Bundesamt
für Sicherheit in der
Informationstechnik

Nutzung von OpenPGP auf Android

Eine Anforderungsanalyse und Studie vorhandener OpenPGP-Implementierungen

Autoren

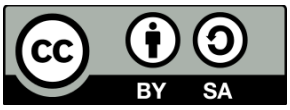
Vincent Breitmoser
OpenKeychain

Dominik Schürmann
Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig
OpenKeychain

Unter Mitwirkung von:

Bernhard Reiter
Emanuel Schütze
Intevation GmbH
Neuer Graben 17
49074 Osnabrück
https://intevation.de

Werner Koch
g10 code GmbH
Hüttenstr. 61
40699 Erkrath
https://g10code.com



Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: bsi@bsi.bund.de
Internet: https://www.bsi.bund.de

Änderungshistorie

Version	Datum	Name	Beschreibung
1.0	11.5.2016	siehe Autoren	Initiale Version für die Veröffentlichung

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Das OpenPGP-Protokoll ist zum aktuellen Zeitpunkt eine stabile Lösung für Ende-zu-Ende-verschlüsselte Kommunikation, welche insbesondere im privaten Sektor verbreitet ist. Eine Verfügbarkeit von Tools ist dabei auf allen gängigen Desktop-Betriebssystemen (Windows, GNU/Linux, Mac OS) bereits gegeben. In den letzten Jahren hat sich für den Erfolg eines Kommunikations-Protokolls jedoch die Verfügbarkeit auf mobilen Endgeräten zur zentralen Anforderung entwickelt. Aus dieser Beobachtung lässt sich die Notwendigkeit entsprechender Software für das Android-Betriebssystem folgern, welches seit einigen Jahren die am weitesten verbreitete Plattform für mobile Endgeräte ist.

1.2 Ziel der Studie

In der vorliegenden Studie soll untersucht werden, inwieweit eine Ende-zu-Ende-Verschlüsselung, mit OpenPGP auf mobilen Plattformen, insbesondere Android, bereits möglich ist und welche Schritte erforderlich sind, um diese zu verbessern.

Hierfür werden zunächst Anforderungen formuliert, welche eine OpenPGP-Implementierung für mobile Endgeräte erfüllen sollte (Kapitel 2), und diese für den konkreten Fall der Android-Plattform ausgearbeitet (Kapitel 3). Dabei wird die Annahme von Desktop-Implementierungen übernommen, dass eine zentrale Komponente (als App) für die Verwaltung von privaten und öffentlichen OpenPGP-Schlüsseln sowie die Bereitstellung darauf aufbauender kryptografischer Methoden zuständig ist. Im Folgenden werden aktuell verfügbare Apps, die OpenPGP-Funktionalität bieten, untersucht (Kapitel 4) und in Hinblick auf die erarbeiteten Anforderungen bewertet (Kapitel 5). Es folgt eine kurze Betrachtung von E-Mail- und Instant-Messaging-Apps, die bereits eigenständig oder durch Nutzung einer OpenPGP-App eine Unterstützung von OpenPGP in Kombination mit ihrem spezifischen Protokoll bieten (Kapitel 6). Im Kapitel 7 wird auf die Vorteile und Machbarkeit einer tieferen Integration von OpenPGP in das Android-Betriebssystem eingegangen. Aufbauend auf diesen Erkenntnissen wird abschließend eine Empfehlung für die Weiterentwicklung von OpenPGP auf Android gegeben (Kapitel 8).

1.3 Ergebnisse

Die Studie hat gezeigt, dass die im Android-Framework verfügbaren Mechanismen zur Interprozess-Kommunikation für die Umsetzung einer App zur zentralen Bereitstellung von OpenPGP-Funktionalität gut geeignet sind.

Unter den aktuell verfügbaren Apps, welche das Grundkonzept einer zentralen OpenPGP-App umsetzen, sind vor allem OpenKeychain und GnuPG for Android von Interesse. GnuPG for Android ist eine Portierung von GnuPG als App, deren Entwicklung allerdings zum Zeitpunkt der Untersuchung in einem frühen Stadium stehen geblieben ist. Dennoch lässt sich an dieser App der technische Ansatz einer Portierung nachvollziehen und evaluieren. OpenKeychain ist eine eigenständige Implementierung, die kryptografisch auf der Bouncy-Castle-Bibliothek aufsetzt und zum Zeitpunkt der Studie die vollständigste Lösung für die Verwendung von OpenPGP darstellt. Neben der Verwaltung von Schlüsseln stellt OpenKeychain eine API zur Verfügung, welche in einer Anzahl von Kommunikations-Apps bereits Verwendung findet.

Eine Unterstützung von OpenPGP ist in einigen Kommunikations-Apps, speziell E-Mail und Instant-Messaging, als ergänzendes Feature für das entsprechende Protokoll vorhanden. Zum aktuellen Zeitpunkt existiert allerdings für keines dieser Nutzungsszenarien eine ausgereifte, umfassende Lösung.

Letztlich werden Empfehlungen für eine Weiterentwicklung von OpenPGP unter Android gegeben. Diese Empfehlungen basieren auf der Weiterentwicklung von OpenKeychain unter Betrachtung der vorher evaluierten Anforderungen und bieten dabei unterschiedliche Aufwands-Nutzen-Abschätzungen.

2 Anforderungen für eine sichere Implementierung und Nutzung von OpenPGP auf mobilen Geräten

In diesem Kapitel werden eine Anzahl von funktionalen (F) und nichtfunktionalen (NF) Anforderungen definiert. Dabei beschreiben funktionale Anforderungen klar abgegrenzte Funktionalitäten, die von einer App unterstützt werden. Nichtfunktionale Anforderungen hingegen beschreiben qualitative Eigenschaften der Implementierung, beispielsweise erhöhte Sicherheitsanforderungen an einzelne Aspekte.

Die hier eingeführten Anforderungen sind für eine OpenPGP-Implementierung unabhängig vom eingesetzten Betriebssystemen allgemeingültig. Erst im folgenden Kapitel werden sie näher beschrieben und im konkreten Android-Anwendungsszenario betrachtet.

Die funktionalen Anforderungen, die an eine OpenPGP-App gestellt werden, lassen sich aufteilen in drei Kerngruppen:

1. Die Verwaltung der privaten Schlüssel des Nutzers (**F1**),
2. Die Verwaltung einer Liste der öffentlichen Schlüssel von Kommunikationspartnern (**F2**) sowie
3. Die Bereitstellung der von OpenPGP unterstützten kryptografischen Operationen für andere Apps über eine Programmierschnittstelle (**F3**).

Eine weitere Gruppe, die allerdings optional ist, beinhaltet die Bereitstellung eigenständiger kryptografischer Operationen (**F4**). Es folgen zwei Gruppen von nichtfunktionalen Anforderungen, aufgeteilt in Sicherheitseigenschaften (**NF5**) und weitere Anforderungen (**NF6**).

2.1 Verwaltung privater Schlüssel (F1)

Die Verwaltung privater Schlüssel lässt sich weiter aufspalten in die Erzeugung (**F1.1**), Veränderung (**F1.2**) sowie die Sicherung und Wiederherstellung (**F1.3**) einzelner privater Schlüssel des Nutzers.

2.2 Verwaltung öffentlicher Schlüssel (F2)

Die Verwaltung öffentlicher Schlüssel von Kommunikationspartnern ist die häufigste direkte Interaktion des Nutzers mit einer OpenPGP-App. Die wesentlichen Vorgänge sind dabei die Suche nach und der Import von öffentlichen Schlüsseln (**F2.1**) sowie die Unterstützung des Nutzers bei der Beglaubigung von Schlüsseln (**F2.2**), wie sie herkömmlich durch den händischen Vergleich von Fingerprints durchgeführt wird. Die Liste bekannter Schlüssel sollte außerdem mit Informationen von den Schlüsselservern aktualisiert werden können, sowohl explizit auf Wunsch des Nutzers (**F2.3**) als auch periodisch als Hintergrundaktivität, um eine eventuelle Veröffentlichung von Widerrufszertifikaten mit möglichst kurzem zeitlichen Verzug bemerken zu können (**F2.4**). Die App sollte ein Konzept für ein Vertrauensmodell besitzen (**F2.5**), bei dem es auch möglich ist, die dafür spezifischen Informationen (also z.B. die Vertrauensverhältnisse zu bestimmten Schlüsseln) mit anderen Geräten des Nutzers zu synchronisieren (**F2.6**).

2.3 Schnittstelle für externe Apps (F3)

Um die kryptografischen Operationen für andere Apps nutzbar zu machen, muss eine entsprechende Schnittstelle zur Verfügung gestellt werden (**F3.1**).

Greift eine App auf die API einer OpenPGP-App zu, sollte sie die alleinige Kontrolle über den Kontrollfluss der Nutzeroberfläche haben. Trotzdem müssen bestimmte Dialoge von der OpenPGP-App direkt durchgeführt werden können, insbesondere die Eingabe einer für den privaten Schlüssel verwendeten Passphrase. Die kryptografischen Operationen sollten jedoch nicht implizit eine Nutzeroberfläche anzeigen, sondern diesen Aufruf explizit in den Kontrollfluss der aufrufenden App zurückreichen (**F3.2**).

Da der Inhalt entsprechender Dateien groß sein kann, beispielsweise bei Daten aus Cloud Storage oder größeren Anhängen, muss es möglich sein, die Daten als Datenstrom zur Bearbeitung zu übergeben. (**NF3.3**)

Das derartige Bereitstellen einer Schnittstelle, welche anderen Apps den Zugriff auf sensitive Daten gewährt, macht eine Verwaltung von Zugriffsberechtigungen notwendig. Diese muss sowohl auf Ebene des Entwicklers als auch des Nutzers eine Entscheidung über die Erlaubnis eines Zugriffs für jede App ermöglichen (**F3.4**).

2.3.1 Betrachtete Clients

Als potentieller Nutzer der kryptografischen API kommt jede Anwendung in Betracht, die sich mit Kommunikation im weitesten Sinne oder vertraulicher Datenspeicherung befasst. Da OpenPGP selbst nicht für ein spezifisches Nutzungsszenario entworfen wurde, kann eine entsprechende API unabhängig vom darüberliegenden Protokoll beschrieben werden.

Wichtig im Zusammenhang mit OpenPGP ist somit, welche Anforderungen eine API erfüllen muss, damit sie in den unterschiedlichen Einsatzgebiete des Standards genutzt werden kann. Neben der klassischen Anwendung in E-Mail-Clients wird OpenPGP insbesondere von folgenden Anwendungen genutzt:

- Dateimanager integrieren OpenPGP, um eine Dateiver- und -entschlüsselung sowie Signaturüberprüfung bereitzustellen.
- Webbrowser können OpenPGP zur Überprüfung von Webseiten-Zertifikaten nutzen, E-Mail-Verschlüsselung in Webmail-Angeboten erlauben oder den Schlüsselimport von Schlüsselservern erleichtern. Diese Features werden oft nicht im Webbrowser selber, sondern durch Add-ons implementiert.
- Instant Messenger, insbesondere die auf dem XMPP-Standard basierenden, können unter Nutzung von XEP-0027 und des in Entwicklung befindlichen Nachfolgers eine OpenPGP-Integration anbieten.

2.4 Eigenständige kryptografische Operationen (F4)

Neben der Bereitstellung kryptografischer Operationen für andere Apps sollten einfache kryptografische Operationen, die nicht auf einem darüberliegenden Protokoll beruhen, dem Nutzer auch eigenständig durch die OpenPGP-App zur Verfügung gestellt werden (**F4.1**). Dazu zählen das einzelne Ver- und Entschlüsseln von Dateien im OpenPGP-eigenen Dateiformat sowie das Erzeugen und Verifizieren von Signaturen zu Dateien und Text. Diese Operationen sollten außerdem mit den dazu vorgesehenen Mechanismen des Betriebssystems aufrufbar sein, wie Verknüpfungen mit URI-Schemata und Dateiendungen (**F4.2**).

2.5 Sicherheitseigenschaften (NF5)

Als sicherheitskritische App unterliegt eine OpenPGP-App erhöhten Sicherheitsanforderungen.

Die Sicherheit des Nutzers hängt direkt von der Geheimhaltung und Integrität des privaten Schlüssels ab. Entsprechend muss das private Schlüsselmaterial vor Zugriff durch andere Apps sowie einer Anzahl besonderer Angriffsszenarien geschützt sein, denen mobile Endgeräte ausgesetzt sind (**NF5.1**).

Ein wichtiger Aspekt, an dem das Vertrauen in den Code einer App (und damit ihre Sicherheit) bemessen werden kann, ist ihre Entwicklergemeinschaft, Struktur und Aktualität der Paket-Betreuung sowie ihr „Track Record“ bezüglich in der Vergangenheit gefundener Sicherheitsprobleme (**NF5.2**).

Letztlich ist wichtig, dass der kryptografische Code der App sich korrekt verhält. Faktoren, die in dieser Hinsicht Vertrauen schaffen, sind Unit-Tests zur Verifikation einzelner Verhaltensmerkmale, Interoperabilitäts-Tests zur Überprüfung der Konformität mit anderen Implementierungen, eine

großflächige Nutzung der App in verschiedenen Anwendungsszenarien sowie Untersuchungen oder im Allgemeinen Aufmerksamkeit durch unabhängige Entwickler (**NF5.3**).

2.6 Weitere Anforderungen (NF6)

Mobile Endgeräte haben im Vergleich zu Desktop-PCs nicht nur eine geringere Prozessor-Leistung, sondern auch beschränkte Energiereserven, was zu einer erhöhten Anforderung an die Performance-Eigenschaften einer OpenPGP-App führt (**NF6.1**). Insbesondere sollte bei der Implementierung berücksichtigt werden, dass auch bei der Durchführung rechenintensiver Algorithmen die Ansprechbarkeit der Nutzeroberfläche gewährleistet ist. Die eingesetzten Bitlängen symmetrischer sowie asymmetrischer Verschlüsselungsalgorithmen sollte dabei gegenüber Desktop-Betriebssystemen nicht reduziert werden. Ein gesondert zu betrachtender Einzelfall ist der „S2K"-Mechanismus, welcher im OpenPGP-Standard als „Key Derivation Function" für das Ableiten symmetrischer Schlüssel aus Passwörtern verwendet wird und eine Parametrisierbarkeit des notwendigen Rechenaufwandes für die einzelne Operation enthält.

Um Hürden bei der Verbreitung der App zu vermeiden, muss sie auf den gängigen Verbreitungswegen verfügbar sein und ohne spezielle Anforderungen an den Nutzer, das Gerät oder das Betriebssystem installierbar sein (**NF6.2**). Eine Alternative zur Verbreitung ist die Möglichkeit, die OpenPGP-App auf Geräten vorzuinstallieren oder als API des Betriebssystems zu integrieren.

3 Analyse der Anforderungen im Kontext des Betriebssystems Android

Die in Kapitel 2 benannten Anforderungen werden in diesem Kapitel im Kontext von Android als Anwendungsszenario betrachtet. Die Anforderungen nehmen dabei konkreten Bezug auf die im Kapitel 2 bezeichneten Referenzen **F** und **NF**.

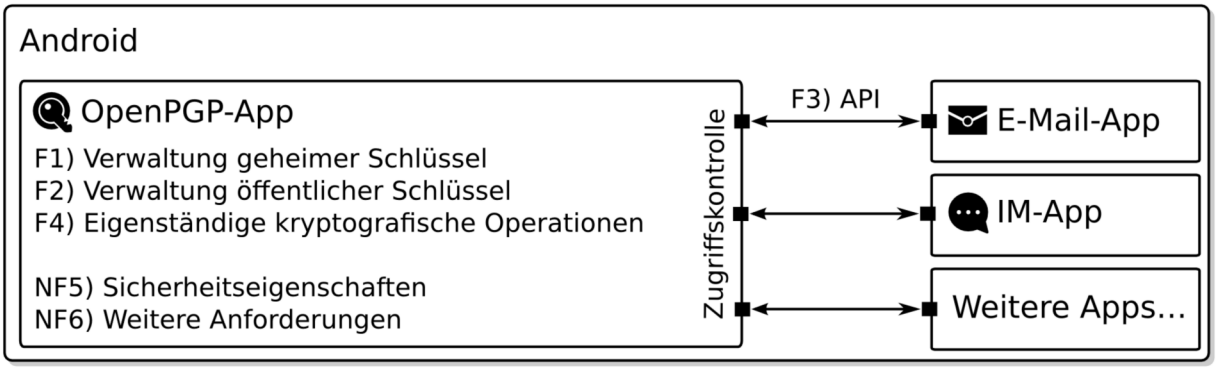


Abbildung 1: Schematische Darstellung der Anforderungen innerhalb einer OpenPGP-App

3.1 Verwaltung privater Schlüssel (F1)

3.1.1 Erstellen eines privaten Schlüssels (F1.1)

Die OpenPGP-App sollte dem Nutzer ermöglichen, einen privaten Schlüssel zu erstellen.

Für die sichere Erstellung von privaten Schlüsseln (und bestimmte andere kryptografische Operationen) ist es wichtig, dass für die Erzeugung der benötigten Zufallszahlen ein gewisses Maß an Entropie zur Verfügung steht, welche üblicherweise vom Betriebssystem bereitgestellt wird. Als Entropie bezeichnet man ein Maß an „echtem“ Zufall, der einem System zur Verfügung steht. Also ein Zufall, der nicht nur aus reproduzierbaren Quellen (wie der Systemzeit) abgeleitet ist, sondern beispielsweise auch aus Latenzinformationen von Disk-Zugriffen, Interrupts oder Nutzerinteraktion. Die zur Verfügung stehende Menge echter Entropie ist naturgemäß begrenzt und wird deswegen gewöhnlich als „Seed“ für einen Pseudo-Zufallszahlengenerator (PRNG) verwendet.

Android-Versionen vor 4.3 (Jelly Bean) waren von einem Fehler betroffen, der allerdings nicht direkt mit der Verfügbarkeit von Entropie zusammenhängt. Dieser führte zu einer inkorrekten Initialisierung des PRNG in der per Default genutzten OpenSSL-Implementierung der Java Cryptography Architecture (JCA). Aufgrund dieses Fehlers lieferte der PRNG in der zur Verfügung gestellten SecureRandom API vorhersagbare Werte, was sich unter anderem auf alle generierten öffentlichen und privaten Schlüsselpaare auswirkte. Dieses Problem ist seit Android-Version 4.3 behoben. Ebenfalls wurde von den zuständigen Entwicklern ein Blogpost mit einem Codeschnipsel „PRNGFixes“ veröffentlicht, mit dem das Problem innerhalb einer App selbst behoben werden kann.

Mobilgeräte stehen in dem Ruf, nur vergleichsweise wenig Entropie zur Verfügung zu haben. Insbesondere kam 2013 die Information in verschiedenen Online-Quellen (1, 2) auf, ein Mangel an Entropie sei für geringe System-Performance verantwortlich. Diese Behauptung wurde im Rahmen einer Untersuchung für das wissenschaftliche Paper Android Low Entropy Demystified (Ding et al. 2014) allerdings hinreichend

widerlegt. In dem Paper werden Zugriffe auf den Entropie-Pool durch Instrumentierung des Kernels nachverfolgt und präzise Untersuchungen über den Zeitverlauf der verfügbaren Entropie angestellt. Ein Ergebnis dieser Untersuchungen ist, dass zum Startzeitpunkt der ersten User-Apps der systemweite PRNG bereits mit ausreichend echter Entropie initialisiert ist, um eine Vorhersage der ausgegeben Werte praktisch unmöglich zu machen. Neben der dort festgestellten frühen Verfügbarkeit von Entropie durch den Kernel über Interrupts und Speicherlatenzen, wird der Entropie-Pool durch das Android-Betriebssystem mittels des EntropyMixer-System-Services unterstützt. Dieser wird als Teil des `system_server`-Prozesses bereits frühzeitig, insbesondere vor Androids Benutzeroberfläche, gestartet. Der EntropyMixer-Service speichert während der Laufzeit periodisch aktuelle Entropie-Daten ab, um sie bei einem neuen Systemstart zusätzlich einzuspeisen, und damit die Qualität der zur Verfügung stehenden Zufallszahlen weiter zu erhöhen. Solange eine OpenPGP-App erst innerhalb der Android-Benutzeroberfläche, und damit bei ausreichender Initialisierung des Entropie-Pools, gestartet werden kann, ist im Allgemeinen nicht davon auszugehen, dass sie von fehlender Entropie beim ersten Gerätestart betroffen werden kann. Dieses Kriterium sollte dennoch bei einer Implementierung bedacht und überprüft werden.

Neben dem frühzeitigen Befüllen des Entropie-Pools ist es außerdem notwendig zu überprüfen, inwieweit die Implementierung im Linux-Kernel Anforderungen erfüllt, welche in „Dokumentation und Analyse des Linux-Pseudozufallszahlengenerators“ diskutiert werden. Aus diesem Dokument geht hervor, dass eine hohe Auflösung der Zeitintervalle (Ticks) eine Voraussetzung für die Verfügbarkeit qualitativ hochwertiger Zufallszahlen ist. Dies wird in aktuellen Kernel-Versionen in der Funktion `random_get_entropy()` implementiert, die hardwareabhängig den CPU-Zeitgeber ausliest.

Abschließend ist festzustellen, dass eine abschließende Bewertung der Sicherheit des Zufallszahlengenerators unter Android nicht zweifelsfrei innerhalb dieser Studie geklärt werden kann.

3.1.2 Modifizieren eines privaten Schlüssels (F1.2)

Das Modifizieren eines privaten Schlüssels umfasst das Hinzufügen und Widerrufen von Nutzer-Identitäten und Unterschlüsseln sowie das Setzen einer neuen primären Nutzer-Identität.

Diese Anforderung umfasst keine Android-spezifischen Aspekte.

3.1.3 Sicherung und Wiederherstellung privater Schlüssel (F1.3)

Die Sicherung der privaten Schlüssel ist eine wichtige Anforderung, da alle mit den dazugehörigen öffentlichen Schlüsseln verschlüsselten Kommunikationsdaten bei Verlust unwiederbringlich verloren gehen. Mobile Endgeräte sind außerdem durch ihre geringe Größe, dem hohen Wert und das mobile Nutzungsszenario einem erhöhten Verlust- und Diebstahlrisiko ausgesetzt.

Für die Sicherung von privatem Schlüsselmaterial wird zunächst ein Ort für die eigentliche beständige Unterbringung der zu sichernden Daten benötigt, die Menge an Daten bewegt sich dabei lediglich im mittleren Kilobyte-Bereich. Geeignete Speicherorte unterscheiden sich hauptsächlich nach Einfachheit ihrer Nutzung und Vertraulichkeit, wie im Folgenden betrachtet.

