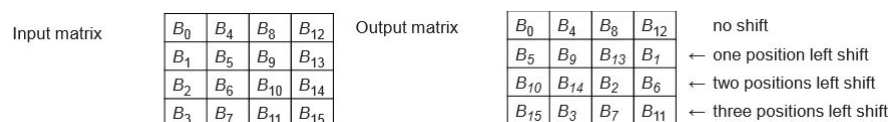


Abbildung 3.3: AES ShiftRow StateMatrix [PPP09, S. 104]

MixColumns

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

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

Abbildung 3.4: AES MixColumns [PPP09, S. 105]

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 möglich in 3 Runden jedes Byte des Klartextes von allen 16 Bytes 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 Runden-schlüssel (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 $W[3] = 4 \times 32 \text{ Bit} = 128 \text{ Bit}$ Schlüssellänge). Der letzte Word-Block einer Runde wird dann durch eine Funktion gegeben und mit den anderen Blöcken XOR-verknüpft.

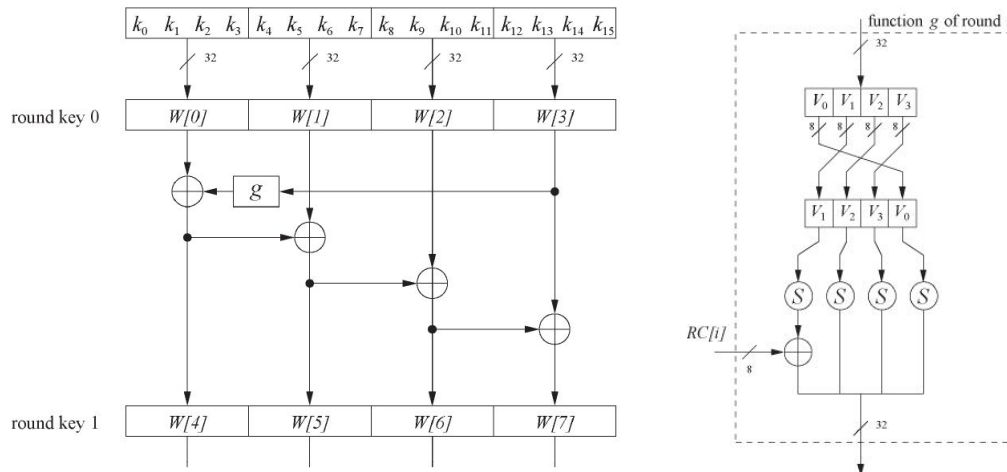


Abbildung 3.5: AES Rundenschlüssel [PPP09, S. 107]

Die Funktion g rotiert hierbei jeweils die Eingabebites 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 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 Verschlüsselung. 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 anschließend zu verwenden. Die Berechnung der Rundenschlüssel selbst ist identisch [PPP09, S. 110ff].

3.1.4 ARC4

RC4 oder auch ARC4 (Arcfour) genannt ist eine Stromverschlüsselung, wird also Bitweise ent- und verschlüsselt [Sch96, S. 455]. 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." [App] 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." [Sch] Darüber hinaus warnen verschiedene Seiten, wie Golem und Heise, die sich mit Informatik beschäftigen, vor der Verwendung von RC4 [Hei, Gol, The]. 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 [fSidI]. 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 sollte: "If people ask, I recommend Twofish instead." [Com] 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 wird der Block in 2 Hälften unterteilt, wobei die eine Teilhälfte immer durch eine Funktion verändert und die andere Teilhälfte in die nächste Runde weitergegeben wird. Dabei werden 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:

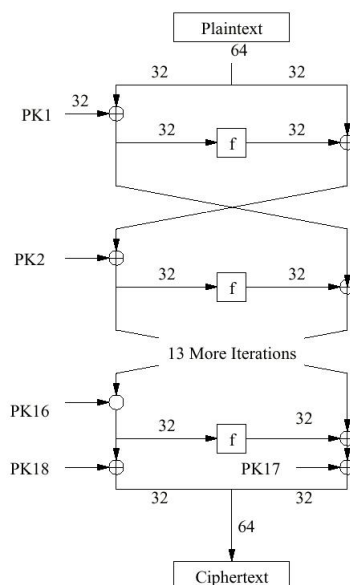


Abbildung 3.6: Aufbau Blowfish [Sch96, S. 389]

Hierbei beschreibt PK_i den Teilschlüssel der aktuellen Runde. Innerhalb der Funktion von Blowfish wird die 32 Bit große Teilhälfte in 4x8 Bit Teilblöcke 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:

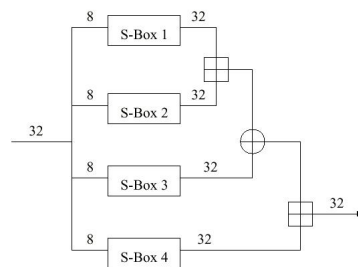


Abbildung 3.7: Funktion Blowfish [Sch96, S. 390]

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 Modulo 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 Teilschlüssel sowie die 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 schließlich insgesamt 4168 Bit Teilschlüsseln besteht [Sch96, S. 390f].

3.2 Asymmetrische Verfahren

Im Gegensatz zu den bisher gezeigten Verschlüsselungsverfahren ist der Ansatz bei asymmetrischen Verfahren ein anderer. Bei symmetrischen Verfahren gibt es einen geheimen Schlüssel, den man sowohl für die Verschlüsselung als auch für die Entschlüsselung verwendet. Die Hauptaufgabe bei diesen Verfahren besteht darin, den geheimen Schlüssel sicher von A nach B zu transportieren, ohne dass ein eventueller Angreifer diesen mitlesen kann. Bei asymmetrischen Verfahren ist der Grundgedanke jener, dass zum Ver- und Entschlüsseln ein jeweils eigener Schlüssel verwendet wird. Der Schlüssel, welcher für die Verschlüsselung zuständig ist, wird Public Key genannt, da er dem Partnern öffentlich zugeteilt werden kann. Das Pendant hierzu ist der private Schlüssel, der im eigenen Besitz bleibt und einzig der Dechiffrierung der verschlüsselten Nachricht dient. Einer der Hauptaufgaben asymmetrischer Verfahren, oder auch Public-Key-Verfahren genannt, besteht darin, 2 Schlüssel zu finden, die jeweils für die Verschlüsselung und Entschlüsselung zu gebrauchen 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* vorstellten [Sch96, S. 525].

3.2.1 RSA

RSA, welches 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 gegebenen, aus zwei Primzahlen errechneten Produkt dessen Faktoren zu bestimmen.

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 $e \in \{1, 2, \dots, \phi(n)-1\}$ und $\text{ggT}(e, \phi(n)) = 1$,
5. berechne d , sodass gilt $d * e \equiv 1 \pmod{\phi(n)}$,
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 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 gemacht 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 = Chiffre bedeutet:

- $C = M^e \pmod{n}$ und
- $M = C^d \pmod{n}$.

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 $M \in \{0, \dots, n-1\}$ 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 aneinandergesetzt.

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, während bei einem großen privaten Schlüssel d die Entschlüsselung entsprechend längere Zeit in Anspruch nimmt. Darüber hinaus ist 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 [PPP09, S. 173ff].

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 [Sch96, S. 543ff].

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 ($1 = \text{ggT}(k, p-1)$) und berechnet die Verschlüsselung wie folgt:

$$\begin{aligned} a &= g^k \bmod p \\ b &= y^k M \bmod p \end{aligned}$$

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

Digitale Signaturen, oder auch elektronische Signaturen sind in der Lage, die im Kryptologie-Grundlagen-Kapitel (vgl. 3), 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 Performancegründen wird lediglich der Hash-Wert (siehe Kapitel Hash-Funktionen) der Nachricht signiert) und verschlüsselt die Nachricht und Signatur mit dem öffentlichen Schlüssel von Bob K_{PubB} .

$$\begin{aligned} \text{sig} &= E(K_{PrivA}, \text{Hash}(M)) \\ C &= E(K_{PubB}, M + \text{sig}) \end{aligned}$$

Der Chiffretext C wird 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.

$$\begin{aligned} M, \text{sig} &= D(K_{PrivB}, C) \\ \text{AliceHash}(M) &= D(K_{PubA}, \text{sig}) \\ \text{AliceHash}(M) &? = \text{Hash}(M) \end{aligned}$$

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) [Eck11, S. 391ff].

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 M_1, M_2, \dots zu finden, die den selben Ausgabewert erzeugen $H(M_1) = H(M_2)$. 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} mögliche Hash-Werte. Dem Angreifer reichen jedoch 2^{32} Nachrichten M und der dazugehörige 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 [Eck11, S. 375ff].

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). Ein 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 [Eck11, S.382ff] [FS03, S. 84ff] [Sch96, S. 498ff, 504ff].

3.3.2 SHA224, SHA256, SHA384, SHA512

Alle 4 Hash-Funktionen gehören zur SHA2-Familie und sind mit der Vorgehensweise zur Berechnung des Hash-Wertes identisch. Unterschiede zwischen SHA256 und SHA512 sind die Blockgröße (512 Bit, 1024 Bit), die Anzahl der Wörter (16x32Bit Wörter, 16x64Bit Wörter) sowie die Anzahl der Konstanten (64 Konstanten, 80 Konstanten). Bei den Verfahren SHA224 sowie SHA384 wird jeweils das größere Hash-Verfahren komplett berechnet, die letzten entsprechenden Bits werden weggelassen. Da sich die Verfahren in 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 initialisiert (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 16×32 -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 ≫ 6) ⊕ (e ≫ 11) ⊕ (e ≫ 25)
ch := (e ∧ f) ⊕ (¬ e ∧ g)
temp1 := h + S1 + ch + k[i] + w[i]
S0 := (a ≫ 2) ⊕ (a ≫ 13) ⊕ (a ≫ 22)
maj := (a ∧ b) ⊕ (a ∧ c) ⊕ (b ∧ 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 [NIS] [FS03, S. 89f].

3.3.3 Message Authentication Code

Hash-Funktionen an sich bieten lediglich die Sicherheit der Integrität der Daten, also dass die Daten beim Empfänger unverändert ankommen. 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 Authentication 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 [FS03, S. 97ff].

3.4 Schlüsselvereinbarung

Das Hauptproblem für alle kryptographischen 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 ermöglichen, einen Schlüssel auszutauschen, ohne dass 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 Angriffe 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 [FS03, S. 211] [Sch96, S. 587].

