Entwicklung einer Android NFC-Signaturkarte

Tobias Uebel

Fraunhofer SIT
tobias.uebel@sit.fraunhofer.de

Hochschule Bonn-Rhein-Sieg
tobias.uebel@inf.h-brs.de

Zusammenfassung

Sichere Chipkarten haben sich als Schlüsselspeicher für die verschlüsselte elektronische Kommunikation bewährt. Jedoch ergeben sich in der Praxis durch ihre Nutzung auch Nachteile: Im Hinblick auf Rechenleistung und Datenübertragungsrate sind Chipkarten stark beschränkt und für ihren Einsatz mit mobilen Endgeräten gibt es keine einheitliche Lösung. Diese Arbeit liefert anhand einer Demoanwendung den Nachweis, dass es möglich ist, die Funktionen einer Signaturchipkarte mit Hilfe eines Smartphones unter Android zu simulieren und das private Schlüsselmaterial in einem vom Betriebssystem abgeschirmten Bereich zu verwalten. Es konnte gezeigt werden, dass kryptographische Funktionen einer Signaturchipkarte standardkonform über ein Kartenterminal nutzbar sind, bei denen keines der beteiligten Systeme direkten Zugriff auf den privaten Schlüssel erhält. Eine zusätzlich erforderliche Benutzerinteraktion kann dabei die missbräuchliche Schlüsselverwendung verhindern.

1 Einleitung

Die zur sicheren Schlüsselspeicherung eingesetzten Chipkarten basieren auf 8-Bit, 16-Bit oder 32-Bit Mikrokontrollern. Ihre Rechenleistung ist damit eher gering, so dass kryptographische Berechnungen von Public-Key Verfahren je nach Schlüssellänge mehrere Sekunden dauern können [RaEf08, S.158 Tabelle 7.7]. Eine Chipkarte bietet in der Regel keine direkte Möglichkeit der Nutzerinteraktion, um Zugriffe auf das Schlüsselmaterial zu kontrollieren: Ist der Zugriff durch die Eingabe einer Geheimzahl (PIN) freigegeben, kann der Benutzer nur durch Ziehen der Karte den Zugriff auf die geöffnete Sitzung sperren. Die mangelnde Unterstützung für den Einsatz von Chipkarten mit mobilen Endgeräten erschwert die sichere Kommunikation im Unternehmensumfeld, wo sich der Einsatz mobiler Endgeräte zur schnellen Kommunikation durchsetzt und stationäre PCs immer weiter verdrängen [DiKl14, S.1]. Im Rahmen dieses Beitrags wird die sichere Schlüsselerstellung und Speicherung auf einem Android-Smartphone und der Einsatz als Signaturchipkarten-Ersatz über Near Field Communication (NFC) mit der Host Card Emulation (HCE) evaluiert. Smartphones sind als Alternative zu Chipkarten besonders interessant, da diese häufig schon, privat oder dienstlich, vorhanden sind. Es wird nur noch das Lesegerät und die Middleware benötigt. Der Einsatz eines Smartphones ermöglicht einen deutlich flexibleren Umgang mit Schlüsselmaterial aufgrund des größeren Speichers und der höheren Rechenleistung. Android bringt ab Version 4.3 alle für eine Signaturkarte benötigten Funktionen mit und bietet seit Version 4.4 zusätzlich die Möglichkeit, diese inner-

P. Schartner (Hrsg.) · D•A•CH Security 2016 · syssec (2016) 150-160.

halb einer vom Betriebssystem abgegrenzten Umgebung wie einem Secure Element oder eines Trusted Execution Environment auszuführen, so dass auf zusätzliche Hardware wie Kryptomodule auf microSD-Karten oder Smartcard-Lesegeräte [Cer15] verzichtet werden kann. Darüber hinaus bieten die Schnittstellen des Smartphones eine weitreichende Benutzerinteraktion für eine zusätzliche Zugriffssteuerung auf die kryptographischen Operationen und können so vor unberechtigt durchgeführten Zugriffen durch Schadsoftware auf dem Client schützen.

2 Stand der Forschung

2.1 Chipkarten

Die Signaturchipkarte ist eine Chipkarte, die besondere Sicherheitsanforderungen erfüllt. Als Chipkarten bezeichnet man Microkontrollerchips, die in der Regel in eine Plastikkarte integriert sind. Aus Kostengründen sind Funktionsumfang und Speicher auf das Aufgabengebiet der Chipkarte angepasst und auf das Wesentliche reduziert. Eine Signaturchipkarte dient als sichere Umgebung zur Speicherung privater Schlüssel und stellt kryptographische Funktionen bereit, welche die privaten Schlüssel benötigen. Zusätzlich enthält sie Speicher für zusätzliche Informationen wie Zertifikate sowie kryptographische Prozessoren für Langzahlarithmetik, um die kryptographischen Funktionen der Public-Key Kryptographie in annehmbarer Zeit berechnen zu können. Neben kommerziellen Signaturanwendungen, welche sich nach dem PKCS#15 Standard und dessen Nachfolger der ISO/IEC 7816-15 richten, gibt es eine DIN-Spezifikation für die deutsche Signaturchipkarte. Diese schreibt unter anderem die zu unterstützenden Hash- und Signaturalgorithmen und die Dateistruktur zur Speicherung von Schlüsseln, Zertifikaten und Informationen vor [RaEf08, S.985 ff.].

