HSB
Hochschule Bremen
City University of Applied Sciences

Trusted Monitoring mit Hilfe eines TPMs durch das Trusted Attestation Protocol

Masterarbeit
14. Juli 2020

Name:	Martin Müller
Matrikelnummer:	358740
Semester:	Sommersemester 2020
Studiengang:	Komplexe Softwaresysteme
E-Mail:	martmueller@stud.hs-bremen.de
	mueller@fireorbit.de

Fakultät Elektrotechnik und Informatik - Hochschule Bremen

1. Gutachter:	Prof. Dr. rer. nat. Richard Sethmann
	richard.sethmann@hs-bremen.de
	+49 421 5905 5483
2. Gutachter:	Prof. Dr.-Ing. Jasminka Matevska
	jasminka.matevska@hs-bremen.de
	+49 421 5905 5425

Abstract

Das Trusted Platform Module (TPM) ist ein von Anwendungen unabhängiger Chip, der weitverbreitet ist und Computer bei kryptografischen Aufgaben auf sichere Weise unterstützen soll. Insbesondere in großen Unternehmen kann das TPM mit dessen spezifischen Trusted Attestation Protocol (TAP) der Trusted Computing Group (TCG) eine Lösung für ein Trusted Monitoring von einer Vielzahl an unternehmenseigenen Computern darstellen. In dieser Arbeit wird aufgezeigt, dass mit Hilfe dieses TAP ein Trusted Monitoring aufgebaut werden kann und dass für dessen Einsatz ein verbindungsorientiertes Netzwerkprotokoll benötigt wird sowie darüber hinaus ein vertrauenswürdiger Zufallszahlengenerator für einen Verifier und ein persistenter Speicher beim Verifier vorhanden sein muss, um die Daten vom Attester mit einem integrierten TPM zu verifizieren. Um dies zu beweisen, wurde der Anwendungsfall des "Attester Remeasurement" vom TAP verwendet sowie der Attester als Service und der Verifier als Prometheus-Exporter im Monitoringsystem implementiert. In der Demonstration des Trusted Monitoring wurden die vom TPM aufgezeichneten Bootvorgänge als Grundlage verwendet. Abschließend werden in dieser Arbeit verschiedene Optimierungsvorschläge und eine Interpretation der Spezifikation für die implementierte TAP-Softwarebibliothek in Rust kritisch diskutiert.

Inhaltsverzeichnis

Abbildungsverzeichnis III

Listingsverzeichnis V

Abkürzungsverzeichnis VII

Glossar IX

1 Einleitung 1
1.1 Motivation . . . 2
1.2 Ziele der Masterarbeit . . . 3
1.3 Aufbau der Masterarbeit . . . 5

2 Grundlagen 7
2.1 TPM Funktionsübersicht . . . 8
2.1.1 Kryptoprozessor . . . 10
2.1.2 Speicher im TPM . . . 11
2.1.3 Einsatzmöglichkeiten . . . 16
2.2 TPM Software Stack . . . 18
2.2.1 Application Programming Interfaces . . . 20
2.2.2 Implementierungen vom TPM Software Stack . . . 22
2.3 TAP Funktionsüberblick . . . 24
2.3.1 Freshness / Nonce . . . 24
2.3.2 Anwendungsfälle . . . 26
2.4 Monitoring . . . 28
2.4.1 grundsätzliche Einsatzmöglichkeiten . . . 28
2.4.2 Komponenten eines Monitoringsystems . . . 30

3 Konzeption der TAP Implementierung 37
3.1 Übermittlung des Protokolls . . . 38
3.2 Abläufe des Protokolles . . . 40

4 Implementierung 45
4.1 Bibliothek . . . 46
4.2 Attester (Service) . . . 47
4.3 Verifier . . . 48

5 Prototypische Realisierung 53
5.1 Aufbau . . . 53
5.2 Auswertung . . . 59

6 Fazit und Ausblick 61

Literaturverzeichnis i

Normenverzeichnis iii

Anhang vii

Abbildungsverzeichnis

2.1 Darstellung der Struktur bzw. der Bestandteile eines TPM 2.0 (vgl. [TCG-Arch, S. 33]) . . . 9
2.2 Intel TXT boot timeline (Figure 22-1 aus [ACG15, S. 336]) . . . 17
2.3 Using TCTI to Connect to Various Target TPMs (Figure 2 aus [TCG-TCTI, S. 10]) . . . 18
2.4 Sequenzdiagramm für Nonce vom Verifier (gekürzte Figure 1 aus [TCG-TAP1, S. 9]) . . . 25
2.5 Sequenzdiagramm für Nonce vom 3rd Party (Figure 2 aus [TCG-TAP1, S. 10]) 25

3.1 Detailliertes Sequenzdiagramm für die Einsatzmöglichkeit “Attester Remeasurement” des TAP . . . 41

4.1 Detailliertes Sequenzdiagramm des Monitoringsystem . . . 49

5.1 Die umgesetze Struktur des Monitoringsystems für Computer mit einem integrieren TPM 2.0 . . . 53
5.2 Ergebnis des Monitoringsystem durch Prometheus visualisiert (Änderung des Wertes einer PCR-Bank) . . . 59
5.3 Alert des Monitoringsystem im Alertmanager, durch Änderung eines Wertes einer PCR-Bank . . . 59

Listingsverzeichnis

4.1 Exemplarischer Auszug der TAP-Bibliothek (*src/tap/mod.rs*) 46

5.1 Ausgabe des TAP-Verifier Exporters (gekürzt, nur für die PCR-Bank Nr. 04) 54
5.2 Konfiguration von Prometheus für Alerts (gekürzt für nur eine Metrik, für die ersten acht PCR-Bänke) 58

1 Hilfeausgabe des TAP-Attester-Service vii
2 Datei mit Umgebungsvariablen/Konfiguration für den TAP-Attester-Service vii
3 Systemd-Service-Datei für das Starten des TAP-Attester-Service viii
4 Systemd-Socket-Datei für "socket-based activation" für den TAP-Attester-Service viii
5 Hilfeausgabe des TAP-Verifier-Exporters ix
6 Datei mit Umgebungsvariablen/Konfiguration für den TAP-Verifier-Exporter x
7 Systemd-Service-Datei für das Starten des TAP-Verifier-Exporters x
8 Exemplarische Konfiguration von Prometheus (mit statischer Liste von "Targets" statt "Service Discovery") xi

Abkürzungsverzeichnis

API	Application Programming Interface
EAP	Extensible Authentication Protocol
ECC	Elliptische-Kurven-Kryptografie
ESAPI	Enhanced System API
FAPI	Feature API
HMAC	Hash Message Authentication Code
HSM	Hardware Security Module
IP	Internet Protocol
ISO	International Organization for Standardization
KDF	Key Derivation Function
PCR	Platform Configuration Registers
RM	Resource Manager
RSA	RSA-Kryptosystem
SAPI	System API
TAB	TPM Access Broker
TAP	Trusted Attestation Protocol
TCG	Trusted Computing Group
TCP	Transmission Control Protocol
TCTI	TPM Command Transmission Interface
TLS	Transport Layer Security
TPM	Trusted Platform Module
TSS	TPM Software Stack
UDP	User Datagram Protocol

Glossar

Extensible Authentication Protocol Extensible Authentication Protocol (EAP) ist ein Framework, um Authentifizierungen auf dem Data-Layer abzuhandeln, ohne das IP benötigt wird (siehe [RFC3748]).