3.4.2 Direkte Vereinbarung

Das Problem der Schlüsselvereinbarung beruht auf dem Problem des unsicheren Kanals. Findet man einen Weg, den Schlüssel direkt über einen sicheren Kanal übertragen zu können, kann gewährleistet werden, dass Schlüssel bei der Übertragung nicht verfälscht werden.

QR-Code

Eine Variante, Daten direkt und sicher zu übertragen sind QR-Codes, welche zweidimensionale Barcodes darstellen. Dieser Barcode wird auf einem Gerät erzeugt und durch die Kamera eines anderen Geräts gelesen. Der Kanal, der hierbei verwendet wird, ist der Weg der Kamera des Lesegerätes zum Display des Ausgabegerätes. Da dieser Austausch nur stattfinden kann, wenn beide Parteien sich im selben Raum befinden und das Abfangen eines Bildes nicht möglich ist, wird dieser Austausch als sicher angesehen. Einzige Möglichkeit, die Sicherheit des Verfahrens zu umgehen, ist es, das Ausgabegerät zu manipulieren, sodass in der Anwendung an anderer Barcode angezeigt wird.

Die hohe Sicherheit birgt jedoch das Problem, dass dieses Verfahren nicht auf größere Entfernung angewandt werden kann.

PGP-Server

Eine weitere Variante, um Schlüssel auszutauschen sind, sogenannte Pretty good privacy-Server. Auf diesen Schlüsselservern werden öffentliche Schlüssel eines asymmetrischen Verfahrens von Nutzern zur Verfügung gestellt. Ein Anwender kann seinen eigenen Schlüssel auf dem Server ablegen und ihn mit einem Namen versehen. Diesen Namen teilt er seinem Gegenüber mit, der dann vom entsprechenden Server den Schlüssel lädt. Die Sicherheit dieses Verfahrens beruht auf dem Vertrauen und der Sicherheit des Servers.

3.5 Zusammenfassung

Es wurde gezeigt, dass die symmetrischen Verfahren AES und Blowfish sowie die asymmetrischen Verfahren RSA und ElGamal als sicher angesehen werden können. Authentifizierung, Integrität und Verbindlichkeit können durch digitale Signaturen sichergestellt werden. Als Hash-Funktionen können alle Verfahren der SHA2-Familie zum Einsatz kommen.

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 einen 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 müssen. Auf beide Verfahren kann man die aufgezählten Betriebsmodi (CBC, OFB, CFB, CTR) verwenden, wodurch Dateien, die größer als die Blockgröße verschlüsselt werden können. 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 Modus fallen so 180 Messungen an (10 Messungen x3 Dateien x3 Schlüssellängen x2 für Ver- und Entschlüsselung). Jede dieser Messungen 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 Modus muss das Smartphone in den Urzustand gebracht werden, d. h. dass der Akku aufgeladen und die Temperatur normalisiert werden müssen. Insgesamt fallen so 1440 Messungen an (180 Messungen x4 Modi x2 Verfahren).

Um die Verfahren mathematisch zu validieren, müssen vorerst die Gewichtungen der einzelnen Messfaktoren 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.

4.1 Ergebnisse

In der nachfolgenden Tabelle werden alle Verfahren mit Punkten von 1-10, wobei 10 das beste und 1 das schlechteste Ergebnis liefert, versehen und mit dessen Gewichtung multipliziert. Die Verteilung der Werte erfolgt durch die Ergebnisse, welche bei der Messung durchgeführt wurden, dabei werden die Punkte 10 und 9 jeweils nur einmal vergeben und die Punkte 8 - 1 jeweils 3 mal. Exemplarisch ist diese Vorgehensweise in nachfolgender Grafik abgebildet.

Sortiert nach Zeit		
Name	Zeit	Punkte
AES 128 CTR	17,68	10
AES 128 CBC	18,02	9
AES 192 CTR	18,05	8
AES 128 CFB	18,08	8
AES 128 OFB	18,12	8
AES 192 CFB	18,42	7
AES 256 CFB	18,60	7
AES 192 CBC	18,67	7
AES 192 OFB	18,71	6
AES 256 OFB	18,82	6
AES 256 CTR	18,88	6
AES 256 CBC	19,25	5
Blowfish 128 CBC	28,03	5
Blowfish 192 CBC	28,41	5
Blowfish 128 OFB	28,72	4
Blowfish 128 CFB	28,81	4
Blowfish 256 CBC	28,95	4
Blowfish 192 OFB	29,16	3
Blowfish 256 OFB	29,18	3
Blowfish 256 CTR	29,27	3
Blowfish 128 CTR	29,41	2
Blowfish 192 CFB	29,43	2
Blowfish 256 CFB	29,52	2
Blowfish 192 CTR	29,53	1

Abbildung 4.1: Zeitvergleich Asymmetrischer Verfahren

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.

Name	Geschwindigkeit Punkte	Geschwindigkeit Ergebnis (Pkt. x3)	Akkuverbrauch Punkte	Akkuverbrauch Ergebnis (Pkt. x2)	Temperatur Punkte	Temperatur Ergebnis (Pkt. x1)	Ergebnis Gesamt
AES 128 CBC	9	27	3	6	2	2	35
AES 128 OFB	8	24	3	6	1	1	31
AES 128 CFB	8	24	2	4	3	3	31
AES 128 CTR	10	30	5	10	3	3	43
AES 192 CBC	7	21	9	18	4	4	43
AES 192 OFB	6	18	4	8	4	4	30
AES 192 CFB	7	21	4	8	6	6	35
AES 192 CTR	8	24	10	20	7	7	51
AES 256 CBC	5	15	8	16	8	8	39
AES 256 OFB	6	18	8	16	6	6	40
AES 256 CFB	7	21	8	16	8	8	45
AES 256 CTR	6	18	7	14	10	10	42
Blowfish 128 CBC	5	15	2	4	2	2	21
Blowfish 128 OFB	4	12	2	4	4	4	20
Blowfish 128 CFB	4	12	1	2	2	2	16
Blowfish 128 CTR	2	6	3	6	3	3	15
Blowfish 192 CBC	5	15	5	10	5	5	30
Blowfish 192 OFB	3	9	7	14	5	5	28
Blowfish 192 CFB	2	6	6	12	6	6	24
Blowfish 192 CTR	1	3	7	14	5	5	22
Blowfish 256 CBC	4	12	4	8	9	9	29
Blowfish 256 OFB	3	9	6	12	8	8	29
Blowfish 256 CFB	2	6	6	12	7	7	25
Blowfish 256 CTR	3	9	5	10	7	7	26

Abbildung 4.2: Auswertung Cryptoverfahren

Wie in der Tabelle hervorgehoben, zeigt der Counter-Modus (CTR) des AES-Verfahrens mit einer Schlüssellänge von 192Bit beste Ergebnisse. Aus diesem Grund wird dieses Verfahren für die komplette Anwendung verwendet.

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 zusammenarbeiten.

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 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. So muss u. a. sichergestellt werden, dass Nebenhören oder Timing-Angriffe auf z. B. Vergleichsfunktionen oder Angriffe auf Zufallszahlengeneratoren nicht durchführbar sind. Alle Angriffsszenarien zu untersuchen und deren Lösung zu implementieren, erfordert hohe Kenntnisse und ist nicht Thema dieser Arbeit. Um Methoden der Kryptographie einzusetzen, soll deshalb in der Arbeit auf standardisierte Bibliotheken, wie die von Android, zurückgegriffen werden in der Annahme, dass die meisten Angriffsszenarien von diesen Bibliotheken bereits abgedeckt werden. Da die Anwendung später durch wissenschaftliche Mitarbeiter am DESY eingesetzt werden soll, die nicht unbedingt über Fachkenntnisse der Bereiche Informatik und Kryptographie verfügen, sollen komplexe Strukturen oder komplizierte Vorgehensweisen möglichst einfach und transparent in der Anwendung durchgeführt werden. Es sollen dem Benutzer keine Abfragen zur Wahl der Verfahren oder ähnliche fachspezifische Fragen 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 vonnöten 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 vonnöten 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) verschleiern. Die Klasse stellt dann nur jene Funktionen zur Verfügung die für den Ablauf der Anwendung vonnöten 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 verbunden 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 Informationen aus den Helper-Klasse abrufen in diese anzeigen. Aktionen durch den Anwender werden entweder von den `BroadcastReceiver` 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 Activities 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 angegebenen 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 [Ora].

5.3.2 SQLite

SQLite ist eine Datenbank, welche in Android standardmäßig zur Verfügung gestellt wird. 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 ersten Start der App eingerichtet werden [SQL].

5.3.3 zxing

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

5.3.4 Bouncy-Castle

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

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.

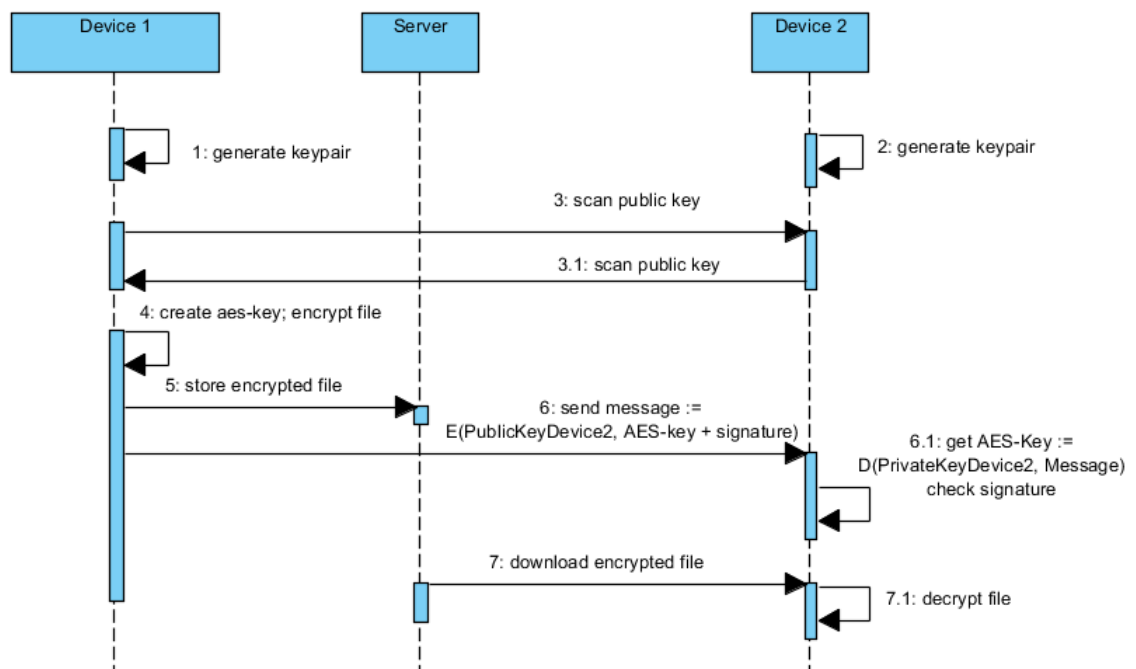


Abbildung 5.1: Sequenzdiagramm

5.4.1 Programmstart

Der Programmstart ist generell zu unterscheiden in zwei Fälle. Der initiale Start, bei dem alle nötigen Elemente angelegt und erstellt werden und den normalen Start, bei der die 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 Key-Value-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 Verbindung 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 Dateexplorer 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 Vergütung 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 192Bit 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 dCache-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 *KeyStoreHelper* abgefragt wird. Der daraus erlangte *SecretKey* wird an den *CryptoHelper* weitergegeben, der die Datei von der SD-Karte liest und mit dem entsprechenden Schlüssel wieder entschlüsselt. Das Ergebnis 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 gegenüber 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 Dateinamen und Hash-Wert des Dateinamens in dem internen auf dem Gerät angelegten KeyStore gespeichert. Beim Download kann der entsprechende Schlüssel dann aus dem KeyStore gelesen werden um die Datei zu entschlüsseln.