2.2 Datenübertragung

Die Kommunikation der Chipkarte mit einem Lesegerät erfolgt entweder kontaktbasiert über Kontakte auf der Chipkarte oder kontaktlos über den Funkstandard ISO/IEC 14443, der auch für Radio Frequency Identification (RFID) verwendet wird [RaEf08, S.263 ff.].

2.2.1 Near Field Communication

Der 2002 durch die Chiphersteller NXP Semiconductors und Sony spezifizierte Near Field Communication Standard basiert auf dem RFID Funkstandard ISO 14443 und bietet eine einfache und schnelle Möglichkeit, zwei Geräte über kurze Distanzen von bis zu zehn Zentimetern kontaktlos miteinander kommunizieren zu lassen [LaRo10, S.87]. Dabei kann ein Gerät als Sender (aktiv) oder Empfänger (passiv) agieren. Zudem gibt es noch eine bidirektionale Übertragung (aktiv-aktiv), bei der das Gerät als Sender und Empfänger fungiert [LaRo10, S.92 ff.]:

Passiv-aktive Kommunikation: Das Smartphone wird über ein Terminal gelesen und verhält sich wie eine NFC-Karte oder ein NFC-Tag.

Aktiv-passive Kommunikation: Das Smartphone liest einen passiven NFC-Tag oder eine NFC-Karte.

Aktiv-aktive Kommunikation: Zwei Android-Smartphones kommunizieren über Android Beam.

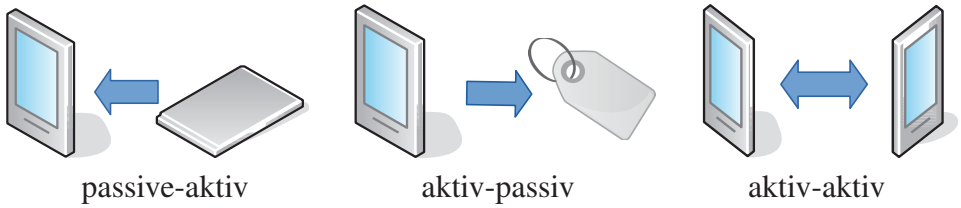

Abb. 1: Kommunikationsarten

2.2.2 Host-based Card Emulation

Viele Smartphones mit NFC-Unterstützung bieten die Möglichkeit eine NFC-Karte zu emulieren. Diese Kartenemulation erfolgt mit Hilfe eines Secure Elements oder einer SIM-Karte. Dieses Secure Element wird durch den Hersteller bzw. den Telekommunikationsanbieter verwaltet und es ist nicht möglich, eigenes Schlüsselmaterial zu importieren oder zu erzeugen [LaRo10, S.145 ff]. Bei einer Kartenemulation mit dem Secure Element wird die Kommunikation des Terminals an dem Betriebssystem vorbei direkt an das Secure Element weitergeleitet [And15b].

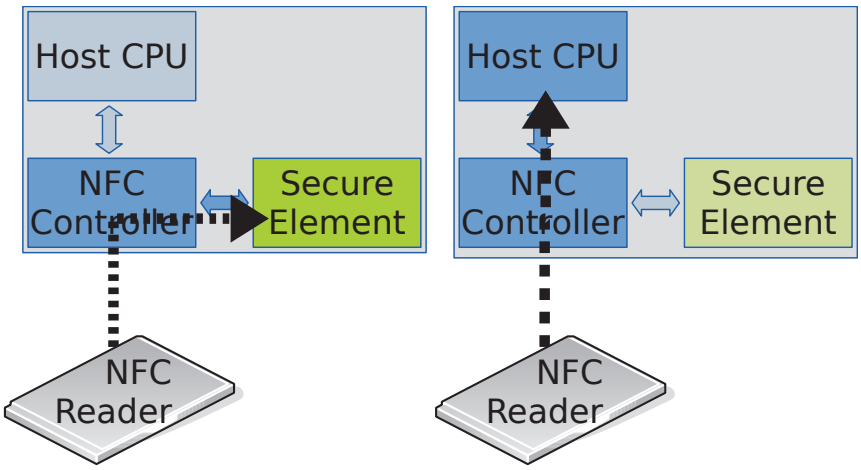

Abb. 2: HCE mit und ohne Secure Element

Mit dem Android API Level 19 (ab Android 4.4 alias Kitkat) kann eine Kartenemulation auch ohne ein Secure Element durchgeführt werden. Die Emulation kann APDUs nach der ISO/IEC 7816-4 Spezifikation verarbeiten. Existiert ein Secure Element, so entscheidet der NFC-Controller anhand der registrierten Application-IDs (AIDs), ob die Kommunikation an die CPU oder an das Secure Element geleitet wird.