Die herkömmliche Variante der Datensicherung legt die Daten auf einem unabhängigen, physikalischen Speichermedium (wie einem USB-Stick oder einer SD-Karte) ab. Der Vorteil dieser Variante liegt darin, dass sie für den Nutzer verständlich und vertrauenswürdig ist, da der physikalische Zugriff während der Lagerung kontrolliert werden kann. Nachteilig ist, dass sie dedizierte Hardware erfordert und die Unterstützung auf aktuellen Plattformen rückgängig ist. In aktuellen Android-Geräten ist nur noch in seltenen Fällen ein Lesegerät für SD-Karten enthalten. USB-Sticks lassen sich nur mit Hilfe eines OTG-Kabels anschließen, welche nicht allzu weit verbreitet sind. Ein weiterer Nachteil ist, dass Nutzer die Lebenszeit von physikalischen Speichermedien nur schlecht einschätzen können, was zu einer eingeschränkten Zuverlässigkeit führt.

Eine andere Möglichkeit zur Speicherung von Daten außerhalb eines Endgeräts ist Cloud-Storage. Dieser Begriff fasst Services zusammen, bei denen Nutzer einen persönlichen Account anlegen können, unter dem sie Daten hinterlegen und flexibel per Login wieder abrufen können. Darunter fallen Dienste wie Dropbox, Google Drive und der Android Backup Service. Weitere Möglichkeiten zur Datensicherung können sich über die Infrastruktur höher liegende Protokolle ergeben, die von API-Clients implementiert werden. Beispielsweise hat ein E-Mail-Client die Möglichkeit, Daten per IMAP in der Mailbox des Nutzers zu speichern. Ebenso kann ein XMPP-Client beliebige Daten serverseitig im Private XML Storage ablegen.

In jedem Fall ist es zur Gewährleistung der Vertraulichkeit und Integrität notwendig, das gesicherte Schlüsselmaterial mit möglichst hoher Sicherheit zu verschlüsseln. Im einfachsten Fall sollte hierzu ein von der App generierter „Backup-Code“ mit hoher Entropie (110 Bits oder mehr) verwendet werden. Dem Nutzer sollte aber explizit nicht die Möglichkeit gegeben werden, diesen selbst zu wählen.

Eine Wiederherstellung von privaten Schlüsseln muss ebenfalls unterstützt werden, sowohl in dem zur Sicherung verwendeten, zusätzlich verschlüsselten Format, als auch in dem von vielen Implementierungen verwendeten Format ohne weitere Verschlüsselung.

3.2 Verwaltung öffentlicher Schlüssel (F2)

3.2.1 Suche und Import öffentlicher Schlüssel (F2.1)

Eine wesentliche Funktion ist der Import öffentlicher Schlüssel in eine Liste lokal bekannter Schlüssel. Dabei müssen sowohl der direkte Import aus Dateien, als auch eine Suche und anschließender Import von Schlüsselservern, möglich sein.

Für die Suche auf Schlüsselservern muss das HKPS-Protokoll unterstützt werden, welches eine auf TLS aufbauende Ergänzung des HKP-Protokolls darstellt und eine vertrauliche Verbindung zu Schlüsselservern ermöglicht. Eine Verwendung des unverschlüsselten HKP-Protokolls ist für besondere Szenarien zulässig, beispielsweise zur Kommunikation mit Schlüsselservern in einem lokalen, vertrauenswürdigen Netzwerk, sollte aber nur nach expliziter Aufforderung durch den Nutzer erfolgen. Alle importierten Schlüssel müssen auf Integrität des enthaltenen Schlüsselmaterials bezüglich der vom Primärschlüssel ausgestellten Self-Signatures überprüft werden, sowohl für User-IDs als auch für Subkeys.

Eine verwandte Funktionalität ist der Import von Schlüsseln als Teil von Beglaubigungsmethoden, bei denen der Fingerprint oder sogar der ganze öffentliche Schlüssel übermittelt wird. Dies passiert beispielsweise beim Einlesen eines QR-Codes, oder dem Empfang von Schlüsselmaterial über eine NFC-Schnittstelle. Es handelt sich hier um einen besonderen Anwendungsfall, dieser ist deshalb Teil der entsprechenden Anforderung (siehe folgender Abschnitt).

3.2.2 Beglaubigung öffentlicher Schlüssel (F2.2)

Um bei einer Ende-zu-Ende-verschlüsselten Kommunikation eine Man-in-the-Middle-Attacke auszuschließen, ist es notwendig sicherzustellen, dass für den beabsichtigten Kommunikationspartner der richtige Schlüssel verwendet wird. Dies benötigt den Austausch einer kleinen Datenmenge über einen authentisierten Kommunikationskanal, der aber nicht vertraulich sein muss. In der einfachsten Umsetzung tauschen die Kommunikationspartner im persönlichen Gespräch Daten aus, die ihre verwendeten öffentlichen Schlüssel eindeutig identifizieren. Traditionell lesen die Parteien sich zu diesem Zweck gegenseitig den „Fingerprint“ ihres Schlüssels vor, der aus einem SHA-1-Hash von 20 Byte Länge, dargestellt als 40 Hexadezimal-Zeichen, besteht. Dieses Verfahren ist sicher, interoperabel und vielen Nutzern bereits bekannt und sollte deswegen von einer OpenPGP-App unterstützt werden.

Verglichen mit dieser händischen und vergleichsweise unbequemen Variante stehen den meisten Android-Geräten erweiterte Möglichkeiten zur Umsetzung der Verifikation zur Verfügung. Nennenswert sind hier

insbesondere eine Validierung mittels QR-Code oder NFC. Bei beiden dieser Möglichkeiten handelt es sich um unmittelbare Methoden der Datenübertragung zwischen den mobilen Endgeräten der Nutzer, welche die Möglichkeit eines Man-in-the-Middle-Angriffs ebenso ausschließen, wie der händische Vergleich zwischen Fingerprints. Zur Nutzung innerhalb eines QR-Codes sollte das URI-Schema `openpgp4fpr` verwendet werden, welches nicht formell standardisiert, aber in der Entwicklergemeinschaft wohl etabliert ist. Eine Unterstützung dieser Verfahren ist für die Vollständigkeit einer OpenPGP-App nicht notwendig, aber wünschenswert. Im Gegensatz zum händischen Vergleich von Fingerprints bringen diese Mechanismen eine höhere Nutzbarkeit mit sich, da sie schneller sind und weniger Aufmerksamkeit erfordern. Besonders die Nutzung von QR-Codes hat in den letzten Jahren in einigen Kommunikations-Apps Verbreitung erfahren, und wird beispielsweise in der Threema Instant-Messaging-App als exklusive Methode zur Verifikation von Kontakten verwendet.

3.2.3 Aktualisierung öffentlicher Schlüssel (F2.3 und F2.4)

Die Aktualisierung einzelner öffentlicher Schlüssel durch den Abruf der Schlüsselinformation von den Schlüsselservern anhand des Fingerprints und anschließender Zusammenfügung mit den vorhandenen Informationen wird umgesetzt. Es ist damit eine recht einfache Operation.

Um auf die Veröffentlichung eines Widerrufszertifikats mit möglichst wenig Verzug reagieren zu können, sollte die bekannte Liste an Schlüsseln durch einen periodisch im Hintergrund laufenden Prozess aktuell gehalten werden. Das Android-Betriebssystem stellt zu diesem Zweck den SyncAdapter-Mechanismus zur Verfügung, mit dem Synchronisierungs-Operationen automatisch auf einen Zeitpunkt verschoben werden können, an dem Konnektivität und ausreichend Strom vorhanden ist und der Nutzer nicht unterbrochen wird. Dabei sollte (optional) auch darauf geachtet werden, nicht durch einen aufeinanderfolgenden Abruf die Liste aller Schlüssel an den Schlüsselserver zu offenbaren.

3.2.4 Vertrauensmodell (F2.5)

Es gibt diverse Ansätze für Vertrauensmodelle zwischen OpenPGP-Schlüsseln, wovon allerdings keiner als De-facto-Standard etabliert ist. Als einfaches Vertrauensmodell reicht es aus, den Nutzer zu ermutigen, Schlüssel von Kommunikationspartnern zu verifizieren und diesen Status beim Empfang signierter Nachrichten anzuzeigen. Auf diese Weise wird ein Vertrauen nur direkt durch den Nutzer und individuell mit jedem Kommunikationspartner aufgebaut. Dieser Status kann, muss aber nicht, durch das Beglaubigen von Schlüsseln abgebildet werden.

Aufbauend auf Beglaubigungen von Schlüsseln, die auch über Schlüsselserver veröffentlicht werden können, gibt es das sogenannte *Web-of-Trust*-Verfahren, bei dem der Nutzer volles oder eingeschränktes Vertrauen in die Beglaubigungen anderer Schlüssel aussprechen kann, so dass auch deren Beglaubigungen als valide im Sinne der Nutzer-Authentisierung angesehen werden. Trotz langer Verfügbarkeit im OpenPGP-Umfeld hat sich dieses Verfahren aufgrund seiner Komplexität, der damit verbundenen Undurchsichtigkeit für den Nutzer und Privacy-Problemen mit dem Veröffentlichen von Vertrauensprofilen nicht durchsetzen können. Es ist dennoch ein legitimer Ansatz, der aktuell von GnuPG implementiert und in einigen festen Nutzerkreisen (wie der Debian-Entwicklergemeinschaft) verwendet wird.

Ein weiterer Ansatz ist das sogenannte „Trust on First Use“ (TOFU). Bei diesem Verfahren wird ein Schlüssel für einen neuen Kommunikationspartner initial als valide angenommen und der Nutzer wird über Änderungen informiert. Auf diese Weise ist Vertrauen in einen Schlüssel nicht von vornherein vorhanden, sondern wird bei fortlaufender Kommunikation unter Verwendung desselben Schlüssels im Laufe der Zeit durch die Sicherheit aufgebaut, dass der Kommunikationspartner immer im Besitz des anfänglich für gültig angenommenen private Schlüssels ist. Das TOFU-Verfahren hat in den letzten Jahren, insbesondere bei synchroner Kommunikation (wie etwa in der Signal-App), Verbreitung gefunden.

Diese Anforderung umfasst keine Android-spezifischen Aspekte.

3.2.5 Synchronisierung des Vertrauenszustands (F2.6)

Verwandt mit der Datensicherung ist die Synchronisierung der Vertrauensverhältnisse des von der App verwendeten Vertrauensmodells. Dies kann recht einfach gelöst werden, wenn der Vertrauenszustand der öffentlichen Schlüssel mittels Beglaubigungen durch den privaten Schlüssel des Nutzers ausgedrückt wird, indem die entsprechenden Beglaubigungsdaten auf den Schlüsselservern hinterlegt werden. Dies gewährleistet aufgrund der zentralen und öffentlichen Infrastruktur allerdings weder Vertraulichkeit noch Verfügbarkeit.

Dies ist jedoch nicht zwangsläufig bei jedem Vertrauensmodell der Fall. Die von GnuPG für den Web-of-Trust-Mechanismus verwendete „Ownertrust"-Datenbank beispielsweise kann zwar importiert und exportiert werden, müsste für eine echte Synchronisierung zwischen mehreren Geräten des Nutzers aber erweitert werden.

Auch für eventuelle Ansätze des TOFU-Verfahrens gibt es kein etabliertes Konzept zur Synchronisierung zwischen Clients.

3.3 Schnittstelle für externe Apps (F3)

Damit externe Apps, wie beispielsweise E-Mail-Clients, mit der OpenPGP-App interagieren können, muss eine Programmierschnittstelle (API) angeboten werden. Diese Schnittstelle sollte vordergründig alle grundlegenden OpenPGP-Operationen implementieren. Dazu gehört die Erstellung und Verifikation von Cleartext und Detached Signatures, die Ver- und Entschlüsselung sowie eine kombinierte Verschlüsselung mit Signaturen. Des Weiteren sollten Methoden bereitgestellt werden, um öffentliche sowie private Schlüssel für die Nutzung für Signaturen und Verschlüsselung durch externe Apps auszuwählen. Alle grundlegenden OpenPGP-Operationen sollten zusätzlich zur binären Ausgabe auch die ASCII-Armor-Kodierung ermöglichen. Mit diesen Methoden kann OpenPGP/MIME (RFC 3156) für E-Mail-Clients implementiert werden. Optional können Methoden zur Anzeige von Schlüsseln sowie zum Herunterladen von fehlenden Schlüsseln angeboten werden. Um den in der Einführung der Anforderungen erwähnten Nachfolger des XEP-0027 zu implementieren, sind zudem Backup-Methoden notwendig, um private Schlüssel in einem symmetrisch verschlüsselten Format zu exportieren/importieren. Das dafür notwendige Format wird im XEP spezifiziert.

Zusätzlich zu den funktionalen Anforderungen sollte eine moderne API so klar wie möglich strukturiert sein und nur wenige Freiheiten bei der Nutzung zulassen, um die „Developer Usability" zu erhöhen. Dies verringert mögliche Implementierungsfehler, wie beispielsweise dargelegt in der Veröffentlichung „Rethinking SSL Development in an Appified World". Da Android-Entwickler mit dem Design der Android-API vertraut sind, sollte auch die OpenPGP-App so weit wie möglich auf existierenden und bekannten Konzepten beruhen. Da Android-Apps bis auf wenige Ausnahmen ausschließlich in Java entwickelt werden, sollte das Design der API ebenfalls auf den in Java üblichen Design-Pattern aufbauen. Beispielsweise sollten für die Ein- und Ausgabe von Daten die Input- bzw. OutputStream-Klassen verwendet werden.

Eingabe von Passwörtern und andere sicherheitskritische Benutzerinteraktionen sollten nicht von den Client-Apps sondern von der OpenPGP-App selber bereitgestellt werden, damit die Client-Apps keinen Zugriff auf die Eingaben haben. Weitere Benutzerinteraktionen, die von der OpenPGP-App implementiert werden können, sind die Behandlung von klassischen Problemfällen, wie beispielsweise die Auswahl eines Schlüssel basierend auf einer E-Mail-Adresse: Entweder es ist kein passender Schlüssel vorhanden und ein passender muss von einem Schlüsselserver heruntergeladen werden oder aber es existieren zwei Schlüssel mit der selben E-Mail-Adresse. Beide Fälle können von der OpenPGP-App selber behandelt werden. Wenn möglich, sollten die Client-Apps bestimmen können, wann die nötige Benutzerinteraktion angezeigt wird. Man betrachte den Fall einer Signaturgenerierung, die im Hintergrund durchgeführt wird, während der Nutzer nicht mit dem Gerät interagiert. Solange das nötige Passwort im Cache vorhanden ist, kann diese Operation ausgeführt werden. Wenn nicht, sollte die Eingabe des Passworts erst angezeigt werden, wenn der Nutzer das Gerät wieder aktiv nutzt – beispielsweise nach dem Entsperren des Bildschirms.

3.3.1 Schnittstelle (F3.1)

Klassische Unix-IPC-Mechanismen (wie System V) stehen unter Android nicht zur Verfügung. Auch moderne Mechanismen, die unter GNU/Linux zur Verfügung stehen (wie das D-Bus-Protokoll), sind nicht implementiert. Local Sockets können zwar genutzt werden, bieten aber keine Möglichkeiten, um den Zugriff auf eine Auswahl an Client-Apps zu begrenzen. Wie zuvor dargelegt, sollte die Schnittstelle auf Android-spezifischen Mechanismen für Inter-Prozess-Kommunikation (IPC) aufsetzen. Somit können Android-spezifische Features, wie beispielsweise Zugriffskontrollen, genutzt werden und die Developer Usability wird erhöht.

Android stellt mit dem Binder-Mechanismus einen modernen Mechanismus zur IPC bereit. Er basiert nicht auf klassischen System-V-Komponenten, ist aber aus Performance und Sicherheitsgründen als Treiber im Kernel implementiert. So kann Speicher direkt ohne Kopien allokiert und dieser zwischen Prozessen geteilt werden. Der Binder-Mechanismus im Kernel wird normalerweise nicht direkt benutzt. Stattdessen gibt es IPC-Methoden, die auf ihm aufbauen und genutzt werden können, um Nachrichten zwischen Prozessen auszutauschen oder um Daten zu streamen. Hierzu gehören Intents, Messenger und Services.

Services, die zwischen unterschiedlichen App-Prozessen kommunizieren, müssen über eine stabile API verfügen, die über die *Android Interface Definition Language* (AIDL) definiert wird. Alle grundlegenden Android-Komponenten, wie beispielsweise Window Manager, Package Manager, Telephony Manager, etc., benutzen somit einen Teil des Binder-Frameworks.

Relevant für dieses Dokument sind die Sicherheitseigenschaften von Androids Binder. Diese beruhen auf einer Kombination aus (vom Betriebssystem bereitgestellten) User ID (UID) und Binder Capabilities. Es wird unterschieden zwischen direkten Capabilities, bei denen beispielsweise Zugriff auf ein bestimmtes Interface erlaubt wird, und indirekten Capabilities, bei denen eine Validierung durch ausgetauschte Tokens stattfindet. Android Permissions werden überprüft, indem unter Nutzung der UID einer Binder-Transaktion die zugeordneten Permissions dieser UID nachgeschlagen werden. Enthält diese Liste die vom Service benötigte Berechtigung, wird die Verbindung zugelassen.

3.3.1.1 Activity Intents

Intents sind eine grundlegende und einfache Methode, um IPC auf Android umzusetzen. Sie bilden eine hohe Abstraktionsschicht, aufbauend auf Androids Binder, und werden in so gut wie allen Android-Apps eingesetzt. Ein Intent ist definiert als Nachrichtenobjekt, das dazu genutzt werden kann, eine Aktion einer anderen App-Komponente anzufordern. Activity Intents werden eingesetzt, um eine Benutzerinteraktion mit einer anderen App-Komponente intern oder auch mit externen Apps anzufordern. Intents können Daten mitgegeben werden, entweder direkt als Parcelable in einer serialisierten Form oder als URI, die auf die Speicherstelle der Daten zeigt.

Eine auf Activity Intents aufbauende API wird von APG und GnuPG for Android bereitgestellt.

Wichtiger Nachteil von Activity Intents ist, dass diese prinzipiell als Methode zum Aufruf einer Nutzerinteraktion vorgesehen sind und dementsprechend bei Ausführung des Intents eine Activity, also eine Komponente mit Nutzeroberfläche, gestartet wird. Es ist möglich, eine Activity ohne sichtbares Interface zu starten und nach Bearbeitung des API-calls wieder zu schließen. Dies widerspricht aber der Intention einer Activity Intents und sollte aus diesem Grund vermieden werden.

Als technische Eigenart ist weiterhin festzustellen, dass durch Parcelables serialisierte Daten durch den Binder in ihrer Größe auf 1 MB begrenzt sind. Diese Begrenzung wird von allen laufenden Transaktionen innerhalb des Prozesses geteilt. Somit können nur kurze Texte, aber keine größeren Dateien oder Anhänge serialisiert als String oder Byte-Array ausgetauscht werden. Auch die von APG und GnuPG for Android verwendete API ist von dieser Limitierung betroffen.

Eine gangbare Alternative wäre hier die Nutzung von FileProvidern. Dazu muss eine App einen FileProvider implementieren, der Dateien verwaltet und dazugehörige Metainformationen über eine einheitliche

Schnittstelle als URIs bereitstellt. Die Zugriffskontrolle von FileProvidern ermöglicht, die Dateien entweder global über Permissions oder für einzelne Dateien über grantUriPermission zu beschränken. Leider funktioniert der `grantUriPermission`-Mechanismus in Android-Versionen vor 4.4 nicht einwandfrei.

Insgesamt ist eine API, welche auf Activity Intents aufbaut, aufgrund der genannten Schwächen und verfügbarer Alternativen mit besseren Eigenschaften, nicht zu empfehlen.

3.3.1.2 AIDL Service

Statt einer Implementierung als Activites kann eine App einen Service bereitstellen, welcher als exportiert gekennzeichnet wird und damit als Schnittstelle von anderen Apps verwendet werden kann. Diese sind dafür vorgesehen, als lang andauernde Prozesse ohne Nutzerinteraktion im Hintergrund zu laufen.

Zur Kommunikation können Intents verwendet werden, diese haben weiterhin die oben genannten Nachteile bezüglich des Datenaustauschs. Als Alternative kann eine direktere Verknüpfung erfolgen über die Android Interface Definition Language (AIDL). Unter Nutzung der AIDL wird ein Interface definiert, welches in beide Apps eingebunden wird, um eine beidseitige Kommunikation zu ermöglichen. Ein Pre-Prozessor aus den Android SDK Tools generiert aus der AIDL-Datei ein Java-Interface, bei dem die innere abstrakte Stub-Klasse innerhalb des Services implementiert wird, von welcher der aufrufende Prozess nach Bindung an den Service eine Instanz erhält. AIDL unterstützt so nativ synchron blockende sowie asynchrone Aufrufe. Als besonderes Merkmal ist es bei der Kommunikation mittels AIDL-Interfaces möglich, mittels der `ParcelFileDescriptor`-Klasse File Descriptor zwischen den Apps zu übergeben. Mittels der Verwendung von Pipes können auf diese Weise Daten wie gewohnt über Input-/OutputStreams direkt zwischen den Prozessen geteilt werden.

Insgesamt ist ein auf AIDL Service aufbauender Service gut geeignet für die Umsetzung einer OpenPGP-API.

3.3.1.3 Stateless vs. Security, Versionierung

Es bleiben einige weitere Merkmale zu beachten beim Design einer derartigen API. Die hier genannten Ansätze eines API-Designs sind in öffentlicher Diskussion und in Zusammenarbeit mit Entwicklern von Guardian Project (GnuPG for Android) und K-9 Mail entstanden. Weitergehende Diskussionen finden sich im Wiki des Guardian Projects. Die genauen von einer API zur Verfügung gestellten Methoden bestimmen sich, aufbauend auf den hier vorgestellten Konzepten, durch die Bedürfnisse der Client-Apps. Eine detaillierte Beschreibung der Methoden liegt deshalb nicht im Rahmen dieses Dokuments.