6 Test

In diesem Kapitel soll gezeigt werden, wie die entwickelte Anwendung auf Fehlverhalten oder andere Probleme getestet wurde. Die gezeigten Mechanismen sollen Fehler innerhalb der Anwendung aufdecken und auch bei Weiterentwicklung des Systems stets diesen Status behalten. Testfälle sind einzeln konzipierte Abläufe die die Anwendung durchlaufen muss und welche das Verhalten der Anwendung auf Richtigkeit oder Fehlerhaftigkeit hin überprüfen. Da Testfälle vom Entwickler selbst implementiert werden müssen, ist es nicht umsetzbar alle möglichen Kombinationen von Eingaben und Aktionen zu überprüfen. Diese Anwendung gilt als Funktionsfähig, wenn die im Kapitel Programmablauf beschriebenen Fälle ohne Absturz oder andere Mängel ausgeführt werden können.

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 Akkuverbrauch 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

Die Programmiersprache Java ist eine objektorientierte Programmiersprache mit Klassen und deren Methoden. Diese Methoden, welche einen mehr oder minder komplexen Algorithmus abbilden müssen auf ihr richtiges Verhalten hin getestet werden. Diese Tests einzelner Funktionen oder Methoden nennt man Funktionstest und wurden innerhalb der Anwendung mit jUnit ausgeführt. jUnit ist ein Test-Framework zum wiederholten Testen einzelner Funktionen. Der Vorteil an der Wiederholbarkeit besteht darin, dass auch bei einer Änderung der Funktion oder des komplexen ganzen, die Funktion erneut auf Ihr richtiges Verhalten hin getestet werden kann.

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.

7 Problemfälle und Lösungen

Folgend soll kurz beschrieben werden, welche Probleme während der Arbeit auftraten und welche Lösungsstrategien sich überlegt wurden.

7.1 Kommunikation

Während der Entwicklung der Software sind verschiedene Problemfälle aufgetreten, die vorher nicht absehbar und geplant wurden. So ist unter anderem die Kommunikation zweier Geräte miteinander nur über einen eigenen externen Server lösbar. Dieser externe Server handhabt die Nachrichten und merkt sich die für diese Anwendung registrierten Geräte und stellt darüber hinaus Funktionen zur Verfügung, dass Anwender andere Geräte anhand von eigens definierten Kriterien wiederfinden. Der eigene Server selbst kommuniziert dann mit dem von Google bereits gestellten Cloud-Messaging-Server und gibt an ihn die zu übertragende Nachricht und die registrierte ID weiter (näheres unter GCM). Da die Implementierung eines eigenen Servers in der Planung nicht berücksichtigt wurde und der zeitliche Rahmen dieser Arbeit dies nicht zulässt wurde die zu übertragende Nachricht in einem separaten Fenster auf dem Bildschirm ausgegeben und automatisch über einen Knopf in die Zwischenablage kopiert. Der Benutzer hat nun die Möglichkeit den kopierten Text über einen beliebigen Kanal an den Empfänger zu versenden (e-Mail, Whatsapp, Facebook etc.). Der Empfänger kann nun über ein ähnliches Fenster den Text in der Anwendung einfügen, die diesen dann entsprechend verarbeitet. Die Kommunikation zweier Geräte ist in den Anforderungen nicht zwingend gestellt und stellt lediglich einen Weg dar, die Schlüssel auszutauschen.

7.2 direkter Schlüsselaustausch

Um den eigenen öffentlichen Schlüssel seinem Gegenüber zu Übertragen so muss dieser denn im Profil enthaltenden QR-Code scannen. Dieses Verfahren kann während der Übertragung der Daten nicht manipuliert werden, da ein Angreifer sich zwischen Display des Anzeigerätes und Kamera des Empfänger-Gerätes schalten müsste. Ein Problem ergibt sich hieraus, wenn beide Person eine größere Entfernung zueinander haben und diese nicht überbrücken können (Person 1 lebt in Europa, Person 2 in Amerika). Für diesen Problemfall gibt es in der Prototypversion nur eine umständliche Lösung. Eine weitere Möglichkeit wird jedoch im Kapitel Ausblick (siehe Schlüsselaustausch) gegeben. Da in der aktuellen Version der Anwendung der QR-Code die einzige Möglichkeit ist, den entsprechenden Schlüssel auszutauschen müssten beide Parteien den QR-Code auf anderem Wege übertragen um ihn dort zu scannen. Es ist möglich in Android über eine Tastenkombination (Lautstärke Hoch + Power-Knopf) ein Bildschirmfoto zu erstellen. Dieses Foto kann man per unsicheren Kanal übertragen und an Empfängerseite einscannen. Da auf diesem Wege die eingehend beschriebene Sicherheit entfällt ist es zwingend notwendig den zum QR-Code angezeigten Finger-

print genau zu überprüfen um sicherzustellen, dass der QR-Code während der Übertragung über den unsicheren Kanal nicht verändert wurde.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurde ein Prototyp entwickelt, der die gestellten Anforderungen erfüllt. Die Anforderung der leichten Bedienbarkeit können bis zum Abschluss dieser Arbeit nicht verifiziert werden, sollen jedoch direkt im Anschluss durch ein Usabilitytest durchgeführt werden (siehe Kapitel Test). Der Prototyp der entwickelt wurde, ist bisher nur auf dem Testgerät selbst und auf verschiedenen emulierten Geräten zum Einsatz gekommen, zeigt jedoch dort keine Probleme.

8.2 Ausblick

Dieses Kapitel soll auf Grundlage des Prototypen einen Ausblick geben, welche Funktionalitäten in der Anwendung zusätzlich zum Einsatz kommen könnten und welche Verbesserungen dadurch erreicht werden. Alle diese gezeigten Punkten stellen Vorschläge dar, die nach Abschluss mit den verantwortlichen Personen diskutiert werden müssen. Eine Implementierung dieser Punkte ist für die Anwendung nicht zwingend notwendig.

8.2.1 Verschlüsselung

In der aktuellen Version der Anwendung werden zu Beginn der Anwendung ein Schlüsselpaar erzeugt und mit diesem Schlüsselpaar zweier Seiten AES-Schlüssel ausgetauscht, die zum Entschlüsseln der Dateien dienen. Jeder Benutzer der die Schlüssel hat, hat auch die Möglichkeit die ihm zugeteilte Datei weiter zu verbreiten. Eine Variante wäre, dass eine Art Rechtesystem eingeführt wird, in dem nur bestimmte Personen gewisse Dateien weiter teilen dürfen. Der Ursprüngliche Inhaber hat das uneingeschränkte Recht die Datei zu verbreiten, alle anderen Teilnehmer, die Zugriff auf die Datei haben dürfen diese jedoch nur lesen und nicht weiter verbreiten. Darüber hinaus ist der Zugriff auf eine Datei zeitlich uneingeschränkt. Solange die Datei auf dem Server gespeichert ist, kann jeder der den nötigen Schlüssel hält diese Datei auch lesen und ggf. verändern oder weiter teilen. Ein zeitlicher Rahmen, in dem auf die Datei zugegriffen werden kann, könnte realisiert werden. Nach Ablauf dieser Zeit gibt es entweder die Möglichkeit den Schlüssel auf den geteilten Geräten zu entfernen, oder die Datei vom Server zu löschen. Um beide gezeigten Varianten jedoch umsetzen zu können, müsste vorher sichergestellt werden, dass die Datei auf Dateisystemebene nicht zugreifbar ist, da der Anwender sonst die Möglichkeit hat die Datei an einen anderen Ort zu verschieben und erneut selbstständig auf den Server zu laden.

8.2.2 GCM

Google Cloud Messaging (kurz GCM) ist ein System von Google mit dem Entwickler an verschiedene Android-Geräte Push-Nachrichten zu versenden können, dabei ist eine Push-Nachricht eine Meldung von einem Server an ein Gerät, ohne dass das Gerät selbst die Initiative ergreifen muss. Ein Smartphone muss auf diesem Wege nicht dauerhaft prüfen ob neue Inhalte für ihn verfügbar sind, sondern es wird ihm vom Server mitgeteilt.