2.3 Android-Sicherheitsarchitektur

Das Open Source Betriebssystem Android baut auf einem angepassten Linux Kernel auf. Neben den durch den Linux Unterbau bestehenden Sicherheitsfunktionen bietet Android zahlreiche zusätzliche Sicherheitsaspekte, welche bei der Sicherheitsbetrachtung der Anwendung zum tragen kommen.

2.3.1 Berechtigungen (Permissions)

Der direkte Zugriff auf bestimmte Funktionen der Systemdienste ist eingeschränkt und wird durch ein Berechtigungsmodel geregelt. Grundsätzlich wird jeder Anwendung der Zugriff auf diese Funktionen verwehrt, wenn diese nicht explizit von der Anwendung während der Installation angefordert wurden. Der Anwender bekommt bei der Installation eine Übersicht der angeforderten Berechtigungen und muss entscheiden, ob er die Anwendung installieren möchte [Elen14, S.14 f].

2.3.2 Sandbox

Der Linux Kernel isoliert Ressourcen und Prozesse über die Benutzer- und Gruppenberechtigungen: Jeder Ressource wird ein Tripel an Berechtigungen zugeordnet (Lesen, Schreiben und Ausführen). Diese Berechtigungen können für einen bestimmten User (UID) oder eine Gruppe (GID) vergeben werden. User können Systemdienste oder auch reale Benutzer sein. Android nutzt diesen Mechanismus um Anwendungen voneinander zu separieren: Jede Anwendung erhält eine eigene UID unter der die Anwendung ausgeführt wird und Dateien im Dateisystem unter dem Pfad /data/data/<paketname> angelegt werden. So erhält die Anwendung nur den Zugriff auf ihre eigenen Dateien und kann auf Dateien und Prozesse anderer Anwendungen nicht zugreifen. In der Datei /data/system/packages.list sind die Paketnamen mit den dazugehörigen Datenverzeichnissen und den UIDs aufgelistet [Elen14, S.12 f]:

```
com.example.test 10142 1 /data/data/com.example.test default none
```

2.3.3 Same Origin Policy

Jede Anwendung unter Android muss durch den Entwickler digital signiert werden. Wird eine Anwendung mit dem selben Paketnamen wie eine bereits auf dem System vorhandene Anwendung installiert, so bleiben UID und die Anwendungsdaten bestehen. Lediglich die Anwendung selber wird ausgetauscht. Da Android Anwendungen über ihren Paketnamen identifiziert werden, gibt dieser auch den Namen für das Datenverzeichnis vor, in dem alle zur Anwendung gehörenden Daten abgelegt sind. Daher muss die Anwendung mit dem selben Schlüssel signiert sein, wie die zu ersetzende Applikation. Unterscheiden sich die Schlüssel, muss die installierte Anwendung zuerst deinstalliert werden, was die Löschung der dazugehörigen Daten im Verzeichnis /data/data/<paketname> bewirkt [Elen14, S.16].

2.4 Android-Keystore

Der Android-Keystore ist ein elektronischer Schlüsselbund, welcher unter Android durch die Java Cryptography Architecture bereitgestellt wird und neben Schlüsseln auch die dazugehörigen Zertifikate verwalten kann. Das Schlüsselmaterial wird dabei durch Sicherheitsmaßnahmen vor unberechtigtem Zugriff geschützt. Dieser Schutz wird in erster Instanz durch das Berechtigungsmodel des Betriebssystems durchgesetzt. Zusätzlich können weitere Sicherheitsmodule in Form eines Secure Elements oder eines Trusted Execution Environments eingesetzt werden [And15a].

2.5 Trusted Execution Environment

Als Trusted Execution Environment (TEE) wird eine Laufzeitumgebung bezeichnet, die parallel zu einer unsicheren Umgebung (in der Regel einem vollwertigen Betriebssystem) eine sichere und vertrauenswürdige Ausführung von Programmen ermöglicht. Die Laufzeitumgebung

wird dabei in eine sichere (secure World: SWd) und eine unsichere Umgebung (normal World: NWd) aufgeteilt. Die sichere Umgebung kann dabei von der unsicheren Umgebung physikalisch getrennt, wie es bei den Smartcards der Fall ist, oder direkt in eine Prozessorarchitektur integriert sein. Bei der integrierten Umsetzung teilen sich NWd und SWd die Systemressourcen und die Segmentierung der Umgebungen erfolgt durch die vollständige Ressourcenverwaltung durch die SWd [Geat15, S.35ff]. Um die Integrität der SWd zu gewährleisten, muss diese durch einen sicheren Bootvorgang vor der NWd gestartet werden. Dabei wird die Integrität der geladenen Software anhand deren Signatur überprüft. Dieser Vorgang ist notwendig, da der Speicher des Smartphones als unsichere Quelle betrachtet werden muss und jederzeit durch die NWd manipuliert werden kann. Erst wenn die SWd die vollständige Kontrolle über die Hardwareressourcen besitzt, kann die NWd gestartet werden. Innerhalb der SWd lassen sich nur vertrauenswürdige Anwendungen ausführen, welche eine gültige Signatur des Herstellers besitzen. Innerhalb der SWd sind die Anwendungen strikt getrennt und haben keinen unkontrollierten Zugriff auf Funktionen anderer installierter Anwendungen. In der Praxis besteht eine TEE-gestützte Anwendung aus einem Teil, der in der NWd läuft und einem möglichst wenig komplexen und sicheren Teil, der in der SWd installiert wird. Das Ziel ist es, die Zeit in der sich das System in der SWd befindet so gering wie möglich zu halten, da bei einem vollständig integrierten Design parallel keine NWd Operationen durchgeführt werden können und die Wechsel zwischen NWd und SWd aufwändige Operationen sind [Glob11, S.10 ff].