Elliptische-Kurven-Kryptografie Elliptische-Kurven-Kryptografie (ECC) ist ein asymmetrisches kryptografisches Verfahren, das mit erheblich kürzeren Schlüsseln als RSA auskommt, allerdings geht die Sicherheit verloren, wenn die zugrunde liegende Kurve nicht sorgfältig ausgewählt wurde. Diese Tatsache ist der Grund, warum die standardisierte Kurve von der NSA wenig vertraut wird und ECC generell nur langsam Einzug in den Anwendungen findet.

Enhanced System API Enhanced System API (ESAPI) wird in 2.2.1 im Detail erklärt.

Feature API Feature API (FAPI), wird in 2.2.1 im Detail erklärt.

Hash Message Authentication Code Hash Message Authentication Code (HMAC) ist eine auf Hash basierende Methode, um den Ursprung von Daten und die Integrität überprüfbar zu machen.

Hardware Security Module Ein Hardware Security Module (HSM) ist eine eigenständige Hardware-Komponente zum Absichern von kryptografischen Verfahren, wie beispielsweise die Smartcard, eToken oder TPM.

Internet Protocol Internet Protocol (IP) ist ein Netzwerkprotokoll, das unter anderem die Grundlage des Internets darstellt (siehe [RFC8200]).

Key Derivation Function Key Derivation Function (KDF) auf Deutsch Schlüsselableitung, ist eine Funktion, mit deren Hilfe man aus einem bestehenden Schlüssel einen neuen ableitet, ohne das ein Rückschluss auf dem ursprünglichen Schlüssel möglich ist. Dabei wird neben den bestehenden geheimen Schlüssel ein Parameter mit angegeben, welcher öffentlich sein darf. Die Idee dahinter ist, dass man die Anzahl der sicher zu speichernden Schlüssel verringert. Da die Schlüssel einfach reproduzierbar sind, gewinnt man mehr Sicherheit, indem für jeden Einsatzzweck andere abgeleitete

Schlüssel verwendet werden.

Resource Manager Resource Manager (RM), wird in 2.2 im Detail erklärt.

RSA-Kryptosystem RSA ist das erste asymmetrische kryptografische Verfahren, welches von Rivest, Shamir und Adleman eigentlich zum Widerlegen der Theorie solcher Public-Key-Kryptografie entwickelt wurde.

System API System API (SAPI), wird in 2.2.1 im Detail erklärt.

TPM Access Broker TPM Access Broker (TAB), wird in 2.2 im Detail erklärt.

Trusted Attestation Protocol Trusted Attestation Protocol (TAP), dient zum beweiskräftigen Sicherstellen des Zustandes und der Integrität des Attesters (z.B. mittels eines TPM) durch einen entfernten Verifier (siehe 2.3 und [TCG-TAP1, S.7]).

Trusted Computing Group Trusted Computing Group (TCG) ist eine von der Industrie betriebene Standardisierungs-Organisation, die einen offenen Standard für Trusted-Computing-Plattformen entwickelt.

Transmission Control Protocol Transmission Control Protocol (TCP) ist ein zuverlässiges, verbindungsorientiertes, paketvermitteltes Netzwerkprotokoll, welches als Grundlage Internet Protocol (IP) nutzt (siehe [RFC7323]).

TPM Command Transmission Interface TPM Command Transmission Interface (TCTI), wird in 2.2 im Detail erklärt.

Transport Layer Security Transport Layer Security (TLS) ist durch den Vorgänger Secure Sockets Layer (SSL) bekannt. TLS wurde für sichere Übertragungen auf Basis von öffentlichen asymetrischen Schlüsseln in Zertifikaten entwickelt. Mit den dadurch entstandenen Zertifikat-Ketten und den Zertifizierungsstellen ist es unter anderem möglich, mit anderen Teilnehmern im Internet ein indirektes Vertrauensverhältnis aufzubauen. (Für Details siehe [RFC8446].)

Trusted Platform Module Trusted Platform Module (TPM) ist ein HSM, dessen Funktionalität von der TCG international standartisiert wird.

TPM Software Stack TPM Software Stack (TSS), wird in 2.2 im Detail erklärt.

User Datagram Protocol User Datagram Protocol (UDP) ist ein minimales, verbindungsloses Netzwerkprotokoll auch für mehr als zwei Teilnehmer, welches als Grundlage Internet Protocol (IP) nutzt (siehe [RFC768]).

Hash "Mittels kryptografisch sicherer Hashfunktionen werden eindeutige, so genannte digitale Fingerabdrücke von Datenobjekten berechnet und zusammen mit dem Objekt versandt bzw. gespeichert, so dass der Empfänger bzw. Objektnutzer anhand dieses Fingerabdrucks die Integrität des Objekts prüfen kann. Das heißt, dass damit unautorisierte Modifikationen aufdeckbar sind." [Eck18, S.365]

Nonce Ein Nonce ist eine Zufallszahl, die in der Kryptografie dafür verwendet wird, um Replay-Attack vorzubeugen. Ein Nonce, wird nur einmal verwendet, um danach durch etwas Sicheres ersetzt zu werden. (Im Kontext zum TAP wird Nonce genauer in Kapitel 2.3.1 erklärt.)

Quote Das TPM bietet die Möglichkeit, Werte zu signieren, wie beispielsweise die aus seinen PCR-Bänken. Diese Signatur wird als Quote bezeichnet. Im Detail wird dies in 2.1.2 erklärt.

Replay-Attack Ein Replay Attack ist ein Angriff, bei dem zum Beispiel der Hash-Wert von Logindaten durch einen Angreifer aufgezeichnet wird. Mit diesen aufgezeichneten Daten wird vom Angreifer versucht, sich ohne Kenntnis der ursprünglichen Logindaten einzuloggen.