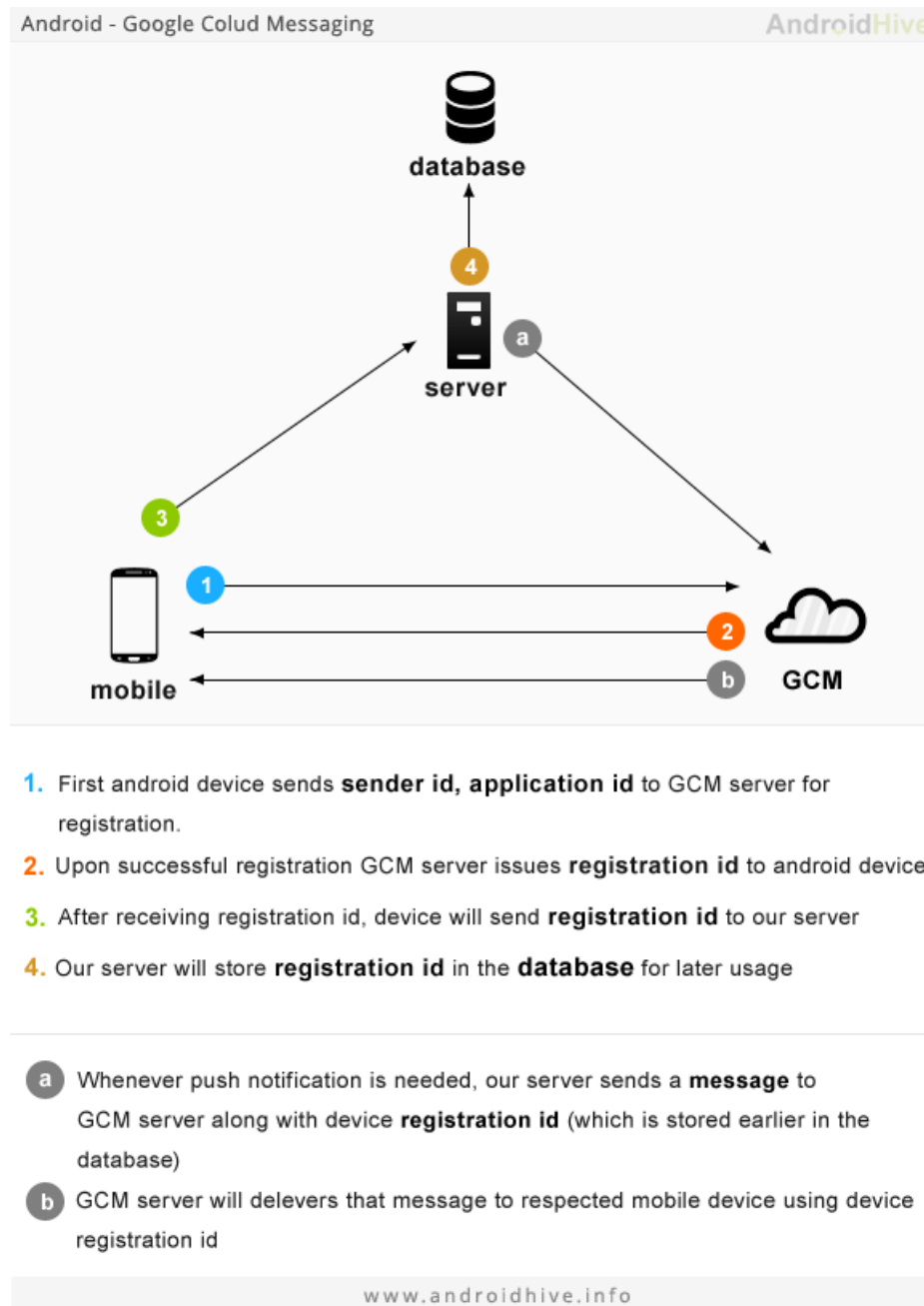


Abbildung 8.1: Aufbau GCM [Tam]

Um Push-Nachrichten nutzen zu können ist es notwendig, dass ein eigener Server vorhanden ist. Zu Beginn registrieren sich die Geräte mit der Anwendung am GCM-Server und erhalten eine eindeutige Registrierungsnummer. Die Nummer identifiziert eindeutig das Gerät und die Anwendung. Diese Nummer muss nun dem eigenen Anwendungsserver mitgeteilt werden, der diese mit weiteren optionalen Informationen wie Name, ID, Telefonnummer oder ähnlichen speichert. Alle Geräte die die Anwendung verwenden registrieren sich so am GCM-Server und speichern ihre erhaltene ID im Anwendungsserver. Sobald eine Kommunikation mit einem anderen Gerät stattfinden soll erstellt der Sender die entsprechende Nachricht und sendet sie mit Empfängerinformationen (Name, ID o. ä.) an den Server. Dieser prüft anhand der Informationen ob ein solcher Benutzer registriert ist, indem er diese in seinem System sucht und prüft ob eine entsprechende Registrierungsnummer vorhanden ist. Anschließend sendet der Server die Nachricht und die Registrierungsnummer an den GCM-Server, der anhand der Registrierungsnummer das Gerät identifizieren kann, welches die Nachricht erhalten soll. Mit diesem System ist sichergestellt, dass ausschließlich der GCM-Server in der Lage ist eine Push-Nachricht an die Geräte zu senden. Die Implementierung eines eigenen Servers ist notwendig um ein System zu schaffen, welches die registrierten Geräte speichert und eine Möglichkeit bietet diese durch eigene Informationen wieder zu finden. In dieser Anwendung wäre das GCM-System denkbar um die verschlüsselte Nachricht, die erzeugt wird um eine Datei zu teilen, an die Empfängerseite zu übertragen [\[Gooa\]](#).

8.2.3 Password-Reset

In der Anwendung werden sensible Daten im KeyStore gespeichert, welcher wiederum im System durch ein Passwort geschützt ist. Sollte es einem Anwender passieren dass er das Master-Passwort, welches zu Beginn der Anwendung gesetzt wird, vergisst, so kann er nicht mehr auf den KeyStore zugreifen. Alle AES-Schlüssel, sowie sein eigener Privater und Öffentlicher Schlüssel sind dann nicht mehr zugreifbar. Einzige Möglichkeit ist das Löschen der KeyStore-Datei um damit die Anwendung dazu zu bringen die Daten entsprechend neu zu initialisieren. Eine Wiederherstellung der Daten ist jedoch nicht möglich. Eine Lösung wäre, die entsprechenden Passwörter der KeyStore-Dateien auf irgendeine Art und Weise gesichert auf einem Server zu hinterlegen, sodass sie bei Verlust wieder hergestellt werden könnten. Ein Angreifer bräuchte dann zum einen Zugriff auf den Server und zum anderen die nötigen KeyStore-Dateien um alle Daten aus dem Gerät auszulesen um die Schlüssel zu erfahren. Eine Speicherung der Passwörter von KeyStore-Dateien würde das System an einer Stelle angreifbar machen und dessen Sicherheit stark beeinflussen.

8.2.4 Schlüsselaustausch

In der aktuellen Version ist es lediglich Möglich seinen eigenen öffentlichen Schlüssel über eine QR-Code anderen Endgeräten mitzuteilen. Wie bereits im Kapitel Problem erläutert, ist dies bei größeren Entfernungen nicht umsetzbar. Um dennoch die Schlüssel austauschen zu können, wäre ein Einsatz von Key-Servern. Auf einem solchen Server können alle Personen ihre öffentlichen Schlüssel hinterlegen und diese anhand eines eindeutigen Strings wiederfinden. Man müsste für einen Austausch der Schlüssel lediglich den eindeutigen String seinem Gegenüber mitteilen, der den öffentlichen Schlüssel dann am Key-Server abrufen kann. Durch einen Fingersprint kann dann geprüft werden, ob der erhaltene Schlüssel tatsächlich jener ist, der zu der Person gehört. Der

Einsatz von Schlüsselserversn erlaubt den Austausch von öffentlichen Schlüsseln über eine größere Entfernung, bietet jedoch nicht die Sicherheit die bei einer direkten Vereinbarung (wie QR-Code) gegeben ist.

9 Schlusswort und Dank

Zu Beginn möchte ich mich bei Prof. Dr. Hermann Heßling bedanken. Er ermöglichte mir durch Zusammenarbeit mit dem DESY Zugriff auf Infrastrukturen, die in dieser Arbeit zum Einsatz kamen. Darüber hinaus möchte ich Ihm für die Zusammenarbeit und die Betreuung während der Arbeit danken, er war bei Fragen und Ideen stets mit hilfreichen Anmerkungen für mich da. Weiterhin danke ich Frau Prof. Dr. Christin Schmidt, die ich durch ihre Herzlichkeit und Freude immer gern bei Fragen kontaktieren konnte. Ich danke beiden für die Betreuung dieser Arbeit und freue mich auch in Zukunft auf gute Zusammenarbeit.

Ganz besonderen Dank gilt jedoch meiner Familie. Meinen Eltern, Carmen und Frank Schubert sowie meinen Großeltern, Ingrid und Egon Bley, die mir nicht nur finanzielle Unterstützung leisteten, sondern auch trotz der großen Entfernung mir immer zur Seite standen, sowohl seelisch als auch psychisch. Ich verdanke Euch die Chance, diesen Schritt im Leben zu gehen und Freue mich, dass Ihr immer für mich da seid.

Als letztes möchte ich allen Freunden danken, die mich während des Studiums und auch während der Arbeit begleiteten und ich freue mich auf noch viele lustige, unterhaltsame Tage und Abende.

Literaturverzeichnis

- [App] Jacob Appelbaum. Rc4 broken. <https://twitter.com/ioerror/status/398059565947699200>. letzter Zugriff: 04.09.2014. 3.1.4
- [BIT13] BITKOM. Nsa-affäre bringt verschlüsselung in mode. [http://www.bitkom.org/files/documents/BITKOM-Presseinfo_Verschluesselung_18_12_2013_v2\(1\).pdf](http://www.bitkom.org/files/documents/BITKOM-Presseinfo_Verschluesselung_18_12_2013_v2(1).pdf), Dezember 2013. letzter Zugriff: 04.09.2014. 1
- [Bou] BouncyCastle. Bouncycastle mainpage. <https://www.bouncycastle.org/>. letzter Zugriff: 04.09.2014. 5.3.4
- [BP13] Arno Becker and Marcus Pant. *Android 4.4: Programmieren für Smartphones und Tablets - Grundlagen und fortgeschrittene Techniken*. dpunkt.verlag GmbH, 3., erw. und akt. aufl. edition, 12 2013. 2.1
- [Com] Computerworld. Bruce almighty: Schneier preaches security to linux faithful. http://www.computerworld.com.au/article/46254/bruce_almighty_schneier_preaches_security_linux_faithful/?pp=3. letzter Zugriff: 04.09.2014. 3.1.5
- [Droa] Dropbox. Dropbox. <https://www.dropbox.com>. letzter Zugriff: 04.09.2014. 1.1
- [Drob] Dropbox. Dropbox hilfe-seite. <https://www.dropbox.com/help/27/de>. letzter Zugriff: 04.09.2014. 1.1
- [Duda] Duden. Krypto. http://www.duden.de/rechtschreibung/krypto_. letzter Zugriff: 04.09.2014. 3
- [Dudb] Duden. Logie. http://www.duden.de/rechtschreibung/_logie. letzter Zugriff: 04.09.2014. 3
- [Eck11] Claudia Eckert. *IT-Sicherheit - Konzepte - Verfahren - Protokolle*. Oldenbourg Verlag, MÄnchen, überarbeitete und erweiterte auflage edition, 2011. 3.1.3, 3.2.3, 3.3, 3.3.1
- [FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography (Computer Science)*. John Wiley Sons, 1. auflage edition, 4 2003. 3, 3.1.1, 3.1.2, 3.3.1, 3.3.2, 3.3.3, 3.4.1
- [fSidI] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren:empfehlungen und schlüssellängen. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2_pdf.pdf?__blob=publicationFile. letzter Zugriff: 04.09.2014. 3.1.4
- [Gol] Golem. Rc4 - schnell, einfach - und unsicher? <http://www.golem.de/news/verschluesselung-was-noch-sicher-ist-1309-101457-2.html>. letzter Zugriff: 04.09.2014. 3.1.4