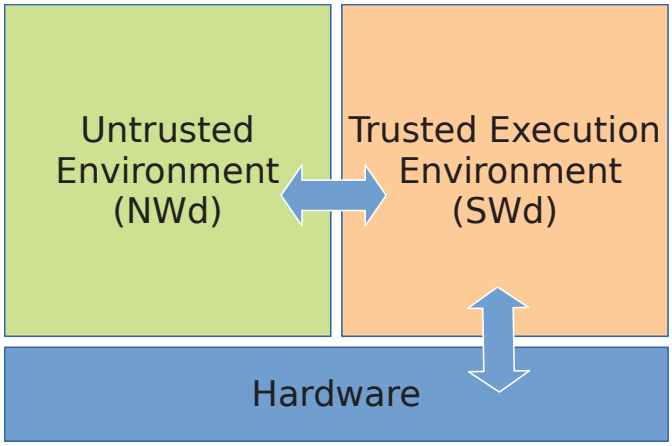

Abb. 3: Architektur eines TEE

3 Signatur-Kartenemulation unter Android

Die im Rahmen dieser Arbeit entwickelte Demoanwendung soll zeigen, dass es möglich ist mit einem Smartphone und dem Betriebssystem Android (Version >= 4.4) eine Signaturchipkarte zu simulieren und das private Schlüsselmaterial in einer sicheren Umgebung aufzubewahren. Der Kern der Anwendung besteht aus einer Android-Service-Klasse (APDU-Service), welche eine sogenannte Application-ID (AID) beim Androidsystem registriert und die Kommunikation über einen Kartenterminal verarbeitet. Das HCE-Framework übernimmt die initiale Kommunikation mit dem Kartenterminal und wartet auf die Auswahl einer Applikation über eine AID. Sobald die registrierte AID vom Terminal ausgewählt wird, übergibt das HCE-Framework jedes folgende Kommando solange an die registrierte Serviceklasse, bis eine andere AID ausgewählt wird. Diese Klasse verarbeitet die Kommandos nach dem Standard ISO-7816 und implementiert

rudimentäre Funktionen zur Dateiverwaltung sowie kryptographische Funktionen zur Signaturbildung und Entschlüsselung.

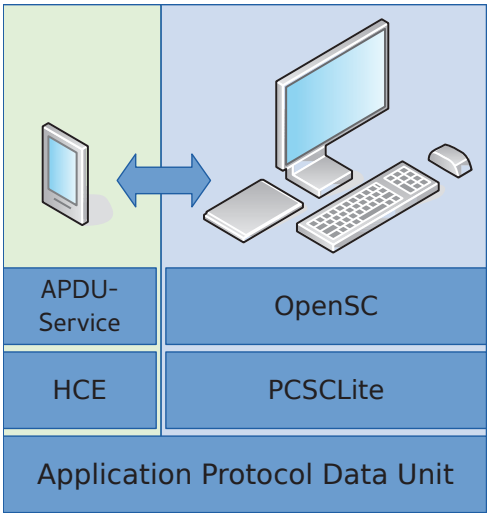

Abb. 4: Systemarchitektur

Auf der Clientseite werden die Kommandos über eine PC/Smartcard-Schnittstelle (PCSClite) entgegen genommen [RaEf08, S.256 f]. Die Implementierung orientiert sich stark an der freien Javacard Signaturanwendung IsoApplet von Philip Wendland [Wend15], um eine spätere Nutzung der PKCS#11 und PKCS#15 Schnittstellen über das OpenSC-Framework möglich zu machen. Die Demoanwendung unterstützt folgenden Befehle:

- `SELECT`
- `CREATE FILE`
- `UPDATE BINARY`
- `READ BINARY`
- `DELETE FILE`
- `GENERATE ASYMETRIC KEYPAIR`
- `PERFORM SECURITY OPERATION`
- `DECIPHER`
- `COMPUTE DIGITAL SIGNATURE`

Die Anwendung generiert und verwaltet das Schlüsselmaterial in einer Android-Keystore Instanz, welche auf einem hardware-gestützten Speicher zurückgreift, sofern dieser auf dem verwendeten Smartphone vorhanden ist. Der Status der Anwendung lässt sich über eine grafische Oberfläche in Form eine Android-Activity ablesen. Sie gibt Auskunft über die gewählten Sicherheitsparameter und die zur Verfügung stehenden Schlüssel sowie deren Hardwarebindung. Jeder Zugriff auf eine kryptografische Funktion wie Schlüsselgenerierung, Signaturbildung oder Entschlüsselung wird über eine zusätzliche Benachrichtigung visualisiert.