- Vermeidung von State in der OpenPGP-App: Der Service sollte, soweit es geht ohne einen zwischengespeicherten State auskommen. Wird eine Operation unterbrochen, beispielsweise für den Aufruf einer Nutzerinteraktion mittels PendingIntent, sollte der zu diesem Zeitpunkt entstandene State als Teil des zwischenzeitlichen Resultats zurückgegeben werden, welches bei einem erneuten Aufruf wieder übergeben wird. Auf diese Weise kann die OpenPGP-App ohne einen Zwischenspeicher für Zwischenergebnisse von Operationen auskommen. Dabei sollte allerdings darauf geachtet werden, dass sensitive Variablen, wie etwa die Passphrase zu einem privaten Schlüssel, vor der aufrufenden App verborgen werden.
- Versionierung der API: Eine AIDL-Datei ist nicht einfach versionierbar, da sie Methoden unveränderlich definiert und somit beide Kommunikationspartner mit der gleichen AIDL-Datei arbeiten müssen. Jede Änderung der Methoden und somit ein Update der OpenPGP-App würde eine Inkompatibilität mit allen externen Apps herstellen, die nicht die neue Version der AIDL-Datei nutzen. Anstatt die Parameter innerhalb der Methodendefinition zu nutzen, bietet es sich an, diese innerhalb eines Bundles zu übergeben. So können ohne weitere Änderungen neue Parameter hinzugefügt werden, ohne dass die AIDL-Datei selber geändert werden muss. Hierbei ist zu beachten, dass ein ParcelFileDescriptor nicht im Bundle übergeben werden kann, was auch begründet, warum diese nicht unter Nutzung einer Handler-Klasse ausgetauscht werden können. Zusätzlich dazu sollten Parcelables versionierbar sein, da auch hier das Hinzufügen weiterer Klassenvariablen die

serialisierte Datenstruktur ändert. Die vom Google-Angestellten Roman Nurik gepflegte App DashClock Widget zeigt beispielhaft, wie solche versionierbaren Parcelables aussehen sollten.

3.3.2 Kontrolle über Kontrollfluss der Nutzeroberfläche (F3.2)

Sollte während der Ausführung eines API-Aufrufs eine Interaktion mit dem Nutzer erforderlich sein, ist es sinnvoll, der Client-App die Möglichkeit zu geben, selbst zu entscheiden, zu welchem Zeitpunkt die entsprechende Bedienoberfläche angezeigt wird. Obwohl technisch möglich, sollte das Starten von Dialogen zur Nutzerinteraktion durch den im Hintergrund laufenden Service vermieden werden, da dies den Fluss der laufenden App ungewollt unterbrechen kann und damit dem Nutzererlebnis abträglich ist. Stattdessen bietet Android das Konzept der PendingIntents an, die in der OpenPGP-App vorbereitet über die API zurückgegeben und dann mittels `startIntentSenderForResult()` zu einem beliebigen Zeitpunkt von der Client-App ausgeführt werden können. Dieses Konzept findet sich in dieser Form auch in Google's In-App Billing API (siehe `getBuyIntent()`) und in Androids AccountManager (`getAuthToken()` mit `KEY_INTENT`).

3.3.3 Bearbeitung der Daten als Datenstrom (NF3.3)

Um eine zuverlässige Verarbeitung von großen Dateien zu gewährleisten, ohne dabei auf temporäre Dateien zurückzugreifen oder einen Speicherüberlauf zu riskieren, sollten die mit der OpenPGP-API ausgetauschten Daten in Form von Datenströmen übertragen werden. Diese Anforderung kann durch die Verwendung eines AIDL-Services (siehe Abschnitt 3.3.1.2) und eine Nutzung der von der Java-API zur Verfügung gestellten `Stream`-Klassen umgesetzt werden.

3.3.4 Berechtigungssystem (F3.4)

Da die OpenPGP-App eine Programmierschnittstelle zur Verfügung stellen soll, welche prinzipiell von jeder installierten App genutzt werden kann, wird ein Konzept benötigt, um entsprechende Zugriffsberechtigungen zu verwalten. Zu diesem Zweck können zwei unterschiedliche Ansätze verfolgt werden: Zunächst bietet das Android-Betriebssystem ein eigenes System zur Verwaltung von Berechtigungen von Apps an, welches ebenfalls die Definition eigener Berechtigungen für eine App erlaubt und damit für diese Aufgabe grundsätzlich geeignet ist. Aufgrund von Einschränkungen dieses Systems, die erst in neueren Android-Versionen behoben sind, ist eine Nutzung der Android-eigenen Berechtigungen für eine OpenPGP-App allerdings problematisch. Als Alternative ist es möglich, ein unabhängiges System zur sicheren Verwaltung von Berechtigungen zu implementieren, indem die entsprechende Funktionalität des Binder-Frameworks direkt genutzt wird.

Die folgenden Abschnitte erläutern beide Ansätze und ihre entsprechenden Vor- und Nachteile.

3.3.4.1 Android-Berechtigungen

Das Android-Betriebssystem besitzt mit Permissions ein eigenes, komplexes System zur Zuweisung und Verwaltung von Berechtigungen – zugeschnitten auf die Bestätigung einzelner Berechtigungen durch den Nutzer. Dieses System hat über die Jahre in verschiedenen Android-Versionen starke Veränderungen durchlaufen, welche insbesondere Auswirkungen auf die Nutzbarkeit für eine OpenPGP-App mit sich bringen. Im Nachfolgenden werden die relevanten Gegebenheiten und Änderungen der Android Releases seit Android 4.3 (Jelly Bean) zusammengefasst.

Das Android-Permission-System hat sich inkrementell verändert. Allen Versionen ist gemein, dass es eine Anzahl von Permissions gibt, die einer App (welche sie anfordert) Zugriff auf gewisse zugriffsbeschränkte Schnittstellen gewähren. Es gibt eine recht große Anzahl Permissions, die vom Betriebssystem vorgegeben werden. Die Definition von neuen Permissions durch Apps zur Installationszeit ist ebenfalls möglich.

Permissions sind in verschiedene „Protection Level“ unterteilt, die wichtigsten davon *normal*, *dangerous* und *signature*. Eine *normal*-Permission wird nur pro forma angefragt und ohne Rückfrage an den Nutzer automatisch gewährt. Gehört eine Permission zum Level *dangerous*, muss sie für jede App, welche nicht vorinstalliert ist, explizit durch den Nutzer bestätigt werden. Permissions des Levels *signature* sind besonders, da sie genau für die Apps verfügbar sind, welche von demselben Entwickler-Key signiert wurden – dann allerdings auch keine Nutzerinteraktion benötigen. Eine Permission wird anhand einer beliebigen ASCII-Zeichenkette identifiziert. Häufig verwenden Apps ihren Paketnamen als Prefix, so dass eine ungewollte Kollision so gut wie ausgeschlossen ist.

Permissions werden individuell auf Kernel-, IPC-, oder App-Ebene durchgesetzt. Insbesondere ist eine explizite Überprüfung als Teil der Bereitstellung einer Schnittstelle möglich. Eine nähere Betrachtung der Umsetzung von Permissions ist im Rahmen dieses Dokumentes unerheblich.

Besonders wichtig für das Design einer API zur Bereitstellung von kryptografischen Methoden ist die Bereitstellung und Anforderung von Custom-Permissions, also solchen, die nicht durch das Betriebssystem vorgegeben sind. Die Nutzbarkeit dieser Sorte Permissions hat sich über die verschiedenen Versionen von Android erheblich verändert und wird im Nachfolgenden in ihrer Verwendung und technischen Nutzbarkeit untersucht und bewertet.

Android vor 5.0

Im Berechtigungsmodell der Android-Versionen vor Version 5.0 kann eine App ausschließlich dann installiert werden, wenn der Nutzer zur Zeit der Installation einer App zustimmt, dass alle von der App angeforderten Permissions gewährt werden. Dies bedeutet insbesondere, dass keine App mit nur teilweise akzeptierten Permissions installiert werden kann. Die Möglichkeit, bestimmte Permissions nach Installation zu verweigern, wird zwar durch bestimmte Custom ROMs und auch in einigen Versionen als offizielles Feature von Android („AppOps“) bereitgestellt, ist aber nicht weitläufig verfügbar und wird deshalb hier nicht weiter betrachtet.

Die für das Bereitstellen einer API besonders wichtigen Custom-Permissions haben bei Android Versionen vor 5.0 (Lollipop) einige Eigenschaften, die eine Nutzung unsicher und damit wenig empfehlenswert machen. Konkret hat jede App die Möglichkeit, Custom-Permissions zu definieren. Dabei legt sie Eigenschaften für die Anzeige (Name, Beschreibung, ...) sowie den Sicherheitstyp der Permission fest. Auf diese Weise definierte Permissions können danach von anderen Apps regulär angefordert werden und unterscheiden sich in ihrer weiteren Behandlung nicht von anderen Permissions.

Die Definition und Anforderung einer Permission können unabhängig voneinander erfolgen. Es ist also möglich, sowohl eine Permission zu definieren (ohne sie selbst anzufordern) als auch sie anzufordern (ohne sie selbst zu definieren). Eine Besonderheit tritt auf, wenn eine App eine Permission anfordert, aber nicht definiert, welche zu diesem Zeitpunkt auch von keiner anderen App definiert wurde. Da die erforderlichen Eigenschaften zur Anzeige der Permission für den Nutzer nicht vorliegen, kann die App in diesem Fall nur ohne die Permission installiert werden. Ein späteres Anfordern der Permission nach Installation der definierenden App ist nicht möglich und kann nur durch eine Neuinstallation erfolgen. Dies erzeugt ein Usability-Problem für Apps, welche optional die Verwendung einer durch eine andere App bereitgestellte Schnittstelle unterstützen.

Wird eine Permission von mehreren Apps definiert, gilt die Definition derjenigen App, welche zum früheren Zeitpunkt installiert wurde. Dies gilt insbesondere auch für das Protection Level der Permission, welches (s.o.) definiert, wie diese von anderen Apps angefordert werden kann. Eine böswillige App hat damit die Möglichkeit, Permissions mit einem niedrigeren Zugriffslevel zu definieren, beispielsweise indem sie eine Permission, welche für das Protection Level *signature* vorgesehen war, als *normal*-Permission definiert. Wird diese böswillige App vor der rechtmäßigen installiert, führt dies effektiv dazu, dass die entsprechende Permission jeder App ohne Rückfrage an den Nutzer zugänglich gemacht wird.

Android 5.0

Unter Android 5.0 (Lollipop) wurde das Berechtigungssystem gegenüber den Vorgängerversionen hinsichtlich des oben genannten Security-Problems nachgebessert. Ab dieser Version müssen zwei Apps, welche dieselbe Permission definieren, von demselben Entwickler-Schlüssel signiert sein, um gleichzeitig installiert sein zu können. Ist bereits eine App installiert, die eine Permission definiert, kann eine App, welche dieselbe Permission definieren soll (aber mit einem anderen Entwickler-Schlüssel signiert ist) nicht mehr installiert werden. Da eine doppelte Belegung von Permissions so gut wie ausgeschlossen ist, behebt dieses neue Verhalten das Sicherheitsproblem durch das frühzeitige Definieren einer Permission mit reduziertem Sicherheitslevel.

Weiterhin ungelöst bleibt das oben genannte Usability-Problem. Fordert eine App eine Permission an bevor diese definiert ist, erhält sie diese nicht und kann sie auch zu keinem späteren Zeitpunkt außer durch Neuinstallation erhalten. Da eine redundante Definition nicht möglich ist, bedeutet dies in der Praxis, dass die App (welche eine Permission nutzt) stets nach der App (welche sie bereitstellt) installiert werden muss. Ausnahme sind hier Apps, welche vom gleichen Entwickler-Schlüssel signiert sind. Hier kann die Permission ohne Probleme redundant definiert werden, was für den Zweck der Interoperabilität zwischen Apps verschiedener Hersteller allerdings keine Option ist.

Android 6.0

Die Neuerungen der Version 6.0 (Marshmallow) von Android bringt eine grundlegende Veränderung des Berechtigungssystems unter dem Namen „Runtime Permissions“ mit sich. Permissions werden jetzt nicht mehr vollständig zur Installationszeit, sondern zur Laufzeit (bei konkreter Verwendung der entsprechenden Berechtigung) abgefragt. Dabei werden auch keine individuellen Permissions abgefragt, sondern lediglich Gruppen verwandter Permissions, die beispielsweise den Zugriff auf Kontakte, den Kalender, die Kamera, Ortungsverfahren oder Sensoren zusammenfassen.

Die Möglichkeit zur Definition eigener Permissions bleibt unverändert. Allerdings kann unabhängig von der Reihenfolge der Installation verschiedener Apps eine Permission jetzt zum Zeitpunkt ihrer ersten Nutzung angefragt werden, bei ggf. vorheriger Installation der bereitstellenden App. Dies löst elegant das Usability-Problem, welches in den vorherigen Versionen bestand, und macht die Verwendung eigener Permissions unter dieser Version uneingeschränkt nutzbar.

Fazit Android Permissions

Aufgrund verschiedener Probleme ist die Nutzung eigener Permissions für eine OpenPGP-App erst unter Android 6.0 (Marshmallow) uneingeschränkt zu empfehlen. Diese Version ist noch sehr neu (Oktober 2015) und hat einen dementsprechend geringen Nutzeranteil. Um eine hohe Reichweite zu gewährleisten, sollten zum aktuellen Zeitpunkt eine Rückwärtskompatibilität bis Android 4.1 gewährleistet sein, womit ein Anteil von ca. 94% aller Devices (Stand: Dezember 2015, siehe Abschnitt 3.6.2) abgedeckt wird.

Ein weiteres Problem bei der Nutzung der systemeigenen Berechtigungen ist der Mangel an Stabilität über die verschiedenen Android-Versionen. Jedes Major-Release der letzten Jahre brachte signifikante Änderungen mit sich, was die Zukunftssicherheit und damit eine Nutzung für sicherheitskritische Komponenten infrage stellt.

3.3.4.2 Eigenes Berechtigungsmanagement

Das unter Android für IPC genutzte Binder-Framework ermöglicht allen Kommunikationsteilnehmern, die Apps, mit welchen sie kommunizieren, eindeutig und zuverlässig zu identifizieren und auch wiederzuerkennen. Dabei können ebenfalls alle wichtigen Eigenschaften der App abgefragt werden, wie der Packagename und die „Signature“ der App.

Der Begriff „Signature“ wird vom Android-Framework etwas fehlleitend verwendet. Es handelt sich dabei präziser ausgedrückt um ein Self-Signed Certificate. In diesem Dokument wird dennoch weiterhin der Begriff Signature verwendet, um dem Android-Terminus gerecht zu bleiben. Die Signature einer App ist der eindeutige Entwickler-Schlüssel, mit welchem die gesamte App signiert ist. Dieser ist zwar nicht durch eine Chain of Trust (wie X.509 Certificate Authorities) abgesichert, kann aber verwendet werden, um eine spezifische App eindeutig und unabhängig von ihrer Version zu identifizieren und insbesondere wiederzuerkennen.

Es ergibt sich die Möglichkeit eines einfachen Kontrollflusses: Greift eine App das erste Mal auf die bereitgestellte Schnittstelle zu, wird vom Nutzer erfragt, ob dieser App der Zugriff erlaubt werden soll. Ist dies der Fall, wird ein Vermerk in einer internen Datenbank hinterlegt, welche der App anhand ihres Paketnamens und ihrer Signature weiterhin den Zugriff erlaubt. Dieser Kontrollfluss bildet das Konzept der *signature*-Permissions nach, ist dabei aber flexibler und unabhängig von der verwendeten Android-Version.

3.4 Eigenständige kryptografische Operationen (F4)

3.4.1 Eigenständige Operationen (F4.1)

Dem Nutzer sollten in der Benutzeroberfläche der OpenPGP-App die eigenständigen Operationen, die das OpenPGP-Protokoll vorsieht, ermöglicht werden. Hierzu zählen insbesondere die Ver- und Entschlüsselung von Dateien sowie das Signieren und Verifizieren von Texten.

3.4.2 Integration in Android (F4.2)

Neben der Verfügbarkeit der eigenständigen kryptografischen Operationen in der Nutzeroberfläche der App selbst sollten diese Operationen über die vom Betriebssystem bereitgestellten Mechanismen nutzbar sein. Zu diesem Zweck gibt es drei zentrale Mechanismen:

- Die „Share“-Operation, ausgeführt durch den `ACTION_SEND`-Intent, ist eine Methode, die unter Android für das allgemeine Versenden von Daten im weitesten Sinne genutzt wird. Die OpenPGP-App stellt hierbei keinen Endpunkt im Sinne eines Empfängers dar, kann aber als Element einer Pipeline betrachtet werden, indem die ver-/entschlüsselten Daten wiederum mittels eines Share-Intents weitergegeben werden. Die Verknüpfung wird bei diesem Intent über den MIME-Type hergestellt.
- Die „View“-Operation, entsprechend dem `ACTION_VIEW`-Intent, ist für das lokale Betrachten gedacht. Diese Operation sollte auf verschlüsselte Dateien sowie Schlüsselmaterial anwendbar sein. Die Verknüpfung sollte hier zusätzlich zum MIME-Type der Datei über ihre Dateiendung erfolgen.
- Neben den expliziten Operationen kann eine App sich auf Schemata von URIs registrieren. Hier sollte mindestens das `openpgp4fpr`-Schema behandelt werden. Ebenfalls denkbar sind Verweise auf öffentliche Schlüssel, wie sie häufig auf persönlichen Websites verwendet werden.

Neben diesen drei Mechanismen, die auf jeden Fall unterstützt werden sollten, gibt es weitere von weniger Bedeutung, von denen zwei der Vollständigkeit halber erwähnt werden:

- Ein einfacher Mechanismus ist die Zwischenablage, mit der einfach Daten zwischen Apps weitergegeben werden können. Dieser Mechanismus hat den Vorteil, dass er für viele Nutzer ein vertrautes Nutzungsmuster (*Strg-C, Strg-V*) verwendet. Ein Nachteil ist allerdings, dass für den Zugriff auf die Zwischenablage keine Permission benötigt wird. Es muss also davon ausgegangen werden, dass die dort hinterlegten Daten durch jede installierte App einseh- und modifizierbar sind. Aus diesem Grund sollten nur signierte Daten über die Zwischenablage geteilt bzw. verarbeitet werden, niemals aber Passwörter oder entschlüsselte Daten.

- Ein neues Feature unter Android 6 ist der `ACTION_PROCESS_TEXT`-Intent. Dieser erlaubt das zusätzliche Bereitstellen von Operationen auf markierten Text neben Kopieren, Ausschneiden und Einfügen. Damit ist es beispielsweise möglich, geschriebenen Text in einer E-Mail vor Versand zu verschlüsseln, ohne Unterstützung von der E-Mail-App zu benötigen. Diese Methode beruht auf der manuellen Selektion von Text durch den Nutzer (sowohl zur Ver- als auch zur Entschlüsselung) und ist damit sehr flexibel. In Bezug auf E-Mails ist diese Variante nur mit dem PGP/INLINE-Format nutzbar, da dem Nutzer in einer E-Mail im PGP/MIME-Format die verschlüsselten Daten niemals als Text angezeigt werden, sondern ggf. als Anhang. Dies bedeutet für einen regelmäßigen Gebrauch aber auch einen deutlich erhöhten Aufwand und damit eingeschränkte Nutzbarkeit gegenüber einer integrierten Unterstützung. Aufgrund der absehbaren Verfügbarkeit von integrierten Lösungen (sowohl für das E-Mail- als auch das Instant-Messaging-Nutzungsszenario) und der relativ geringen Verbreitung von Android 6 ist auch dieses Feature nur optional.

3.5 Sicherheitseigenschaften (NF5)

3.5.1 Sicherheit des privaten Schlüssels (NF5.1)

Eine OpenPGP-App unterliegt naturgemäß erhöhten Sicherheitsanforderungen – ein Feld bei dem mobile Endgeräte viele Besonderheiten aufweisen. Insbesondere die Sicherheit des privaten Schlüsselmaterials in möglichst weitreichenden Angriffs-Szenarien sollte gewährleistet sein, ebenso wie die unverschlüsselten Daten vor und nach kryptografischen Operationen. Auch die Liste an öffentlichen Schlüsseln kann je nach Nutzerszenario als vertraulich eingestuft werden.

Das private Schlüsselmaterial des Nutzers kann direkt auf dem Gerät erzeugt, oder aus einer verschlüsselten Datensicherung der App wiederhergestellt werden. Es besteht weiterhin die Möglichkeit, den privaten Schlüssel von einer anderen Implementierung zu importieren (siehe F1.1 und F1.3). Es gibt allerdings kein vorgesehenes Format im OpenPGP-Standard für eine Übertragung von privatem Schlüsselmaterial, das Vertraulichkeit und Integrität gewährleistet. Aus diesem Grund sollte dem Nutzer für dieses Nutzungsszenario eine Methode zur sicheren Datenübertragung empfohlen werden. Die Ausarbeitung einer solchen Methode geht allerdings über dieses Dokument hinaus.

Mobile Endgeräte sind von einer Vielzahl allgemeiner Angriffsszenarien betroffen, zusätzlich aber auch von einigen Speziellen die auf anderen Geräteformen nicht oder weniger relevant vorkommen:

- Durch ihren kleinen Formfaktor und das Nutzungsszenario, dass das Gerät häufig am Körper mitgeführt wird, sind mobile Endgeräte einfacher durch Diebstahl zu entwenden.
- Die unachtsame Verwendung eines Gerätes in der Öffentlichkeit erlaubt das sogenannte „Shoulder Surfing“, bei welchem der Angreifer einen Nutzer während der Eingabe geheimer Daten (insbesondere Passwörtern) unbemerkt beobachtet und sich so Zugriff auf die passwortgeschützten Daten verschafft.
- Die Verwendung von Touch Displays als Eingabemedium führt zur „Smudge Attack“, da getätigte Eingaben häufig erkennbare Spuren hinterlassen. Dies wirkt sich insbesondere bei der wiederholten Eingabe von festen Mustern aus, wie bei der Eingabe von PINs oder Passwörtern.
- Ein weiterer Angriffsvektor entsteht durch das kurzzeitige Weitergeben des Geräts, beispielsweise um Bilder oder Videos vorzuzeigen, was bei mobilen Endgeräten ein übliches Szenario ist.

Traditionell werden private Schlüssel mit einer komplexen Passphrase geschützt. Auf mobilen Endgeräten ist diese Möglichkeit aber für den Nutzer mit hoher Unannehmlichkeit verbunden, da üblicherweise keine Hardware-Tastatur verfügbar ist und viele Anwendungsfälle deutlich kurzfristiger sind als bei vergleichbaren Anwendungen auf Desktop-PCs. Die genannten Angriffsszenarien können nur bedingt durch die App verhindert werden, sollten aber in der Entwicklung konzeptuell bedacht werden.