1 Einleitung

Das Trusted Platform Module (TPM) ist ein Hardware Security Module (HSM), dessen erste Version im Jahr 2009 von der Trusted Computing Group (TCG) entwickelt und von der International Organization for Standardization (ISO) als Standard herausgegeben wurde [ISO 11889-1]. Heutzutage ist ein solches Modul in so gut wie jedem Computer verbaut. Somit ist schon allein aufgrund der hohen Verbreitung dessen genaue Betrachtung von Interesse. Das TPM wurde damals entwickelt, um kryptografische Aufgaben zu übernehmen und Schlüssel sicher aufzubewahren. Indem dafür ein unabhängiger Chip verwendet wird, der sicher vor Kompromittierung ist, kann dieser als Vertrauensanker für den Computer verwendet werden. Die zugrunde liegende Idee des TPM ist, dass Aufgaben wie die Verschlüsselung, die Signierung bis hin zu dem Vermessen und Überprüfen des Bootvorganges des Computers manipulationssicher direkt auf diesem im Computer verbauten Chip durchgeführt werden können. Da das TPM fest verbaut ist, sind kryptografische Zuordnungen / Authentifizierungen ausschließlich auf Geräteebene möglich und nicht auf der personenbezogenen Ebene, wie es mit anderen HSM - beispielsweise Smartcards - möglich ist. Alle Schlüssel, auch die zur Authentifizierung benötigten, werden auf dem TPM generiert und so aufbewahrt, dass Sie das TPM nicht mehr verlassen können. Lediglich die entstehenden Klar- und Ciphertexte können das TPM verlassen.

In Unternehmen mit vielen Computern, bei denen der Zustand des TPM ausschließlich lokal ausgelesen werden kann, ist die Überwachung des Zustandes von allen Computern nur schwer möglich. Um diesem Problem zu begegnen, soll ein Monitoringsystem für den Zustand der TPMs aufgebaut werden. Dabei soll das in das TPM gelegte Vertrauen weiter bestehen bleiben und so erweitert werden, dass den Aussagen des Monitoringsystems auf Basis der TPMs ebenfalls vertraut werden kann.

Die Entwicklung des TPM wird von verschiedenen Organisationen und Unternehmen, die in der Trusted Computing Group (TCG) vertreten sind, vorangetrieben. Letztere hat den minimalen Funktionsumfang und die Kommunikation von solchen Chips standardisiert, damit die Herstellung der TPMs sowie die Entwicklung von Software, welche TPMs benutzen, unabhängig vonstattengehen können.

Im Herbst 2019 hat die TCG einen weiteren Standard veröffentlicht: das Trusted Attestation Protocol (TAP) [TCG-TAP2]. Dieses Protokoll soll das Vertrauen in das TPM über den jeweiligen Computer hinaus ermöglichen. Dazu wurden zunächst die möglichen Anwendungsfälle einer solchen Erweiterung für das TAP in [TCG-TAP1] spezifiziert. Beispiele für solche Anwendungsfälle ist eine Authentifizierung durch das TPM und die sichere Übertragung von Messdaten beispielsweise des Bootvorganges. Durch das TAP und damit die über ein Netzwerk erreichbaren Messungen, sollte es möglich sein, den Zustand des TPM zu überwachen.

In dieser Arbeit soll überprüft werden, ob es möglich ist das TAP für ein solches Trusted Monitoring zu verwenden.

1.1 Motivation

Ein Anwendungsfall des benannten TAP sieht das Neuvermessen des TPM vor, mit dessen Hilfe es möglich ist, eine Vielzahl an Computern zu überwachen. Mit dieser Lösung des Problems kann ein Unternehmen das Vertrauen in seine Infrastruktur verbessern, indem eine Manipulation beispielsweise des Bootvorganges auf einem System durch das Monitoringsystem erkannt wird. Durch das Erkennen von Fehlverhalten in der Infrastruktur lassen sich etwaige Angriffe frühzeitig feststellen. So können rechtzeitig Gegenmaßnahmen eingeleitet werden, bevor ein Schaden entsteht. Dies ist jedoch bisher lediglich hypothetisch möglich, da noch keine konkrete Implementierung des TAP existiert. Folglich soll diese Arbeit an genau dieser Leerstelle ansetzen. Wie konkret diese Forschungslücke gefüllt werden soll, wird im folgenden Abschnitt detailliert dargestellt.

1.2 Ziele der Masterarbeit

Mit Hilfe dieser Arbeit soll bewiesen werden, unter welchen Voraussetzungen das vertrauenswürdige Monitoring von Computern mit einem integrierten TPM durch TAP möglich ist und welche Anforderungen an die einzelnen Komponenten bestehen.

Dabei muss erwähnt werden, dass die Spezifikation des TAP sehr unkonkret ist, um Herstellern einen Spielraum für Innovationen zu geben. Daher wird zunächst ein Konzept für eine funktionierende Implementierung entwickelt. Für dieses Konzept müssen unter anderem folgende Entscheidungen getroffen werden:

1. Welches Protokoll soll als Basis zur Übertragung des TAP über das Netzwerk genutzt werden?

2. Wie und durch wen soll eine Verbindung aufgebaut werden?

3. Wie soll die Encodierung der TAP-Nachrichten stattfinden?

Zur Beweisführung, dass es mit dem TAP möglich ist, ein Trusted Monitoring aufzubauen, soll dies prototypisch realisiert werden. Bei der verwendeten Software für das Monitoringsystem handelt es sich um Prometheus, da dies derzeit eine der meistverwendeten Monitoringsoftware ist. Eine Evaluierung anderer existierender Software für Monitoringsysteme wurde nicht vorgenommen, da dies den Rahmen einer Masterarbeit überschreiten würde und es zur Bearbeitung der Kernfrage dieser Arbeit keinen Mehrwert leistet. Für eine kritische Reflexion der Ergebnisse, kann ein Bezug zu anderer Funktionsweise als von der hier verwendeten Monitoringsoftware spannend sein. Daher ist diese Reflexion auch im Ausblick 6 zu finden.

Für die konkrete Umsetzung des Aufbauens eines solchen Monitoringsystems bedarf es aber noch zweier Softwarekomponenten, die ebenfalls erst entwickelt werden müssen:

Die erste Softwarekomponente soll mit dem aktuellen TPM 2.0 kommunizieren und als Service auf jedem zu überwachenden Computer laufen. Dieser Service soll dann die Funktionen des TPM durch das TAP im Netzwerk zur Verfügung stellen.

Die zweite Komponente, der Exporter, wird in eine zentrale Monitoringumgebung eingebettet, welche regelmäßig den Zustand einzelner Computer überprüfen soll. Der Exporter nutzt das TAP, um auf die erste Komponente zuzugreifen. Er dient als Proxy zwischen dem TAP und dem vom Monitoringsystem vordefinierten Datenformat.