Entwickelt wurde die Anwendung mit der Java-Entwicklungsumgebung Eclipse und dem Android Software Development Kit. Der Funktionstest erfolgte auf einem LG D821 (Nexus 5) mit dem Betriebssystem Cyanogenmod 12.1, welches auf Android 5.1 basiert. Die Verbindung

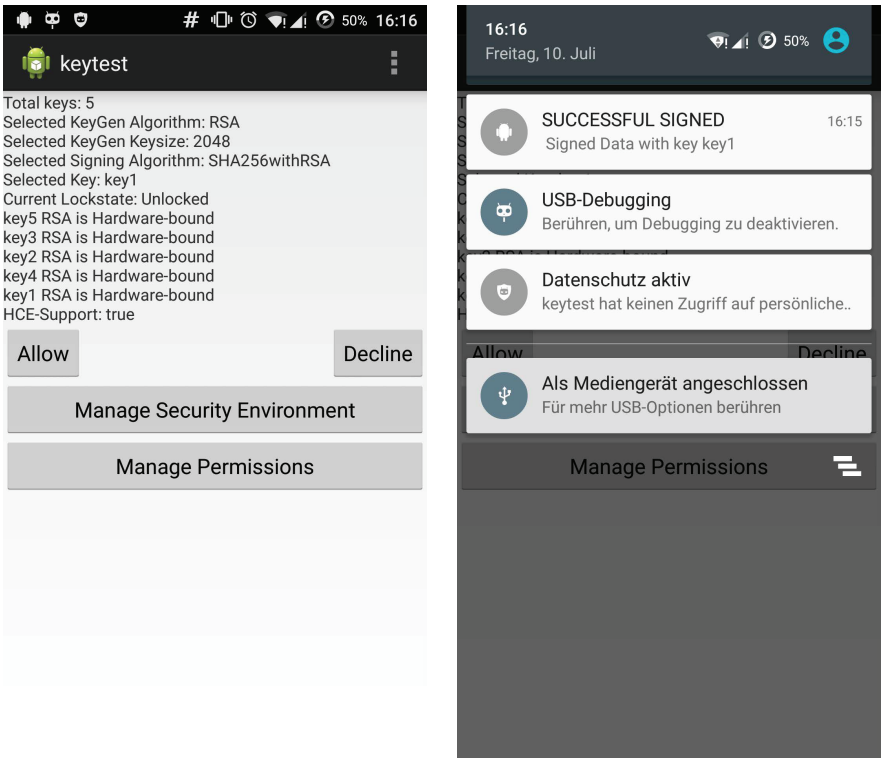

Abb. 5: Anwendung und Benachrichtigung

Tab. 1: Berechnungsdauer in Milisekunden über 1000 Messungen

Funktion	Durchschnitt	Minimal	Maximal
Schlüsselgenerierung (RSA 2048 Bit)	2573 ms	532 ms	10821 ms
Signaturbildung (SHA256withRSA)	291 ms	189 ms	425 ms
Entschlüsselung (RSA/ECB/PKCS1Padding)	293 ms	195 ms	416 ms

zwischen dem Smartphone und dem Clientrechner über NFC wurde mit einem Dual-Interface Smartcard-Reader CardMan 5321 der Firma Omnikey realisiert.

4 Sicherheitsbetrachtung

Ein Smartphone leistet als Signaturkartenersatz nicht die Sicherheitseigenschaften einer Smartcard. Im folgenden werden die Software-seitige Schlüsselspeicherung (Abschnitt 4.1), die Chipkarten-seitige Schlüsselspeicherung (Abschnitt 4.2) und die Hardware-gestützte Schlüsselspeicherung (Android Keystore, Abschnitt 4.3) betrachtet.

4.1 Software-seitige Schlüsselspeicherung

Werden die Schlüssel rein in Software gespeichert, müssen diese entweder direkt im Dateisystem (beispielsweise im PKCS#12-Format) oder über das Betriebssystem in einem Schlüsselbund abgelegt werden. Befindet sich das Schlüsselmaterial nicht im Anwendungsverzeichnis, welches durch Dateiberechtigungen geschützt ist (z.B. auf einer SD-Karte), reicht einem Angreifer bereits der Zugriff auf das Dateisystem aus, um an das Schlüsselmaterial zu gelangen. In der Regel liegen die privaten Schlüssel nicht im Klartext vor, sondern sind durch einen