- [Gooa] Google. Google cloud messaging for android. <https://developer.android.com/google/gcm/index.html>. letzter Zugriff: 04.09.2014. 8.1
- [Goob] Google. zxing. <https://github.com/zxing/zxing/>. letzter Zugriff: 04.09.2014. 5.3.3
- [Hei] Heise. Nsa entschlüsselt webserver-daten angeblich in echtzeit. <http://www.heise.de/newsticker/meldung/NSA-entschluesselt-Webserver-Daten-angeblich-in-Echtzeit-2041383.html>. letzter Zugriff: 04.09.2014. 3.1.4
- [NIS] NIST. Secure hash standard (shs). <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>. letzter Zugriff: 04.09.2014. 3.3.2
- [ope] openhandsetalliance. Openhandsetalliance. <http://www.openhandsetalliance.com/index.html>. letzter Zugriff: 04.09.2014. 2
- [Ora] Oracle. Keystore manual. <http://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html>. letzter Zugriff: 04.09.2014. 5.3.1
- [PPP09] Bart Preneel, Christof Paar, and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer Science Business Media, Berlin Heidelberg, 2009. (document), 3.1.3, 3.1, 3.1, 3.2, 3.3, 3.4, 3.5, 3.5, 3.2.1
- [Sch] Bruce Schneier. The nsa is breaking most encryption on the internet. https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html. letzter Zugriff: 04.09.2014. 3.1.4
- [Sch96] Bruce Schneier. *Angewandte Kryptographie . Protokolle, Algorithmen und Sourcecode in C (Informationssicherheit)*. Addison-Wesley, 5. aufl. edition, 5 1996. (document), 3, 3, 3.1, 3.1.1, 3.1.1, 3.1.1, 3.1.1, 3.1.1, 3.1.4, 3.6, 3.7, 3.1.5, 3.2, 3.2.2, 3.3.1, 3.4.1
- [SQL] SQLite. Sqlite main. <http://www.sqlite.org/>. letzter Zugriff: 04.09.2014. 5.3.2
- [Staa] Stackoverflow. What crypto algoithms does android support. <http://stackoverflow.com/questions/7560974/what-crypto-algroithms-does-android-support>. letzter Zugriff: 04.09.2014. (document), 2.2
- [Stab] Statista. Anteil der verschiedenen android-versionen an allen geräten mit android os weltweit im zeitraum 06. bis 12. august 2014. <http://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/>. letzter Zugriff: 04.09.2014. (document), 2.1
- [Stac] Statista. Anzahl der smartphone-nutzer in deutschland in den jahren 2009 bis 2014 (in millionen). <http://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonennutzer-in-deutschland-seit-2010/>. letzter Zugriff: 04.09.2014. (document), 1.1

- [Tam] Ravi Tamada. Android push notifications using google cloud messaging (gcm), php and mysql. <http://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/>. letzter Zugriff: 04.09.2014. (document), 8.1
- [The] TheRegister. That earth-shattering nsa crypto-cracking: Have spooks smashed rc4? http://www.theregister.co.uk/2013/09/06/nsa_cryptobreaking_bullrun_analysis/. letzter Zugriff: 04.09.2014. 3.1.4

10 Anhang: Kryptoverfahren Android

Version 2.3.3

```

provider: AndroidOpenSSL
  algorithm: SHA-1
  algorithm: SHA-384
  algorithm: SSLv3
  algorithm: SSL
  algorithm: SHA-512
  algorithm: TLSv1
  algorithm: TLS
  algorithm: Default
  algorithm: SHA-256
  algorithm: MD5
provider: DRLCertFactory
  algorithm: X509
provider: BC
  algorithm: PBEWITHSHAAND192BITAES-CBC-BC
  algorithm: SHA-256
  algorithm: PKCS12PBE
  algorithm: DH
  algorithm: DES
  algorithm: PBEWITHHMACSHA
  algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
  algorithm: PBEWITHSHA1ANDRC2
  algorithm: PBEWITHSHA1ANDDES
  algorithm: SHA256WithRSAEncryption
  algorithm: DESEDE
  algorithm: PBEWITHMD5ANDDES
  algorithm: PKIX
  algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
  algorithm: DESEDE
  algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
  algorithm: PBEWITHSHAAND128BITRC2-CBC
  algorithm: PBEWITHSHA1ANDDES
  algorithm: HMACSHA512
  algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
  algorithm: PKIX
  algorithm: PKCS12
  algorithm: BouncyCastle
  algorithm: PBEWITHSHAAND40BITRC2-CBC
  algorithm: PBEWITHSHAAND40BITRC4
  algorithm: DSA
  algorithm: PBEWITHSHAAND128BITRC4
  algorithm: AESWRAP
  algorithm: PBEWITHSHAAND128BITAES-CBC-BC
  algorithm: HMACSHA1
  algorithm: OAEP
  algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
  algorithm: ARC4
  algorithm: AES
  algorithm: DSA
  algorithm: PBEWITHHMACSHA1
  algorithm: Collection
  algorithm: DES
  algorithm: PBKDF2WithHmacSHA1
  algorithm: X.509
  algorithm: SHA512WithRSAEncryption
  algorithm: NONEWITHDSA
  algorithm: PBEWITHSHAAND256BITAES-CBC-BC

```

algorithm: HMACMD5
algorithm: DSA
algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
algorithm: SHA384WithRSAEncryption
algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
algorithm: PBEWITHSHAAND128BITRC4
algorithm: DH
algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
algorithm: BLOWFISH
algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
algorithm: PBEWITHSHAAND128BITRC2-CBC
algorithm: HMACSHA384
algorithm: RSA
algorithm: PBEWITHMD5ANDRC2
algorithm: PBEWITHSHAAND256BITAES-CBC-BC
algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
algorithm: HMACMD5
algorithm: PBEWITHSHAAND40BITRC4
algorithm: SHA-384
algorithm: PBEWITHSHAANDTWOOFISH-CBC
algorithm: PBEWITHMD5ANDDES
algorithm: AES
algorithm: SHA-512
algorithm: PBEWITHSHAANDTWOOFISH-CBC
algorithm: PBEWITHSHAAND40BITRC2-CBC
algorithm: MD5WithRSAEncryption
algorithm: BLOWFISH
algorithm: BKS
algorithm: HMACSHA256
algorithm: RSA
algorithm: DES
algorithm: AES
algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
algorithm: HMACSHA1
algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
algorithm: BLOWFISH
algorithm: HMACSHA256
algorithm: PBEWITHSHAAND192BITAES-CBC-BC
algorithm: DES
algorithm: HMACSHA384
algorithm: SHA1WithRSAEncryption
algorithm: DH
algorithm: DESEDE
algorithm: MD5
algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
algorithm: PBEWITHMD5ANDRC2
algorithm: PBEWITHSHA1ANDRC2
algorithm: DH
algorithm: DESEDEWRAP
algorithm: PBEWITHHMACSHA1
algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
algorithm: PBEWITHSHAAND128BITAES-CBC-BC
algorithm: DH
algorithm: SHA1withDSA
algorithm: SHA-1
algorithm: DESEDE

```
algorithm: DSA
algorithm: RSA
algorithm: RC4
algorithm: HMACSHA512
algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
provider: Crypto
  algorithm: SHA1PRNG
  algorithm: SHA1withDSA
  algorithm: DSA
  algorithm: SHA-1
provider: HarmonyJSSE
  algorithm: X509
  algorithm: SSLv3
  algorithm: SSL
  algorithm: X509
  algorithm: TLS
  algorithm: TLSv1
```

11 Anhang: Kryptoverfahren Android

Version 4.1.1

```

provider: AndroidOpenSSL
  algorithm: SHA-384
  algorithm: RSA
  algorithm: SSLv3
  algorithm: TLSv1.1
  algorithm: TLS
  algorithm: DSA
  algorithm: SHA256WithRSAEncryption
  algorithm: RSA
  algorithm: TLSv1
  algorithm: MD5WithRSAEncryption
  algorithm: SHA512WithRSAEncryption
  algorithm: SHA1WithRSAEncryption
  algorithm: SHA-256
  algorithm: Default
  algorithm: SHA1withDSA
  algorithm: SHA-1
  algorithm: SSL
  algorithm: SHA-512
  algorithm: SHA384WithRSAEncryption
  algorithm: TLSv1.2
  algorithm: MD5
provider: DRLCertFactory
  algorithm: X509
provider: BC
  algorithm: DH
  algorithm: DESEDE
  algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
  algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
  algorithm: RSA
  algorithm: EC
  algorithm: MD5WithRSAEncryption
  algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
  algorithm: HMACSHA384
  algorithm: BLOWFISH
  algorithm: DSA
  algorithm: DESEDE
  algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
  algorithm: HMACSHA384
  algorithm: PBEWITHSHA1ANDRC2
  algorithm: HMACSHA256
  algorithm: Collection
  algorithm: PKIX
  algorithm: EC
  algorithm: DES
  algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
  algorithm: SHA1WithRSAEncryption
  algorithm: AES
  algorithm: PBEWITHSHAAND40BITRC4
  algorithm: HMACMD5
  algorithm: SHA256WithRSAEncryption
  algorithm: PBEWITHSHAAND40BITRC2-CBC
  algorithm: SHA-1
  algorithm: DES
  algorithm: PBEWITHHMACSHA1
  algorithm: PBEWITHSHAAND128BITAES-CBC-BC
  algorithm: BKS