3.5.1.1 Sicherheit der Anwendungsdaten

Die Abschirmung der Nutzerdaten jeder App ist in Android nicht nur vorgesehen, sondern wird bereits auf Betriebssystem-Ebene strikt durchgesetzt. Der Prozess jeder App läuft als ein eigener Nutzer, also mit eigener UID, und hat zunächst nur Zugriff auf ihren eigenen Internal Storage. Ein Zugriff auf den Internal Storage einer App durch andere Apps ist grundsätzlich nicht vorgesehen. Stattdessen erfolgt ein Austausch von Daten mit anderen Apps, im Vergleich zu den meisten anderen Betriebssystemen, ausschließlich auf explizite Weise. Dies passiert entweder durch die Bereitstellung von IPC-Schnittstellen oder durch das Hinterlegen von Daten im External Storage. Für die direkte Kommunikation mit anderen Apps wird das Binder-Framework als vereinheitlichte IPC-Schnittstelle verwendet und darauf aufbauende Komponenten wie das Storage Access Framework (SAF), siehe dazu Abschnitt 2.3. Für eine indirekte Bereitstellung kann eine App Daten im External Storage hinterlegen, wofür allerdings eine eigene Permission (siehe dazu Abschnitt 3.3.4) angefragt werden muss. Diese Möglichkeit ist außerdem seit Einführung des SAF nicht mehr vorgesehen und als Legacy-Schnittstelle zu verstehen.

Ein Feature, welches unter Android von verschiedenen Apps bereitgestellt wird, ist das Sichern der Anwendungsdaten anderer Apps. Zu diesem Zweck verwendete Apps nutzen dafür entweder die Backup-Funktionalität von „ADB“ (Android Debug Bridge) oder benötigen „Root“-Zugriff. Da eine OpenPGP-App sicherheitskritische Daten verwaltet, sollte die Möglichkeit eines derartigen Backups mittels ADB explizit verboten werden – durch Setzen des `androidBackup:false` Flags in der Manifest-Datei. Entsprechend **F1.3** sollte eine Backup-Funktionalität direkt durch die App unterstützt werden.

Wie üblich bei der Betrachtung von Datensicherheit, ist es notwendig zu erwähnen, dass der Zugriff auf Daten durch eine App mit Root-Zugriff nicht verhindert werden kann und damit aus der Betrachtung fällt. Da allerdings die Installation von Apps mit Root-Zugriff unter Android nicht offiziell unterstützt wird und nur mit erhöhtem Aufwand überhaupt möglich ist, kann für den allgemeinen Fall davon ausgegangen werden, dass sich die Gruppe derartiger Apps auf solche beschränkt, die vom Betriebssystem selbst bereitgestellt werden.

3.5.2 Entwicklungsprozess und Entwicklergemeinschaft (NF5.2)

Die Verfügbarkeit der Applikation als Freie Software ist keine feste Anforderung, vereinfacht aber eine Beurteilung erheblich und schafft durch die Möglichkeit unabhängiger Reviews viel Vertrauen in die Qualität der Programmquellen. Ähnliches gilt für einen offenen Entwicklungsprozess. Insbesondere durch einen öffentlichen Bug-Tracker wird sichergestellt, dass Sicherheitsprobleme zeitnah Beachtung finden oder zumindest bekannt sind. Auch die Möglichkeit, über die Android Developer Console direkt durch Endnutzern Crash-Reports zu erhalten, sollte wahrgenommen und entsprechende Reports in den Bug-Tracker übernommen werden.

Unabhängig vom Entwicklungsprozess ist es wichtig, dass die App sich in aktiver Entwicklung befindet und regelmäßige Releases erfolgen, in denen wenigstens gefundene Fehler behoben werden.

3.5.3 Korrektheit des kryptografischen Codes (NF5.3)

Für den kryptografischen Code der OpenPGP-App sollte Wert auf die Korrektheit gelegt werden. Es existieren verschiedene Methoden, die zum Vertrauen in ein korrektes Verhalten des kryptografischen Codes beitragen:

- Für die kryptografischen Algorithmen sowie für Operationen auf der OpenPGP-Paketstruktur ist es förderlich, wenn eine hohe Abdeckung an Unit-Tests angestrebt wird. Vom Gradle-Buildsystem, welches üblicherweise für Android-Projekte verwendet wird, werden Plattform-Tests auf emulierten oder angeschlossenen Testgeräten unterstützt sowie einfachere Unit-Tests, welche direkt in einer JVM ausgeführt werden (seit Android Studio 1.1).

- Um eine möglichst einheitliche Interpretation von OpenPGP-Datenpaketen in verschiedenen OpenPGP-Implementierungen zu gewährleisten, ist es nützlich, Interoperabilitäts-Tests zu verwenden. Diese sollten eine möglichst vielfältige Menge an Test-Eingaben in allen getesteten Implementierungen auf dieselbe, erwartete Ausgabe überprüfen. Dieser Punkt ist insbesondere deshalb wichtig, da der OpenPGP-Standard in vielen Punkten nicht allzu präzise Vorgaben enthält, sondern Entscheidungen im Detail der Implementierung überlässt.
- Um das Vertrauen in die Sicherheit der App in einer bestimmten Version zu erhöhen, ist es weiterhin förderlich, wenn ein Security Audit durch einen unabhängigen Dienstleister durchgeführt wird. Die dabei gefundenen Sicherheitsprobleme sollten zeitnah behoben und anschließend veröffentlicht werden.

3.6 Weitere Anforderungen (NF6)

3.6.1 Performance (NF6.1)

Mobile Endgeräte sind im Allgemeinen weniger leistungsstark als Desktop-PCs derselben Generation. Diese Einschränkung liegt nicht am Android-Betriebssystem an sich, ist aber dennoch wichtig zu betrachten beim Design einer OpenPGP-App und wird deshalb hier kurz untersucht.

3.6.1.1 Datendurchsatz symmetrischer Blockchiffren

Die wichtigste Performance-kritische Operation ist die Anwendung von symmetrischen Blockchiffren, da diese für die Ver- oder Entschlüsselung größerer Datenmengen den signifikanten Teil der Bearbeitungszeit einnehmen. Für eine einfache Übersicht wurden Benchmarks mit OpenSSL (OpenSSL 1.0.2d, keine Hardwarebeschleunigung) durchgeführt – für die Operationen AES-128 im CBC-Modus und eine fortlaufend Berechnung eines Digests mittels SHA-256.

Prozessor	Betriebssystem	Jahr	Algorithmus	Datendurchsatz
Thinkpad x200 (Core2Duo P8600)	Debian 8 (Jessie)	2008	AES128	175MB/s
Thinkpad x200 (Core2Duo P8600)	Debian 8 (Jessie)	2008	SHA256	188MB/s
Nexus 7 (Cortex-A9)	Stock Android 4.4	2012	AES128	18MB/s
Nexus 7 (Cortex-A9)	Stock Android 4.4	2012	SHA256	27MB/s

Die Ergebnisse bestätigen das Resultat, dass (wenigstens unter Benchmark-Bedingungen) die Leistung mobiler Endgeräte erheblich niedriger liegt – hier um eine Größenordnung, trotz deutlich neueren Baujahres des Nexus 7.

In einem etwas realistischeren Test wurde die in Java geschriebene OpenPGP-Implementierung von Bouncy Castle verwendet, um 10 Megabyte zufällige Daten zu verschlüsseln und entschlüsseln. Beide Operationen verwenden AES-128 als Blockchiffre im OpenPGP-spezifischen modifizierten CFB-Betriebsmodus sowie SHA-1 als Digest für einen Integritätscheck. Die Zeitmessung ist gemittelt auf insgesamt jeweils 10 Operationen:

Prozessor	Betriebssystem	Jahr	Operation	Benötigte Zeit
Thinkpad x200 (Core2Duo P8600)	Debian 8.2 (Jessie)	2008	10MB Encrypt	530ms
Thinkpad x200 (Core2Duo P8600)	Debian 8.2 (Jessie)	2008	10MB Decrypt	690ms
Thinkpad x220 (Core i7-3520M)	Debian 8.2 (Jessie)	2012	10MB Encrypt	340ms
Thinkpad x220 (Core i7-3520M)	Debian 8.2 (Jessie)	2012	10MB Decrypt	420ms
Nexus 7 (Cortex-A9)	Stock Android 4.4	2012	10MB Encrypt	7070ms

Nexus 7 (Cortex-A9)	Stock Android 4.4	2012	10MB Decrypt	9140ms
Nexus 5 (Krait 400)	Stock Android 5.1.1	2013	10MB Encrypt	3290ms
Nexus 5 (Krait 400)	Stock Android 5.1.1	2013	10MB Decrypt	3370ms

Da es sich um Rahmenbedingungen der Hardware handelt, ist es in Software natürlich nur begrenzt möglich, dieses Defizit auszugleichen. Was aber folgt, ist ein Anspruch an die Berücksichtigung beim Design und die Ansprechbarkeit des Nutzerinterfaces während der Durchführung kryptografischer Operationen bereits auf Datenmengen im niedrigen Megabyte-Bereich, welche in E-Mail-Anhängen üblich sind.

3.6.1.2 S2K

Eine Besonderheit im Bezug auf Performance stellen Key Derivation Functions (KDF) dar – im Falle von OpenPGP konkret das S2K-Verfahren, welches zur Erzeugung von Session-Schlüsseln aus Passphrases dienen. Dieses wird verwendet, um private Schlüssel mit einer Passphrase zu verschlüsseln, sowie für die symmetrische Verschlüsselung von Daten. Wie für eine KDF üblich, ist der „Work Factor“ des S2K-Verfahrens parametrisierbar durch einen „Iteration Count“, mit welchem der für eine Operation notwendige Rechenaufwand bestimmt werden kann. Dies ist gedacht als Schutz vor Offline-Angriffen, also gegen einen Angreifer der bereits symmetrisch verschlüsselte Daten abgefangen hat, aber nicht die zur Verschlüsselung verwendete Passphrase. Ein hoher Work Factor erhöht dabei effektiv die Schwierigkeit, Passwörter mit niedriger Entropie durch eine Brute-Force-Attacke zu berechnen.

Die Verwendung des S2K-Algorithmus ist deswegen auf mobilen Endgeräten interessant zu betrachten, da der verwendete Work Factor meist dynamisch so gewählt wird, dass die S2K-Operation als Ganzes einen festen Zeitraum benötigt. Beispielsweise passt GnuPG den verwendeten Iteration Count so an, dass die S2K-Operation in 100 Millisekunden durchführbar ist. Die Anzahl hierbei erreichbarer Iterationen unterscheidet sich dabei um mehr als eine Größenordnung zwischen typischen Prozessoren in mobilen Endgeräten und Desktop-PCs. Zur Übersicht wurde ein Benchmark durchgeführt, welcher auf verschiedenen Plattformen die Anzahl S2K-Iterationen misst, die innerhalb dieser Zeitspanne erreichbar sind.

Gerät / Prozessor	**Betriebssystem**	**Jahr**	**Implementierung (Single-Core)**	**Iterationen in 100ms**
Intel Core2Duo P8600	Debian 8.2 (Jessie)	2008	GnuPG 2.1.9	$11,53*10^6 $
Intel Core2Duo P8600	Debian 8.2 (Jessie)	2008	Bouncy Castle 1.51	$ 5,76*10^6 $
Intel Core i7-3520M	Debian 8.2 (Jessie)	2009	Bouncy Castle 1.51	$ 9,96*10^6 $
Nexus 7 (Cortex-A9)	Stock Android 4.4	2012	Bouncy Castle 1.51	$ 0,40*10^6 $
Nexus 5 (Krait 400)	Stock Android 5.1.1	2013	Bouncy Castle 1.51	$ 1,44*10^6 $

Neben der Beobachtung, dass eine JVM-basierte Implementierung auf demselben Rechner bereits einen Faktor 2 Geschwindigkeitsnachteil hat, ist die Differenz zu mobilen Endgeräten (auch das Baujahr betrachtend) bereits mehr als eine Größenordnung. Zwar ist eine Wartezeit von 100 Millisekunden in einer Nutzeroberfläche durchaus akzeptabel, erzeugt ein Nutzer aber beispielsweise einen privaten Schlüssel auf einem Desktop-PC und importiert ihn anschließend auf einem mobilen Endgerät, benötigt dieses für das Entschlüsseln des privaten Schlüsselmaterials die entsprechend vielfache Zeit. Eine OpenPGP-App sollte die Ansprechbarkeit der Nutzeroberfläche auch für diese Operation innerhalb von weniger als einer halben Sekunde gewährleisten, beispielsweise indem betroffene Schlüssel mit einer entsprechend niedrigeren Anzahl Iterationen neu verschlüsselt werden.

3.6.2 Verbreitungswege der App (NF6.2)

Der übliche Weg, eine App unter Android zur Verfügung zu stellen, erfolgt über den Google Play Store. Um die Verfügbarkeit einer App zu gewährleisten, ist eine Veröffentlichung durch Google Play unumgänglich. Die App muss dafür allerdings mit den entsprechenden Richtlinien kompatibel sein.

Neben Google Play existieren verschiedene alternative Plattformen für die Verbreitung von Apps unter Android. Beispiele hierfür sind der Amazon Appstore sowie das in Freier Software implementierte F-Droid. Einige dieser Stores werden durch die Vertreiber von Hardware auf den entsprechenden Geräten vorinstalliert. Zum aktuellen Zeitpunkt ist der Anteil an Android-Geräten, welche ausschließlich einen der alternativen Vertriebsplattformen enthalten, jedoch vernachlässigbar gering. Der Vertrieb einer App über alternative Stores erhöht dessen Verfügbarkeit somit nur unwesentlich und ist daher optional.

Stand 14.12.2015 sieht die Verteilung installierter Android-Versionen so aus, dass die Android-Versionen, welche älter sind als Version 4.0.X (Ice Cream Sandwich) insgesamt nur noch etwa 6% der aktiven Geräte ausmachen. Auf dem nächst höheren API-Level 4.1.X (Jelly Bean) hingegen verteilen sich noch 10% der aktiven Installationen. Diese sollte also noch unterstützt werden.

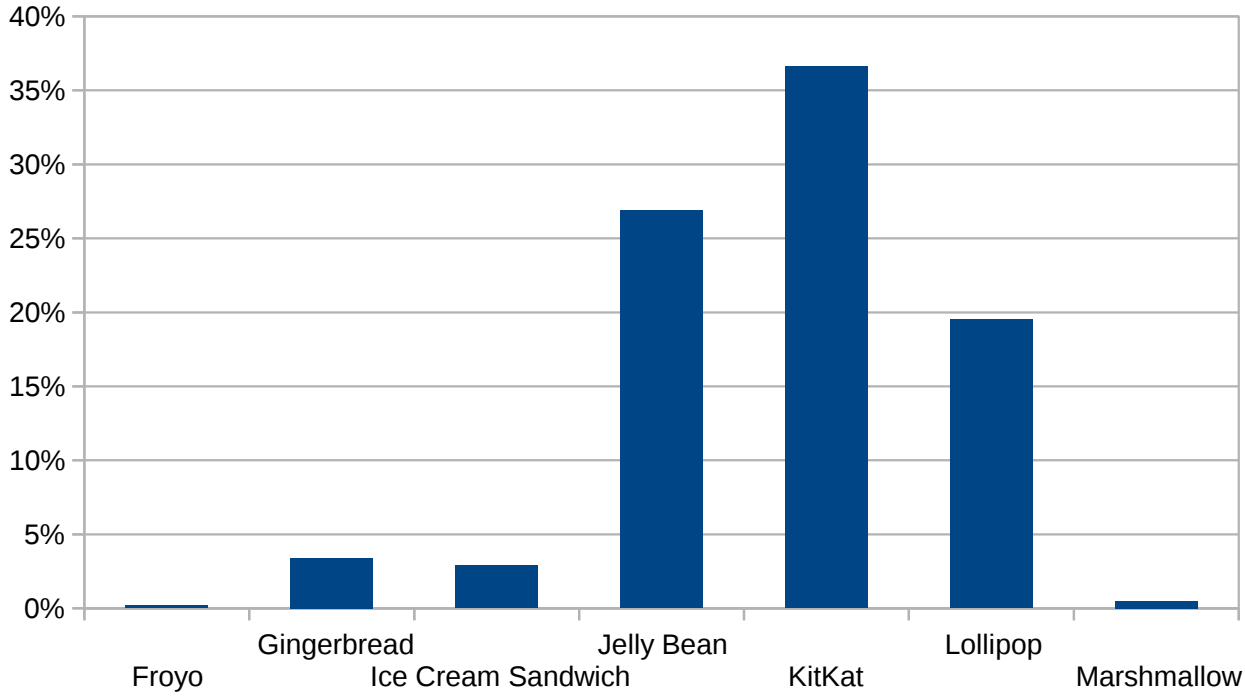


Abbildung 2: Übersicht zur Verteilung der Android-Versionen, Stand Dezember 2015

Version	Codename	API	Distribution
2.2	Froyo	8	0,2%
2.3.3 - 2.3.7	Gingerbread	10	3,4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2,9%
4.1.x	Jelly Bean	16	10,0%
4.2.x		17	13,0%
4.3		18	3,9%
4.4	KitKat	19	36,6%
5.0	Lollipop	21	16,3%
5.1		22	13,2%
6.0	Marshmallow	23	0,5%

Eine weitere zu beachtende Eigenart der Installation ist, dass sie mit dem üblichen Berechtigungslevel für Apps installiert werden können. Dazu zählt insbesondere, dass keine Root-Rechte benötigt werden, da dies

in den üblichen, durch Endgerät-Hersteller verbreiteten Versionen des Android-Betriebssystem nicht unterstützt wird. Keine der hier aufgeführten Anforderungen erfordert den Gebrauch von Root-Rechten.

Potentielle Vorteile durch die Vorinstallation von einer OpenPGP-App oder Integration einer OpenPGP-API in das Android-Betriebssystem werden in Kapitel 7 diskutiert.

4 Beschreibung vorhandener OpenPGP-Implementierungen

In diesem Kapitel werden eine Anzahl existierender Implementierungen des OpenPGP-Standards unter Android vorgestellt und betrachtet. Es werden hier Apps betrachtet, deren Hauptfunktionen die Schlüsselverwaltung und Bereitstellung einer kryptografischen API sind.

4.1 Übersicht

Die betrachteten Apps wurden ausgewählt, da sie entweder eine ausreichend große Anzahl an Installationen in Google Play aufweisen oder interessante Features oder API-Funktionalitäten anbieten, die von anderen nicht abgedeckt werden. Es wurde auf Google Play nach Suchbegriffen „PGP“ und „OpenPGP“ gesucht. Apps, die einen E-Mail-Client oder andere PGP-basierte Apps implementieren, werden in Kapitel 6 betrachtet.

Nicht betrachtet wurden die folgenden Apps:

- OpenPGP Manager (Harpagosoft), da ihre letzte Aktualisierung im Juni 2013 erfolgte, nur weniger als 5000 Installationen aufweist und keine besonderen Features hat.
- Symantec PGP Viewer (Symantec Corporation), welche einen Benutzeraccount auf einem Symantec Encryption Management Server benötigt und somit grundsätzlich nicht für die definierten Szenarien genutzt werden kann.
- PGPTools (SJ Software), PGPFiles (SJ Software), und Juggler (William R. Moore) weisen jeweils weniger als 500 Installationen auf und enthalten keine besonderen Merkmale, die eine Betrachtung dennoch rechtfertigen.

Somit bleiben die folgenden Apps, die genauer untersucht wurden (sortiert nach der Anzahl der Installationen):

App	Entwickler	Lizenz	Letztes Release	Installationen
APG (Android Privacy Guard)	'Thialfiar'	GPLv3	1.1.1 (2014-03-24)	100.000 - 500.000
Gnu Privacy Guard for Android	Guardian Project	GPLv3	0.3.1 (2014-04-07)	100.000 - 500.000
OpenKeychain	V. Breitmoser, D. Schürmann	GPLv3	3.7 (2015-11-23)	50.000 - 100.000
PGP KeyRing	A. Wasserman	proprietär	2.1 (2015-09-21)	500 - 1.000

4.2 APG (Android Privacy Guard)

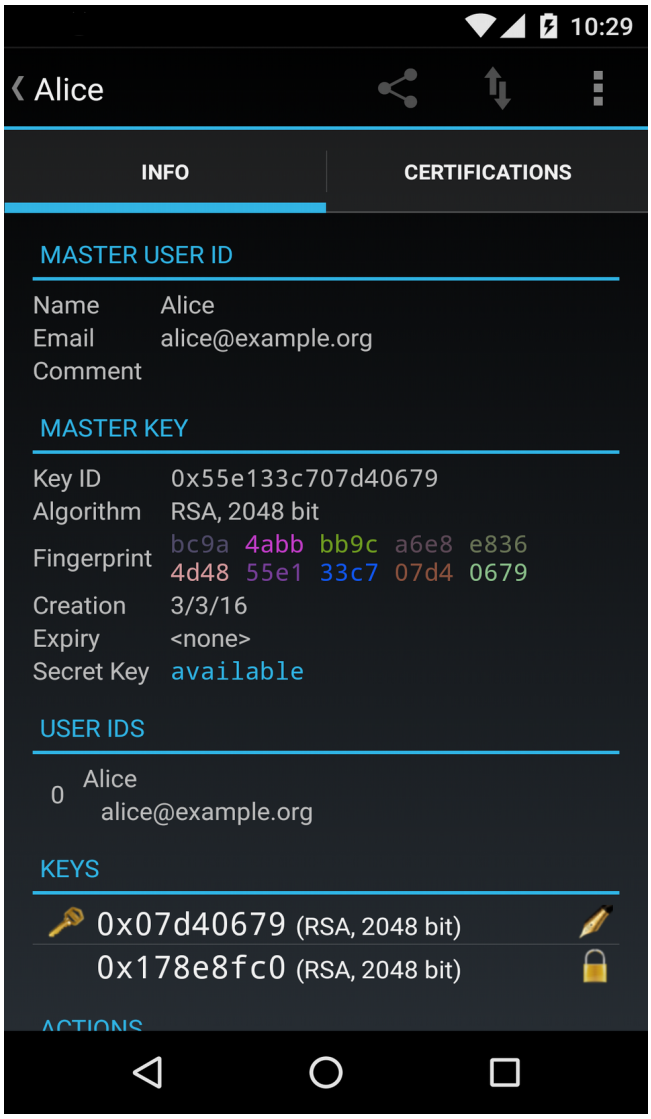


Abbildung 3: APG: Anzeige eines privaten Schlüssels

Die erste stabile Version 1.0 von APG ist im Juni 2010 erschienen. APG stellt somit die älteste untersuchte Standalone-Implementierung dar. Die Entwicklung stagnierte im Dezember 2010 mit der Version 1.0.8. Die im Oktober 2013 erschienene Version 1.0.9 enthält nur zwei Fehlerkorrekturen, wobei eine die Integration der sicherheitsrelevante PRNG-Korrektur-Klasse (vgl. Abschnitt 3.1.1) darstellt. Bis zu Version 1.0.9 wurde APG unter der Apache Lizenz v2 entwickelt. Version 1.1.0 und 1.1.1 basieren fast vollständig auf OpenKeychain, was sich nebenläufig entwickelte. APG Version 1.1.1 umfasst alle Features, die in der zum Release-Zeitpunkt aktuellen Development-Version von OpenKeychain enthalten waren. Die Unterschiede beschränken sich auf ein verändertes Farbschema sowie den Erhalt der *Intent API* (siehe unten), welche in OpenKeychain entfernt und durch die neue OpenPGP-API ersetzt wurde. Da OpenKeychain die Lizenz wechselte, steht ab diesem Release auch APG unter der GPLv3. Seit März 2014 gab es keine weiteren Releases auf Google Play oder F-Droid, somit wurden auch keine Sicherheitsprobleme behoben, die bei einem Audit von OpenKeychain gefunden wurden (siehe Abschnitt 4.4). APG ist auf Google Play und F-Droid verfügbar.