Für die Kommunikation zwischen den beiden Komponenten wird das TAP implementiert, wobei dies auf Basis der oben erwähnten konzeptionellen Entscheidungen geschehen soll. Bei der Entwicklung der beiden Komponenten soll der Quellcode so strukturiert werden, dass eine Software-Bibliothek für das TAP entsteht, welche von beiden Komponenten genutzt werden kann. Durch die TAP-Software-Bibliothek sollen für die Zukunft weitere Verwendungszwecke ebenfalls ermöglicht werden. Beispiele für eine Implementierung in andere Software für Monitoringsysteme könnten sein: Icinga, Check MK, InfluxDB.

1.3 Aufbau der Masterarbeit

Die hier vorliegende Arbeit ist in folgende vier Teile gegliedert.

Im ersten Teil werden die Grundlagen des TPM (siehe Kapitel 2.1 und 2.2, um dafür Software zu entwickeln), TAP (2.3) und Monitoring (2.4) erläutert, um ein Verständnis für den Themenkomplex der Arbeit und die darauf aufbauenden Kapitel zu entwickeln. Manche Begriffe aus der IT-Sicherheit und Kryptografie werden dabei allerdings als bekannt vorausgesetzt, um der eigentlichen Fragestellung der Arbeit fokussiert nachgehen zu können. Trotzdem werden in diesem Kapitel viele Fachbegriffe benötigt: Daher befindet sich gleich zu Beginn dieser Arbeit ein Abkürzungsverzeichnis (S. VII) sowie das Abbildungs- und Listingsverzeichnis und ein Glossar (S. IX) mit kurzen Beschreibungen der Fachbegriffe.

Die Fragestellungen für das Konzept der TAP-Implementierung werden im zweiten Teil (Kapitel 3) bearbeitet und die notwendigen Entscheidungen für eine Umsetzung des Trusted Monitoring getroffen. Dabei werden die Spezifikation [TCG-TAP2] und die Anwendungsfälle [TCG-TAP1] des TAP analysiert, sodass anschließend fundiert Entscheidungen für den Einsatz des Protokolls gefällt werden können. Das Ziel dieses Teils ist die Darstellung des erarbeiteten Konzeptes, wie das TAP implementiert und für ein Trusted Monitoring genutzt werden kann.

Im darauffolgenden dritten Teil (Kapitel 4) wird die Umsetzung der Implementation des Konzepts dargestellt. Hierbei wird für das TAP eine Bibliothek entwickelt und diese zur Verfügung gestellt. Diese Bibliothek wird generisch sein und kann auch für andere Anwendungsfälle eingesetzt werden, als jener, der in dieser Arbeit zugrunde gelegt wird. Ergänzend werden die Struktur der Bibliothek sowie die Entscheidungen, die zu dieser Struktur führten, im Kapitel 4.1 festgehalten. Des Weiteren werden die beiden entwickelten Softwarekomponenten TAP-Service im Kapitel 4.2 und TAP-Exporter für Prometheus in 4.3 beschrieben.

Im vierten Teil (Kapitel 5) werden alle Komponenten zusammengesetzt und das Trusted Monitoring als Ganzes System betrachtet. Bei der Integration soll abschließend begutachtet werden, ob alle Komponenten wie erwartet miteinander agieren. Darüber hinaus soll gezeigt werden, wie man durch solch ein Monitoringsystem das Vertrauen in die eigene Infrastruktur weiter festigen kann (siehe Kapitel 5.1).

Mit einem Fazit und Ausblick (Kapitel 6), in welchem die Zusammenfassung der Erkenntnisse sowie einige Ideen für weitere Optimierungen an dem Protokoll, den beteiligten Komponenten und dem Trusted Monitoring im Ganzen zu finden sind, endet diese Arbeit. Am Ende der Arbeit sind als getrennte Quellenverzeichnisse das Literaturverzeichnis (S. i) und das Normenverzeichnis (S. v) aufgeführt.

2 Grundlagen

In diesem Kapitel wird zunächst der grundlegende Sachverhalt erläutert, welcher als notwendiges Fundament dieser Arbeit genutzt wird.

Um zu zeigen, dass mit dem TAP ein Trusted Monitoring aufgebaut werden kann, ist es zunächst nötig das vom TAP (in Kapitel 2.3) durch den TPM Software Stack (TSS) (in Kapitel 2.2) verwendete Trusted Platform Module (TPM) (in Kapitel 2.1) und seine Funktionsweise zu erläutern. Des Weiteren wird die Grundidee von Monitoringsystemen und die dort verwendeten Mechanismen in Kapitel 2.4 erläutert, um dieses Wissen für die Umsetzung des TAP für ein Trusted Monitoring verwenden zu können. Auf diese Weise soll das im TPM gesetzte Vertrauen nun bis in das Monitoringsystem reichen.

In diesem Grundlagenkapitel wird es immer wieder kursive Absätze geben, wie hier. In solchen Absätzen wird die Relevanz der erläuterten Grundlage für die Arbeit dargelegt und dabei wird ein Bezug bis hin zu einem Fazit für diese spätere Arbeit gezogen.

2.1 TPM Funktionsübersicht

Ein Trusted Platform Module (TPM) 2.0 bietet mehr als ein übliches Hardware Security Module (HSM) wie zum Beispiel eine Smartcard. Es ist direkt im Computer verbaut, und meist sogar ein Bestandteil des Motherboards. Neben dem sicheren Speichern von Schlüsseln ist mittels eines Kryptoprozessors mehr möglich, als symmetrische und asymmetrische Verschlüsselung durchzuführen.

Das Ziel eines Hardware Security Module (HSM) und TPM ist es, immun gegen Angriffe durch Software zu sein, ein physicher Angriff liegt im Verantwortungsbereich des Herstellers (vgl. [ACG15, S. 2]). Ein Angriff auf die Kryptografie in der Software wäre zum Beispiel das Auslesen der privaten Schlüssel aus dem Arbeitsspeicher.[1] Da der private Schlüssel nicht extrahierbar im TPM gespeichert wird, könnte man solch einen Angriff durch die Benutzung des TPM verhindern. Ein weiterer Vorteil liegt in der Implementierung kryptografischer Algorithmen. Hier besteht keine Notwendigkeit mehr, ggf. unsichere Bibliotheken oder Eigen-Implementationen zu nutzen, denn es können die kryptografischen Funktionen des TPM genutzt werden. Da diese Algorithmen in Hardware umgesetzt wurden, können sie nicht manipuliert werden, doch eventuelle Fehler in der Implementierung der Algorithmen, auch nicht durch ein Softwareupdate korrigiert werden.[2] Es gibt verschieden Typen von TPMs, von eigenständiger Hardware, über integrierte Hardware bis hin zu virtualisierten oder sogar simulierten TPMs (mehr dazu siehe [TCG-Brief]).

