Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

ERHEBUNG STATE-OF-THE-ART REMOTE-ATTESTATION

Version 1.0 vom 17.06.2019
Bernd Prünster – bernd.pruenster@a-sit.at

Abstract/Zusammenfassung: Remote-Attestation und Trusted-Computing gewinnen insbesondere im Mobilbereich und im DRM-Kontext zunehmend an Bedeutung. Die Umsetzung der zu Grunde liegenden Konzepte unterscheidet sich jedoch zwischen Mobil- und Desktop-Computing-Bereich. Im Rahmen dieses Projekts wurde der aktuelle Stand von Remote-Attestation-Verfahren im Desktop und Mobilbereich erhoben, wobei insbesondere die Praktikabilität im Kontext offener Systeme – ohne explizite Enrolment-Prozeduren – beleuchtet wurde. Dieses Dokument gibt einen Überblick über grundlegende Trusted-Computing-Konzepte, stellt Intels Software Guard Extensions (SGX) als Beispiel für Trusted-Computing im Desktopumfeld vor und beschreibt den Stand der Dinge im Mobilbereich. Darüber hinaus wird ein Trusted-Computing-Modell für aktuelle Android-Geräte mit Remote-Attestation-Unterstützung vorgestellt, welches in diesem Projekt erarbeitet wurde und im Juli diesen Jahres im Rahmen der 16th International Conference on Security and Cryptography präsentiert wird.

Inhaltsverzeichnis

1. Einleitung

Remote-Attestation ist ein integraler Bestandteil von Trusted-Computing-Konzepten [1] und erlaubt es, verlässlich festzustellen, ob ein System zum Zeitpunkt der Überprüfung integer ist. Theoretisiert wurde über solche Systeme bereits vielfach; konzeptionell und innerhalb abgeschlossener Szenarien waren diese auch bereits bisher umsetzbar. Ein ungelöstes Problem war lange Zeit

jedoch, wie es auf globaler Ebene möglich sein soll, die Integrität eines entfernten Systems nachzuweisen, ohne zu diesem zuvor Kontakt gehabt zu haben und ohne dass dies lediglich innerhalb eines geschlossenen Gesamtsystems stattfinden soll. Folglich handelt es sich dabei um Fragen bei der Umsetzung theoretisch bereits gelöster Probleme in der Praxis. Konkret stellt sich die Frage nach einem Vertrauensmodell, das allgemein anerkannt ist, bzw. dessen Root-of-Trust anerkannt ist. Die erste Implementierung, die nennenswert Verbreitung fand, und deren Trust Anchor breites Vertrauen genoss, war Intels *Software Guard Extensions*[1], wobei diese Technologie zwar prinzipiell betriebssystemunabhängig, jedoch an einen Chiphersteller (Intel) gebunden ist.
In den letzten Jahren wurde auch im Mobilbereich an entsprechenden Lösungen gearbeitet [2]. In diesem Kontext ergibt sich insofern ein Vorteil gegenüber Desktop-Systemen, als dass verifizierte Bootvorgänge, restriktive Sicherheitsmodelle und Sandboxing im Mobilbereich bereits etabliert sind [3]. Entgegen Trusted-Computing-Konzepten im Desktopbereich muss so nicht erst ein vom Betriebssystem isolierter Bereich geschaffen werden, um eine vertrauenswürdige Umgebung für die Ausführung von sensiblem Programmcode bereitzustellen. Stattdessen wurde im Mobilbereich das Android-Betriebssystem selbst schrittweise um Trusted-Computing-Features und Schnittstellen zu kryptografischen Hardwaremodulen erweitert [4] [5] [6]. Die im Rahmen dieses Projekts entwickelten Konzepte nutzen diese Funktionalität auf eine Art und Weise, die es auch praktisch ermöglicht, den Integritätszustand von Betriebssystem und Applikationen aus der Ferne festzustellen, ohne dass beispielsweise Mobile-Device-Management-Lösungen erforderlich sind. Das zugehörige Vertrauensmodell basiert auf der Annahme, dass ein unmodifiziertes Betriebssystem vertrauenswürdig ist, und die Hardware wie vorgesehen funktioniert. Fragen nach vertrauenswürdigen IO-Schnittstellen (Sensoren, Bildschirm, Eingabegeräte) stellen sich durch die hohe Integration der Komponenten miteinander (und mit dem Betriebssystem) im Gegensatz zu traditionellen Trusted-Computing-Modellen nicht.

Dieses Dokument gibt einen Überblick über aktuelle Remote-Attestation-Verfahren sowohl im Desktop- als auch im Mobilbereich. Dabei wurde insbesondere die Praktikabilität, bzw. die Flexibilität der Lösungen untersucht. Der nachfolgende Abschnitt skizziert Trusted-Computing-Grundlagen und diskutiert insbesondere die Notwendigkeit von Remote-Attestation in diesem Zusammenhang. Anschließend wird in Abschnitt 3 Remote-Attestation im Desktopbereich am Beispiel von Intel SGX als Trusted-Computing-Ansatz diskutiert. Abschnitt 4 beschreibt die Situation im Mobilbereich und stellt ein im Kontext dieses Projekts entstandenes Verfahren für umfassende Remote-Attestation-Funktionalität unter Android vor. Den Abschluss dieses Dokuments bildet Abschnitt 5 mit einer Zusammenfassung der Projektergebnisse.

2. Grundlagen von Trusted Computing und Remote-Attestation

Remote-Attestation ist im Zusammenhang von Trusted Computing essentiell – ein vertrauenswürdiges, integres System hat im Rahmen von Cloud-Computing, oder anderen verteilten Anwendungen schlichtweg kaum Mehrwert, wenn der Nachweis dieses Systemzustands nicht aus der Ferne möglich ist. Da jedoch besonders dieser Aspekt ein in der Praxis lange Zeit ungelöstes Problem darstellte, wurde dieser Facette von Trusted Computing im Rahmen dieses Projekts besondere Aufmerksamkeit gewidmet. Grundsätzlich ist jedoch festzuhalten, dass tatsächlich nie die Integrität eines gesamten Systems nachgewiesen wird, bzw. werden kann oder soll, sondern dass zwischen nicht vertrauenswürdigen Systemkomponenten und vertrauenswürdigen Teilen differenziert wird und lediglich zu letzterem eine Aussage getroffen wird. Die Summe dieser vertrauenswürdigen Teile wird als *Trusted Computing Base* (TCB) bezeichnet [1]. Als Abgrenzung dieser beiden Systemteile voneinander bietet sich folgende Definition an: Die Trusted Computing Base ist jene Menge an Software und Hardware von der die Systemsicherheit direkt abhängig ist, im Gegensatz zu den restlichen Komponenten, deren (Fehl)verhalten keinen Einfluss auf die Systemsicherheit hat [7].

Auf Basis dieser Definition lässt sich ein allgemeiner Anwendungsfall für Trusted Computing skizzieren: Ziel ist es, dass kritische Operationen, welche nicht lokal ausgeführt werden können oder sollen, auf entfernten Geräten abgearbeitet werden, wobei sichergestellt werden können muss, dass

[1] https://software.intel.com/en-us/sgx

diese ausgelagerten Operationen auch tatsächlich in unveränderter Form ausgeführt werden. Dies umfasst per Definition den funktional korrekten Ablauf von Programmcode, oftmals stehen jedoch insbesondre Datenschutz-, Privatsphäre- und Sicherheitsaspekte im Vordergrund. Wenn etwa aufwändige Rechenoperationen aus Kostengründen und/oder mangels lokal verfügbarer Ressourcen in die Cloud ausgelagert werden sollen, diese Operationen aber die Verarbeitung sensibler Daten beinhalten, wäre es wünschenswert, dass mittels kryptografischer Methoden unerlaubter Zugriff auf diese Daten (auch durch den Service-Provider) verhindert werden kann. Besonders in solchen Situationen ist Remote-Attestation kritisch: Wenn nachgewiesen werden kann, dass in die Cloud ausgelagerter Programmcode innerhalb einer Trusted Computing Base ausgeführt wird, können die Vorteile von Cloud Computing genutzt werden, ohne dass unerlaubter Zugriff auf ausgelagerte Daten befürchtet werden muss. Dadurch können die typischerweise mit Cloud Computing in Verbindung gebrachten Sicherheitsbedenken zerstreut werden. Ein vorheriger Kontakt mit Cloud-Computing-Instanzen im Rahmen von etwaigen Enrolment-Prozeduren ist in derartigen Szenarien per Definition ausgeschlossen.