APG stellt eine Liste mit allen öffentlichen sowie privaten Schlüsseln zur Verfügung. Dieser Hauptbildschirm kann genutzt werden, um Schlüssel anzuzeigen, zu löschen oder zu exportieren. Jede einzelne Schlüsselansicht zeigt die dem Schlüssel zugehörige primäre User ID, den Hauptschlüssel mit dem

Fingerprint und alle zugehörigen User IDs an. Über das Overflow-Menü kann der Fingerprint oder der gesamte Schlüssel via QR-Codes, NFC oder Zwischenablage geteilt werden. Des Weiteren werden Möglichkeiten angeboten, den Schlüssel vom Keyserver zu aktualisieren oder ihn zu exportieren. Eigene private Schlüssel können hier editiert werden. Zu den Editierungsmöglichkeiten gehört das Ändern der Passphrase sowie das Hinzufügen von neuen User IDs und Unterschlüsseln. Die gleichen Funktionalitäten bietet APG bei der Schlüsselerzeugung an.

Texte und Dateien können direkt aus der App heraus verschlüsselt und signiert werden. Das Resultat der Verschlüsselung kann gespeichert oder mit anderen Apps über Androids `ACTION_SEND`-Intent geteilt werden. Verschlüsselter Text kann zusätzlich auch in der Zwischenablage zwischengespeichert werden. APG erlaubt eine Datei-Entschlüsselung und -Verifikation unter Nutzung eines integrierten Dialogs um Dateien zu öffnen. Eine Verifikation von Detached Signatures ist nicht möglich. Verschlüsselter Text kann aus der Zwischenablage eingefügt werden.

APG integriert sich in das Android-Betriebssystem indem es `ACTION_VIEW` mit den MIME-Types `application/octet-stream`, `application/pgp-keys` bereitstellt, sowie `ACTION_SEND`/`ACTION_SEND_MULTIPLE` mit allen MIME-Types (`*/*`) verknüpft.

APG ist komplett in Java geschrieben und nutzt die Kryptografie-Bibliothek Bouncy Castle. Da das Android-Betriebssystem schon Teile der Bouncy-Castle-API mitbringt, wurde eine umbenannte Version namens Spongy Castle integriert, um Paketkonflikte zu vermeiden. Spongy Castle wird von Roberto Tyley auf GitHub betreut. Neben Spongy Castle verwendet APG eine Anzahl weiterer Bibliotheken, die zum größten Teil spezielle Widgets für die Nutzeroberfläche bereitstellen und keine kryptografische Relevanz haben.

Die Entwicklung findet auf GitHub statt. Außer den Kommentaren im Quelltext existiert keine Dokumentation oder Wiki. APGs Webseite enthält nur wenig Informationen. Der Autor 'Thialfiar' tritt auch hier nicht mit einem Klarnamen auf.

4.2.1 Intent-API

APG stellt eine API basierend auf Activity Intents bereit. Diese unterliegt allen Einschränkungen, die Activity Intents als IPC-Mechanismus mit sich bringen. Dies gilt insbesondere, da die zu verarbeitenden Daten nicht über einen FileProvider oder ähnlichen Mechanismus, sondern direkt als Teil des Intents übergeben werden und damit einer limitierten Größe von (höchstens) 1 MB unterliegen (siehe Abschnitt 3.3.1.1). Die Intent-API wurde in früheren Versionen von K-9 Mail (siehe Abschnitt 6.1.1) implementiert. Diese Unterstützung ist in Version 5.001 von K-9 Mail jedoch entfernt und durch die OpenPGP-API ersetzt worden. Seither gibt es nach Wissen der Autoren keine App mehr, welche die Intent-API für kryptografische Operationen verwendet, weshalb diese API hier nur eingeschränkt betrachtet wird.

4.3 Gnu Privacy Guard (GnuPG) for Android

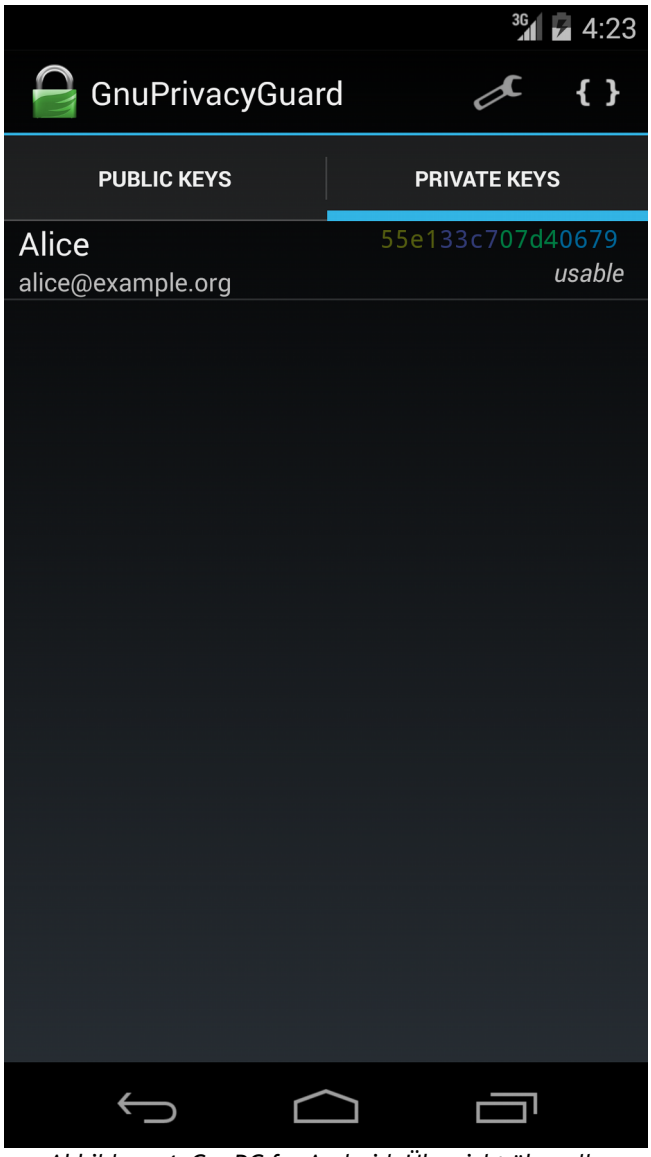


Abbildung 4: GnuPG for Android: Übersicht über alle privaten Schlüssel

Im Dezember 2011 haben die Entwickler Hans-Christoph Steiner und Abel Luck angefangen, GnuPG auf Android zu portieren. Beide Autoren gehören der Entwicklergemeinschaft Guardian Project an, die aus Spenden und projektgebundenen Geldern finanziert wird. Die letzte Version wurde im April 2014 auf Google Play und F-Droid veröffentlicht. Diese funktioniert leider nicht auf Android-Versionen ab 5.0. Der zusätzliche Installationsvorgang, der beim erstmaligen Starten ausgeführt wird, bricht ab. Außerdem ist das nativ kompilierte GnuPG nicht als Position Independent Executables (PIE) Binary kompiliert, was aber unter Android 5 vorausgesetzt wird. Diese Probleme scheinen partiell im aktuellen Entwicklungszweig auf GitHub behoben worden zu sein. Die letzten git-Commits wurden im April 2015 eingecheckt, bringen aber nur geringfügige Änderungen mit sich, die auch nicht im Play Store veröffentlicht wurden.

Die Benutzeroberfläche von GnuPG for Android ist in drei Hauptbildschirme aufgeteilt. Ein Tab enthält die Liste über alle öffentlichen Schlüssel, ein Tab alle privaten Schlüssel und der letzte Tab erlaubt die Suche nach Schlüsseln auf Schlüsselservern. Des Weiteren integriert GnuPG for Android sich in Androids Kontakte und stellt eine Verknüpfung zu den öffentlichen Schlüsseln her. Die Schlüsselsuche funktioniert und erlaubt

das Importieren ausgewählter Schlüssel in die eigene Liste. Die Funktionalitäten in der Liste an öffentlichen Schlüsseln wiederum beschränken sich auf das Hochladen sowie Aktualisieren von Schlüsselservern. Das Erstellen von neuen Schlüsseln hat auf keinem verwendeten Testgerät funktioniert. Getestete Android-Versionen umfassen Android 4.3 und 4.4. Selbst auf der Kommandozeile über `adb shell` stürzt `gpg --gen-key` mit der Fehlermeldung „`Key generation failed: End of file`“ ab. Dieser Fehler wird so auch in dem Fehler 3188 im Issue-Tracker des Guardian Projects beschrieben und ist nicht gelöst. Um Dateien mit einem öffentlichen Schlüssel zu verschlüsseln, muss der Weg über Androids Kontakte genommen werden. Nach dem Anklicken von „Encrypt file to...“, kann eine Datei gewählt werden, die nun verschlüsselt wird. Weitere Funktionalitäten sind unter dem „Debug Menu“ zu finden.

Wie einleitend erwähnt, basiert GnuPG for Android auf einer für Android kompilierten Version von GnuPG und nicht wie APG, OpenKeychain und PGP KeyRing auf einer Java-Implementierung. Da eine Entwicklung von Benutzerschnittstellen unter Android ohne die Nutzung von Java schwierig ist, setzt GnuPG for Android auf eine Hybridlösung. Android-spezifische Teile sind somit in Java implementiert und kommunizieren über Java Native Interface (JNI) mit der GPGME-Bibliothek. Die nativen Exectuables und Libraries müssen mit dem Android Native Development Kit (NDK) kompiliert werden. Das NDK enthält die Header-Dateien zu Libraries (die in AOSP integriert wurden), GCC- und Clang-Cross-Compiler sowie Buildtools und Skripte, um den Prozess der Kompilierung zu vereinfachen. Die Entwicklung wird von Google betreut und das NDK kann von der offiziellen Android-Webseite kostenlos heruntergeladen werden. Im Falle von GPGME wird nicht das vom NDK bevorzugte Buildsystem unter Nutzung von Android.mk und Application.mk Build-Dateien genutzt. Stattdessen existiert ein spezifisches Makefile, welches das NDK als Standalone Toolchain einbindet. Der JNI-Code hingegen wird, wie von Google empfohlen, gebaut. Die Nutzung von nativem Code bedeutet, dass für jede Architektur cross-kompiliert werden muss. Im Moment unterstützt GnuPG for Android nur ARM-Geräte. x86- und MIPS-Geräte werden aktuell von den Build-Dateien von GnuPG for Android nicht unterstützt. Die Abhängigkeiten curl, gnupg, gpgme, libassuan, libassuan, libgcrypt, libgpg-error, libksba, npth und openldap werden fast unverändert eingebunden. Die Inklusion erfolgt über git-Submodules im Unterordner „external“; kleinere Patches finden sich dort ebenfalls. Pinentry wurde geforkt und so abgeändert, dass es unter Android eine neue Activity aus dem Hintergrund startet.

Während die git-Repositories auf GitHub gehostet sind, werden Issue-Tracker und Wiki durch die Infrastruktur des Guardian Projects bereitgestellt. Das Wiki enthält geplante Features und Diskussionen in einer stichpunktartigen Form. Die Mailingliste `guardian-dev@lists.mayfirst.org` des Projektes wird genutzt, um zukünftige Entwicklungen aller Unterprojekte des Guardian Projects zu diskutieren. Hier finden auch übergreifende Diskussionen mit nicht-angehörigen Projekten statt und sind gewünscht.

4.4 OpenKeychain

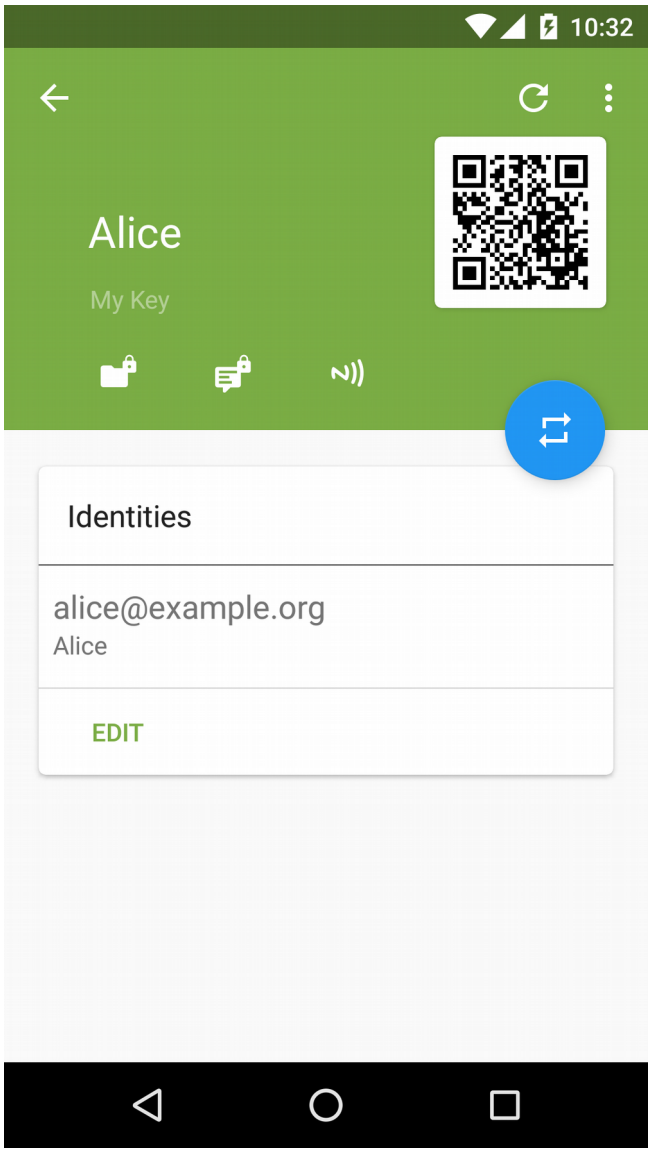


Abbildung 5: OpenKeychain: Anzeige eines privaten Schlüssels

Wie bereits in der Beschreibung von APG eingeführt, entstand OpenKeychain als Fork von APG im März 2012. Die erste Version 2.0 wurde im Januar 2013 veröffentlicht. Der Vetrieb erfolgt seither über den Play Store sowie auf F-Droid. Eine der wichtigsten Änderungen in der aktuellen Version 3.7 ist die Schließung einer Reihe von Sicherheitslücken, die bei einem Sicherheitsaudit der Firma Cure53 gefunden wurden. Wie auch APG, wird OpenKeychain in Java entwickelt und nutzt die Spongy Castle Library, um kryptografische Funktionen zu implementieren.

Dadurch, dass die letzte Version von APG auf OpenKeychain basiert, hat OpenKeychain die selben Grundfunktionalitäten – ist dem APG-Entwicklungsstand aber ca. 1,5 Jahre voraus. OpenKeychains Oberfläche wurde, basierend auf den Material Design Guidelines, grundlegend umgestaltet. Die Oberflächen, um Schlüssel anzuzeigen und zu editieren sowie die Ver- und Entschlüsselungsbildschirme wurden aktuellen Richtlinien angepasst. Viele Funktionalitäten, die sich vorher in der normalen Anzeige eines Schlüssel befunden haben, wurden in eine erweiterte Darstellung verschoben, wenn sie für den durchschnittlichen Nutzer als nicht relevant eingestuft wurden. So ist die Liste von Unterschlüsseln sowie

die Liste von Beglaubigungen nun in dieser erweiterten Darstellung zu finden. Dies und weitere Verbesserungen führen zu einer vereinfachten Nutzbarkeit für die Anwender. Zu den neuen Funktionalitäten gehört die Unterstützung von Elliptischen Kurven sowie Security Tokens, wie den YubiKey NEO. Der YubiKey NEO ist ein Token in der Größe eines USB-Sticks, der den OpenPGP-Standard unterstützt und über NFC kommunizieren kann. OpenKeychain kann unter Nutzung dieser Tokens Nachrichten/Dateien signieren sowie entschlüsseln. Neben dem Schlüsselaustausch über QR-Codes und NFC wird nun ein Austausch über das SafeSlinger-Protokoll angeboten. Dieses erlaubt einen Austausch von Schlüsseln von bis zu 10 Personen und kann somit klassische Key-Signing-Parties ersetzen. Wie auch in GnuPG for Android sind Schlüssel mit Androids Kontakten verknüpft, was es dem Nutzer erlaubt, von der Kontaktliste aus, zu den Schlüsseln zu navigieren. Damit Widerrufszertifikate rechtzeitig die Nutzer erreichen, wurde eine periodische Aktualisierung über einen Hintergrund-Dienst implementiert. Diese Aktualisierung kann optional auch über Tor oder andere Proxy-Server erfolgen. Um E-Mail-Applikationen rudimentär zu unterstützen, die die angebotene API bisher nicht direkt nutzen, wurde ein MIME-Parser in OpenKeychain integriert. Somit können MIME-Anhänge aus dem Clienten heraus mit OpenKeychain geöffnet werden, was nach einer Entschlüsselung Text und Anhänge in OpenKeychain darstellt. Durch Unit-Tests und Continuous Integration wurde die automatische Testbarkeit von OpenKeychain wesentlich verbessert. Die in der Einleitung erwähnten geschlossenen Sicherheitslücken umfassen die Implementierung von HKPS-Schlüsselservern (HKP über TLS), das Pinnen von TLS-Zertifikaten sowie Prüfungen, ob unsichere veraltete Kryptografie verwendet wurde. Die Intent-API von APG wurde entfernt, stattdessen wurde eine AIDL API entwickelt und umgesetzt, die von zahlreichen anderen Apps (wie beispielsweise K-9 Mail, Conversations oder Password Store) genutzt wird.

OpenKeychain bietet eine Webseite mit ausführlichen FAQs und Kontaktmöglichkeiten (wie Mailingliste und IRC Channel). Die Entwicklung findet auf GitHub statt, wo Bug-Reports verfasst werden können und auch weitere technische Details im Wiki zu finden sind.

4.5 PGP KeyRing

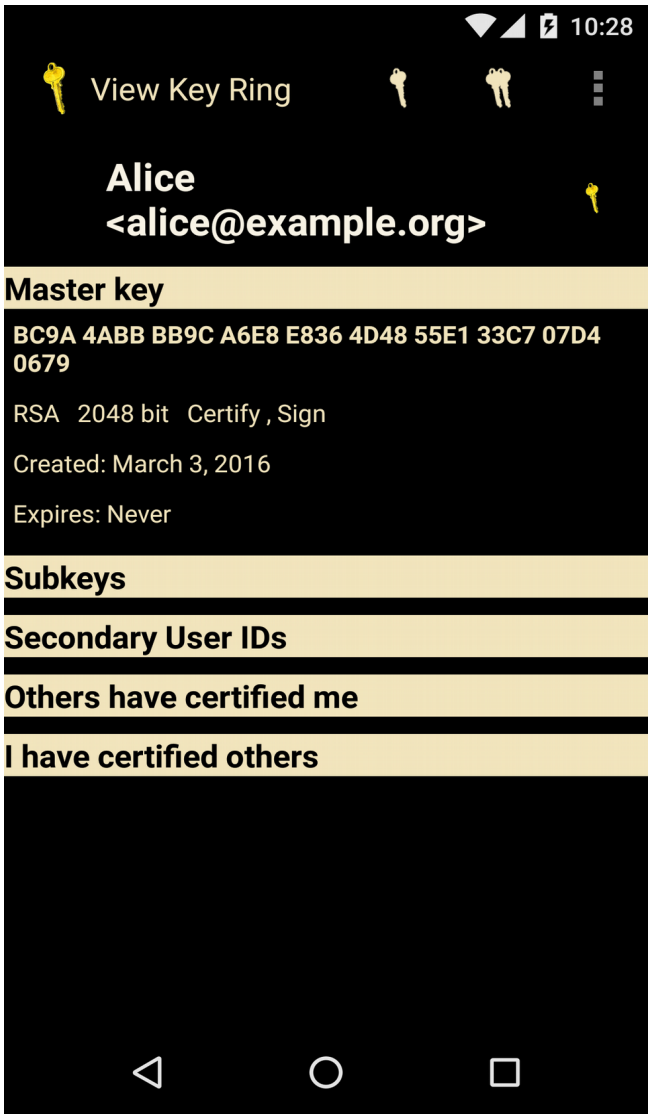


Abbildung 6: PGP KeyRing: Anzeige eines privaten Schlüssels

PGP KeyRing ist die einzige untersuchte Standalone-App, die keine Freie Software ist und proprietär vertrieben wird. Somit ist sie auch nur auf Google Play zu erwerben und nicht auf F-Droid verfügbar. PGP KeyRing ist in Java geschrieben und unterstützt alle wichtigen Funktionalitäten einer OpenPGP-Implementierung. Beim erstmaligen Start wird man mit einem kurzen Einleitungstext begrüßt, der auf die wichtigsten Funktionen hinweist. Im Overlow-Menu können viele Funktionen, wie die Erstellung von neuen privaten Schlüsseln, ausgewählt werden. Öffentliche Schlüssel können von Schlüsselservern heruntergeladen und anschließend beglaubigt werden. Es bietet alle nötigen Funktionalitäten, um private Schlüssel zu verwalten (wie beispielsweise das Hinzufügen von neuen User IDs, Unterschlüssel oder das Widerrufen). Im Vergleich zu den anderen untersuchten Anwendungen wird das Web-of-Trust unterstützt, d.h. es kann die Vertrauenswürdigkeit einzelner Schlüssel festgelegt werden, sowie Einstellungen vorgenommen werden, ab welcher Mindestgrenze an Zertifikaten einem Schlüssel vertraut wird. Außerdem wurde die Erstellung sowie Verifikation von Detached Signatures implementiert, was bei den anderen Apps fehlt. Weitere interessante Features, die in anderen Implementierungen fehlen, ist die Übertragung von öffentlichen Schlüsseln per Bluetooth. Leider hält sich PGP KeyRing nicht an Material Design Guidelines

und weicht stark vom normalen Design anderen Apps ab. Was im Vergleich zu anderen Apps fehlt, ist der Austausch von Schlüsseln durch QR Codes und die Unterstützung von Security Tokens.