Verschlüsselungsschlüssel abgesichert. Ist der Verschlüsselungsschlüssel nicht im Dateisystem hinterlegt, da er in der Anwendung abgefragt wird oder bei der Speicherung im Betriebssystem-Schlüsselbund aus dem Sperrpasswort des Gerätes abgeleitet wird, bleibt dem Angreifer nur das Ausprobieren mittels Brute-Force [Elen14, S.175]. Bekommt der Angreifer zusätzlich Zugriff auf den Arbeitsspeicher des Gerätes, kann er den Schlüssel jedoch abfangen, sobald der private Schlüssel entschlüsselt im Speicher liegt. Der Angreifer benötigt für das Auslesen des Arbeitsspeichers Root-Zugriff auf das Smartphone, welcher in der Regel nicht vorliegt. Es gibt wenige Nutzer, die ihr System gerootet haben, um ihr System individuell anzupassen und so die Restriktion der Hersteller umgehen zu können. Jedoch laufen auf vielen Android-Smartphones veraltete Android-Versionen, die durch Exploits angreifbar sind. Mitte 2014 veröffentlichte George Hotz das Tool Towelroot mit dem sich nahezu alle zu dem Zeitpunkt im Umlauf befindlichen Android-Versionen durch eine Sicherheitslücke im Linux-Kernel rooten ließen [Hei14]. Gelangt ein entsprechendes Programm, welches sich Rootrechte auf dem Smartphone verschafft und die Schlüssel aus dem Arbeitsspeicher extrahiert, auf das Gerät, gelangt der Angreifer in den Besitz der privaten Schlüssel.

4.2 Schlüsselspeicherung auf einer Chipkarte

Auf einer Smartcard gespeicherte Schlüssel werden innerhalb eines von einem Chipkartenbetriebssystem verwalteten und nicht öffentlich zugänglichem Speicherbereich abgelegt. Wenn der private Schlüssel nicht ausgelesen werden soll, verhindert das Chipkartenbetriebssystem, dass das Schlüsselmaterial über eine externe Schnittstelle die Karte verlässt. Neben Sicherheitslücken im Betriebssystem der Karte, welche den unberechtigten Zugriff auf den Schlüssel ermöglichen, kann das Schlüsselmaterial durch direkten Speicherzugriff, Seitenkanalangriffe oder Fehleranalyse extrahiert werden. Der technische Aufwand liegt hier um ein vielfaches höher als bei der rein Software-basierten Schlüsselspeicherung: Um Zugriff auf die Hardware zu bekommen und so die Kommunikation über die Datenbusse oder Speicherbausteine direkt abzugreifen, muss deren Position und Anordnung bekannt sein. Ist ein direkter Zugriff auf das Schlüsselmaterial so nicht möglich, besteht die Möglichkeit die Chipkarte bei der Verarbeitung des Schlüsselmaterials zu beobachten und mit Hilfe der gesammelten Daten Rückschlüsse auf den Schlüssel zu erlangen. Mögliche Messwerte für die Beobachtung sind Stromverbrauch, elektromagnetische Abstrahlung oder Berechnungsfehler von kryptografischen Berechnungen [KoJJ99, BoDL99]. Gegen viele Angriffe sind auf der Smartcard bereits Gegenmaßnahmen implementiert, so dass wirksame Angriffe entsprechend aufwendig und teuer sind. Zudem braucht der Angreifer dafür physikalischen Zugriff auf die Karten sowie die PIN, um Zugriff auf die kryptographischen Funktionen zu erhalten, die von dem privaten Schlüssel Gebrauch machen. Sobald der Zugriff auf die Karte durch die Eingabe einer PIN freigegeben ist, hat der Nutzer keine Möglichkeit die Nutzung des Schlüsselmaterials zu überwachen und ist dabei auf Zusatzgeräte wie Kartenterminals mit Display und PIN-Eingabefeld angewiesen.

4.3 Hardware-gestützte Schlüsselspeicherung

Für die Schlüsselgenerierung und -speicherung sowie die kryptographischen Funktionen bei denen die sicheren Schlüssel zum Einsatz kommen, wird auf dem Nexus 5 eine TEE Implementierung der Firma ARM namens TrustZone eingesetzt. Dabei wird, wie bei einer virtuellen Maschine, ein gesicherter und ein ungesicherter Bereich auf dem selben Prozessor ausgeführt. Überwacht wird der ganze Prozess über einen Monitor. Dieser Monitor analysiert, in welchem Kontext (sicher/unsicher) sich die gerade ausführten Befehle befinden und verhindert den Zu-

griff auf gewisse Speicherbereiche. TrustZone teilt sich die Hardware vollständig mit dem unsicheren System und sorgt lediglich für die Separierung. Um zu verhindern, dass die sichere Umgebung manipuliert werden kann, muss deren Integrität durch einen abgesicherten Bootvorgang vor dem Starten des eigentlichen Bootloader gewährleistet sein [ARM08, S.5-5ff.].