Um die eben beschriebene Funktionalität umzusetzen, ist immer Hardwareunterstützung notwendig. Soll beispielsweise das Betriebssystem Teil der Trusted Computing Base sein, ist zwingend ein verifizierter Bootvorgang notwendig, im Rahmen dessen die Hardware sicherstellt, dass nicht unbemerkt ein modifiziertes Betriebssystem gestartet werden kann. Dies wird typischerweise erreicht, indem integre Betriebssystem-Images vom Hersteller signiert werden und die Signaturprüfung im Rahmen des Bootvorgangs von der Hardware vorgenommen wird [8]. Traditionellerweise wird dies im Mobilbereich [9] und bei Spielkonsolen [10] eingesetzt. In beiden Fällen umfasst die Trusted Computing Base das Gesamtsystem, wobei im Kontext von Spielkonsolen die Durchsetzung von Kopierschutzmaßnahmen und Eindämmung von Piraterie im Vordergrund steht. Remote-Attestation ist hier insbesondere im Rahmen von Online-Spieleinhalten und Funktionalitäten wir Online-Multiplayer kritisch. So kann beispielsweise die Verwendung aktueller Betriebssystem- und Softwareversionen erzwungen werden. Wie im Mobilbereich erlauben erst die hohe Integration der Komponenten (Hardware, Betriebssystem, Softwaretitel, Serverinfrastruktur) und die Restriktionen bezüglich der vom Benutzer, bzw. der Benutzerin durchführbaren Aktionen (z.B. kein Installieren eigener Betriebssysteme) die Umsetzung einer durchgängigen, das gesamte System umfassenden, Trusted Computing Base.
Im Desktop- und Cloud-Computing-Bereich bietet sich aus historischen Gründen ein völlig entgegengesetztes Bild. Serviceprovider, sowie Nutzerinnen und Nutzer haben üblicherweise Vollzugriff auf die Hardware und können beliebige Betriebssysteme installieren, diese nach eigenen Wünschen anpassen, sowie auch die Hardware modifizieren. Dadurch ergibt sich ein sehr heterogenes Umfeld und eine umfassende Trusted Computing Base ist schlicht nicht umsetzbar. Nähere Details zur Umsetzung im Desktopbereich werden im nachfolgenden Abschnitt erläutert.

3. Remote-Attestation im Desktopumfeld

Im Desktopumfeld wird aufgrund der historisch bedingten Offenheit der Plattform versucht, die Trusted Computing Base möglichst klein und unabhängig vom Betriebssystem zu halten [11]. Da sich vorerst lediglich Intels *Software Guard Extensions* (SGX) durchsetzen konnten, widmet sich dieser Abschnitt SGX, um exemplarisch Trusted Computing und insbesondere Remote-Attestation-Verfahren im Desktopbereich zu analysieren.

3.1. SGX als Trusted Computing Base

SGX definiert das Konzept von so genannten (*secure*) *Enclaves* als von der CPU zur Verfügung gestellte, gesonderte Ausführungsumgebung für sensible Operationen [12]. Aktuelle Intel CPUs verfügen hierzu über spezielle Instruktionen zum Erstellen von innerhalb solcher Enklaven lauffähigem Programmcode. Enklaven werden zwar betriebssystemunterstützt erstellt, die Codeausführung erfolgt jedoch vollkommen abgeschottet von Betriebssystem und anderen Anwendungen. Schnittstellen um Daten mit Enklaven austauschen zu können, müssen vom Programmierer, bzw. der Programmiererin erstellt werden, da die CPU prinzipiell jeglichen Zugriff auf Enklaven von außen verhindert, um deren Sicherheit zu garantieren. Da die aus Teilen der CPU und den Enklaven bestehende, vertrauenswürdige Hard- und Software nicht über separaten

(eventuell speziell vertrauenswürdigen) Hauptspeicher verfügt, sondern sich diesen mit dem Betriebssystem teilt, wird der von Enklaven benutzte Speicher von der CPU verschlüsselt, sodass sich eine logisch vom restlichen System isolierte Trusted Computing Base ergibt. Für Attestation-Funktionalität, Zugriffskontrolle, und Integritätsprüfungen muss der in einer Enklave auszuführende Code vom Entwickler, bzw. der Entwicklerin signiert werden. Insbesondere für die Nutzung von Remote-Attestation, muss hierfür eine Lizenz von Intel erworben werden [13].
Dieses technisch komplexe Konzept führt insgesamt zu einem wohldefinierten Sicherheits- und Vertrauensmodell: Die Integrität von Code, der in Enklaven ausgeführt wird, kann überprüft werden, das restliche System (Hardware und Software) wird als nicht vertrauenswürdig angesehen. Dies gilt insbesondere auch für Ein- und Ausgabegeräte (wobei es vertrauenswürdige Kommunikations-kanäle zwischen CPU und (integrierter) GPU gibt [14]), was bedeutet, dass es keine Möglichkeit gibt, innerhalb des Vertrauensmodells Benutzereingaben zu erhalten. Darüber hinaus sind die Ressourcen (insbesondere Speicher), die innerhalb einer Enklave genutzt werden können, begrenzt. Tatsächlich wird das Auslagern klar abgegrenzter, aber kritischer Funktionalität (wie z.B. kryptografischer Operationen) als Einsatzgebiert von SGX propagiert [15].

Weitere Details zur Funktionalität und zum Vertrauens- und Sicherheitsmodell der Software Guard Extensions können den von Intel zur Verfügung gestellten Ressourcen[2] entnommen werden. Nachfolgend wird der Remote-Attestation-Prozess im Rahmen von SGX diskutiert.

3.2. SGX Remote-Attestation

Intel unterscheidet strikt zwischen lokaler Attestation (zwischen zwei Enklaven, welche auf demselben System ausgeführt werden) und Remote-Attestation. Die Aussagekraft der Resultate beider Verfahren ist jedoch ident und bestätigt laut Intel [16] im SGX-Kontext die folgenden Eigenschaften einer Enklave:

1. Die Identität der Enklave

2. Die Integrität des Codes der Enklave

3. Dass die Enklave tatsächlich auf echter Intel-Hardware ausgeführt wird (und nicht etwa innerhalb einer virtuellen Maschine, oder eines Emulators)

4. Dass die Enklave tatsächlich isoliert und vom restlichen System geschützt ausgeführt wird

Unabhängig vom Anwendungsfall, im Rahmen dessen Remote-Attestation benötigt wird, ist im SGX-Kontext ein von Intel betriebenes Service involviert[3]. Technisch liegt dies in den verwendeten kryptografischen Verfahren basierend auf anonymen Gruppensignaturen begründet [16], wobei die genauen Implementierungsdetails bisher nicht von Intel veröffentlicht wurden [15]. Lokal (auf der CPU, die eine zu attestierende Enklave ausführt) ist eine so genannte *Quoting Enclave*, welche auf an Intel gebundenes Schlüsselmaterial Zugriff hat, dafür zuständig, die Integrität der zu attestierenden Enklave zu ermitteln. Die Ergebnisse dieser Operation werden verschlüsselt an einen von Intel betriebenen Attestation-Service übertragen, welcher diese auswertet und schließlich im Rahmen des Attestation-Prozesses an die Stelle übermittelt, welche den Attestation-Prozess angestoßen hat, um die Integrität einer Enklave zu ermitteln.
Konkrete Details können unter anderem einem (ausführlichen, jedoch technisch komplexen) Begleitdokument zu einem von Intel zur Verfügung gestelltem Codebeispiel zum Thema Remote-Attestation entnommen werden [16]. Das von SGX eingesetzte Remote-Attestation-Verfahren ist insgesamt als komplex zu bewerten und erfordert tiefgehendes technisches Verständnis der Materie, sowie Programmierkenntnisse in C/C++ und Intel-Assembler.