```

algorithm: SHA256WITHECDSA
algorithm: SHA384WithRSAEncryption
algorithm: OAEP
algorithm: X.509
algorithm: DH
algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
algorithm: PBEWITHHMACSHA
algorithm: HMACSHA512
algorithm: PBEWITHSHAAND128BITRC2-CBC
algorithm: PBEWITHMD5ANDRC2
algorithm: SHA-256
algorithm: PBKDF2WithHmacSHA1
algorithm: DSA
algorithm: PBEWITHSHA1ANDRC2
algorithm: DSA
algorithm: BouncyCastle
algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
algorithm: PBEWITHMD5ANDDES
algorithm: SHA-512
algorithm: MD5
algorithm: RSA
algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
algorithm: AES
algorithm: PBEWITHSHAAND256BITAES-CBC-BC
algorithm: PBEWITHSHAAND192BITAES-CBC-BC
algorithm: DH
algorithm: ARC4
algorithm: DESEDE
algorithm: PBEWITHSHAANDTWOOFISH-CBC
algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
algorithm: DES
algorithm: BLOWFISH
algorithm: HMACSHA1
algorithm: DESEDE
algorithm: PKCS12PBE
algorithm: PBEWITHMD5ANDDES
algorithm: AESWRAP
algorithm: PBEWITHSHAAND128BITAES-CBC-BC
algorithm: PBEWITHSHAANDTWOOFISH-CBC
algorithm: ECDH
algorithm: PBEWITHSHA1ANDDES
algorithm: AES
algorithm: DES
algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
algorithm: ECDSA
algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
algorithm: PBEWITHHMACSHA1
algorithm: RSA
algorithm: HMACSHA256
algorithm: ARC4
algorithm: DESEDEWRAP
algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
algorithm: PBEWITHSHAAND40BITRC2-CBC
algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
algorithm: PBEWITHSHAAND128BITRC2-CBC

algorithm: HMACMD5
algorithm: NONEWITHDSA
algorithm: DH
algorithm: PBEWITHSHAAND40BITRC4
algorithm: DSA
algorithm: SHA-384
algorithm: PKCS12
algorithm: SHA384WITHECDSA
algorithm: PBEWITHSHAAND128BITRC4
algorithm: BLOWFISH
algorithm: SHA1withDSA
algorithm: PBEWITHSHAAND256BITAES-CBC-BC
algorithm: PBEWITHMD5ANDRC2
algorithm: PKIX
algorithm: SHA512WITHECDSA
algorithm: NONEwithECDSA
algorithm: HMACSHA512
algorithm: DH
algorithm: PBEWITHSHAAND128BITRC4
algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
algorithm: SHA512WithRSAEncryption
algorithm: PBEWITHSHAAND192BITAES-CBC-BC
algorithm: HMACSHA1
algorithm: PBEWITHSHA1ANDDES
provider: Crypto
algorithm: DSA
algorithm: SHA1withDSA
algorithm: SHA-1
algorithm: SHA1PRNG
provider: HarmonyJSSE
algorithm: AndroidCAStore
algorithm: TLSv1
algorithm: SSLv3
algorithm: X509
algorithm: SSL
algorithm: TLS
algorithm: X509

12 Anhang: Kryptoverfahren Android

Version 4.4.2

```

provider: AndroidOpenSSL
type: SSLContext algorithm: SSL
type: SSLContext algorithm: SSLv3
type: SSLContext algorithm: TLS
type: SSLContext algorithm: TLSv1
type: SSLContext algorithm: TLSv1.1
type: SSLContext algorithm: TLSv1.2
type: SSLContext algorithm: Default
type: MessageDigest algorithm: SHA-1
type: MessageDigest algorithm: SHA-256
type: MessageDigest algorithm: SHA-384
type: MessageDigest algorithm: SHA-512
type: MessageDigest algorithm: MD5
type: KeyPairGenerator algorithm: RSA
type: KeyPairGenerator algorithm: DSA
type: KeyPairGenerator algorithm: EC
type: KeyFactory algorithm: RSA
type: KeyFactory algorithm: DSA
type: KeyFactory algorithm: EC
type: KeyAgreement algorithm: ECDH
type: Signature algorithm: MD5WithRSA
type: Signature algorithm: SHA1WithRSA
type: Signature algorithm: SHA256WithRSA
type: Signature algorithm: SHA384WithRSA
type: Signature algorithm: SHA512WithRSA
type: Signature algorithm: SHA1withDSA
type: Signature algorithm: NONEwithRSA
type: Signature algorithm: ECDSA
type: Signature algorithm: SHA256withECDSA
type: Signature algorithm: SHA384withECDSA
type: Signature algorithm: SHA512withECDSA
type: SecureRandom algorithm: SHA1PRNG
type: Cipher algorithm: RSA/ECB/NoPadding
type: Cipher algorithm: RSA/ECB/PKCS1Padding
type: Cipher algorithm: AES/ECB/NoPadding
type: Cipher algorithm: AES/ECB/PKCS5Padding
type: Cipher algorithm: AES/CBC/NoPadding
type: Cipher algorithm: AES/CBC/PKCS5Padding
type: Cipher algorithm: AES/CFB/NoPadding
type: Cipher algorithm: AES/CFB/PKCS5Padding
type: Cipher algorithm: AES/CTR/NoPadding
type: Cipher algorithm: AES/CTR/PKCS5Padding
type: Cipher algorithm: AES/OFB/NoPadding
type: Cipher algorithm: AES/OFB/PKCS5Padding
type: Cipher algorithm: DESEDE/CBC/NoPadding
type: Cipher algorithm: DESEDE/CBC/PKCS5Padding
type: Cipher algorithm: DESEDE/CFB/NoPadding
type: Cipher algorithm: DESEDE/CFB/PKCS5Padding
type: Cipher algorithm: DESEDE/ECB/NoPadding
type: Cipher algorithm: DESEDE/ECB/PKCS5Padding
type: Cipher algorithm: DESEDE/OFB/NoPadding
type: Cipher algorithm: DESEDE/OFB/PKCS5Padding
type: Cipher algorithm: ARC4
type: Mac algorithm: HmacMD5
type: Mac algorithm: HmacSHA1
type: Mac algorithm: HmacSHA256
type: Mac algorithm: HmacSHA384

```

type: Mac algorithm: HmacSHA512
 type: CertificateFactory algorithm: X509
 provider: DRLCertFactory
 type: CertificateFactory algorithm: X509
 provider: BC
 type: MessageDigest algorithm: MD5
 type: Mac algorithm: HMACMD5
 type: KeyGenerator algorithm: HMACMD5
 type: MessageDigest algorithm: SHA-1
 type: Mac algorithm: HMACSHA1
 type: KeyGenerator algorithm: HMACSHA1
 type: Mac algorithm: PBEWITHHMACSHA
 type: Mac algorithm: PBEWITHHMACSHA1
 type: SecretKeyFactory algorithm: PBEWITHHMACSHA1
 type: SecretKeyFactory algorithm: PBKDF2WithHmacSHA1
 type: SecretKeyFactory algorithm: PBKDF2WithHmacSHA1And8BIT
 type: MessageDigest algorithm: SHA-256
 type: Mac algorithm: HMACSHA256
 type: KeyGenerator algorithm: HMACSHA256
 type: MessageDigest algorithm: SHA-384
 type: Mac algorithm: HMACSHA384
 type: KeyGenerator algorithm: HMACSHA384
 type: MessageDigest algorithm: SHA-512
 type: Mac algorithm: HMACSHA512
 type: KeyGenerator algorithm: HMACSHA512
 type: AlgorithmParameters algorithm: PKCS12PBE
 type: AlgorithmParameters algorithm: AES
 type: Cipher algorithm: AES
 type: Cipher algorithm: AESWRAP
 type: KeyGenerator algorithm: AES
 type: Cipher algorithm: PBEWITHSHAAND128BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHAAND192BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHAAND256BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
 type: Cipher algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
 type: Cipher algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND192BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND256BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHA256AND256BITAES-CBC-BC

type: Cipher algorithm: ARC4
 type: KeyGenerator algorithm: ARC4
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITRC4
 type: SecretKeyFactory algorithm: PBEWITHSHAAND40BITRC4
 type: Cipher algorithm: PBEWITHSHAAND128BITRC4
 type: Cipher algorithm: PBEWITHSHAAND40BITRC4
 type: Cipher algorithm: BLOWFISH
 type: KeyGenerator algorithm: BLOWFISH
 type: AlgorithmParameters algorithm: BLOWFISH
 type: Cipher algorithm: DES
 type: KeyGenerator algorithm: DES
 type: SecretKeyFactory algorithm: DES
 type: AlgorithmParameters algorithm: DES
 type: Cipher algorithm: PBEWITHMD5ANDDES
 type: Cipher algorithm: PBEWITHSHA1ANDDES
 type: SecretKeyFactory algorithm: PBEWITHMD5ANDDES
 type: SecretKeyFactory algorithm: PBEWITHSHA1ANDDES
 type: Cipher algorithm: DESEDE
 type: Cipher algorithm: DESEDEWRAP
 type: KeyGenerator algorithm: DESEDE
 type: AlgorithmParameters algorithm: DESEDE
 type: SecretKeyFactory algorithm: DESEDE
 type: Cipher algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
 type: Cipher algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHMD5ANDRC2
 type: SecretKeyFactory algorithm: PBEWITHSHA1ANDRC2
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITRC2-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND40BITRC2-CBC
 type: Cipher algorithm: PBEWITHMD5ANDRC2
 type: Cipher algorithm: PBEWITHSHA1ANDRC2
 type: Cipher algorithm: PBEWITHSHAAND128BITRC2-CBC
 type: Cipher algorithm: PBEWITHSHAAND40BITRC2-CBC
 type: Cipher algorithm: PBEWITHSHAANDTWOOFISH-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAANDTWOOFISH-CBC
 type: CertificateFactory algorithm: X.509
 type: AlgorithmParameters algorithm: DSA
 type: AlgorithmParameterGenerator algorithm: DSA
 type: KeyPairGenerator algorithm: DSA
 type: KeyFactory algorithm: DSA
 type: Signature algorithm: SHA1withDSA
 type: Signature algorithm: NONEWITHDSA
 type: KeyPairGenerator algorithm: DH
 type: KeyAgreement algorithm: DH
 type: KeyFactory algorithm: DH
 type: AlgorithmParameters algorithm: DH
 type: AlgorithmParameterGenerator algorithm: DH
 type: KeyAgreement algorithm: ECDH
 type: KeyFactory algorithm: EC
 type: KeyPairGenerator algorithm: EC
 type: Signature algorithm: ECDSA
 type: Signature algorithm: NONEwithECDSA
 type: Signature algorithm: SHA256WITHECDSA
 type: Signature algorithm: SHA384WITHECDSA