Darüber hinaus gibt es eine Vielzahl von Einsatzmöglichkeiten für das TPM, eine Aufzählung wäre umfangreich, und wahrscheinlich niemals vollständig. Um die Möglichkeiten des TPM im Detail zu verstehen, ist daher eine Auseinandersetzung mit den Bestandteilen notwendig.

[1] Ein sehr bekanntes Beispiel ist der Heartbleed-Bug in der Transport Layer Security (TLS)-Bibliothek OpenSSL, wo dies zwischen 2012 bis 2014 (Version 1.0.1 bis 1.0.1f) möglich war (siehe `https://heartbleed.com/`).

[2] Der von Infineon hergestellte Zufallszahlen-Generator für Primzahlen für RSA-Schlüssel, war zum Beispiel nicht hinreichend zufällig, sodass der private Schlüssel relativ einfach aus dem öffentlichen Schlüssel berechnet werden konnte. (Dieser Bug erhielt den Namen ROCA und ist unter CVE-2017-15361 zu finden.)

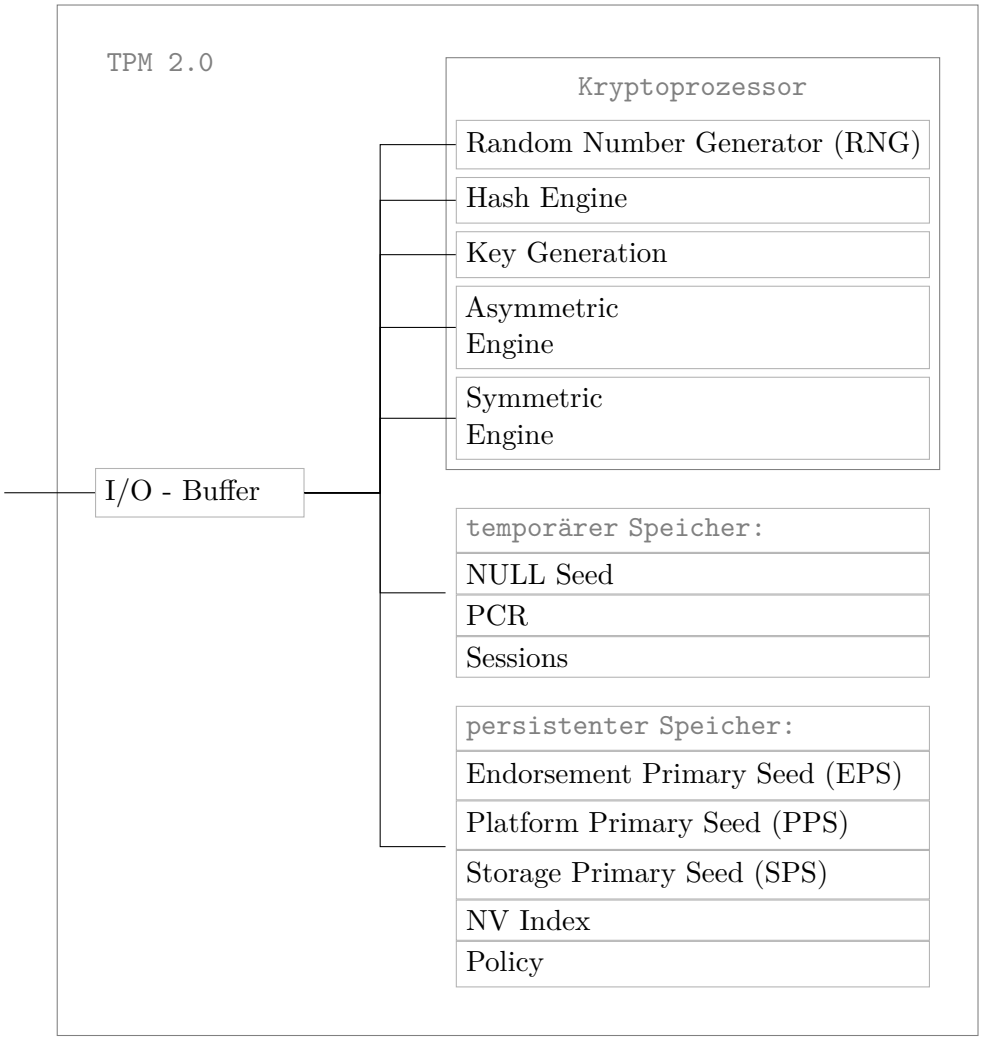

Abbildung 2.1: Darstellung der Struktur bzw. der Bestandteile eines TPM 2.0 (vgl. [TCG-Arch, S. 33])

In der Abbildung 2.1 wird eine grobe Struktur vom TPM dargestellt. Auf der linken Seite der Abbildung befindet sich zunächst ein I/O-Buffer, der für die Kommunikation mit dem Hostsystem zuständig ist. Die Aufgabe des I/O-Buffers ist die Validierung, ob die Kommunikation der Spezifikation der TCG entspricht. Ein Beispiel für eine solche Validierung ist die Überprüfung des Hash-Wertes der Daten eines Befehles. Des Weiteren gibt es einen manipulationssicheren Speicher, der in der Abbildung mittig dargestellt ist. Er setzt sich aus einem temporären und einem persistenten Speicher zusammen. Hier werden unter anderem kryptografische Schlüssel gespeichert, die nachfolgend genauer erläutert werden. Darüber hinaus gibt es einen Teil für kryptografische Operationen, den sogenannten Kryptoprozessor. Er ist in der Abbildung rechts dargestellt.

2.1.1 Kryptoprozessor

Ein Hauptbestandteil des TPM ist der Kryptoprozessor, dieser besitzt verschiedenste Funktionen, die auch Engines genannt werden.

Im Folgenden werden diese Engines mit jeweils einer kurzen Beschreibung ihrer Funktion aufgelistet:

Random Number Generator (RNG) Der RNG ist einer der wichtigsten Bestandteile eines TPM und wird von anderen Engines benötigt. Hier wird laut Spezifikation eine maximal 32 Byte große Zufallszahl auf einmal generiert (vgl. [TCG-Arch, S. 46]).

Hash Engine Die Hash Engine wird zum Ausführen von Hashfunktionen auf externe und interne Daten verwendet. Neben den bekannten Secure Hash Algorithm (SHA) - Varianten unterschiedlicher Bitlängen (siehe. [TCG-Algo, S. 22ff]), unterstützt diese Engine den Hash Message Authentication Code (HMAC) (vgl. [TCG-Arch, S. 38]).