[2] https://software.intel.com/en-us/sgx

[3] Tatsächlich kann auch ein eigenes Attestation-Service betreiben werden [12], allerdings ergibt sich dadurch wieder ein geschlossenes System, ohne allgemein akzeptierten Trust-Anchor, weshalb dieses Szenario im Rahmen dieses Dokuments im Hinblick auf breite Anwendbarkeit in offenen Systemen nicht weiter betrachtet wird.

Durch die zwar auf technologischer, aber nicht notwendigerweise auf konzeptioneller Ebene enge Bindung an Intel-spezifische Entwicklungen stellt sich die Frage der Nachhaltigkeit dieses Ansatzes. Eine Wiederverwendbarkeit z.B. auf AMD- oder ARM-CPUs ist de-facto ausgeschlossen, ebenso wie die einfache Integration von SGX-Remote-Attestation-Abläufen in bestehende Anwendungen. Im Mobilbereich herrscht unter Android (siehe Abschnitt 4.2) eine vollkommen andere Situation. Dies ist, wie nachfolgend erläutert, insbesondere auf ein dem Desktopbereich gegensätzliches Vertrauensmodell zurückzuführen.

4. Remote-Attestation im Mobilbereich

Aktuell gibt es mit Apples *iOS* und Googles *Android* zwei Betriebssysteme mit signifikantem Marktanteil im Mobilbereich. Beiden Plattformen gemein sind restriktive Sicherheitsmodelle auf Basis verifizierter Bootvorgänge und Sandboxing. Aus Nutzersicht bedeutet das, dass nur auf jene Ressourcen zugegriffen werden kann, welche vom Betriebssystem, bzw. von Applikationen explizit zur Verfügung gestellt werden. Datenaustausch zwischen Applikationen ist ebenfalls nur möglich, wenn dies von allen beteiligten Applikationen unterstützt wird. Darüber hinaus kommt flächendeckend Dateisystemverschlüsselung zum Einsatz. Eine Entschlüsselung findet im Rahmen des Bootvorgangs nach Eingabe eines Passworts statt. Die Verwaltung von Schlüsselmaterial übernimmt ein eigens dafür vorgesehenes Hardwaremodul. Zwar unterscheiden sich die Umsetzungen dieser Konzepte im Detail zwischen iOS und Android, diese Aspekte sind jedoch im Zusammenhang mit Remote-Attestation nicht weiter relevant. Während Sandboxing, die Isolation von Applikationen voneinander, sowie der restriktive Zugriff auf Ressourcen rein mittels Softwaremechanismen durchgesetzt wird, ist Dateisystemverschlüsselung an die Hardware gebunden. Die spezifischen Details zum Sicherheitsmodell des jeweiligen Systems, insbesondere im Hinblick auf Trusted Computing und Remote-Attestation unter praktischen Aspekten, sind den nachfolgenden Abschnitten zu entnehmen.

4.1. iOS

Das iOS-Sicherheitsmodell lässt im regulären Betrieb keine Installation von Applikationen zu, welche nicht direkt aus dem von Apple betriebenen *AppStore* bezogen werden [9]. Die auf diesem Weg verteilte Software hat einen von Apple durchgeführten Review-Prozess durchlaufen, der auch (potentiell nur teilweise) im Falle von Applikationsupdates wiederholt wird. Die Details zu diesem Prozess sind nicht öffentlich dokumentiert. Das erklärte Ziel ist laut Apple, hohe Qualität, möglichst Fehlerfreiheit und Sicherheit zu garantieren. Lediglich im Rahmen der Applikationsentwicklung ist es möglich, Software direkt, ohne die Einbindung des AppStore auf iOS-Geräten zu installieren. In Kombination mit einem verifizierten Bootvorgang, welcher mit Unterstützung kryptografischer Hardwaremodule das Installieren und Starten ausschließlich von von Apple signierten Betriebssystemabbildern erlaubt, ergibt sich ein geschlossenes Ökosystem.
Im Prinzip entspricht dies einer das gesamte Gerät umfassenden Trusted Computing Base. Folglich kann (theoretisch; s.u.) davon ausgegangen werden, dass Applikationen, welche über den AppStore vertrieben werden, auf einer Trusted Computing Base ausgeführt werden. So lange die Integrität der TCB nicht kompromittiert ist, lassen sich auch sensible Operationen (und Daten) auf iOS-Geräte auslagern. Schlüsselmaterial für bestimmte kryptografische Verfahren wird darüber hinaus automatisch innerhalb eines kryptografischen Hardwaremoduls erzeugt und kann dies nicht verlassen, wodurch selbst Fehler im Betriebssystem keine Extraktion ermöglichen [9].
Im Prinzip ergibt sich durch das restriktive Betriebssystem- und Softwarevertriebskonzept ein hohes Sicherheitsniveau, jedoch stellt iOS keinerlei Remote-Attestation-Möglichkeiten bereit, wodurch die Integrität des Systems aus der Ferne lediglich angenommen werden kann. Selbst der Nachweis, dass Schlüsselmaterial nicht extrahierbar, innerhalb kryptografischer Hardware erzeugt wurde, kann nicht erbracht werden. Implementierungsfehler werden unter iOS traditionell erfolgreich ausgenutzt, um unautorisierte Änderungen am Betriebssystem vorzunehmen [17]. Diese als *Jalbreak* bekannten Verfahren werden inklusive Anleitung im Internet verbreitet und erlauben es unter anderem, nicht über den AppStore vertriebene Software zu installieren, oder auf das gesamte Dateisystem – und damit auch auf eigentlich geschützte Applikationsdaten – zuzugreifen.

Zusammenfassend lässt sich festhalten, dass durch das geschlossene Ökosystem um iOS und das durchgängige Sicherheitskonzept basierend auf verifizierten Bootvorgängen und kryptografischen Hardwaremodulen zwar ein hohes Sicherheitsniveau erreicht wird, der Zustand, bzw. die Integrität dieses Systems und der von Applikationen jedoch nicht aus der Ferne nachgewiesen werden kann. Ein erfolgreicher Jailbraik reicht aus, um z.B. mittels Reflection zur Laufzeit auch den Code eigentlich integrer Applikationen zu modifizieren. Somit kann aus der Ferne keine Aussage über den Zustand eines iOS-Geräts und der darauf ausgeführten Software getroffen werden. Die iOS-Plattform implementiert somit kein durchgängiges Trusted-Computing-Konzept und kann nicht als unter praktischen Umständen auch als solche einsetzbare Trusted Computing Base klassifiziert werden. Daran ändert auch der Umstand, dass (Sicherheits-)updates von Apple rasch und regelmäßig ausgerollt werden, nichts.