Da sich keine Informationen über die verwendeten Libraries in der App oder auf der Homepage finden, wurde die App dekompiliert. Es zeigt sich, dass Spongy Castle genutzt wird, sowie Apache Commons. Auch wurden die PRNGFixes von Google (siehe Abschnitt 3.1.1) eingebunden. Die bereitgestellte API besteht aus Activity Intents und einem Service, der über IPC aufgerufen werden kann und durch eine Custom Android Permission geschützt ist. Diese API wird im Moment ausschließlich von Squeaky Mail genutzt, was ein K-9-Mail-Fork ist, der vom selbigen Autor gepflegt wird.

Informationen zu PGP KeyRing finden sich ausschließlich auf Google Play und auf der dazugehörigen Webseite. Es existiert kein öffentlicher Bug-Tracker oder eine Mailingliste. Support ist ausschließlich über die E-Mail-Adresse des Autors möglich.

5 Bewertung vorhandener OpenPGP-Implementierungen

In diesem Kapitel werden die im vorherigen Kapitel 4 betrachteten OpenPGP-Implementierungen hinsichtlich der Erfüllung der erarbeiteten Anforderungen (Kapitel 2 und 3) evaluiert.

5.1 Übersicht

Die in diesem Kapitel diskutierte Tabelle gibt eine Übersicht über die Evaluationsergebnisse und basiert auf den folgenden funktionalen und nichtfunktionalen Anforderungen dieser Studie:

- **F1**: Verwaltung privater Schlüssel (vgl. 3.1)
  - **F1.1**: Erstellen eines privaten Schlüssels
  - **F1.2**: Modifizieren eines privaten Schlüssels
  - **F1.3**: Sicherung und Wiederherstellung privater Schlüssel
- **F2**: Verwaltung öffentlicher Schlüssel (vgl. 3.2)
  - **F2.1**: Suche und Import öffentlicher Schlüssel
  - **F2.2**: Beglaubigung öffentlicher Schlüssel
  - **F2.3**: Explizite Aktualisierung öffentlicher Schlüssel
  - **F2.4**: Periodische Aktualisierung öffentlicher Schlüssel
  - **F2.5**: Vertrauensmodell
  - **F2.6**: Synchronisierung des Vertrauenszustands
- **F3**: Schnittstelle für externe Apps (vgl. 3.3)
  - **F3.1**: Schnittstelle
  - **F3.2**: Kontrolle über Kontrollfluss der Nutzeroberfläche
  - **NF3.3**: Bearbeitung der Daten als Datenstrom
  - **F3.4**: Berechtigungssystem
- **F4**: Eigenständige kryptografische Operationen (vgl. 3.4)
  - **F4.1**: Eigenständige Operationen
  - **F4.2**: Integration in Android
- **NF5**: Sicherheitseigenschaften (vgl. 3.5)
  - **NF5.1**: Sicherheit des privaten Schlüssels
  - **NF5.2**: Entwicklungsprozess und Entwicklergemeinschaft
  - **NF5.3**: Korrektheit des kryptografischen Codes
- **NF6**: Weitere Anforderungen (vgl. 3.6)
  - **NF6.1**: Performance
  - **NF6.2**: Verbreitungswege der App

App	F1.1	F1.2	F1.3	F2.1	F2.2	F2.3	F2.4	F2.5	F2.6	F3.1	F3.2	NF3.3	F3.4	F4.1	F4.2	NF5.1	NF5.2	NF5.3	NF6.1	NF6.2
GnuPG for Android	⚠	○	◐	●	○	○	○	○	○	◐	○	○	○	⚠	⚠	◐	⚠	◐	●	◐
OpenKeychain	●	●	●	●	●	●	●	●	◐	●	●	●	●	●	●	●	●	◐	◐	●
PGP KeyRing	●	●	◐	●	●	○	○	●	○	●	○	○	◐	●	○	◐	○	○	◐	◐

Die Evaluation ist aufgespalten in vier Ergebniszustände:

- Eine erfüllte Anforderung (●) wird von der App vollständig umgesetzt.
- Bei einer teilweise erfüllten Anforderung (◐) hingegen fehlen wesentliche Teile, etwa wenn ein Feature zwar technisch vorhanden, aber eine Nutzung in der Benutzeroberfläche nur mit Einschränkungen möglich ist.
- Eine nicht erfüllte Anforderung (○) beschreibt ein nicht vorhandenes Feature.
- Bei einer fehlerhaften Anforderung (⚠) hingegen ist das Feature zwar vorhanden, aber beispielsweise Aufgrund eines persistenten Absturzes unbenutzbar.

In der Evaluation wird APG aus folgenden Gründen nicht weiter betrachtet: Bei APG handelt es sich um eine zwei Jahre altes Release von OpenKeychain, wobei die letzte Version von APG schon auf OpenKeychains Quellcode basierte. Es findet keine aktive Weiterentwicklung auf GitHub statt – seit Mai 2014 wurden keine neuen Releases auf Google Play oder F-Droid veröffentlicht und Probleme, die im Sicherheitsaudit von Cure53 beschrieben sind, wurden nicht behoben. APG stellt weiterhin eine auf Activity-Intents basierende Intent-API bereit. Da diese API aktuell von keiner Client-App mehr implementiert (siehe Abschnitt 4.2) wird, kann sie jedoch als obsolet betrachtet werden. Des Weiteren ist in APG teilweise Unterstützung für die von OpenKeychain implementierten OpenPGP-API (siehe Abschnitt 5.3.3) vorhanden. Diese ist allerdings aufgrund von Fehlern, die unweigerlich zu Abstürzen führen, in der Praxis nicht nutzbar.

5.2 GnuPG for Android

Viele Operationen in GnuPG for Android verbergen sich hinter dem „Debug Mode“ im Optionsmenü. Die dort durchführbaren Operationen werden nur teilweise mit grafischer Oberfläche durchgeführt, insbesondere besteht ihre Ausgabe lediglich aus einer Auflistung des von GnuPG ausgegebenen Textes. Diese Operationen sind damit zwar technisch vorhanden, für den üblichen Nutzer jedoch nicht verwendbar.

5.2.1 Verwaltung privater Schlüssel (F1)

Im Test ist es den Autoren nicht gelungen, mit GnuPG for Android auf einem Nexus 7 unter Android 4.4 ein Schlüsselpaar zu erstellen. Bemerkenswerterweise stellt die Nutzeroberfläche diese Operation nur hinter der Menüoption „Debug Mode“ überhaupt zur Verfügung, hinter der sich unter anderem die „Create Key“ Option verbirgt. Im folgenden Dialog kann man Daten zur Nutzer-Identität eingeben, es folgt eine Abfrage zur Passphrase. Beim eigentlichen Erstellen eines Schlüssels bleibt die App dann in einem modalen Dialog mit der Meldung „This process can currently take up to 15 minutes“ hängen. Der Vorgang wurde im Test nach 45 Minuten abgebrochen.

Ein Export von Schlüsseln ist möglich, aber ebenfalls nur im „Debug Mode“ zu finden. Der Export sieht keinen zusätzlichen Schutz des Schlüsselmaterials vor.

Eine Nutzeroberfläche für das Editieren von privaten Schlüsseln ist nicht vorhanden.

5.2.2 Verwaltung öffentlicher Schlüssel (F2)

Die Nutzeroberfläche unterstützt einen Import von öffentlichen Schlüsseln, der von der Nutzeroberfläche her rudimentär ist, aber wie erwartet funktioniert. Eine Beglaubigung sowie die manuelle oder automatische Aktualisierung von Schlüsseln wird hingegen nicht unterstützt.

Aufgrund der Nutzung von GnuPG als Basis werden die entsprechenden unterstützten Vertrauensmodelle theoretisch auch von GnuPG for Android unterstützt. Diese sind allerdings nicht in die Nutzeroberfläche integriert.

5.2.3 Schnittstelle für externe Apps (F3)

GnuPG for Android bietet als Schnittstelle für externe Apps eine Intent-API an, welche dasselbe Interface unterstützt wie APG. Die Schnittstelle unterstützt die erforderlichen Operationen. Aufgrund des Designs als Intent-API wird allerdings keine der weiterführenden Anforderungen (Nutzerinterations-Kontrollfluss, Berechtigungsmanagement, Streambarkeit) erfüllt.

5.2.4 Eigenständige kryptografische Operationen (F4)

Die App unterstützt die erwarteten eigenständigen kryptografische Operationen, welche außerdem an verschiedenen Stellen in das Betriebssystem integriert sind. Die Verschlüsselung wurde getestet, indem in der Contacts-App von Android auf GnuPGs Eintrag „Encrypt file to…“ geklickt wurde und anschließend eine Datei zum Verschlüsseln ausgewählt wurde. Ein anderer Weg, Dateien zu verschlüsseln, konnte nicht gefunden werden. Die Entschlüsselung wurde getestet, indem die vorher verschlüsselte Datei in einem Dateimanager geöffnet wurde. Nachdem GnuPG ausgewählt wurde, konnte die Datei erfolgreich entschlüsselt werden. Die Benutzeroberfläche stellt sich hier als sehr spartanisch heraus, da keine Icons oder weiteren Informationen zum Status der Verschlüsselung oder Signatur angezeigt werden.

5.2.5 Sicherheitseigenschaften (NF5)

Für die Sicherheit von GnuPG for Android kann die Android-App getrennt von GnuPG als Unterbau betrachtet werden.

GnuPG selbst wird seit langer Zeit in verschiedenen Umgebungen produktiv verwendet, hat weitreichende Unit-Tests, und wird aktiv und offen entwickelt und betreut. Die Nutzung von GnuPG als Unterbau ist aufgrund seiner Stabilität gut geeignet, Fehler auf kryptografischer Ebene zu vermeiden.

GnuPG for Android selbst wird seit 2 Jahren nicht mehr aktiv entwickelt oder betreut, eine Weiterentwicklung ist aktuell nicht in Planung. Insbesondere führt ein Aufruf der App auf Android-Versionen, welche neuer sind als 5.0, unweigerlich zu einem Absturz. Die App ist aus diesem Grund nicht in der Praxis nutzbar und wird hier nur historisch für ihre Architektur betrachtet. Damit sind etwaige Sicherheitsaspekte, die auf der tragenden Entwicklergemeinschaft beruhen, nicht mehr relevant. Das Konzept der damaligen Entwicklergemeinde war vergleichbar mit dem von GnuPG. Eine Implementierung von Tests oder die Durchführung eines Security-Reviews ist allerdings auch in der Zeit aktiver Entwicklung nicht erfolgt.

5.2.6 Weitere Anforderungen (NF6)

Da das verwendete GnuPG nicht in einer Java-VM, sondern als nativer Code läuft, hat GnuPG for Android bessere Performance-Eigenschaften als Implementierungen, die auf Bouncy Castle aufbauen. Dies gilt insbesondere bei Operationen, die ohne Datenfluss von GnuPG selbst abgewickelt werden können, wie das eigenständige Ver- oder Entschlüsseln von Dateien. Die höhere Geschwindigkeit wirkt sich ebenfalls auf den S2K-Algorithmus aus, behebt die Problematik aber nicht.

GnuPG for Android ist im Google Play Store sowie bei F-Droid erhältlich. Die letzte tatsächlich unterstützte Android-Version ist Android 4.4, welche 2013 auf den Markt gebracht wurde. Es ist möglich, das Paket selbst zu bauen, wobei einige Anpassungen am Bauvorgang des nativ kompilierten GnuPG-Programms notwendig sind, um dieses auf aktuellen Android-Versionen lauffähig zu machen.

5.3 OpenKeychain

OpenKeychain befindet sich von allen betrachteten Apps die längste Zeit in aktiver Entwicklung und besitzt bei den meisten Anforderungen den höchsten Reifegrad. Viele der gestellten Anforderungen werden bereits erfüllt, die Nutzeroberfläche hält sich in weiten Teilen an die Vorgaben des auf Android üblichen Material-Designs.

5.3.1 Verwaltung privater Schlüssel (F1)

OpenKeychain unterstützt das Erstellen und Verändern von privaten Schlüsseln. Das Sichern von privaten Schlüsseln wird ebenfalls unterstützt. Hier wird eine Verschlüsselung des exportierten Schlüsselmaterials erzwungen, und die dafür verwendete Passphrase in festem Format von der App erzeugt.

5.3.2 Verwaltung öffentlicher Schlüssel (F2)

OpenKeychain unterstützt die Suche und den Import von Schlüsseln.

Eine Beglaubigung von öffentlichen Schlüsseln wird in OpenKeychain ebenfalls unterstützt und ist dabei stets verknüpft mit einem Verifikations-Verfahren. Unterstützte Methoden sind der händische Vergleich des Fingerprints, das Scannen eines QR-Codes (welcher von der zu verifizierenden Person angezeigt wird), NFC und eine Methode, die auf dem Safeslinger-Paper beruht und eine gemeinsame Verifikation mit bis zu 10 Personen gleichzeitig ermöglicht.

OpenKeychain bietet eine manuelle Aktualisierung von Schlüsseln und aktualisiert alle drei Tage mittels Android-Alarm-Service in einem gestaffelten Verfahren die Liste aller öffentlichen Schlüssel. Als zusätzliches Feature für die Privacy des Nutzers kann die Aktualisierung wahlweise unter Verwendung der Orbot-Tor-Implementierung erfolgen.

Das Vertrauensmodell in OpenKeychain ist ein einfaches und direktes Modell, in dem jeder private Schlüssel als vertrauenswürdig angenommen wird. Damit wird jeder öffentliche Schlüssel als verifiziert angenommen, der eine Beglaubigung durch einen privaten Schlüssels des Nutzers enthält. Eine einfache Synchronisierung dieser Vertrauensverhältnisse erfolgt optional über die Schlüsselserver, indem eine einmal ausgestellte Beglaubigung dort als Teil des öffentlichen Schlüssels gelagert wird und dadurch verfügbar bleibt. Dieses Verfahren ist einfach und funktional, bringt aber den Nachteil mit sich, dass die Vertrauensverhältnisse des Nutzers dabei veröffentlicht werden. Eine Unterstützung von komplexeren Vertrauensmodellen, wie dem Web of Trust, wird von OpenKeychain aktuell nicht umgesetzt.

5.3.3 Schnittstelle für externe Apps (F3)

Mit der „OpenPGP-API-Lib“ stellt OpenKeychain seine kryptografischen Operationen für die Nutzung durch andere Apps auf Basis der OpenPGP-API bereit. Hierzu zählen das Verschlüsseln und Signieren sowie das Entschlüsseln und Verifizieren von Daten. Zur Verwendung muss eine App lediglich die API-Library einbinden, welche die Operationen rückwärtskompatibel zur Verfügung stellt. Die hierzu veröffentlichte Library ist nicht OpenKeychain-spezifisch, sondern unterstützt theoretisch alle verfügbaren Apps, welche die OpenPGP-API im Rahmen der offenen Spezifikation bereitstellen. Die Spezifikation der API wurde abgestimmt mit der Entwicklung von GnuPG for Android, ist dort aber vor Einstellung der Entwicklung nie implementiert worden. Das Design und die genaue Spezifikation der API haben sich bis zur aktuellen Version 11 geringfügig geändert, entsprechen aber weitgehend den in Abschnitt 3.3.1.2 beschriebenen Kriterien und Überlegungen.

Intern kommuniziert die Library über ein AIDL-Interface, welches eine Verarbeitung der Daten direkt als Stream ermöglicht und so eine Verwendung von großen Zwischenspeichern vermeidet.

Für die Umsetzung von Interaktion mit dem Nutzer, etwa für die Eingabe einer Passphrase, kann als Ergebnis einer Operation eine Nutzerinteraktion in Form eines PendingIntents zurückgegeben werden. Dieser kann von der aufrufenden App angezeigt aber nicht manipuliert werden. Es wird dann ein Zwischenergebnis zurückgegeben, welches wiederum als Eingabe der Operation verwendet wird. Auf diese Weise bleibt der Kontrollfluss auch während etwaiger Nutzerinteraktionen bei der aufrufenden App, ohne dass diese Zugriff auf kryptografische Daten und Details erhält.

Da die Nutzung von Android-Berechtigungen, insbesondere auf eine rückwärtskompatible Weise, nur mit Einschränkungen möglich ist (siehe Abschnitt 3.3.4), ist für die OpenPGP-API die Verwendung eines eigenen, wenn auch einfachen, Berechtigungs-Systems vorgesehen. Zu diesem Zweck ist das Ergebnis eines ersten Zugriffs durch eine App (identifiziert anhand ihrer App-Signatur) stets eine Nutzerinteraktion, welche den Nutzer auffordert, den Zugriff durch die entsprechende App zu bestätigen.

5.3.4 Eigenständige kryptografische Operationen (F4)

OpenKeychain unterstützt als eigenständige Operationen das Ver- und Entschlüsseln sowie Signieren und Verifizieren von Dateien und Text sowie zusätzlich das unverschlüsselte Signieren von Text. Ein Ausstellen unabhängiger Signaturen für Dateien, sowie die Verifikation derselben, wird aktuell nicht unterstützt. Als zusätzliches Feature wird die Auflösung von (in verschlüsselten Blöcken enthaltenen) MIME-Strukturen unterstützt, wodurch es möglich ist, im PGP/MIME-Format verschlüsselte Teile von E-Mails als Anhang zu öffnen.

Alle Operationen sind mit den von Android bereitgestellten Mechanismen verknüpft. Zusätzlich werden verifizierte Schlüssel mit Kontakten aus dem Adressbuch des Nutzers anhand von E-Mail-Adressen verknüpft, um eine einfache Zuordnung zu ermöglichen und einen direkten Weg von einem Kontakt zum Schlüssel des Nutzers zu bieten.

5.3.5 Sicherheitseigenschaften (NF5)

Die Entwicklung von OpenKeychain wird über einen öffentlichen Issue-Tracker auf GitHub abgewickelt. Ebenfalls existiert eine Mailingliste für technische Rückfragen. Die App wird aktiv entwickelt, über 2014 und 2015 erfolgten jeweils 5 Feature-Releases, nebst kleineren Bugfix-Releases.

Als kryptografische Basis verwendet OpenKeychain die Bouncy-Castle-Bibliothek, welche über umfangreiche Unit-Tests verfügt und durch Verwendung in weiteren Anwendungen mit verschiedenen Plattformen und Nutzungsszenarien ein hohes Vertrauen mit sich bringt. Alle darüber liegenden, kryptografischen Operationen sind in OpenKeychain in gekapselte Operationen mit festem Input/Output-Interface getrennt, die jeweils durch Unit-Tests abgesichert sind. Es wird außerdem eine Reihe von Interoperabilitäts-Tests angewendet, die ebenfalls von End-to-End genutzt werden und Interoperabilität von Schlüsseln und verschlüsselten Nachrichten verschiedener Implementierungen überprüfen. Im November 2015 erhielt OpenKeychain von der Firma Cure53 ein Security Audit. Im Rahmen dieses Audits wurden keine **kritischen** Sicherheitslücken gefunden. Die niedriger priorisierten Sicherheitslücken wurden in einem zeitnahen Release behoben und vor Veröffentlichung des Reports erneut überprüft. Trotz dieser Eigenschaften bleibt als Nachteil zu beachten, dass es sich bei dem Quellcode-Teil, der das OpenPGP-Datenformat und Protokoll in OpenKeychain umsetzt, um eine, im Vergleich zu GnuPG, sehr junge Implementierung handelt. Es ist wichtig festzustellen, dass die meisten im Audit gefundenen Sicherheitslücken auf Fehlern beruhen, die durch die Implementierung der Benutzeroberfläche und Nutzung der Android API entstanden und nicht im kryptografischen Teil der Software lagen. Eine Sicherheitslücke beispielsweise, die später in der Veröffentlichung „Surreptitious Sharing on Android“ weiter untersucht wurde, konnte in Verbindung mit Fehlern in der Benutzeroberfläche dazu führen, dass private Schlüssel extrahiert werden. Für GnuPG for Android und PGP KeyRing fehlt ein Security Audit, das die Implementierung der Oberfläche und Nutzung der Android-APIs evaluiert, sodass wir in Bezug auf die Sicherheit der privaten Schlüssel OpenKeychain besser bewerten.

5.3.6 Weitere Anforderungen (NF6)

Die Geschwindigkeits-Eigenschaften von OpenKeychain beruhen asymptotisch auf der Implementierung der kryptografischen Algorithmen in Bouncy Castle. Verglichen mit dem nativ kompilierten GnuPG ist die Java-basierte Bouncy-Castle-Implementierung etwas langsamer (siehe Abschnitt 3.6.1). Trotz des relativen Geschwindigkeitsnachteils gegenüber einer nativen Implementierung ist bei der Verarbeitung von Daten im niedrigen Kilo- und Megabyte-Bereich, beispielsweise beim Lesen von E-Mails, keine Einschränkung der Nutzbarkeit durch starke Verzögerungen zu verzeichnen.

Der Vertrieb von OpenKeychain erfolgt als kostenlose App sowohl über den Google Play Store, als auch über F-Droid. Als Freie Software-Anwendung ohne proprietäre Teile kann OpenKeychain uneingeschränkt von Dritten überprüft, modifiziert und gebaut werden.

5.4 PGP KeyRing

Auf den ersten Blick unterstützt PGP KeyRing viele Operationen, hat ein komplexes Vertrauensmodell und stellt eine eigene API bereit. Die Verfügbarkeit dieser Funktionen wird leider eingeschränkt durch Mängel im Design der Nutzeroberfläche, die vor allem für technisch nicht versierte Nutzer eine Hürde darstellen, sowie technische Fehler im Detail.

5.4.1 Verwaltung privater Schlüssel (F1)

PGP KeyRing unterstützt sowohl das Erstellen als auch das Modifizieren von privaten Schlüsseln in technisch vollem Umfang, allerdings mit unübersichtlicher Benutzeroberfläche.

Alle Operationen zum Editieren von Schlüsseln werden unterstützt, mit akzeptabler Nutzeroberfläche. Eine kleinere Einschränkung ist, dass Grenzfälle (wie ein fehlender Unterschlüssel, der für eine Modifikation notwendig wäre) nicht abgefangen werden, sondern in fehlleitende Fehlermeldungen resultieren.

Ein Export von privaten Schlüsseln ist verfügbar, es erfolgt allerdings keine Verschlüsselung des Schlüsselmaterials als Teil der Export-Operation.