Key Generation ist eine weitere Engine, die sich um das Erstellen von neuen Schlüsseln kümmert, dabei verwendet Sie je nach Kontext den RNG oder HMAC aus der Hash-Engine als Key Derivation Function (KDF), wie es durch das National Institute of Standards and Technology (NIST) in [SP 800-108] und [SP 800-56A] vorgesehen wurde. Im Normalfall wird die KDF als Schlüsselableitung verwendet, da das TPM nur genau drei Schlüssel permanent speichert. Für mehr Informationen siehe auch : Verwalten des TPM-Speichers unter 2.1.2.

Asymmetric Engine Die Asymmetric Engine ist für sämtliche asymmetrische Operationen zuständig. Darunter zählen zum Beispiel, Verschlüsselung und Entschlüsselung, genauso wie das Signieren und Validieren von Signaturen. Die hierfür verwendbaren Algorithmen sind RSA und ECC auf Basis von Primzahlen. Eine genaue Auflistung befindet sich in [TCG-Algo].

Symmetric Engine Die Symmetric Engine ist für symmetrische Operationen zuständig. Hierfür werden die Algorithmen AES, SM4, Camellia und Triple DES zur Verfügung gestellt. Mögliche Parameter sind unter [TCG-Algo, S. 27] aufgelistet. Mit dieser Engine ist es ebenfalls möglich zu signieren und Signaturen zu verifizieren. Hierfür wird HMAC verwendet (vgl. [TCG-Arch, S. 38]).

2.1.2 Speicher im TPM

Neben dem Kryptoprozessor gibt es noch zwei Speicherbereiche. Hier werden unter anderem die Schlüssel aufbewahrt. Es gibt sowohl einen persistenten Speicher (Non-Volatile Memory), als auch einen temporärer Speicher (Volatile Memory), welcher nach dem Neustarten eines Computers ungültig wird. (vgl. [TCG-Arch, S. 33])

Dictionary Attack Protection

Um den unbefugten Zugriff auf den geschützten Speicher zu verhindern, besitzt das TPM eine Dictionary Attack Protection. Bei einem "Wörterbuch-Angriff" wird versucht, wird beispielsweise versucht, ein Passwort dadurch zu erraten, indem man die wahrscheinlichsten Passwörter aus einer Liste (Wörterbuch) der Reihe nach ausprobiert. Ein Schutz davor im TPM sieht vor, dass der Zugriff auf die verschiedenen Ressourcen nach zu häufigen fehlerhaften Zugriffsversuchen für eine Weile gesperrt wird. Auf diese Weise wird ein solcher Angriff so sehr verlängert, dass sich diese Methode nicht mehr lohnt.

Dies wird umgesetzt, indem für jeden relevanten Speicherbereich auch immer ein Zähler für fehlerhafte Versuche angelegt wird. Dieser Zähler lässt sich konfigurieren, indem man eine maximale Versuchsanzahl angibt, ab den man keinen Zugriff erhalten darf und damit die Daten gesperrt sind. Um den Zähler eines Speicherbereiches nach Ablauf einer definierter Zeit wieder herunterzuzählen und somit der gesperrte Bereich wieder freizugeben, kann die sogenannte **Selbstheilung** konfiguriert werden. (vgl. [TCG-Arch, S. 131ff])

Seeds

Im Speicher des TPM werden ebenfalls vier Seeds gespeichert, die Grundlage für verschiedene Schlüssel sind. Ein TPM selbst speichert die Seeds in einem Speicherbereich, der nicht ausgelesen werden kann. Ein Seed ist, vereinfacht gesagt, lediglich eine große Zufallszahl, die verwendet wird, um mittels einer KDF zunächst die sogenannten Primary Keys abzuleiten. Erst durch diese Ableitung wird das Format der symmetrischen oder asymmetrischen Algorithmen entschieden. Auf diese Weise ist wenig Speicher notwendig, während viele Algorithmen unterstützt werden können. (vgl. [TCG-Arch, S. 72ff])

Wie alle Schlüssel kann man auch die Primary Keys wiederum ableiten. Je nach Verwendungszweck ist es so möglich, mehrere reproduzierbare Schlüssel zu erstellen, ohne dass die Schlüssel das TPM verlassen. Eine Ausnahme stellen hier natürlich Public Keys dar.

Es ist vorgesehen, dass die drei der insgesamt vier Seeds im Non-Volatile Memory gespeichert werden.

Endorsement Primary Seed (EPS) Aus dem EPS kann der Endorsement Key (EK) abgeleitet werden. Die Besonderheit dieses Seeds ist, das er vom Hersteller erstellt und signiert wurde. Auf diese Weise wird es unter anderem möglich, zu verifizieren, ob man mit einem echten TPM oder z.B. mit einem simulierten TPM kommuniziert.

Mann kann diesen Seed neu generieren, doch dadurch werden sämtliche Schlüssel und Zertifikate ungültig. Der automatisch neu generierte Seed ist dann nicht mehr vom Hersteller signiert, wodurch auch das Vertrauen verloren geht, mit einem echten TPM zu kommunizieren. (vgl. [TCG-Arch, S. 73f])

Durch die Verwendung dieses Seed wird die Privatsphäre beeinträchtigt, da sich die hieraus entstandenen Signaturen korrelieren lassen und so einen einzelnen TPM zuordnen. (vgl. [ACG15, S. 108f])

Platform Primary Seed (PPS) Der PPS wird verwendet, um den Platform Key abzuleiten. Dieser Seed und seine Schlüssel sind für die Platform Firmware wie zum Beispiel UEFI vorgesehen, seine Verwendung soll nicht dem Betriebssystem oder irgendwelchen Programmen zur Verfügung stehen. Er darf nur vom Hersteller zurückgesetzt werden. (vgl. [TCG-Arch, S. 74])

Storage Primary Seed (SPS) Der SPS wird verwendet, um den Storage Root Key (SRK) abzuleiten. Da dieser Seed vom Eigentümer generiert wird, wird er auch als Owner Seed bezeichnet. Dieser Seed ist für die Verwendung vom Betriebssystem und Programmen vorgesehen. (vgl. [TCG-Arch, S. 74f])

NULL Seed Der NULL Seed, ist der einzige seed, welcher im temporären Speicher gespeichert wird. Dieser NULL Seed wird bei jedem Reset neu generiert. Ein Reset kann durch ein Stromverlust oder das Neustarten des Computers ausgelöst werden. Die Schlüssel, welche aus dem NULL Seed abgeleitet werden, nennt man Ephemeral Keys. Sie gehen durch das Neugenerieren des Seeds verloren. (vgl. [ACG15, S. 113] und [TCG-Arch, S. 75])

persistenter Speicher (Non-Volatile Memory)