4.2. Android

Bei Android handelt es sich um eine sehr viel offenere, heterogenere Plattform als iOS: Geräte werden von unterschiedlichen Herstellern produziert, Modifikationen am Betriebssystem können vom Hersteller vorgenommen werden, Applikationen können auch aus nicht vom Hersteller oder von Google kontrollierten Quellen installiert werden und der Bootloader vieler Geräte kann unproblematisch entsperrt werden, um unsignierte Betriebssystemabbilder ohne verifizierten Bootvorgang zu starten. Im Gegensatz zu iOS kann man als Distributor von Applikationen unter Android – selbst wenn nur integre Geräte in Betracht gezogen werden – nicht ausschließen, dass modifizierte Versionen der eigenen Anwendung im Umlauf sind. Grund dafür ist, dass es keine technischen Maßnahmen gibt, die verhindern, dass Android-Softwarepakete modifiziert werden, bevor diese auf ansonsten unkompromittierten Geräten installiert werden. Auf Grund umfassender Remote-Attestation-Verfahren aktueller Android-Geräte gibt es dennoch Möglichkeiten, verlässlich festzustellen, ob eine vertriebene Applikation in unmodifizierter Form auf einem integren Gerät, auf dem ein unmodifiziertes, vom Hersteller signiertes, im Rahmen eines verifizierten Bootvorgangs überprüftes Betriebssystemabbild gebootet wurde, ausgeführt wird. Wie dies bewerkstelligt werden kann, wurde im Rahmen dieses Projekts erarbeitet[4] und wird Ende Juli 2019 im Rahmen der 16th International Conference on Security and Cryptrography präsentiert [18].

Nachfolgend wird das Sicherheitsmodell von Android beschrieben, um einen Überblick über die Umgebung, in der Android-Applikationen ausgeführt werden, zu geben. Anschließend wird herausgearbeitet, wie die Integrität von Android-Geräten im Sinne einer Trusted Computing Base verifiziert werden kann.

4.2.1. Android-Systemsicherheitsmodell

Android basiert auf Linux und baut auf dessen Mehrbenutzerkonzept auf, um Applikationen voneinander zu isolieren, sowie Zugriff direkten auf Systemressourcen zu unterbinden. Konkret wird – im Gegensatz zum Desktopbereich – jeder Applikation eine eigene Benutzer-ID, –gruppe, sowie ein eigenes Verzeichnis im Dateisystem zugewiesen [19]. Die Zugriffsrechte auf dieses Verzeichnis sind auf das Minimum reduziert, sodass lediglich die entsprechende Applikation Daten lesen und schreiben kann. Der Benutzer, bzw. die Benutzerin eines Geräts hat keine besonderen Zugriffsrechte, wodurch direkte Zugriffe auf System- und Applikationsdaten unterbunden werden. Mit Android 4.3 wurde zusätzlich *Mandatory Access Control* (MAC) mittels *SELinux* eingeführt [20], um diese Isolationsmechanismen zu verstärken. Da diese Sicherheitsfeatures jedoch lediglich softwarebasiert umgesetzt sind, können diese, sofern keine zusätzlichen, hardwaregestützten Maßnahmen vorhanden sind, (per Definition) ohne Hardwareeingriffe umgangen werden [21]. Folglich können sensible Applikationsdaten sowohl vom Angreifer, als auch von Gerätenutzern und Gerätenutzen extrahiert werden.
Aktuelle Android-Geräte verfügen typischerweise über dedizierte Hardwaremodule, welche einerseits für kryptografische Operationen zuständig sind, insbesondere jedoch im Rahmen eines *Android Verified Boot* (AVB) genannten verifizierten Bootprozesses zum Einsatz kommen. Dadurch

[4] Das im Rahmen dieses Dokuments erörterte Verfahren wurde bisher nicht als praktisch umsetzbar beschrieben und stellt insbesondere eine Erweiterung des *Android Platform Security Model* [3] dar.

wird sichergestellt, dass lediglich vom Hersteller signierte Betriebssystemabbilder gestartet werden können. Unter anderem bietet AVB auch eine als *Rollback Protection* titulierte Funktionalität, welche lediglich das Aktualisieren des Betriebssystems zulässt, nicht jedoch das Einspielen älterer Betriebssystemversionen [22]. Dadurch soll verhindert werden, dass nach einem Systemupdate gezielt ältere Betriebssystemabbilder installiert werden, welche potentiell bekannte Schwachstellen enthalten.

Dieses Sicherheitsmodell hat jedoch nur Gültigkeit, solange der Bootloader eines Geräts nicht entsperrt wurde. Da dies jedoch bei manchen Geräten möglich ist, ergibt sich diesbezüglich ein heterogenes Umfeld für die Distribution von Applikationen. Aussagekräftige Remote-Attestation-Verfahren sind daher in jedem Fall notwendig, um aus der Ferne feststellen zu können, ob ein Gerät auch tatsächlich nicht modifiziert wurde und das Android-Sicherheitsmodell als intakt angenommen werden kann. Im Hinblick auf die Sicherheit von kryptografischem Material hat dieser Umstand ebenfalls Einfluss, wenn auch nur bedingt. Nachfolgend wird daher die Funktionalität des Android-Keystore vorgestellt, auf Basis derer auch Remote-Attestation umgesetzt werden kann. In diesem Kontext wird nicht auf Root-Exploits eingegangen, da diese im Gegensatz zu iOS dank entsperrbarer Bootloader unter Android im Sinne kontrollierter, von Benutzer, bzw. der Benutzerin durchgeführter Systemmodifikationen zunehmend an Relevanz verlieren.

4.2.2. Funktionalität des Android-Keystore

Der Android-Keystore bietet die Möglichkeit, kryptografisches Schlüsselmaterial zu erstellen und zu verwalten. Auch in diesem Zusammenhang sorgt die Heterogenität im Androidgeräteumfeld für zusätzliche Komplexität. Sofern kein entsprechendes Hardwaremodul im Gerät vorhanden ist, erfolgt die Schlüsselverwaltung rein in Software. In diesem Fall können Schlüssel extrahiert werden, sobald Zugriff auf die entsprechenden Teile des Dateisystems erlangt wird. Nahezu alle aktuellen Androidgeräte (all jene, welche über biometrische Sensoren verfügen [4], sowie darüber hinaus z.B. das *Nokia 1*[5]) verfügen jedoch über ein Modul als Teil der CPU, das ein TEE zur Verfügung stellt und für die Erstellung und Verwaltung kryptografischer Schlüssel, sowie für die Ausführung bestimmter kryptografischer Operationen zuständig ist. Einige wenige Geräte (z.B. die Geräteserien Google Pixel 2 und Google Pixel 3) sind darüber hinaus mit einem diskreten *Hardware Security Module* (HSM) ausgestattet, wodurch eine zusätzliche Sicherheitsbarriere und weiter erhöhter Schutz des Schlüsselmaterials erreicht werden soll. Tatsächlich müssen diese von Google als *StrongBox* bezeichneten Module nach dem Schutzprofil *BSI-CC-PP-0084-2014* zertifiziert werden [6]. Die Einführung dieser weiteren Klasse an hardwaregestützter Schlüsselverwaltung wird unter anderem im Hinblick auf Attacken wie *Spectre* begründet [23]. Eine Übersicht über die unterschiedlichen Keystore-Typen ist in Abbildung 1 dargestellt.