Der hardware-gestützte Schlüsselspeicher verhält sich ähnlich wie ein Softwarespeicher in Kombination mit einem Verschlüsselungsschlüssel auf einer Smartcard: Tim Cooijmans hat in seiner Arbeit die Schlüsselspeicherung auf Basis der TrustZone analysiert und herausgefunden, dass für jedes generierte Schlüsselpaar unter dem Verzeichnis /data/misc/keystore/<user_id> zwei Dateien angelegt werden. Die Dateien tragen die UID der Anwendung von welcher sie generiert wurden, gefolgt von USRCERT für das Zertifikat bzw. USRPKEY für den privaten Schlüssel und dem gewählten Schlüssel-Alias:

```
-rw------- keystore keystore  708 2015-06-07 21:23 10142_USRCERT_key1
-rw------- keystore keystore  708 2015-06-07 21:24 10142_USRCERT_key2
-rw------- keystore keystore 1652 2015-06-07 21:23 10142_USRPKEY_key1
-rw------- keystore keystore 1652 2015-06-07 21:24 10142_USRPKEY_key2
```

Verschiedene Tests mit den Dateien, die von Cooijmans durchgeführt wurden, lassen vermuten, dass ein geräteabhängiger Verschlüsselungsschlüssel genutzt wird, um die privaten Schlüssel im Dateisystem zu verschlüsseln [Cooi14, S.30 ff.]. Nikolay Elenkov hat die Trustzone Implementierung eines Nexus 4 untersucht und vermutet, dass die privaten Schlüssel dort mit einem 128 Bit AES Schlüssel verschlüsselt sind [Elen14, S.179]. Dieser ist im TEE untergebracht und entschlüsselt den privaten Schlüssel bevor dort die kryptographischen Operationen durchgeführt werden. Durch ein zurücksetzten des Gerätes wird dieser Schlüssel nicht verändert: Werden die Daten unter dem Verzeichnis /data/misc/keystore/<user_id> nach dem Zurücksetzten des Gerätes wiederhergestellt, steht das Schlüsselmaterial weiterhin zur Verfügung. Gelangt der Angreifer an Dateisystemzugriff, so bekommt er lediglich Zugriff auf die verschlüsselten Schlüssel. Dies entspricht dem Szenario der in Software gespeicherten Schlüssel ohne Arbeitsspeicherzugriff: Gelangt ein Angreifer an Rootzugriff, erhält er zwar keinen direkten Zugriff auf den privaten Schlüssel, er kann jedoch an dem Sandbox-Prinzip von Android vorbei das Schlüsselmaterial nutzen.

Um an den privaten Schlüssel zu kommen, muss der Angreifer an den geräteabhängigen Schlüssel aus dem TEE gelangen. Dazu kann er über die Hardware den Speicher auslesen. Wie die Smartcards besitzt die Smartphone-Hardware Debugging-Schnittstellen. Der Aufbau der Smartphone-Hardware ist um ein vielfaches komplexer und erschwert dadurch die Suche nach dem Schlüsselmaterial erheblich. Seitenkanalanalysen sind aufgrund der Vielzahl an Operationen auf unterschiedlichen Kernen mit ständig wechselnden Taktraten ebenfalls deutlich gegenüber der Smartcard erschwert [DFLM+14, Kapitel 13]. Da jedoch bei Berechnungen in der sicheren Umgebung der TrustZone nur Anwendungen im sicheren Kontext ausgeführt werden, könnte dies Seitenkanalangriffe begünstigen.

Einen weiteren Angriffspunkt bietet die Firmware des TEE. Gelingt es dem Angreifer diese auszutauschen, bekommt er Zugriff auf den geräteabhängigen Schlüssel. Die von Cooijmans gefundenen Treiber für die TrustZone Keystore Anbindung enthielten digitale Signaturen. Diese werden während des Bootvorgangs überprüft und nur mit gültiger Signatur in das TEE geladen. Fehlerhafte Implementierungen der Signaturüberprüfung können dazu führen, dass eine ungültige Signatur als gültig erkannt wird [CVE14]. Ein Fehler in der Speicherverwaltung des TEE kann dazu führen, dass ein Angreifer Zugriff auf den Speicherbereich der sicheren Umge-

bung erhält und den geräteeigenen Schlüssel auslesen kann.

Erhält der Angreifer schon vor der Schlüsselgenerierung Vollzugriff auf das Gerät, so kann er das Betriebssystem so manipulieren, dass das Schlüsselmaterial nicht, wie vom Benutzer erwartet, innerhalb der TEE generiert wird, sondern im unsicheren Betriebssystem.

Ist der Angreifer der Hersteller oder kann in den Fertigungsprozess eingreifen, so kann er die sichere Software innerhalb des TEE und den geräteeigenen Schlüssel manipulieren. Sollte der nicht änderbare geräteeigene Schlüssel irgendwann kompromittiert werden, kann das Gerät mit dem darauf befindlichen Schlüsselmaterial nicht mehr verwendet werden.