type: Signature algorithm: SHA512WITHECDSA
type: AlgorithmParameters algorithm: OAEP
type: Cipher algorithm: RSA
type: KeyFactory algorithm: RSA
type: KeyPairGenerator algorithm: RSA
type: Signature algorithm: MD5WITHRSA
type: Signature algorithm: SHA1WITHRSA
type: Signature algorithm: SHA256WITHRSA
type: Signature algorithm: SHA384WITHRSA
type: Signature algorithm: SHA512WITHRSA
type: KeyStore algorithm: BKS
type: KeyStore algorithm: BouncyCastle
type: KeyStore algorithm: PKCS12
type: CertPathValidator algorithm: PKIX
type: CertPathBuilder algorithm: PKIX
type: CertStore algorithm: Collection
provider: Crypto
type: MessageDigest algorithm: SHA-1
type: SecureRandom algorithm: SHA1PRNG
type: Signature algorithm: SHA1withDSA
type: KeyFactory algorithm: DSA
provider: HarmonyJSSE
type: SSLContext algorithm: SSL
type: SSLContext algorithm: SSLv3
type: SSLContext algorithm: TLS
type: SSLContext algorithm: TLSv1
type: KeyManagerFactory algorithm: PKIX
type: TrustManagerFactory algorithm: PKIX
type: KeyStore algorithm: AndroidCAStore
provider: AndroidKeyStore
type: KeyStore algorithm: AndroidKeyStore
type: KeyPairGenerator algorithm: RSA

13 Anhang: Kryptoverfahren Android

Version 5.0.0

```

provider: AndroidOpenSSL
type: SSLContext algorithm: SSL
type: SSLContext algorithm: SSLv3
type: SSLContext algorithm: TLS
type: SSLContext algorithm: TLSv1
type: SSLContext algorithm: TLSv1.1
type: SSLContext algorithm: TLSv1.2
type: SSLContext algorithm: Default
type: MessageDigest algorithm: SHA-1
type: MessageDigest algorithm: SHA-224
type: MessageDigest algorithm: SHA-256
type: MessageDigest algorithm: SHA-384
type: MessageDigest algorithm: SHA-512
type: MessageDigest algorithm: MD5
type: KeyPairGenerator algorithm: RSA
type: KeyPairGenerator algorithm: DH
type: KeyPairGenerator algorithm: DSA
type: KeyPairGenerator algorithm: EC
type: KeyFactory algorithm: RSA
type: KeyFactory algorithm: DH
type: KeyFactory algorithm: DSA
type: KeyFactory algorithm: EC
type: KeyAgreement algorithm: ECDH
type: Signature algorithm: MD5WithRSA
type: Signature algorithm: SHA1WithRSA
type: Signature algorithm: SHA224WithRSA
type: Signature algorithm: SHA256WithRSA
type: Signature algorithm: SHA384WithRSA
type: Signature algorithm: SHA512WithRSA
type: Signature algorithm: SHA1withDSA
type: Signature algorithm: NONEwithRSA
type: Signature algorithm: ECDSA
type: Signature algorithm: SHA224withECDSA
type: Signature algorithm: SHA256withECDSA
type: Signature algorithm: SHA384withECDSA
type: Signature algorithm: SHA512withECDSA
type: SecureRandom algorithm: SHA1PRNG
type: Cipher algorithm: RSA/ECB/NoPadding
type: Cipher algorithm: RSA/ECB/PKCS1Padding
type: Cipher algorithm: AES/ECB/NoPadding
type: Cipher algorithm: AES/ECB/PKCS5Padding
type: Cipher algorithm: AES/CBC/NoPadding
type: Cipher algorithm: AES/CBC/PKCS5Padding
type: Cipher algorithm: AES/CFB/NoPadding
type: Cipher algorithm: AES/CTR/NoPadding
type: Cipher algorithm: AES/OFB/NoPadding
type: Cipher algorithm: DESEDE/ECB/NoPadding
type: Cipher algorithm: DESEDE/ECB/PKCS5Padding
type: Cipher algorithm: DESEDE/CBC/NoPadding
type: Cipher algorithm: DESEDE/CBC/PKCS5Padding
type: Cipher algorithm: DESEDE/CFB/NoPadding
type: Cipher algorithm: DESEDE/OFB/NoPadding
type: Cipher algorithm: ARC4
type: Mac algorithm: HmacMD5
type: Mac algorithm: HmacSHA1
type: Mac algorithm: HmacSHA224
type: Mac algorithm: HmacSHA256

```


type: Mac algorithm: HmacSHA384
 type: Mac algorithm: HmacSHA512
 type: CertificateFactory algorithm: X509
 provider: BC
 type: MessageDigest algorithm: MD5
 type: Mac algorithm: HMACMD5
 type: KeyGenerator algorithm: HMACMD5
 type: MessageDigest algorithm: SHA-1
 type: Mac algorithm: HMACSHA1
 type: KeyGenerator algorithm: HMACSHA1
 type: Mac algorithm: PBEWITHHMACSHA
 type: Mac algorithm: PBEWITHHMACSHA1
 type: SecretKeyFactory algorithm: PBEWITHHMACSHA1
 type: SecretKeyFactory algorithm: PBKDF2WithHmacSHA1
 type: SecretKeyFactory algorithm: PBKDF2WithHmacSHA1And8BIT
 type: MessageDigest algorithm: SHA-224
 type: Mac algorithm: HMACSHA224
 type: KeyGenerator algorithm: HMACSHA224
 type: MessageDigest algorithm: SHA-256
 type: Mac algorithm: HMACSHA256
 type: KeyGenerator algorithm: HMACSHA256
 type: MessageDigest algorithm: SHA-384
 type: Mac algorithm: HMACSHA384
 type: KeyGenerator algorithm: HMACSHA384
 type: MessageDigest algorithm: SHA-512
 type: Mac algorithm: HMACSHA512
 type: KeyGenerator algorithm: HMACSHA512
 type: AlgorithmParameters algorithm: PKCS12PBE
 type: AlgorithmParameters algorithm: AES
 type: AlgorithmParameters algorithm: GCM
 type: Cipher algorithm: AES
 type: Cipher algorithm: AESWRAP
 type: Cipher algorithm: GCM
 type: KeyGenerator algorithm: AES
 type: Cipher algorithm: PBEWITHSHAAND128BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHAAND192BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHAAND256BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND128BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND192BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHSHA256AND256BITAES-CBC-BC
 type: Cipher algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
 type: Cipher algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
 type: Cipher algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND128BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND192BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHMD5AND256BITAES-CBC-OPENSSL
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND192BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND256BITAES-CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHA256AND128BITAES-CBC-BC

type: SecretKeyFactory algorithm: PBEWITHSHA256AND192BITAES-
 CBC-BC
 type: SecretKeyFactory algorithm: PBEWITHSHA256AND256BITAES-
 CBC-BC
 type: Cipher algorithm: ARC4
 type: KeyGenerator algorithm: ARC4
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITRC4
 type: SecretKeyFactory algorithm: PBEWITHSHAAND40BITRC4
 type: Cipher algorithm: PBEWITHSHAAND128BITRC4
 type: Cipher algorithm: PBEWITHSHAAND40BITRC4
 type: Cipher algorithm: BLOWFISH
 type: KeyGenerator algorithm: BLOWFISH
 type: AlgorithmParameters algorithm: BLOWFISH
 type: Cipher algorithm: DES
 type: KeyGenerator algorithm: DES
 type: SecretKeyFactory algorithm: DES
 type: AlgorithmParameters algorithm: DES
 type: Cipher algorithm: PBEWITHMD5ANDDES
 type: Cipher algorithm: PBEWITHSHA1ANDDES
 type: SecretKeyFactory algorithm: PBEWITHMD5ANDDES
 type: SecretKeyFactory algorithm: PBEWITHSHA1ANDDES
 type: Cipher algorithm: DESEDE
 type: Cipher algorithm: DESEDEWRAP
 type: KeyGenerator algorithm: DESEDE
 type: AlgorithmParameters algorithm: DESEDE
 type: SecretKeyFactory algorithm: DESEDE
 type: Cipher algorithm: PBEWITHSHAAND3-KEYTRIPLEDES-CBC
 type: Cipher algorithm: PBEWITHSHAAND2-KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND3-
 KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND2-
 KEYTRIPLEDES-CBC
 type: SecretKeyFactory algorithm: PBEWITHMD5ANDRC2
 type: SecretKeyFactory algorithm: PBEWITHSHA1ANDRC2
 type: SecretKeyFactory algorithm: PBEWITHSHAAND128BITRC2-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAAND40BITRC2-CBC
 type: Cipher algorithm: PBEWITHMD5ANDRC2
 type: Cipher algorithm: PBEWITHSHA1ANDRC2
 type: Cipher algorithm: PBEWITHSHAAND128BITRC2-CBC
 type: Cipher algorithm: PBEWITHSHAAND40BITRC2-CBC
 type: Cipher algorithm: PBEWITHSHAANDTWOOFISH-CBC
 type: SecretKeyFactory algorithm: PBEWITHSHAANDTWOOFISH-CBC
 type: CertificateFactory algorithm: X.509
 type: AlgorithmParameters algorithm: DSA
 type: AlgorithmParameterGenerator algorithm: DSA
 type: KeyPairGenerator algorithm: DSA
 type: KeyFactory algorithm: DSA
 type: Signature algorithm: SHA1withDSA
 type: Signature algorithm: NONEWITHDSA
 type: Signature algorithm: SHA224WITHDSA
 type: Signature algorithm: SHA256WITHDSA
 type: KeyPairGenerator algorithm: DH
 type: KeyAgreement algorithm: DH
 type: KeyFactory algorithm: DH
 type: AlgorithmParameters algorithm: DH
 type: AlgorithmParameterGenerator algorithm: DH
 type: KeyAgreement algorithm: ECDH

type: KeyFactory algorithm: EC
type: KeyPairGenerator algorithm: EC
type: Signature algorithm: ECDSA
type: Signature algorithm: NONEwithECDSA
type: Signature algorithm: SHA224WITHECDSA
type: Signature algorithm: SHA256WITHECDSA
type: Signature algorithm: SHA384WITHECDSA
type: Signature algorithm: SHA512WITHECDSA
type: AlgorithmParameters algorithm: OAEP
type: Cipher algorithm: RSA
type: KeyFactory algorithm: RSA
type: KeyPairGenerator algorithm: RSA
type: Signature algorithm: MD5WITHRSA
type: Signature algorithm: SHA1WITHRSA
type: Signature algorithm: SHA224WITHRSA
type: Signature algorithm: SHA256WITHRSA
type: Signature algorithm: SHA384WITHRSA
type: Signature algorithm: SHA512WITHRSA
type: KeyStore algorithm: BKS
type: KeyStore algorithm: BouncyCastle
type: KeyStore algorithm: PKCS12
type: CertPathValidator algorithm: PKIX
type: CertPathBuilder algorithm: PKIX
type: CertStore algorithm: Collection
provider: Crypto
type: SecureRandom algorithm: SHA1PRNG
provider: HarmonyJSSE
type: KeyManagerFactory algorithm: PKIX
type: TrustManagerFactory algorithm: PKIX
type: KeyStore algorithm: AndroidCAStore
provider: AndroidKeyStore
type: KeyStore algorithm: AndroidKeyStore
type: KeyPairGenerator algorithm: RSA