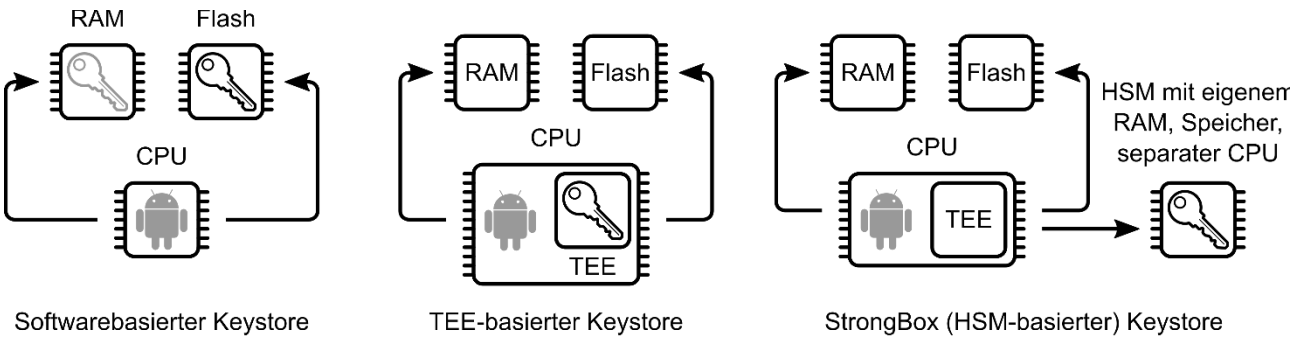

Abbildung 1: Unterschiedliche Keystore-Typen unter Android, inklusive Speicherort von Schlüsselmaterial

In jedem Fall bieten hardwarebasierte Keystores im Gegensatz zu rein softwarebasieren Lösungen erhöhten Schutz vor Schlüsselextraktion, da einmal von der Hardware generierte Schlüssel nicht exportiert werden können und damit an das Gerät, das sie erstellt hat, gebunden sind. Dadurch steigert sich insbesondere die Effektivität von Dateisystemverschlüsselung, da eine Entschlüsselung

[5] https://www.nokia.com/phones/en_int/nokia-1

nur am Gerät möglich ist. Auch kryptografische Bindungen an einzelne Geräte lassen sich dadurch realisieren. Die Nutzung von Schlüsselmaterial, bzw. der Zugriff auf die entsprechenden Systemschnittstellen wird jedoch vom Betriebssystem geregelt. Das Erreichen der von Google propagierten Sicherheitsniveaus setzt folglich ein integres Betriebssystem voraus. Vorgesehen ist, dass lediglich die Applikation, welche Schlüsselmaterial erstellt hat, dieses auch benutzen darf. Zugriffskontrolle wird auch nutzerseitig unterstützt: Beispielsweise kann die Nutzung von Schlüsselmaterial an die Eingabe eines Passworts, oder das Auslösen des Fingerabdrucksensors (sofern vorhanden) gekoppelt sein. Dadurch soll sichergestellt werden, dass sensible Operationen nicht von Angreifern oder Angreiferinnen, sondern lediglich vom Besitzer, bzw. der Besitzerin des Geräts ausgeführt werden können, sofern dies seitens der betreffenden Applikation gewünscht ist. Ausgehend von der Rollback Protection für Betriebssystemabbilder, ist eine ähnliche Funktionalität auch für kryptografische Schlüssel verfügbar: Wird ein Schlüssel entsprechend generiert, kann auf diesen nach dem Aufspielen einer älteren Betriebssystemversion nicht mehr zugegriffen werden.

Ein letzter fehlender Aspekt bezüglich der Betriebsumgebung insbesondere von Backend-gebundenen Android-Applikationen betrifft Maßnahmen, welche direkt auf die Sicherheit und Integrität von Applikationen abzielen. Nähere diesbezügliche Details und bestehende (softwarebasierte) Konzepte um Applikationsintegrität zu gewährleisten werden nachfolgend beschrieben.

4.2.3. Überprüfung und Kompromittierung von Applikationsintegrität unter Android

Typischerweise zielen Maßnahmen gegen Modifikation von Applikationen auf die Detektion von Fehlverhalten ab, das dem Nutzer, bzw. der Nutzerin schaden soll. Im Gegensatz zu iOS handelt es sich hierbei um ein verbreitetes Problem, da Applikationen z.B. von Geräten mit modifiziertem Betriebssystem und Vollzugriff („gerooteten“ Geräten) extrahiert, modifiziert und erneut zum Download angeboten werden können. Zwar sind Android-Applikationen prinzipiell vom Distributor signiert, allerdings verhindert dies lediglich das Modifizieren einer Applikation im Rahmen von Updates, nicht jedoch das initiale installieren von modifizierter Software. Durch den Umstand, dass Applikationen auch abseits des *Play Store*, (dem offiziellen, von Google unterstützten Distributionskanal für Hardware, Software und andere Multimediale Inhalte) wird die Verbreitung derartig kompromittierter Applikationen erheblich erleichtert – insbesondere, da es alternative Marktplätze – z.B. betrieben von Amazon– gibt, welche Nutzerinnen und Nutzer dazu auffordern, die Installation von Software aus alternativen Quellen zu erlauben.
Eine ganze Reihe von Gegenmaßnahmen wurde bisher vorgeschlagen, um dieses Problem einzudämmen, wobei einige im Rahmen von Googles *Play Protect* eingesetzt werden [24]. Dabei handelt es sich um ein Framework (dessen genaue Funktionsweise nicht öffentlich dokumentiert ist), das unter anderem Applikationen im Rahmen ihrer Installation, sowie teilweise auch zur Laufzeit auf bösartiges Verhalten überprüft. Dadurch können zwar einige bekannte Angriffe abgewehrt werden und nutzerseitig ergibt sich ein sicherheitstechnischer Mehrwert, allerdings ist die Verwendung von Play Protect optional. Folglich kann beispielswiese ein Applikationsbackend nicht davon ausgehen, dass Play Protect eingesetzt wird, und somit nicht sicher sein, dass keine modifizierten Client-Applikationen im Einsatz sind, welche potentiell das Backend selbst als Angriffsziel haben. Unter anderem wurde 2018 gezeigt, dass mittels einer gezielt modifizierten Mobilapplikation zum Minen einer Kryptowährung möglich ist, eine nahezu unbegrenzte Menge an Kryptowährungstoken auszuschütten [25].
Derartige Lücken sollen mit Hilfe von Googles *SafetyNet* Remote-Attestation-Service geschlossen werden [26]. Auch in diesem Fall gibt es keine öffentlich verfügbare Dokumentation bezüglich der Funktionalität dieses Dienstes. Bekannt ist lediglich, dass Kontakt zu Google-Servern aufgenommen wird und im Hintergrund gesammelte Informationen (wie z.B. Modifikationen der Systempartition, das Vorhandensein von Root-Binaries, sowie Installation von Standardapplikationen) übertragen werden, um ein Attestation-Resultat zu erhalten. Im Rahmen dieses Projekts durchgeführte Versuche haben jedoch gezeigt, dass der Einsatz von *Magisk*[6] um Root-Rechte zu erhalten, nach wie vor ein unbedenkliches Attestation-Resultat produziert, obwohl Magisk einen entsperrten Bootloader voraussetzt, und weitreichende Modifikationen der Systempartition (im Rahmen einer

[6] https://topjohnwu.github.io/Magisk/

Overlay-Dateisystems) zulässt. In Kombination mit dem *Xposed*-Framework[7] lässt sich mittels Code-Injection der Ablauf aller Applikationen (inklusive Systemapplikationen) beliebig modifizieren, ohne dass dies im Rahmen einer SafetyNet-Attestation als bedenklich klassifiziert wird.