5.4.2 Verwaltung öffentlicher Schlüssel (F2)

Die Suche und der Import einzelner Schlüssel ist möglich, aber auch hier mit mangelnder Nutzeroberfläche. Beispielsweise erhält der Nutzer während der Ladezeiten des Imports keine Rückmeldung über den laufenden Vorgang, so dass nicht erkennbar ist, ob eine Suche nach einem Schlüssel oder ein Import gestartet wurde, noch verarbeitet wird, oder mit einem Fehler abgebrochen wurde.

Das Beglaubigen von Schlüsseln ist möglich, diese ist jedoch nicht mit einer Aufforderung an den Nutzer verbunden, den Schlüssel auch zu verifizieren. Die Funktionalität wird somit bereitgestellt, ist aber ohne fundiertes Wissen des Nutzers schwer benutzbar und kann im schlimmsten Fall zur Ausstellung von unbedachten Beglaubigungen führen.

Eine Operation zum Aktualisieren eines einzelnen Schlüssels ist nicht ersichtlich, selbiges gilt für das Aktualisieren aller bekannten Schlüssel.

PGP KeyRing zeigt in der Detailansicht für jeden Schlüssel sowohl eingehende als auch ausgehende Beglaubigungen bekannter Schlüssel an, und erlaubt in einer etwas versteckten Option die Zuordnung von Vertrauens-Leveln zu bestimmten Schlüsseln. Auf diese Weise wird das Web-of-Trust-Vertrauensmodell umgesetzt. Ein Abstrich dabei ist, dass in der Nutzeroberfläche zum tatsächlichen Durchführen von kryptografischen Operationen der Vertrauensstatus fremder Schlüssel nicht angezeigt wird.

Ein Export oder eine Synchronisation der zugeordneten Vertrauens-Einstellungen wird nicht unterstützt.

5.4.3 Schnittstelle für externe Apps (F3)

PGP KeyRing stellt eine eigene, undokumentierte API-Schnittstelle zur Verfügung, die von demselben Entwickler in der App Squeaky Mail, basierend auf K-9 Mail, implementiert wird.

Als Zugriffskontrolle verwendet PGP KeyRing eine Berechtigung im Android-Berechtigungssystem, die von Squeaky Mail angefordert wird. Dies führt zu den oben erwähnten Nutzbarkeitsproblemen (siehe Abschnitt 3.3.4), wenn Squeaky Mail vor PGP KeyRing installiert wird, und Sicherheitsproblemen, wenn eine böswillige App die Berechtigung vor der Installation von PGP KeyRing definiert.

Für kryptografische Operationen notwendige Nutzerinteraktionen werden von Squeaky Mail unmittelbar durchgeführt. Dies ermöglicht eine gute Integration und hält den Kontrollfluss beim Client, bringt aber als Sicherheitsmakel mit sich, dass die Squeaky Mail App als Teil des Vorgangs Zugriff auf die Passphrase des Nutzers erhält.

Die Kommunikation von Daten erfolgt über Pfade zu temporären Dateien. Diese Lösung funktioniert auch mit großen Dateien, ist aber durch den notwendigen Zugriff auf Sekundärspeicher verhältnismäßig langsam.

5.4.4 Eigenständige kryptografische Operationen (F4)

Alle erwarteten kryptografischen Operationen werden unterstützt, verlangen dem Nutzer aber stellenweise Entscheidungen mit lückenhafter Information ab. Einige Fallbeispiele:

- Für das Verschlüsseln von Daten wird die Auswahl der Empfänger-Schlüssel über ein Dropdown-Menü bereitgestellt, in dem ausschließlich die User IDs aller Schlüssel angezeigt werden, nicht jedoch das dazugehörige Vertrauensverhältnis.
- In einem zweiten Dropdown-Menü kann der Nutzer den zu nutzenden Unterschlüssel für die Verschlüsselung anhand seiner Schlüssel-ID auswählen.
- Eine Möglichkeit, von diesem Menü aus zu einer Detailansicht des verwendeten Schlüssels zu gelangen, ist nicht gegeben.
- Um Daten zu Signieren, wählt der Nutzer den zu verwendenden Schlüssel anhand seiner Schlüssel-ID aus einem weiteren Dropdown-Menü aus.
- Hinter der Option „With integrity check?“ verbirgt sich die Aktivierung des Modification Detection Code (MDC) Pakets, ein Feature des OpenPGP-Standards, welches aufgrund bekannter Attacken bedingungslos aktiviert sein sollte (RFC4880, Seite 83).

Eine Integration mit den Android-eigenen Mechanismen ist lediglich für den Import öffentlicher oder privater Schlüssel implementiert, ein Ver- oder Entschlüsseln von Daten mittels der Öffnen- oder Teilen-Operationen wird nicht unterstützt.

5.4.5 Sicherheitseigenschaften (NF5)

PGP KeyRing verfolgt ein geschlossenes, proprietäres Entwicklungs- und Vertriebsmodell, und wird nicht von einer offenen Entwicklergemeinschaft getragen. Da relativ wenige Informationen zu Implementierungsdetails dokumentiert sind, ist es schwer, Aussagen über Sicherheitsaspekte der App zu treffen.

Der Entwicklungsprozess von PGP KeyRing ist nicht offen, ebenso existiert kein öffentlicher Bug-Tracker. Die Entwicklung ist dennoch aktiv mit mehreren Releases im Jahr. Da diese aber ohne Changelog ausgeliefert werden, ist es schwierig, den Fortschritt der Entwicklung zu beurteilen. Die Website des Produkts bietet nur eine rudimentäre Übersicht über Features der App, jedoch keine weiteren Informationen.

Eine mögliche Nutzung von Unit- oder Interoperabilitäts-Tests ist, aufgrund des geschlossenen Entwicklungsprozesses, nicht bekannt. Ebenso liegt kein veröffentliches Security Audit für PGP KeyRing vor. Durch die geringe Nutzeranzahl und die geschlossene Entwicklung ist außerdem wenig Aufmerksamkeit durch unabhängige Entwickler anzunehmen.

5.4.6 Weitere Anforderungen (NF6)

Die Geschwindigkeits-Eigenarten von PGP KeyRing werden, ähnlich wie bei OpenKeychain, durch Bouncy Castle vorgegeben.

Der Vertrieb und Support von PGP KeyRing erfolgt ausschließlich über den Google Play Store. Ein selbstständiges Kompilieren und ein Vertrieb über F-Droid ist aufgrund des geschlossenen Entwicklungsmodells nicht möglich.

6 Betrachtung aktuell verfügbarer Client-Apps

In diesem Kapitel werden Apps betrachtet, die OpenPGP zum Zweck der Kommunikations-Verschlüsselung verwenden. Dies schränkt die Auswahl betrachteter Apps im Speziellen auf E-Mail-Clients und Instant-Messaging-Apps ein. Andere Nutzungsszenarien, wie Datensicherung, wurden nicht betrachtet.

Die betrachteten Apps unterscheiden sich anhand ihrer OpenPGP-Unterstützung in drei Gruppen: Im einfachsten Fall implementieren die Apps OpenPGP selbst und kommen ohne Installation weiterer Komponenten aus (Standalone). Andere Apps verwenden die API einer der vorgestellten OpenPGP-Apps. Und einige Apps lagern die OpenPGP-Funktionalität in ein Plugin aus, das sich wie ein Teil der App verhält, aber einzeln installiert werden muss.

6.1 E-Mail-Clients

Für die betrachteten E-Mail-Clients wurden solche Apps ausgewählt, die aktiv weiterentwickelt werden und eine ausreichend große Installationsbasis vorweisen. Gesucht wurde, wie bei den Standalone-Implementierungen, nach „PGP“ und „OpenPGP“. Apps, die keinen E-Mail-Client implementieren, aber bei der Suche gefunden wurden, werden separat im nächsten Abschnitt 6.2 betrachtet. Nicht betrachtet wurden die folgenden Apps, die weniger als 5000 Installationen in Google Play aufwiesen:

- PGP Mail (Secure Mail GPG Team)
- Secure Email (Secure Group Inc.)
- SMail - Secure Email (Gimy)
- Xafely - A Secure Email Client (The Xafely Team)

Weiterhin nicht betrachtet wurden verschiedene Derivate von K-9 Mail, die sich in ihrer OpenPGP-Funktionalität nicht vom Ursprungsprodukt unterscheiden:

- K-@ Mail (1gravity LLC)
- Kaiten Mail (C. Ketterer)
- AnyEmail (DickLucifer)

Die App GMX Mail (GMX) wurde ebenfalls nicht betrachtet, da sie der WEB.DE Mail (WEB.DE) App im Funktionsumfang gleicht und von demselben Hersteller (United Internet AG) veröffentlicht wird.

Somit bleiben die folgenden Apps, die genauer betrachtet wurden (sortiert nach der Anzahl der Installationen):

App	Entwickler	Benötigt	Lizenz	Letztes Release	Installationen
K-9 Mail	C. Ketterer	OpenKeychain/APG	Apache License v2	5.006 (2015-06-09)	5.000.000 - 10.000.000
WEB.DE Mail	WEB.DE	-	proprietär	3.6 (2015-10-28)	5.000.000 - 10.000.000
MailDroid	Flipdog Solutions, LLC	Plugin	proprietär	4.10 (2015-10-15)	1.000.000 - 5.000.000
R2Mail2	rundQuadrat OG	-	proprietär	2.17 (2015-07-28)	10.000 - 50.000
Squeaky Mail	A. Wasserman	PGP KeyRing	Apache License v2	5.001.08 (2015-04-28)	1.000 - 5.000

6.1.1 K-9 Mail

Mit über 5 Millionen Installationen ist K-9 Mail einer der meistgenutzten, als Freie Software lizenzierten E-Mail-Clients unter Android – neben dem E-Mail-Client, der im Android Open Source Project (AOSP) inkludiert ist. Da der AOSP E-Mail-Client voraussichtlich nicht aktiv von Google weiterentwickelt wird, ist K-9 Mail eine wichtige Freie Software-Alternative geworden.

Die Unterstützung von OpenPGP unter K-9 Mail beruhte bis September 2014 auf der von APG implementierten Intent-API, die lediglich ein simples Ersetzen von PGP/INLINE formatiertem Inhalt durch den entschlüsselten Text anbot. Aufgrund von Design-Entscheidungen im von K-9 Mail genutzten Datenbank-Format ist eine Implementierung von PGP/MIME nur mit erhöhtem Aufwand möglich, indem das Datenbank-Format verändert wird. In den aktuellen Versionen ab Version 5.000 wurde die Intent-API durch die OpenPGP-API ersetzt. Eine Unterstützung von PGP/MIME fehlt jedoch weiterhin.

E-Mails im S/MIME-Format werden von K-9 Mail nicht unterstützt. Pläne für eine Umsetzung bestehen zwar, zum Zeitpunkt dieser Studie fand hier aber noch keine Entwicklung statt.

K-9 Mail ist unter der Apache Lizenz v2 lizenziert, welche auch Forks erlaubt, die ihrerseits keine Freie Software sind. Aus diesem Grund existiert eine Anzahl von E-Mail Apps im Google Play Store, die auf K-9 Mail basieren und neue Features hinzufügen. Die Entwicklung von K-9 Mail wird hauptsächlich von Christian 'cketti' Ketterer betreut, der Pull-Requests auf GitHub bearbeitet und neue Release-Versionen veröffentlicht.

Unterstützung und Hilfe finden sich in einem ausführlichen Nutzerhandbuch auf GitHub, einer Mailingliste (Google Groups) sowie einer Google+ Community. Diskussionen unter Entwicklern finden auf einer gesonderten Mailingliste sowie auf dem IRC-Kanal #k-9 im Freenode-Netzwerk statt.

6.1.2 WEB.DE Mail

Die WEB.DE-Mail-App wird von United Internet entwickelt und bietet nur Zugang zu WEB.DE-E-Mail-Konten an. Andere Anbieter werden nicht unterstützt.

Seitdem WEB.DE und GMX OpenPGP-Support unter Nutzung des Mailvelope-Browser-Plugins eingeführt haben, wurde auch die dazugehörige App um OpenPGP-Support erweitert. Die Generierung von privaten Schlüsseln wird nicht unterstützt, stattdessen wird ein bereits eingerichtetes WEB.DE-Konto vorausgesetzt. Das Schlüsselpaar wird auf einem Desktop-Betriebssystem mit Mailvelope generiert und dann automatisch mit einem Backup-Code verschlüsselt und dieses Backup in einen Cloud-Speicher bei WEB.DE hochgeladen. Der Backup-Code wird dem Benutzer als druckbare Version angezeigt, wobei auch ein entsprechender QR-Code auf der Seite dargestellt wird, für die die App einen entsprechenden Scanner zum Einlesen enthält. Der so heruntergeladene und entschlüsselte private Schlüssel wird nun in der App gespeichert und kann genutzt werden.

Die WEB.DE-App basierte ursprünglich auf K-9 Mail, wurde aber in Hinblick auf Design und Funktionalität weiterentwickelt. Die ursprüngliche Unterstützung der APG Activity-Intents und auch die neuere OpenPGP-API finden bei WEB.DE keine Anwendung. Stattdessen wurde eine eigene proprietäre OpenPGP-Implementierung basierend auf Spongy Castle in die App entwickelt, die PGP/MIME unterstützt. S/MIME wird nicht unterstützt.

Die WEB.DE App ist proprietär und der Quellcode ist geschlossen. Eine öffentliche Hilfeseite zur App konnte nicht gefunden werden.

6.1.3 MailDroid

MailDroid ist eine E-Mail-App mit sehr vielen Features, wie beispielsweise IMAP-Idle-Push und einem WYSIWYG-Text-Editor. Sie wird als kostenlose, werbefinanzierte Version sowie als Pro-Version (die im Vergleich zu vielen anderen E-Mail-Apps mit 10,50€ recht teuer ist) auf Google Play vertrieben. Um OpenPGP- und S/MIME-Support zu erhalten, muss die App FlipdogSolutions Crypto Plugin zusätzlich installiert werden. Es werden PGP/MIME und PGP/INLINE unterstützt.

Schlüssel können aus Dateien importiert sowie von Schlüsselservern heruntergeladen werden. Hier wird für den Transfer allerdings kein TLS (HKPS) unterstützt. Eine rudimentäre Darstellung der Schlüssel innerhalb der App wird unterstützt. Das Generieren von neuen Schlüsseln ist hingegen nicht möglich.

Hilfe findet sich ausschließlich auf einer Mailingliste (Google Groups). Die Entwicklung und der Quellcode ist geschlossen.

6.1.4 R2Mail2

R2Mail2 ist eine proprietäre E-Mail-App, welche integrierte Unterstützung von OpenPGP und S/MIME bietet. Es existiert eine Testversion, die maximal 5 E-Mails darstellen kann. Um den vollen Funktionsumfang freizuschalten, muss eine Lizenz in Form einer Zusatz-App für 4,80€ gekauft und installiert werden.

R2Mail2 unterstützt S/MIME, PGP/MIME und PGP/INLINE. Wenn die „combined“-Funktion aktiviert wurde, werden auch kombinierte Modi aus PGP und S/MIME unterstützt (Details dazu auf der Webseite). Schlüssel können per Datei und über LDAP- sowie Schlüsselserver importiert werden – auch hier wird kein TLS (HKPS) unterstützt. Das Generieren von privaten Schlüsseln wird ebenfalls nicht unterstützt. Schlüssel werden in einer verschlüsselten Datenbank gespeichert, die wiederum mit einem Master-Passwort verschlüsselt wird. Dieses muss nach dem Öffnen der App eingegeben werden und wird dann zwischengespeichert. Als besonderes Feature ist hervorzuheben, dass Empfänger-Regeln festgelegt werden können, um beispielsweise zu erzwingen, dass Nachrichten an spezielle Empfänger ausschließlich verschlüsselt versendet werden. Weitergehende Informationen zu den unterstützen kryptografischen Features finden sich auf der dazugehörigen Webseite.

Die Implementierung basiert auf Bouncy Castle. Da der Quellcode nicht verfügbar ist, sind weitere Details zur Implementierung nicht bekannt. Das Unternehmen rundquadrat entwickelt die App nicht-öffentlich.

6.1.5 Squeaky Mail

Die App Squeaky Mail basiert auf K-9 Mail und ist kostenlos aus dem Google Play Store installierbar. Die OpenPGP-Unterstützung wird umgesetzt durch eine Integration mit der App PGP KeyRing – für eine Beschreibung der Schnittstelle siehe Abschnitt 5.4.3.

Während der Quellcode von PGP KeyRing nicht öffentlich verfügbar ist, ist Squeaky Mail unter der Apache Lizenz v2 lizenziert und der Quellcode auf GitHub veröffentlicht. Es handelt sich bei dieser App um einen Fork von K-9 Mail, der primär um die Nutzung der API von PGP KeyRing erweitert wurde.

Trotz langer Verfügbarkeit im Play Store und aktiver Entwicklung hat Squeaky Mail mit knap über 1000 Installationen eine sehr kleine Nutzerbasis. Eine Hilfe existiert nicht. Auf der Webseite von PGP KeyRing werden keine weiteren Informationen über Squeaky Mail verfügbar.

6.2 Weitere OpenPGP-basierte Apps

In diesem Abschnitt werden relevante Apps betrachtet, die keine Standalone-Implementierung und kein E-Mail-Client sind, aber bei der Suche nach „PGP“ und „OpenPGP“ gefunden wurden. Ausgewählt wurden Apps, die aktiv weiterentwickelt werden und eine ausreichend große Installationsbasis vorweisen. Nicht betrachtet wurden die Apps PGP SMS (woodkick), SMSSecure (SMSSecure) sowie Tutanota (Tutao GmbH), da diese nicht auf dem OpenPGP-Standard basieren, auch wenn sie bei einer Suche nach „PGP“ im Play Store als Suchergebnisse erscheinen. Safe Messenger PGP (Apps4us) weist weniger als 1000 Installationen auf und die Apps PGPClipper (Minori Hiraoka) sowie GP ID (RTEK) nutzen die API von OpenKeychain für sehr spezielle Szenarien, die hier nicht weiter betrachtet werden.

Des Weiteren wurde untersucht, ob Browser-Add-ons, die für Desktop-Browser entwickelt wurden, auch auf Android installiert werden können: Chrome auf Android unterstützt bisher keine Add-ons. Firefox hingegen stellt einen Großteil der Add-on-APIs auch unter Android zur Verfügung. Eine Suche nach „PGP“ oder „OpenPGP“ auf addons.mozilla.org unter Nutzung von Firefox auf Android ergibt aber keine relevanten Ergebnisse. Insbesondere von Mailvelope ist laut Issue 119 auf GitHub keine Portierung für Firefox auf Android geplant.

Somit bleiben die folgenden zwei Apps, die genauer betrachtet wurden (sortiert nach der Anzahl der Installationen):

App	Entwickler	Benötigt	Lizenz	Letztes Release	Installationen
Kontalk	Daniele Ricci	-	GPLv3	3.1.1 (2015-11-09)	10.000 - 50.000
Conversations	Daniel Gultsch	OpenKeychain	GPLv3	1.7.2 (2015-10-31)	5.000 - 10.000

6.2.1 Kontalk

Kontalk ist ein Instant-Messenger, der als Freie Software unter der GPLv3 lizenziert ist. Er wird seit 2011 von Daniele Ricci entwickelt und bis heute aktiv von ihm gepflegt. Kontalk basiert auf den Standards XMPP und OpenPGP und nutzt diese Standards zusammen mit einer Reihe von Protokoll-Erweiterungen, um beispielsweise Gruppenchats oder Push-Benachrichtigungen besser umsetzen zu können. Kontalk verwendet für die OpenPGP-Verschlüsselung eine eigene, nicht standardisierte Erweiterung des XMPP-Protokolls. Nach der Installation von Kontalk wird ein XMPP-Account basierend auf der Telefonnummer generiert. Eine Verifizierung der Telefonnummer erfolgt, wie auch bei anderen bekannten Messengern wie WhatsApp, über einen zentralen externen Dienstleister. Diese Verifikation erlaubt es dem Betreiber von Kontalk, OpenPGP-Schlüssel für Telefonnummern auszustellen. Dieses Verfahren ist zwar nicht sicher gegen einen bösartigen Dienstleister, erlaubt aber ein Mindestmaß an Vertrauen in Schlüssel der Kommunikationspartner und verhindert die Überflutung von OpenPGP- und XMPP-Servern mit gefälschten Accounts und Schlüsseln. Zum Schutz der Privatsphäre werden Telefonnummern nicht direkt als User IDs genutzt, sondern als SHA-1 Hash gespeichert. Kontalk erlaubt das Versenden von Bildern, Kontakten und Audio-Dateien sowie die Nutzung von Emojis. Auch Gruppenchats sind möglich, wobei XMPP-Nachrichten dabei an alle Teilnehmer verschlüsselt werden. Eine genauere Analyse der eingesetzten OpenPGP-Schlüssel zeigt, dass für den Hauptschlüssel, der Zertifizieren und Signieren kann, RSA mit 2048 Bits und für den Unterschlüssel zur Verschlüsselung Elliptic-Curve-Diffie-Hellman (ECDH) mit der Kurve P-256 genutzt wird.

6.2.2 Conversations

Wie Kontalk ist auch Conversations ein Instant Messenger, der unter der GPLv3 lizenziert ist. Er wird aktiv auf GitHub von Daniel Gultsch entwickelt. Auf Google Play kann Conversations für 2,38€ gekauft oder kostenlos von F-Droid bezogen werden. Im Gegensatz zu Kontalk setzt Conversations auf die klassische Nutzung oder Erstellung von XMPP-Accounts mittels JIDs. Nach dem Login per XMPP-Account kann über

eine moderne Benutzeroberfläche, die weitestgehend Androids aktuellen Material-Design-Guidelines entspricht, mit anderen gechattet werden. Neben den typischen zu erwartenden Funktionen, wie beispielsweise dem Versenden von Bildern und anderen Dateien, unterstützt Conversations „Gelesen“-Markierungen und das dynamische Nachladen von älteren Nachrichten. Diese Funktionen sind sonst nur in proprietären Instant-Messengern wie WhatsApp vorhanden. Conversations wurde speziell für Mobilgeräte optimiert und implementiert spezielle XMPP-Features, die den Energieverbrauch reduzieren (XEP-0352), auch bei instabilen Netzwerk-Bedingungen verhindern, dass XMPP-Nachrichten verloren gehen (XEP-0198) und Nachrichten zwischen mehreren Endgeräten des Nutzers synchronisieren (XEP-0280, XEP-0313).

Für eine Ende-zu-Ende-Sicherheit implementiert Conversations drei unterschiedliche Verschlüsselungsarten. Diese sind Off-the-Record Messaging (OTR), OpenPGP und das neu entwickelte OMEMO Multi-End Message and Object Encryption (OMEMO). Während OTR und OMEMO vollständig integriert sind, nutzt Conversations für die OpenPGP-Implementierung die von OpenKeychain bereitgestellte OpenPGP-API. Die für OpenPGP genutzte XMPP-Protokoll-Erweiterung ist XEP-0027, welche jedoch aufgrund ihres Alters und dadurch mangelnder Konformität mit aktuellen Erweiterungen durch das XMPP Council im März 2014 für deprecated erklärt wurde.