5 Fazit und Ausblick

Mit dieser Arbeit wurde die Machbarkeit einer Signaturfunktionalität auf Basis eines Android-Smartphones mit Hilfe der Host-based Card Emulation demonstriert. Sie stellt alle kryptographischen Funktionen bereit, welche einen privaten Schlüssel benötigen. Die Schlüsselerzeugung und Speicherung erfolgt dabei über eine Android-Keystore-Implementierung, die, sofern vorhanden, auf eine Hardwareunterstützung zurückgreift und den direkten Zugriff auf den privaten Schlüssel aus dem Androidsystem heraus verhindert. Ein transparentes Dateisystem ermöglicht das Ablegen von Binärdateien, so dass eine Middleware in der Lage ist, zusätzliche Informationen wie Zertifikate auf der Signaturkartenemulation abzulegen. Die Benachrichtigung bei Zugriff auf die kryptographischen Funktionen erhöht den Schutz vor Missbrauch durch Schadsoftware über den Client. Ist das Android-Smartphone nicht gerootet und der Bootloader gesperrt, so ist das Schlüsselmaterial innerhalb des TEE für viele Anwendungsszenarien ausreichend geschützt. Auf gerooteten Geräten besteht die Gefahr, dass Schadsoftware das Schlüsselmaterial unbemerkt nutzen kann. Die Anlehnung an eine existierende Open-Source JavaCard Implementierung macht die spätere Nutzung bereits existierender Treiber und Middleware möglich, so dass diese nicht zusätzlich entwickelt werden müssen. Um das Smartphone im vollen Umfang als Signaturkartenersatz nutzen zu können, muss durch Implementierung der noch fehlenden Funktionen eine vollständige Kompatibilität zum IsoApplet hergestellt werden. Die Demoanwendung stellt hier derzeit nur die nötigsten Befehle zur Umsetzung eines transienten Dateisystems und Grundfunktionen der asymmetrischen Kryptographie bereit. Durch die strikte Trennung des Dateisystems und des Schlüsselmaterials steht derzeit der öffentliche Schlüssel nur über den Rückgabewert bei der Schlüsselerzeugung zur Verfügung. Es wäre denkbar, das Dateisystem vollständig über den Keystore-Inhalt abzubilden. So stände neben der Signaturkartensimulation über die NFC-Schnittstelle dem Androidsystem der Keystore als Equivalent zur Verfügung.

Um die Sicherheit der Anwendung zu erhöhen, ließe sich das Schlüsselmaterial zusätzlich mit einem Benutzerschlüssel, der beispielsweise aus einer PIN abgeleitet wird, verschlüsseln. So würde der Zugriff auf das Gerät allein nicht mehr ausreichen, um das Schlüsselmaterial nutzen zu können.

Literatur

[And15a] Android Developer Website: Keystore System (2015). http://developer.android.com/training/articles/keystore.html, abgerufen am 06.05.2016.

[And15b] Android Developer Website: Host-based Card Emulation (2015).

https://developer.android.com/guide/topics/connectivity/nfc/hce.html, abgerufen am 06.05.2016.

[ARM08] ARM Security Technology: Trustzone Security Whitepaper (2008). http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf , abgerufen am 11.7.2015.

[BoDL99] D. Boneh, R. A. DeMillo, R. J. Lipton: On the Importance of Checking Cryptographic Protocols for Faults. http://crypto.stanford.edu/ dabo/abstracts/faults.html (1999), abgerufen am 06.05.2016.

[Cer15] certgate: Produktübersicht. http://www.certgate.com/de/produkte (2015), abgerufen am 06.05.2016.

[Cooi14] T. Cooijmans: Secure Key Storage and Secure Computation in Android. Radboud Univerity Nijmegen (2014).

[CVE14] CVE-2014-0973: Incomplete signature parsing during boot image authentication leads to signature forgery. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0973 (2014), abgerufen am 11.7.2015.

[DFLM+14] J. J. Drake, P. O. Fora, Z. Lanier, C. Mulliner, S. A. Ridley, G. Wicherski: Android Hacker's Handbook. Wiley & Sons, Inc. (2014).

[DiKl14] G. Disterer, C. Kleiner: Mobile Endgeräte im Unternehmen. Springer Fachmedien Wiesbaden (2014).

[Elen14] N. Elenkov: Android Security Internals. No Starch Press (2014).

[Geat15] J. Geater: Trusted Computing for Embedded Systems. Springer International Publishing (2015).

[Glob11] GlobalPlatform: The Trusted Execution Environment (2011). http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf, abgerufen am 06.05.2016.

[Hei14] Heise Security: Towelroot knackt Android in Sekunden (2014). http://www.heise.de/security/meldung/Towelroot-knackt-Android-in-Sekunden-2225143.html, abgerufen am 11.7.2015.

[KoJJ99] P. Kocher, J. Jaffe, B. Jun: Differential Power Analysis (1999). https://www.rambus.com/differential-power-analysis, abgerufen am 06.05.2016.

[LaRo10] J. Langer, M. Roland: Anwendung und Technik von Near Field Communication (NFC). Springer Verlag (2010).

[RaEf08] W. Rankl, W. Effing: Handbuch der Chipkarten. Carl Hanser Verlag GmbH & Co. KG (2008).

[Wend15] P. Wendland: Iso Applet Landing Page. http://www.pwendland.net/IsoApplet (2015), abgerufen 06.05.2016.