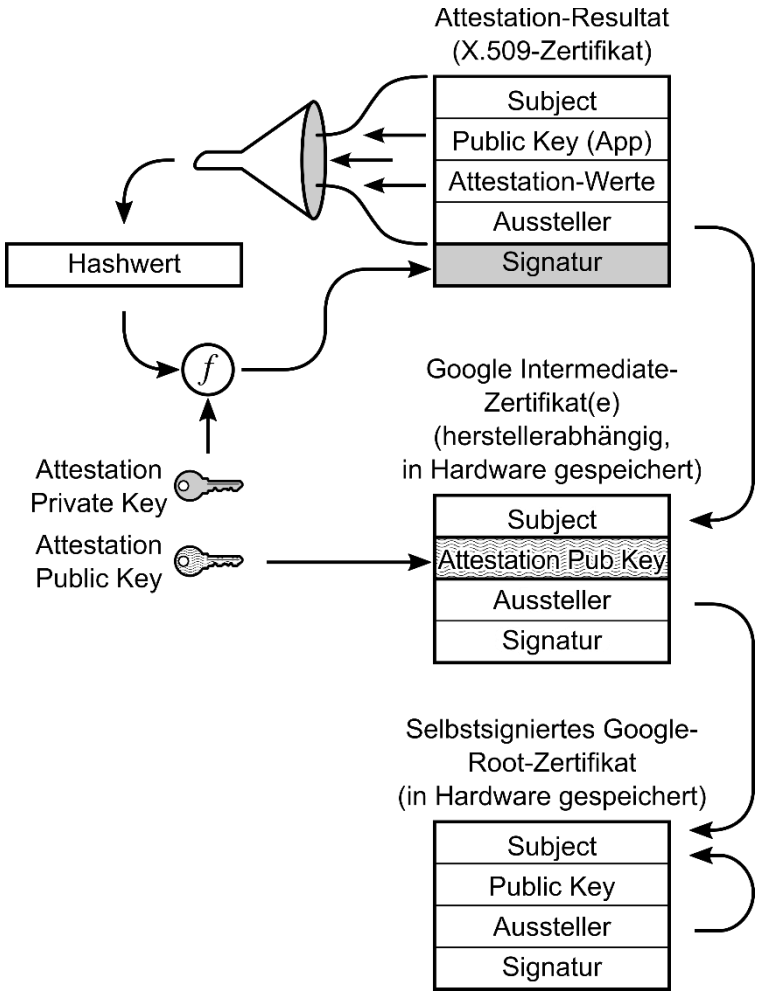

Abbildung 2: Zertifikatskette im Kontext von Schlüsselattestierung

Nachfolgend wird ein alternativer, im Rahmen dieses Projekts erarbeiteter, Remote-Attestation-Ansatz für Android vorgestellt, der lediglich auf Keystore-Funktionalität basiert. Zwar ist dieser nicht auf allen aktuell im Einsatz befindlichen Android-Geräten einsetzbar, allerdings werden keine Heuristiken, sondern ein simples Vertrauensmodell, sowie kryptografische Verfahren eingesetzt.

4.2.4. Remote-Attestation auf Basis des hardwaregstützten Keystore

Aktuelle Android-Versionen unterstutzen die Attestierung von Eigenschaften kryptografischer Schlüssel. Das entsprechende Attestation-Resultat wird innerhalb des kryptografischen Hardwaremoduls (wo auch das zu attestierende Schlüsselmaterial gespeichert ist) erstellt und signiert [4]. Ausgehend vom dem Signaturschlüssel („Attestation-Key“) zugehörigen Zertifikat („Attestation-Zertifikat“) lässt sich eine Zertifikatskette bis zu einem Google-Root-Zertifikat aufbauen, und so die Authentizität von Attestation-Resultaten prüfen. Dieser Schlüssel wird im Zuge des Produktionsprozesses in die Hardware eingebracht und kann nicht extrahiert werden.
Das Attestation-Resultat selbst ist ebenfalls in Form eines X.509-Zertifikats, bzw. Zertifikats-erweiterungen umgesetzt, und beinhaltet alle Informationen, die notwendig sind, um den attestierten Schlüssel zu identifizieren und zu charakterisieren [27]. Folglich kann im Rahmen einer Zertifikats-validierung überprüft werden, ob ein von einer Android-Applikation erstellter kryptografischer Schlüssel auch tatsächlich auf einem echten Gerät (und nicht mittels Emulator) innerhalb

[7] https://repo.xposed.info/

kryptografischer Hardware generiert wurde. Die hierzu notwendige Vertrauenskette und die Zusammenhänge zum zu attestierenden Schlüsselmaterial sind in Abbildung 2 dargestellt.

Die attestierten Characterstika umfassen dabei mehr als nur die Eigenschaften des betreffenden Schlüssels. Insbesondere werden auch Keystore-, Betriebssystem-, Bootloader- und Applikationsinformationen als Teil des resultierenden Zertifikats aufgenommen. Im Detail sind dies folgende Daten:

- **Sicherheitsstufe der Attestierung**, beschreibt im Wesentlichen die KeyStore-Umsetzung des Geräts. Nimmt einen der folgenden Werte an:
  - Software
  - TEE
  - StrongBox

- **Grundlegende kryptografische Eigenschaften des attestierten Schlüssels** (wie z.B. den öffentlichen Schlüssel, falls ein Public-Private-Key-Pair attestiert wurde, sowie Details zum kryptografischen Algorithmus)

- **Rollback-Resistance**; ob der Schlüssel unter Rollback-Protection fällt

- **Bootloader-Status**; ob der Bootloader gesperrt („locked“) ist und nur vom Hersteller signierte Betriebssystemabbilder bootet

- **Verified-Boot-Status**:
  - *Verified*; sowohl vom Hersteller signierter Bootloader, als auch Betriebssystemabbild
  - *SelfSigned*; nicht vom Hersteller signierter Bootloader im Einsatz
  - *Unverified*; vollkommen frei veränderliche Boot-Kette ohne verifizierten Bootprozess
  - *Failed*; nie im Rahmen eines Attestation-Resultats möglich

- **Betriebssystemversion** und **Patch-Level**

- **Applikationsmetadaten**:
  - Name des Applikationspakets (des APKs)
  - Version
  - Hash des Zertifikats, dessen Private-Key zum Signieren des Applikationspakets verwendet wurde

- Die in Abbildung 2 skizzierte **Zertifikatskette** zur direkten Verifikation des Attestation-Resultats

- Eine laut Spezifikation geforderte **Nonce**, die im Rahmen einer Attestation-Anfrage an den KeyStore übergeben werden muss

Abbildung 3 illustriert den Attestierungsprozess und veranschaulicht welche Teile des Attestation-Resultats von der Hardware und welche von der Software erstellt werden. In diesem Zusammenhang ist insbesondere die Auswertung der einzelnen Datenfelder kritisch: Ein Resultat mit der Sicherheitsstufe *Software* bietet keinen Mehrwert an Sicherheit, da diese Werte manipuliert werden können, sobald Vollzugriff auf das entsprechende Gerät erlangt wird. Der Attestierungsprozess selbst läuft wie folgt ab:

1. Beim Gerätestart verifiziert das kryptografische Hardwaremodul die Signatur des verwendeten Bootloaderabbilds (signiert mittels Verified-Boot-Key) und protokolliert, ob ein unmodifiziertes Abbild, bzw. ein gesperrter Bootloader eingesetzt wird.

2. Der Bootloader verifiziert die Signatur des Betriebssystemabbilds, um dessen Integrität zu überprüfen. Im Zuge dieses Prozesses wird die Betriebssystemversion und das Patch-Level an das kryptografische Hardwaremodul übermittelt.

3. Im Rahmen einer jeden Applikationsinstallation verifiziert das Betriebssystem die Signatur des zu installierenden Applikationspakets, wobei in diesem Zusammenhang mangels Konfiguration eines Trust Anchor lediglich die kryptografische Korrektheit überprüft wird. Insbesondere wird jedoch im Rahmen etwaiger Applikationsupdates sichergestellt, dass sich der Signaturschlüssel nicht ändert.