14 Anhang: Rohdaten Validierung

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

5MB: Blowfish448/OFB/NoPadding ENC : time=19117 ; acc=72,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=19358 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=20154 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=19506 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=19727 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=19836 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=19680 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding ENC : time=20593 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14049 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=13930 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14474 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14254 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14712 ; acc=71,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14077 ; acc=70,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14857 ; acc=70,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14302 ; acc=70,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=15213 ; acc=70,000000 ; temp=33,000000
5MB: Blowfish448/OFB/NoPadding DEC : time=14238 ; acc=70,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=75561 ; acc=70,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=76816 ; acc=69,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=75766 ; acc=69,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=75634 ; acc=69,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=76575 ; acc=68,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=77588 ; acc=68,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=76283 ; acc=68,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=76680 ; acc=67,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=75881 ; acc=67,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding ENC : time=77280 ; acc=66,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57165 ; acc=66,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57368 ; acc=66,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=56066 ; acc=66,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57953 ; acc=65,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57022 ; acc=65,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=58520 ; acc=65,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57254 ; acc=64,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=57209 ; acc=64,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=60051 ; acc=64,000000 ; temp=33,000000
20MB: Blowfish448/OFB/NoPadding DEC : time=58535 ; acc=64,000000 ; temp=33,000000

[illegible]

[illegible]

5MB: AES256/CBC/PKCS7Padding ENC : time=12065 ; acc=78,000000 ; temp=37,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=15150 ; acc=78,000000 ; temp=37,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=12027 ; acc=78,000000 ; temp=37,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=12512 ; acc=78,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=11758 ; acc=78,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=11812 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=10669 ; acc=77,000000 ; temp=37,000000
5MB: AES256/CBC/PKCS7Padding ENC : time=11884 ; acc=77,000000 ; temp=37,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=11636 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10245 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=12139 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10215 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10762 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10497 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10514 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=12334 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10199 ; acc=77,000000 ; temp=38,000000
5MB: AES256/CBC/PKCS7Padding DEC : time=10418 ; acc=77,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=46321 ; acc=76,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=46026 ; acc=76,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=46515 ; acc=76,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=45841 ; acc=75,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=44881 ; acc=75,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=45543 ; acc=74,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=45410 ; acc=74,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=46536 ; acc=74,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=44221 ; acc=74,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding ENC : time=45044 ; acc=73,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42118 ; acc=73,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42814 ; acc=73,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42932 ; acc=72,000000 ; temp=38,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=43280 ; acc=72,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42411 ; acc=72,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=43309 ; acc=71,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42133 ; acc=71,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42623 ; acc=71,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42545 ; acc=70,000000 ; temp=37,000000
20MB: AES256/CBC/PKCS7Padding DEC : time=42376 ; acc=70,000000 ; temp=37,000000

[illegible]

[illegible]

5MB: AES256/CTR/NoPadding ENC : time=10653 ; acc=82,000000 ; temp=37,000000
5MB: AES256/CTR/NoPadding ENC : time=11469 ; acc=82,000000 ; temp=36,000000
5MB: AES256/CTR/NoPadding ENC : time=10929 ; acc=82,000000 ; temp=36,000000
5MB: AES256/CTR/NoPadding ENC : time=11946 ; acc=82,000000 ; temp=36,000000
5MB: AES256/CTR/NoPadding ENC : time=11911 ; acc=82,000000 ; temp=36,000000
5MB: AES256/CTR/NoPadding ENC : time=12641 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding ENC : time=10955 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding ENC : time=11108 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10942 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10401 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10545 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=11338 ; acc=82,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10633 ; acc=81,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10495 ; acc=81,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10300 ; acc=81,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10611 ; acc=81,000000 ; temp=35,000000
5MB: AES256/CTR/NoPadding DEC : time=10379 ; acc=81,000000 ; temp=34,000000
5MB: AES256/CTR/NoPadding DEC : time=10304 ; acc=81,000000 ; temp=34,000000
20MB: AES256/CTR/NoPadding ENC : time=42880 ; acc=81,000000 ; temp=34,000000
20MB: AES256/CTR/NoPadding ENC : time=43554 ; acc=80,000000 ; temp=34,000000
20MB: AES256/CTR/NoPadding ENC : time=43432 ; acc=80,000000 ; temp=34,000000
20MB: AES256/CTR/NoPadding ENC : time=46259 ; acc=80,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=45283 ; acc=79,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=43045 ; acc=79,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=46593 ; acc=79,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=44498 ; acc=79,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=45899 ; acc=78,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding ENC : time=44567 ; acc=78,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=44178 ; acc=78,000000 ; temp=32,000000
20MB: AES256/CTR/NoPadding DEC : time=44784 ; acc=77,000000 ; temp=32,000000
20MB: AES256/CTR/NoPadding DEC : time=42483 ; acc=77,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=42959 ; acc=76,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=42680 ; acc=76,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=41589 ; acc=76,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=41383 ; acc=76,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=41162 ; acc=75,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=40515 ; acc=75,000000 ; temp=33,000000
20MB: AES256/CTR/NoPadding DEC : time=42126 ; acc=75,000000 ; temp=33,000000

[illegible]

[illegible]

5MB: Blowfish448/CTR/NoPadding ENC : time=18907 ; acc=75,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=18983 ; acc=75,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=19115 ; acc=75,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=19068 ; acc=75,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=18932 ; acc=75,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=19405 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=18958 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding ENC : time=18903 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14760 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14885 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14980 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=15186 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=15270 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14414 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14234 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14811 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14434 ; acc=74,000000 ; temp=33,000000
5MB: Blowfish448/CTR/NoPadding DEC : time=14259 ; acc=74,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=76406 ; acc=73,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=79406 ; acc=73,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=77417 ; acc=72,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=79548 ; acc=72,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=78567 ; acc=72,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=78033 ; acc=71,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=78739 ; acc=71,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=77295 ; acc=70,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=76336 ; acc=70,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding ENC : time=76375 ; acc=70,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=59066 ; acc=69,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=57387 ; acc=69,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=57647 ; acc=69,000000 ; temp=34,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=56985 ; acc=68,000000 ; temp=34,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=57089 ; acc=68,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=58656 ; acc=68,000000 ; temp=33,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=57609 ; acc=68,000000 ; temp=34,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=56603 ; acc=67,000000 ; temp=34,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=56078 ; acc=67,000000 ; temp=34,000000
20MB: Blowfish448/CTR/NoPadding DEC : time=56283 ; acc=67,000000 ; temp=34,000000

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

5MB: AES256/CFB/NoPadding ENC : time=11157 ; acc=75,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11421 ; acc=75,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11001 ; acc=75,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11923 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11297 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11854 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11359 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding ENC : time=11054 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10301 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10311 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10398 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10607 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10665 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10450 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10275 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10622 ; acc=74,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10417 ; acc=73,000000 ; temp=35,000000
5MB: AES256/CFB/NoPadding DEC : time=10533 ; acc=73,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding ENC : time=43776 ; acc=73,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding ENC : time=44340 ; acc=73,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding ENC : time=45622 ; acc=72,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=43763 ; acc=72,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=44045 ; acc=72,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=44037 ; acc=72,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=44363 ; acc=71,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=43917 ; acc=71,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=43453 ; acc=71,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding ENC : time=43718 ; acc=70,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding DEC : time=41063 ; acc=70,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding DEC : time=42765 ; acc=70,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding DEC : time=40819 ; acc=69,000000 ; temp=36,000000
20MB: AES256/CFB/NoPadding DEC : time=40763 ; acc=69,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=40896 ; acc=69,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=41825 ; acc=68,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=41459 ; acc=68,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=42357 ; acc=68,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=41185 ; acc=67,000000 ; temp=35,000000
20MB: AES256/CFB/NoPadding DEC : time=41119 ; acc=67,000000 ; temp=35,000000

[illegible]

[illegible]

5MB: AES256/OFB/NoPadding ENC : time=11030 ; acc=72,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11381 ; acc=72,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11106 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11141 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11266 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11074 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=11130 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding ENC : time=12047 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10739 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10326 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=11074 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10370 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=11187 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10472 ; acc=71,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10876 ; acc=70,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10566 ; acc=70,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=11067 ; acc=70,000000 ; temp=36,000000
5MB: AES256/OFB/NoPadding DEC : time=10907 ; acc=70,000000 ; temp=36,000000
20MB: AES256/OFB/NoPadding ENC : time=42893 ; acc=70,000000 ; temp=36,000000
20MB: AES256/OFB/NoPadding ENC : time=43876 ; acc=70,000000 ; temp=36,000000
20MB: AES256/OFB/NoPadding ENC : time=44544 ; acc=69,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=43726 ; acc=69,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=43255 ; acc=69,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=43709 ; acc=68,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=45128 ; acc=68,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=43083 ; acc=68,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=44464 ; acc=67,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding ENC : time=43907 ; acc=67,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=43790 ; acc=67,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=44550 ; acc=66,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42812 ; acc=66,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42999 ; acc=66,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42555 ; acc=65,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=41935 ; acc=65,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42369 ; acc=65,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=41469 ; acc=65,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42924 ; acc=64,000000 ; temp=37,000000
20MB: AES256/OFB/NoPadding DEC : time=42494 ; acc=64,000000 ; temp=37,000000