7 Integration von OpenPGP in das Android-Betriebssystem

In diesem Kapitel werden Möglichkeiten betrachtet, die sich durch eine eigene Weiterentwicklung und Distribution des Android-Betriebssystems ergeben. Neben einer Vorinstallation der OpenPGP-App als System-App, ist es möglich Androids Keystore API zu erweitern oder eine OpenPGP-API in das System zu integrieren. Dabei muss allerdings festgehalten werden, dass der Vertrieb einer eigenen Android-Distribution ein Unterfangen von erheblicher Tragweite darstellt, sowohl was die initiale Entwicklung angeht als auch den laufenden Wartungsaufwand.

7.1 Installation als System-App

Für den Entwickler und Vertreiber einer eigenen Android-Distribution besteht die besondere Möglichkeit, Apps als Teil des Systems vorzuinstallieren. Eine App mit dem System auszuliefern kann auf zwei Weisen erfolgen, die jeweils unterschiedliche Vorteile bieten.

Zunächst kann eine App als „System-App“ vorinstalliert sein. Die Installation als System-App ist (seit Android 4.4) an sich nicht mit zusätzlichen Berechtigungen verbunden, sondern führt lediglich dazu, dass die App nicht deinstalliert werden kann. Ist eine App auf diese Weise vorinstalliert, fallen die in Anforderung F3.4 (siehe Abschnitt 3.3.4) beschriebenen Probleme bei der Nutzung der Android-Permissions weg, da die Definition der Permission durch das Betriebssystem schon vor der Installation jeglicher Apps durch den Nutzer erfolgt ist. Dies eröffnet die Möglichkeit, für das Berechtigungsmanagement der API ohne Einschränkung Android-Permissions nutzen zu können, anstatt eines eigenen Systems. Anzumerken ist jedoch, dass diese Probleme ab Android 6.0 mit der Einführung von Runtime-Permissions ebenfalls behoben sind. Der Vorteil dieser Variante ergibt sich daraus, dass andere Apps für das entsprechende System unter der Annahme entwickelt werden können, dass die OpenPGP-App immer installiert und verfügbar ist.

Als zweite Form der vorinstallierten App gibt es „Privileged Apps”, welche über die besondere *signatureOrSystem*-Permission verfügen. Diese Form von Berechtigung wird vom Android-System verwendet, um besondere Funktionalität wie z.B. das Installieren und Verwalten von Paketen als Apps umsetzen zu können, die dennoch in den meisten Belangen wie normale Apps behandelt werden können, also beispielsweise Updates erhalten. Abseits von dieser zusätzlichen Berechtigung weisen Privileged Apps keine weiteren Besonderheiten auf. Für den Betrieb einer OpenPGP-App ist zum Zeitpunkt dieser Betrachtung keine der Permissions, welche auf diese Weise erteilt werden, von nennenswertem Nutzen – entsprechend einer aus dem Android-Quellcode extrahierten Liste von Permissions.

Unabhängig von der Art der Vorinstallation ist es möglich, eine vorinstallierte App als Teil des Assistenten zur initialen Einrichtung zu integrieren, optional auch in Kombination mit einer oder mehreren Client-Apps. Diese Form der Integration ist technisch mit relativ wenig Aufwand verbunden, kann aber verwendet werden, um die OpenPGP-App enger in das Nutzererlebnis der Android-Distribution einzubinden.

7.2 Erweiterung der Keystore API

Als eine gezieltere Form der Betriebssystem-Integration mit OpenPGP wäre eine Nutzung des Android Keystore Systems möglich, mit der durch Hardware-Unterstützung verbesserte Sicherheitseigenschaften erzielt werden können. Der Android Keystore Provider wurde in Android 4.3 veröffentlicht und in Android 6.0 stark erweitert (1, 2). Androids Keychain API, die den Android Keystore als Backend nutzt, ist hier explizit nicht relevant, da sie speziell für X.509-Zertifikate entwickelt wurde und außerdem Nutzerinteraktion erfordert.

Denkbare wäre es zu diesem Zweck, das Android-Keystore-System um OpenPGP-Unterstützung zu erweitern und diese Methoden für die Speicherung von privatem Schlüssel-Material zu verwenden. Die

Schlüssel, welche auf diese Weise im Keystore gespeichert werden, sind vor einer Extraktion durch einen Angreifer geschützt. Für eine solche Umsetzung müssten alle nötigen Cipher-Suites, die für OpenPGP nötig sind, über die Keystore-API verfügbar gemacht werden. Seit Android 6.0 wurden diese zwar stark erweitert, es fehlen dennoch viele, teilweise OpenPGP-spezifische Ciphers – beispielsweise die unterschiedlichen Blockciphers (IDEA, 3DES, CAST5,...) mit dem modifizierten CFB-Modus, welcher in OpenPGP verwendet wird. Da OpenPGP-Schlüssel aus mehreren Subkeys bestehen, muss das dazugehörige private Schlüsselmaterial jeweils einzeln im Keystore abgelegt werden.

Ohne Nutzung des Keystores müsste ein Angreifer sich zur Extraktion eines privaten Schlüssels zunächst durch Ausnutzung einer Sicherheitslücke Root-Zugriff auf dem Gerät verschaffen und das Schlüsselmaterial entweder in verschlüsselter Form extrahieren und per Brute-Force-Attacke entschlüsseln, oder es alternativ in unverschlüsselter Form während der Verwendung aus dem RAM auslesen. Ein Hardware-gestützter Keystore hält den Angreifer zwar davon ab, den Schlüssel selbst zu extrahieren, dennoch würde der gleiche Root-Zugriff es ermöglichen, die entsprechenden Operationen direkt auf dem Keystore durchzuführen. Dies ist besonders einfach möglich, da die Keystore-Hardware im Regelfall mit dem Gerät verbaut und dadurch zu jeder Zeit verfügbar ist – im Gegensatz zu beispielsweise einer Smartcard, die nur zur Nutzzeit verbunden wird. Aufgrund der Vielfalt von Android-Geräten und hohem Verbreitungsgrad älterer Android-Versionen wäre eine Nutzung des Android-Keystores in jedem Fall als optionales Feature für zusätzliche Sicherheit zu sehen, ein Betrieb der App bei nicht vorhandenem (oder inkompatiblem) Keystore muss weiterhin möglich sein.

In einer alternativen Implementierung wäre es möglich, die OpenPGP-Schlüssel weiter in der OpenPGP-App selbst abzulegen, diese aber symmetrisch zu verschlüsseln und nur den symmetrischen Schlüssel im Android-Keystore abzulegen. Beim Nutzen der Schlüssel würde nun kurzfristig der symmetrische Datenbankschlüssel aus dem Keystore geladen werden. Diese Variante hält die Komplexität des Keystores selbst gering, bietet aber dennoch einen zusätzlichen Schutz der Schlüssel, da nun die Sicherheit der Datenbank an die Authentisierung des Android-Betriebssystems gekoppelt werden kann. Die Datenbank der OpenPGP-App erhält so einen zusätzlichen Schutz über die Screen-Lock-Mechanismen des Systems, wie beispielsweise PIN, Pattern, Password oder Fingerprint. Zum Zeitpunkt dieser Studie ist die Wirksamkeit dieser Mechanismen unter Android leider recht begrenzt, da alle für Screen-Lock üblichen Mechanismen einfach zu merkende Informationen mit geringer Entropie verwenden, und in aktuellen Android-Versionen kein hardwaregestützter Schutz vor Brute-Force-Attacken existiert.

7.3 Integration in das Android SDK

Aktuell gibt es keine Unterstützung von OpenPGP im Android SDK. Ebenfalls sind die OpenPGP- und S/MIME-APIs der OpenSSL- und Bouncy-Castle-Bibliotheken, die teilweise in Android integriert wurden, nicht Teil der öffentlich nutzbaren Android-API. Statt die kryptografischen Operationen einer OpenPGP-App als Bibliothek zur Verfügung zu stellen, wäre eine direkte Integration der entsprechenden Methoden in das Android SDK denkbar. Die auf diese Weise zur Verfügung gestellten Methoden würden technisch identisch funktionieren zu denen einer Bibliothek – also ebenfalls mit einer App kommunizieren, die die eigentliche Funktionalität zur Verfügung stellt. Eine API als Teil des SDKs bringt aus diesem Grund an sich keine technischen Vorteile mit sich, sondern erhöht lediglich die Reichweite der API, da diese direkt ohne Einbindung einer Bibliothek genutzt werden kann. Für eine erfolgreiche Integration in das Android-Betriebssystem auf diese Weise müsste die entsprechende API letztendlich als Teil des Android Open Source Projects (AOSP) akzeptiert werden, damit es wiederum Teil des offiziellen SDKs wird. Ein paralleler Vertrieb der API als Teil eines alternativen SDKs und als Bibliothek für andere Systeme wäre als Übergangslösung ebenfalls denkbar. Dennoch halten die Autoren diese Variante aufgrund des schlechten Kosten-Nutzen-Verhältnisses für nicht empfehlenswert.

8 Risko- und Aufwandsabschätzung zur Weiterentwicklung

Aufbauend auf den Erkenntnissen der vorhergehenden Kapitel werden in diesem Kapitel Empfehlungen für die Weiterentwicklung von OpenPGP auf Android gegeben.

8.1 Integration von OpenPGP in das Android-Betriebssystem

Im vorherigen Kapitel wurden die Möglichkeiten der Integration einer OpenPGP-App in das Android-Betriebssystem diskutiert. Eine Vorinstallation der OpenPGP-App sowie eine Integration in den Einrichtungsassistenten, welcher zum ersten Systemstart durchgeführt wird, kann dazu beitragen, ein ganzheitliches Nutzererlebnis in Bezug auf OpenPGP zu liefern. Für den Fall, dass bereits eine eigene Android-Distribution gepflegt wird, ist diese Möglichkeit mit nur geringem Aufwand verbunden und deshalb besonders empfehlenswert.

Die weiterführenden Varianten unter den vorgestellten, wie die Erweiterung der Keystore API oder die Integration in das SDK, sind aufgrund ihrer ungünstigen Kosten-Nutzen-Abschätzung nicht zu empfehlen.

8.2 Weiterentwicklung von OpenKeychain

Aus der Evaluation der verfügbaren OpenPGP-Apps geht hervor, dass sich OpenKeychain am sinnvollsten als Basis für die Weiterentwicklung einer OpenPGP-App eignet. GnuPG for Android demonstriert zwar die technische Umsetzbarkeit einer App, die GnuPG als nativen Programmteil enthält und als kryptografischen Kern verwendet, beinhaltet aber keine ausgereifte Benutzeroberfläche und die benötigten Funktionalitäten um eine API und Integration unter Android bereitzustellen. Eine Weiterentwicklung von PGP KeyRing wird ausgeschlossen, da der Quelltext nicht offen vorliegt und einige Anforderungen, aufgrund von fehlenden Funktionalitäten, nicht erfüllt werden. Es wird also empfohlen, OpenKeychain als Basis für eine Weiterentwicklung zu nutzen, da es bereits die meisten benötigten Funktionalitäten implementiert, einschließlich einer ausgereiften und bereits genutzten API. Ebenfalls existiert ein Security Audit, bei dem die Android-Integration und Benutzeroberfläche untersucht wurde. Bedarf an einer Weiterentwicklung wurde bei den Anforderungen Synchronisierung des Vertrauenszustands (F2.6), Korrektheit des kryptografischen Codes (NF5.3) und Performance (NF6.1) identifiziert. Somit konzertieren sich die folgenden Vorschläge zur Weiterentwicklung primär auf diese Anforderungen.

8.2.1 Nutzung von GnuPG als kryptografischen Unterbau von OpenKeychain (Verbesserung NF5.3, NF6.1)

Ein erster Vorschlag zur Weiterentwicklung verfolgt die Idee, GnuPG als kryptografischen Unterbau für OpenKeychain zu verwenden. GnuPG for Android enthält hier eine nutzbare, technische Lösung für die interne Verwendung einer nativen Portierung von GnuPG auf dem Android-Betriebssystem und die Interprozess-Kommunikation mit einer darüberliegenden App mittels libgpgme. Der für Android angepasste Teil von GnuPG, der im Zuge von GnuPG for Android entstanden ist, kann so möglicherweise in OpenKeychain weiterverwendet werden. Anzumerken ist, dass OpenKeychain unter der GPLv3 Lizenz steht, eine Integration von GnuPG setzt also voraus, dass die daraus resultierende App ebenfalls unter die GPLv3 oder eine kompatible Lizenz gestellt wird.

Die Nachteile einer OpenPGP-App auf Basis von GnuPG liegen in der höheren Komplexität der Implementierung aufgrund der notwendigen Interprozess-Kommunikation mit GnuPG und die konzeptuelle Diskrepanz zwischen verschiedenen unter Android üblichen Mechanismen und GnuPG selbst. Die tatsächlichen Vor- und Nachteile sind jedoch recht schwer einzuschätzen. Dazu einige Überlegungen:

- Die Auflistung von Schlüsseln sollte Android-gemäß mittels eines Content Providers erfolgen. Das dafür verwendete Interface ist auf die Nutzung einer darunterliegenden SQL-Datenbank optimiert, kann also nur mit eingeschränkter Funktionalität implementiert werden oder erfordert die Nutzung einer Datenbank als zusätzlichem Caching Layer.
- Werden Features benötigt, die nicht bereits von der libgpgme-API unterstützt werden, führt dies zu der Notwendigkeit, Patches auf dem eingepflegten GnuPG zu pflegen. Dies tritt beispielsweise auf, wenn Informationen an das Frontend weitergegeben werden, die von der ungeänderten API nicht unterstützt werden. In diesem Zusammenhang würde die genutzte libgpgme-Version von der ursprünglichen divergieren, was allerdings durch Koordination mit GnuPG und dortigem Einpflegen der Patches verhindert werden kann.
- Die Unterstützung von NFC-basierten Security Token ist vermutlich nicht mit dem von GnuPG unterstützten Mechanismus nutzbar, da im Android-Betriebssystem ein direkter Zugriff auf die NFC-Hardware nicht vorgesehen ist.
- Eine zusätzliche Unterstützung von S/MIME ist aufgrund der dafür vorhandenen Unterstützung in GnuPG mit relativ wenig technischem Aufwand verbunden, bringt aber signifikanten Änderungsbedarf in der Benutzeroberfläche mit sich.

8.2.2 Auslagerung des OpenKeychain-Backends als Bibliothek (Verbesserung NF5.3)

Eine Alternative zur Nutzung von GnuPG als kryptografisches Backend ist, den entsprechenden Teil von OpenKeychain direkt weiterzuentwickeln. Hier empfehlen sich besonders Maßnahmen zur Erhöhung der Softwarequalität, speziell für die Teile des Codes, welche aufbauend auf Bouncy Castle die übergeordneten kryptografischen Funktionen bereitstellen. Dieser Bedarf rückt vor allem dadurch in den Vordergrund, dass OpenKeychain im Vergleich zu GnuPG bisher verhältnismäßig wenig produktive Nutzung durch Anwender erfahren hat.

Die in Java-Anwendungen sehr etablierte Bouncy-Castle-Bibliothek umfasst viele kryptografische Algorithmen und enthält ein Modul, welches OpenPGP gemäß RFC 4880 implementiert. Diese Unterstützung ist zwar eine vollständige Umsetzung des OpenPGP-Standards, setzt aber mit einer sehr niedrigen Abstraktionsschicht direkt an den OpenPGP-Datenstrukturen an. Grund hierfür ist, dass viele Implementierungsdetails, die für vollständige kryptografische Operationen auf OpenPGP-Schlüsseln notwendig sind, in RFC 4880 offen gelassen sind, und Bouncy Castle darüber hinaus keine Unterstützung für eine Datenbank von Schlüsseln („Keyring“) beinhaltet.

Als konkreten Vorschlag wird daher empfohlen, die kryptografischen Programmteile oberhalb von Bouncy Castle aus dem OpenKeychain-Code auszugliedern und als eigenständige Bibliothek verwendbar zu machen. Der hauptsächliche Aufwand einer Umsetzung dieses Konzeptes liegt darin, alle Android-spezifischen Teile des Codes durch eine Abstraktion zu ersetzen. Der Gedanke einer entsprechenden Kapselung wurde bei der ursprünglichen Implementierung der relevanten Programmteile bereits bedacht, ist jedoch an einigen Stellen nur mit Kompromissen umgesetzt worden. Diese Auslagerung führt zu einer unabhängigen, vollständig in Java implementierten High-Level-Bibliothek zur Nutzung von OpenPGP.

Eine derartige Bibliothek lässt sich nicht nur besser testen und warten, sondern ist auch in anderen Java-basierten Projekten wiederverwendbar, was besonders dem Java-Enterprise-Umfeld zugute kommt und eine dortige Verbreitung von OpenPGP begünstigt. Eine Nutzung in unabhängigen Projekten wiederum bringt Aufmerksamkeit von unabhängigen Entwicklern mit sich und wirkt sich damit positiv auf die praktische Stabilität des Programm-Codes aus. Durch eine klare Abgrenzung der kryptografischen Methoden vom Rest der App vereinfacht sich weiterhin die Möglichkeit eines umfassenden Sicherheitsaudits. Optional ist es eventuell erstrebenswert, die so erhaltene Bibliothek als Modul in Bouncy Castle zurückfließen zu lassen.

Insgesamt ist der geschätzte Aufwand, der zum Erreichen einer stabilen und zuverlässigen OpenPGP-Implementierung auf Android notwendig ist, mit dieser Möglichkeit geringer als bei der auf GnuPG

basierenden Variante. Die Entscheidung, eine separate Implementierung von OpenPGP zu veröffentlichen und zu pflegen, ist allerdings recht weitreichend und sollte mit Bedacht getroffen werden.

8.2.3 Verwendung nativer kryptografischer Routinen (Verbesserung NF6.1)

Um die Erfüllung der nichtfunktionalen Eigenschaft NF6.1 zu verbessern, sollte die Laufzeit der kryptografischen Algorithmen verbessert werden. Dies gilt speziell für die verwendeten Blockchiffren, sowie Hashing-Algorithmen. In der aktuellen Implementierung werden hier die in Java implementierten Routinen aus der Bouncy-Castle-Bibliothek verwendet. Durch die Nutzung der Provider-Struktur der Java Cryptography Architecture (JCA) in Bouncy Castle ist es möglich, diese verwendeten Routinen ohne weitere Anpassungen an Bouncy Castle selbst auszutauschen. Dadurch ist es möglich, die entsprechenden Routinen beispielsweise aus OpenSSL zu verwenden. Die genaue Wahl des verwendeten Providers sollte als Teil der Weiterentwicklung untersucht werden.

8.2.4 Synchronisierung (Verbesserung F2.6)

Voraussetzung für eine nutzbare Verwendung von OpenPGP ist ein Mechanismus zur Synchronisierung des Vertrauensstatus öffentlicher Schlüssel, sowie einer Liste bekannter Schlüssel. Die aktuell in OpenKeychain verwendete Methode mittels öffentlicher Schlüsselserver ist unzureichend, da sie weder Vertraulichkeit noch Verfügbarkeit gewährleistet, und zudem nicht geeignet ist für eine Synchronisierung der Liste bekannter Schlüssel.

Eine Weiterentwicklung ist hier empfehlenswert aufgrund der Wichtigkeit für das Nutzungsszenario mehrerer Geräte durch denselben Nutzer, und um eine reibungslose Integration mobiler Endgeräte bei der Verwendung von OpenPGP gewährleisten zu können. Für die Erfüllung dieser Anforderung gibt es aktuell keine etablierte Methode. Eine Entwicklung sollte deshalb auf Interoperabilität achten und nach Möglichkeit mit anderen Implementierungen koordiniert werden.

Ein Ansatz für die Implementierung wäre die Auslagerung der Synchronisierung auf verwendete Client-Apps, da diese oftmals bereits in der Lage sind, Daten zentral zu hinterlegen und auf diese Weise zwischen beteiligten Endgeräten auszutauschen. Für das E-Mail-Nutzungsszenario wäre es beispielsweise denkbar, die entsprechenden Daten durch den E-Mail Client im Postfach des Nutzers abzulegen. Um die Vertraulichkeit und Integrität zu gewährleisten bietet es sich in jedem Fall an, die Daten mit dem öffentlichen Schlüssel des Nutzers zu verschlüsseln. In jedem Fall erfordert eine Weiterentwicklung in diesem Bereich zunächst eine genauere Ausarbeitung der Umsetzung und eventuell Abwägung anderer möglicher Mechanismen.

8.3 Weiterentwicklung von Anwendungen mit OpenPGP-Support

Eine Spezifikation und Implementierung einer API für die Nutzung von OpenPGP ist seit Juni 2014 verfügbar. Zum Zeitpunkt der Studie wird diese API von verschiedenen Apps für die Bereitstellung von OpenPGP-basierten Features genutzt, es gibt allerdings keinen E-Mail-Client mit vollständiger Unterstützung (siehe Kapitel 6).

Die OpenPGP-Unterstützung in K-9 Mail wird seit Dezember 2015 durch ein Projekt des Open Technology Fund unterstützt, entwickelt von Vincent Breitmoser (ebenfalls Autor dieses Dokuments, siehe Ankündigung des Projekts). Im Rahmen dieses Projektes vorgesehen ist eine vollständige Neuentwicklung der bisherigen OpenPGP-Unterstützung, weiterhin auf Basis der von OpenKeychain implementierten OpenPGP-API, mit einem Fokus auf guter Nutzbarkeit der App. Die Weiterentwicklung beschränkt sich weitgehend auf K-9 Mail. Änderungen an OpenKeychain sind nur vorgesehen soweit unmittelbar benötigt. Fertigstellung dieses Projekts ist für Anfang 2017 geplant. Ein zwischenzeitliches Release soll Mitte 2016 erfolgen.

Im Bereich des Instant Messaging ist mit Conversations eine funktionierende Implementierung vorhanden. Diese beruhen auf der veralteten Spezifikation von OpenPGP für XMPP (XEP-0027, siehe Abschnitt 6.2.2). Eine Implementierung des neueren „OX“-Standards, welcher sich zum Zeitpunkt der Studie in der Ausarbeitung durch die XMPP Standards Foundation (XSF) befindet, ist parallel dazu ebenfalls in der Entwicklung.

Die zur Zeit laufenden Entwicklungen sind zu beobachten, um dann Risiken und Aufwand kurz vor einer geplanten Weiterentwicklung genauer einschätzen zu können.