4. Die Applikation erstellt ein Public-Private-Key-Pair und stößt über die Keystore-API einen Attestierungsprozess an.

5. Das Betriebssystem übermittelt Applikationsmetadaten an die Hardware, welche diese, sowie die zuvor vom Bootloader und der Hardware selbst attestierten Werte, in ein Attestation-Resultat aufnimmt.

6. Die Hardware signiert das Attestation-Resultat und retourniert es über die Betriebssystem- bzw. Keystore-API an die Applikation.

7. Dieses Resultat kann an ein eventuelles Backend weitergegeben werden, bzw. von beliebiger Seite überprüft werden, um eine Aussage über den Integritätszustand von Betriebssystem und Applikation zu treffen.

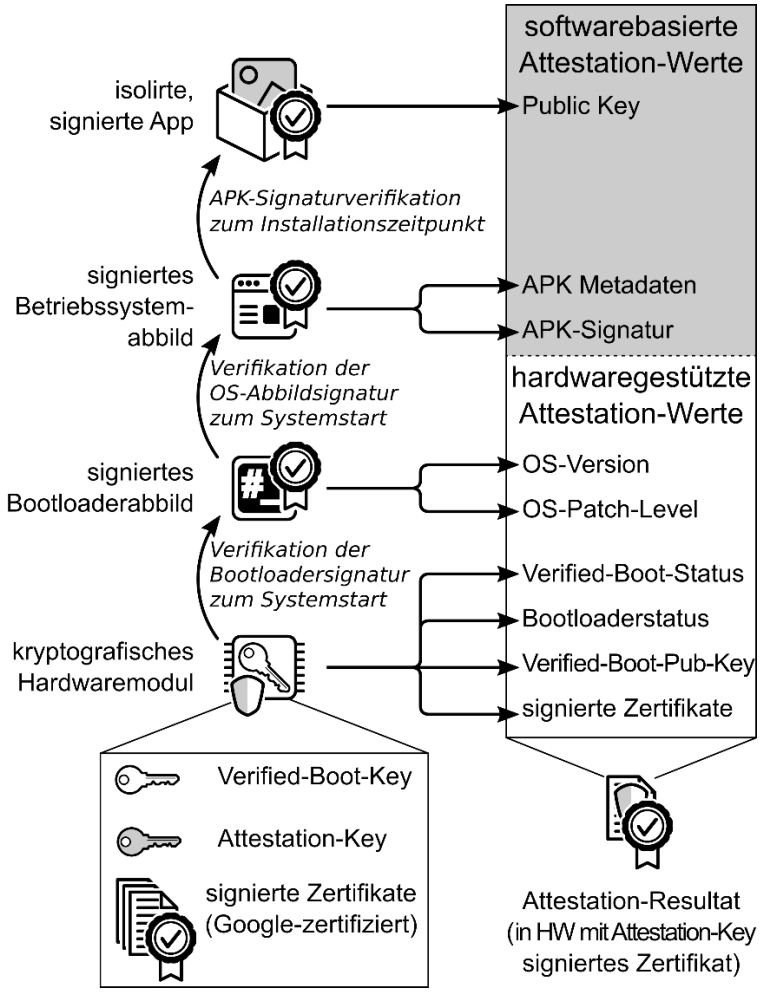

Abbildung 3: Stuktur eines Attestation-Resultats

Augenscheinlich ist ein Attestation-Resultat nur aussagekräftig, wenn die Integrität des Betriebssystems und des Bootloaders von einem Hardwaremodul (entweder TEE oder StrongBox) bescheinigt wird. Daher werden nur Geräte mit kryptografischer Hardware in Betracht gezogen. Insgesamt lassen sich aus einem solchen Resultat folgende Eigenschaften ableiten:

- **Status des verifizierten Bootprozesses**: Spiegelt ein Attestation Resultat einen gesperrten („locked“) Bootloader und einen verifizierten („Verified“) Boot-Status wider, bedeutet dies, dass lediglich vom Hersteller signierte Betriebssystemabbilder gestartet werden können. Diese müssen laut den Android-Kompatibilitätsrichtlinien zwingend konform zum Android-Sicherheitsmodell sein und dürfen keine nutzerseitigen Systemmodifikationen (insbesondere Root-Zugriff) erlauben.
- **Integrität des Gesamtsystems als Trusted Computing Base**: Da die Systemintegrität ausgehend vom verifizierten („locked“, „Verified“, s.o.) Bootprozess von dedizierter Hardware attestiert wird, kann das Betriebssystem als Trusted Computing Base angesehen werden. Folglich können auch die softwarebasierten Attestation-Werte, wie z.B. die Metadaten des Softwarepakets, als aussagekräftig angesehen werden.
- **Applikationsintegrität**: Ausgehend von einer Trusted Computing Base, kann auf Basis der im Attestation-Resultat eingebetteten Applikationsmetadaten sichergestellt werden, dass die Applikation, welche eine Attestierung angestoßen hat, auch tatsächlich die von Distributor in Umlauf gebrachte ist. Dabei ist der Installationsweg irrelevant. Diese Überprüfung geschieht wie folgt: Im Zuge der Bereitstellung eines Applikationspakets wird dieses vom Distributor signiert. Dieser Signaturwert kann anschließend mit dem im Attestation-Resultat vorhandenen verglichen werden. Stimmen die Werte überein, ist auch die Integrität der Applikation garantiert.

Wird der Bootloader entsperrt, oder die Applikation, welche das Attestation-Resultat angefordert hat, deinstalliert, ist kein Zugriff auf das zuvor attestierte Schlüsselmaterial mehr möglich, wodurch nachfolgende Modifikationen ausgeschlossen werden können. Um darüber hinaus einen sicheren Kommunikationskanal zu einer Applikation auf Basis eines Attestation-Resultats aufbauen zu können, muss das attestierte Schlüsselmaterial direkt z.B. in ein Diffie-Hellman-Verfahren einfließen. Diese Form von Remote-Attestation wurde bisher in der Literatur nicht als praktisch gangbarer Weg diskutiert. Tatsächlich kann der beschriebene Prozess nur bedingt eingesetzt werden, da ein aktuelles Android-Gerät mit kryptografischer Hardware benötigt wird. Dieses Problem verliert jedoch mit zunehmender Verbreitung immer aktuellerer Geräte an Bedeutung. Ein weiterer limitierender Faktor sind jedoch Softwarefehler, welche auch ohne das Entsperren des Bootloaders (was im Attestation-Resultat aufscheinen würde) Vollzugriff und somit unbemerkte System- und Applikationsmodifikationen zulassen würden. Je nach Situation kann dem insofern entgegengewirkt werden, als dass z.B. nur attestiert aktuelle Betriebssystemversionen als vertrauenswürdig angesehen werden. Nachdem die das Betriebssystem betreffenden Werte hardwarebasiert überprüft werden, wirken sich eventuelle unerlaubte Systemmodifikation nicht auf diesen Teil des Attestation-Resultats aus. Je nach angestrebtem Sicherheitsniveau kann durch die Wahl eines Schwellwerts bezüglich nicht vollständig aktueller Betriebssystemversionen ein Mittelweg zwischen Kompatibilität und Vertrauenswürdigkeit gewählt werden.