Der persistente Speicher oder auch Non-Volatile (NV) Memory, ist ein Speicher, der die dort abgelegten Daten dauerhaft speichern soll. Hierbei ist es ebenfalls möglich den performanteren temporären Speicher als Cache für den persistenten Speicher zu nutzen. Die Entscheidungen über dieses Implementierungs-Detail wird den Herstellern überlassen. (vgl. [TCG-Arch, S. 52f])

NV Index und Berechtigungen

Im **NV Index** kann man selbst Daten speichern. Dabei muss der benötigte Speicherplatz vorher reserviert werden. Die hierbei definierte Größe kann im Nachhinein nicht mehr verändert werden. Sobald der Speicherplatz im NV Index definiert wurde, ist es möglich dort Daten abzulegen. Beispiele hierfür könnten z.B. ein Counter, ein Passwort oder sogar ein Transport Layer Security (TLS)-Zertifikat sein. (vgl. [TCG-Arch, S. 211ff])

Zudem lässt sich jeder Speicherplatz im NV Index einzeln absichern, dies geschieht ebenfalls beim Reservieren des Speicherplatzes. Es ist durch das Aufstellen verschiedener Regeln für die Lese- und/oder Schreibrechte möglich, den Zugriff zu beschränken. Beispiele solcher Regeln sind u.a. PIN, PCR-Bänke (siehe “temporärer Speicher”) oder sogar komplexe Policies. (vgl. [TCG-Arch, S. 211ff])

Policies

Bei einer Policy, wird eine **Session** ggf. mit einem **Context** zwischen verschiedenen Operationen mitgenommen und deren Resultate fließen dann in die aktuelle Session ein, sodass weitere Operationen wie das Lesen eines NV-Indexes abgesichert wird. Diese Funktionalität wird Enhanced Authorization (EA) genannt. (vgl. [TCG-Arch, S. 115ff])

Temporärer Speicher (Volatile Memory)

Der temporäre Speicher kann durch ein Reset, Stromverlust oder einen Neustart zurückgesetzt werden. Neben der Bezeichnung temporärer Speicher (engl. Volatile Memory) wird dieser Speicher oft auch Random Access Memory (RAM) genannt.

Damit das TPM die oben erwähnten Funktionalitäten umsetzen kann, wird der temporäre Speicher für die Umsetzung benötigt. Beispielsweise werden die Zustände der Sessions, welche für die Policies benötigt werden, im temporären Speicher abgelegt. Des Weiteren werden hier aber auch die verschiedenen Schlüssel, welche gerade verwendet werden, zwischengespeichert. (vgl. [TCG-Arch, S. 33])

Platform Configuration Registers (PCR)

Der Platform Configuration Registers (PCR) ist einer der geschützten Bereiche des temporären Speichers, doch es wird nicht der vollständige temporäre Speicher vor Zugriffen geschützt.

PCRs bieten die Möglichkeit, den Zustand von Software aufzunehmen und so zu überwachen. Dazu gibt es verschiedene PCR-Bänke, in denen die Software Datensätze hinzufügen (engl. extends) kann. Der für PCRs vorgesehene Speicherbereich ist allerdings vor dem direkten Beschreiben oder Ersetzen geschützt. Bei den Extends einer PCR-Bank wird eine Hashfunktion auf den alten Wert der PCR-Bank zusammen mit dem neuen Wert ausgeführt, das Ergebnis wird dann wieder in der PCR-Bank abgelegt. Auf diese Art lassen sich viele Schritte mit ihren Zuständen in einer PCR-Bank zusammenfassen. Es fällt ein abweichender Zustand ab dem ersten Schritt auf, da durch das extends ein anderer Wert in der PCR-Bank vorgefunden wurde. (vgl. [ACG15, S. 151f] und [TCG-Arch, S. 50f])

Quote

Neben dem Auslesen der aktuellen Werte der PCR-Bänke, gibt es die Möglichkeit eine Quote über ausgewählte PCR-Bänke und deren aktuelle Werte zu erstellen. Hierbei werden diese Werte zusammen mit einem gewünschten Nonce durch das TPM signiert. Auf diese Wiese ist es möglich, den Zustand der PCR-Bänke zu kennzeichnen und so lässt sich der Zustand der Software über einen unsicheren Kommunikationsweg von außerhalb überprüfen. Durch das Nonce wird die Aktualität der PCR-Bänke sichergestellt und eine Replay-Attack verhindert. (vgl. [ACG15, S. 156ff])

2.1.3 Einsatzmöglichkeiten

Die Einsatzmöglichkeiten eines TPM sind vielfältig. Eine vollständige Liste kann hier aus Platzgründen nicht dargestellt werden. Im Folgenden Abschnitt werden nun allerdings eine kleine Anzahl von häufigen Einsatzmöglichkeiten erläutert.

Eine naheliegende Möglichkeit ist zum Beispiel die Verwendung der asymmetrischen Funktionen des TPM für TLS-basierte Übertragungen. Dadurch lässt sich einerseits der private Schlüssel sicher im TPM aufbewahren als auch die kryptografischen Berechnungen im TPM durchführen. So ist es zum Beispiel möglich einen Mailserver, Webserver oder eine VPN-Verbindung mittels DTLS[3] abzusichern.

Measured Boot

Durch die PCRs ist eine Methode entstanden, bei dem das TPM die Möglichkeit bietet, den Bootvorgang eines Computers vor Manipulation sicherzustellen, indem es diesen überprüfbar macht. Hierbei wird die beim Booten ausgeführte Software vermessen und je nach erwarteten Ergebnis gehandelt. Die Funktion des Vermessens während des Bootvorgangs muss vom Prozessor ausgeführt werden, was mit modernen Prozessoren aber problemlos möglich ist. Diese Funktionalität ist je nach Prozessorhersteller unter Intel Trusted Execution Technology (Intel TXT), ARM TrustZone und AMD Secure Technology bekannt. (vgl. [ACG15, S. 331ff])

[3] Datagram Transport Layer Security (DTLS) wurde für UDP entwickelt und basiert auf TLS. Damit sollte auch in verbindungslosen und unzuverlässigen Übertragungen kryptografische Sicherheit einziehen. (Für Details siehe [RFC6347].)