5. Fazit

Die Umsetzung von Trusted-Computing-Konzepten und zugehöriger Remote-Attestation-Verfahren variiert stark zwischen Mobil- und Desktopbereich. Während im Desktopumfeld mit Hilfe von Intels Software Guard Extensions prinzipiell beliebige Funktionalität isoliert vom Betriebssystem in einer sicheren Enklave ausgeführt werden kann, ziehen Mobilbetriebssysteme wie iOS und Android dedizierte kryptografische Hardware lediglich für ausgewählte Prozeduren heran. Diese Limitierung stellt jedoch im Fall von Android keine Einschränkung dar, da die Integrität des Gesamtsystems in Sinne einer Trusted Computing Base dank ausreichender Remote-Attestation-Verfahren überprüft werden kann. Tatsächlich bietet diese auf Integration basierende Herangehensweise im Gegensatz zu dem auf zusätzliche Isolation zwischen Betriebssystem und abgesicherter Betriebsumgebung für kritische Operationen einen entscheidenden Kompatibilitätsvorteil: Im Gegensatz zum Desktopbereich ist es dadurch unter Android möglich, bestehenden Applikationscode im Prinzip unverändert weiter zu nutzen, da dieser lediglich um Aufrufe der Keystore-API für die Erzeugung eines Attestation-Resultats ergänzt werden muss. SGX-Code ist hingegen CPU-Serien-spezifisch und kann nur unter Verwendung des von Intel bereitgestellten SDK produziert werden. Darüber hinaus ist im Speziellen auch der Attestierungsprozess selbst im Rahmen von SGX durch den Einsatz eines kryptografisch aufwändigen Protokolls unter Einbezug eines Dienstes, welcher von

Intel betrieben wird, nicht nur komplex, sondern auch von Intel-Infrastruktur abhängig und somit nicht dezentral (z.B. im Peer-to-Peer-Kontext) umsetzbar. Eine Bewertung über die tatsächlich erreichbaren Sicherheitsniveaus beider Ansätze bedarf einer gesonderten Analyse und wurde im Rahmen dieses Projekts nicht durchgeführt. Dennoch zeigt sich, dass Trusted Computing auch im Mobilbereich für erhöhte Sicherheitsgarantien sorgen kann.

Referenzen

[1] Department of Defense, „Trusted Computer System Evaluation Criteria“. USA Patent DoD 5200.28-STD, 12 1985.

[2] AOSP, „Trusty TEE,“ [Online]. Available: https://source.android.com/security/trusty/. [Zugriff am 11 06 2019].

[3] R. Mayrhofer, J. Vander Stoep, C. Brubaker und N. Kralevich, „The Android Platform Security Model,“ 11 04 2019. [Online]. Available: http://arxiv.org/abs/1904.05572. [Zugriff am 11 06 2019].

[4] AOSP, „Android 7.0 Compatibility Definition,“ 18 04 2017. [Online]. Available: https://source.android.com/compatibility/7.0/android-7.0-cdd.pdf. [Zugriff am 11 06 2019].

[5] AOSP, „Android 8.0 Compatibility Definition,“ 01 05 2018. [Online]. Available: https://source.android.com/compatibility/8.0/android-8.0-cdd.pdf. [Zugriff am 11 06 2019].

[6] AOSP, „Android 9.0 Compatibility Definition,“ 08 02 2019. [Online]. Available: https://source.android.com/compatibility/9/android-9-cdd.pdf. [Zugriff am 11 06 2019].

[7] B. Lampson, M. Abadi, M. Burrows und E. Wobber, „Authentication in Distributed Systems: Theory and Practice,“ *ACM Transactions on Computer Systems,* Bd. 10, Nr. 4, pp. 265-310, 1992.

[8] Microsoft, „Secure Boot,“ 05 10 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot. [Zugriff am 11 06 2019].

[9] Apple, „iOS Security,“ 05 2015. [Online]. Available: https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf. [Zugriff am 11 06 2019].

[10] fail0verflow, „Console Hacking 2010 - PS3 Epic Fail,“ 29 12 2010. [Online]. Available: https://media.ccc.de/v/27c3-4087-en-console_hacking_2010#t=384. [Zugriff am 11 06 2019].

[11] M. Hohmuth, M. Peter, H. Härtig und J. S. Shapiro, „Reducing TCB size by using untrusted components: small kernels versus virtual-machine monitors,“ in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, Leuven, Belgien, ACM, 2004.

[12] Intel, „Enhanced Security for Applications and Data In-use,“ 2019. [Online]. Available: https://software.intel.com/sites/default/files/managed/6a/85/intel-sgx-product-brief-2019.pdf. [Zugriff am 11 06 2019].

[13] Intel, „Intel® Software Guard Extensions Commercial Licensing FAQ,“ 04 10 2019. [Online]. Available: https://software.intel.com/en-us/articles/intel-software-guard-extensions-product-licensing-faq. [Zugriff am 11 06 2019].

[14] M. Hoekstra, R. Lal, P. Pappachan, C. Rozas, V. Phegade und J. d. Cuvillo, „Using Innovative Instructions to Create Trustworthy Software Solutions,“ 14 08 2013. [Online]. Available: https://software.intel.com/en-us/articles/using-innovative-instructions-to-create-trustworthy-software-solutions. [Zugriff am 11 06 2011].

[15] Kudelski Group, „SGX Secure Enclaves in Practice: Security and Crypto Review,“ 2016. [Online]. Available: https://www.kudelskisecurity.com/sites/default/files/files/publication_SGXSecureEnclavesInPractice_EN.pdf. [Zugriff am 11 06 2019].

[16] J. M. „Code Sample: Intel® Software Guard Extensions Remote Attestation End-to-End Example,“ 04 07 2018. [Online]. Available: https://software.intel.com/en-us/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example. [Zugriff am 11 06 2019].

[17] Apple, „Unauthorized modification of iOS can cause security vulnerabilities, instability, shortened battery life, and other issues,“ 15 06 2019. [Online]. Available: https://support.apple.com/en-gb/HT201954. [Zugriff am 11 06 2019].

[18] B. Prünster, G. Palfinger und C. Kollmann, „Fides - Unleashing the Full Potential of Remote Attestation,“ in *Proceedings of the 16th International Security and Cryptography (SECRYPT)*, Prag, Tschechien, SciTePress - Science and Technology Publications,, 2019.

[19] AOSP, „Security,“ [Online]. Available: https://source.android.com/security/. [Zugriff am 11 06 2019].

[20] AOSP, „Security-Enhanced Linux in Android,“ [Online]. Available: https://source.android.com/security/selinux/. [Zugriff am 11 06 2019].

[21] M. Sabt und J. Traoré, „Breaking into the keystore: A practical forgery attack against Android keystore,“ in *European Symposium on Research in Computer Security*, Springer, 2016, pp. 531-548.

[22] AOSP, „Android Verified Boot 2.0,“ [Online]. Available: https://android.googlesource.com/platform/external/avb/. [Zugriff am 11 06 2019].

[23] AOSP, „Android keystore system,“ 23 01 2019. [Online]. Available: https://developer.android.com/training/articles/keystore. [Zugriff am 11 06 2019].

[24] AOSP, „Android - Google Play Protect,“ [Online]. Available: https://www.android.com/play-protect/. [Zugriff am 11 06 2019].

[25] D. Ziegler, B. Prünster, A. Marsalek und C. Kollmann, „Spoof-of-Work - Evaluating Device Authorisation in Mobile Mining Processes,“ in *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications*, Portugal, SciTePress - Science and Technology Publications, 2017, pp. 380-387.

[26] J. Kozyrakis, „SafetyNet: Google's tamper detection for Android,“ 17 09 2015. [Online]. Available: https://koz.io/inside-safetynet/. [Zugriff am 11 06 2019].

[27] AOSP, „Verifying hardware-backed key pairs with Key Attestation,“ [Online]. Available: https://developer.android.com/training/articles/security-key-attestation. [Zugriff am 11 06 2019].