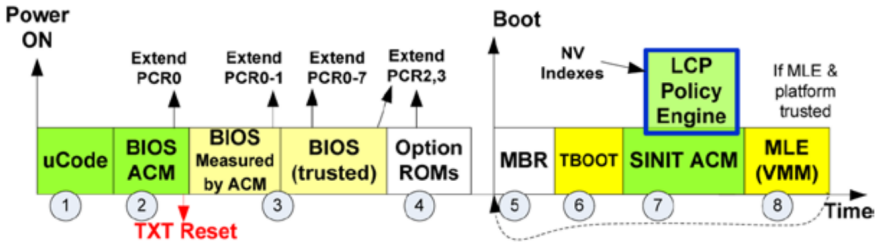

Abbildung 2.2: Intel TXT boot timeline (Figure 22-1 aus [ACG15, S. 336])

Wie in Abbildung 2.2 zu sehen ist, wird die beim Booten involvierte Software vermessen und deren Ergebnisse in verschiedene PCR-Bänke ablegt. In Schritt 7 ist es möglich, Daten im NV-Index durch eine Policy freizugeben. Diese Policy kann die Messung des Bootvorganges bis zu diesen Zeitpunkt als Bedingung haben. Ein Beispiel für Daten, die so geschützt werden können, ist der Schlüssel einer verschlüsselten Festplatte. Dieser Schlüssel kann automatisch freigegeben werden und auf diese Weise wird sichergestellt, dass keine Schadsoftware bis zu diesen Zeitpunkt geladen wurde, die den Schlüssel abgreift und veröffentlicht.

Projekt - SiDaFab

An der Hochschule Bremen in der "Forschungsgruppe Rechnernetze und Informationssicherheit" (FRI) gab es das Projekt "Sichere Datenkommunikation für die verteilte Fabrik der Zukunft" (SiDaFab).

Innerhalb des SiDaFab-Projektes wurde ein Demonstrator für eine wirksame und effiziente hardwarebasierte IT-Sicherheitslösung erstellt, der im Industrie 4.0 Umfeld eingesetzt werden kann und die speziellen Charakteristika eines Prozesssteuerungsnetzes berücksichtigt.

Zum Schutz der beteiligten Systeme wurden TPMs als Vertrauensanker eingesetzt. So wurde eine sichere Kommunikation mit Ende-zu-Ende-Verschlüsselung über physikalisch gesichertes Schlüsselmaterial gewährleistet. Zudem kann die Integrität von Hard- und Softwaresystemen sichergestellt und somit die Sicherheit gegenüber reinen Softwarelösungen deutlich erhöht werden.

2.2 TPM Software Stack

Nachdem die Spezifikation der Hardware TPM mit ihren Funktionalitäten erläutert wurde, stellt sich nun die Frage, wie das TPM verwendet werden kann.

Hierfür gibt es eine Reihe von Spezifikationen, doch zunächst zu denen, die für das Betriebssystem relevant sind. Ein Computer kann mehrere TPMs haben, egal ob Sie als Software oder Hardware vorhanden sind. Diese TPMs müssen genauso wie die darauf zugreifenden Anwendungen mit ihren einzelnen Aufgaben verwaltet werden.

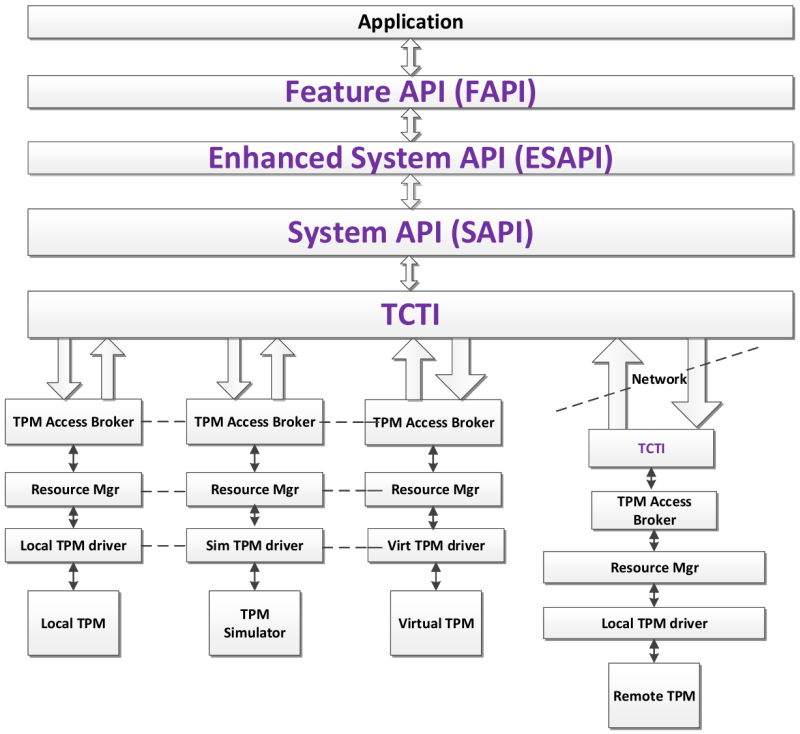

Abbildung 2.3: Using TCTI to Connect to Various Target TPMs (Figure 2 aus [TCG-TCTI, S. 10])

Die Gesamt-Struktur wird in Abbildung 2.3 mit einigen möglichen Konstellationen an TPMs dargestellt. Zudem werden die benötigten Komponenten im Betriebssystem und auch die spezifizierten Bibliotheken bis zur Verwendung in einer Anwendung dargestellt. Nachfolgend sollen die einzelnen Komponenten der Abbildung 2.3 von unten beginnend erläutert werden.

TPM Access Broker

Die genannten Aufgaben werden vom TPM Access Broker (TAB) übernommen. Er sorgt dafür, das zur gleichen Zeit nur ein Befehl (TPM Command) zum TPM übermittelt wird. Dazu werden die einzelnen TPM Commands von verschiedenen Anwendungen in Contexten verwaltet. Damit nun verschiedene Anwendungen gleichzeitig auf den TPM zugreifen können, werden die Commands atomar gehalten und deren Context je nach Bedarf der Anwendungen gewechselt. (vgl. [ACG15, S. 95] und [ACG15, S. 290])

Resource Manager

Die Ressourcen des TPM sind limitiert, daher werden sie mit Bezug auf den Context vom Resource Manager (RM) verwaltet. Ein TPM hat zum Beispiel nur wenig Speicherplatz und kann auch nur maximal drei Sessions halten. Daher übernimmt der RM die Verwaltung und Steuerung. Er lädt die benötigten Daten je nach Bedarf aus und in dem TPM. (vgl. [ACG15, S. 95] und [ACG15, S. 291])

Da beide Funktionalitäten verwandt sind, wurden Sie zusammen im [TCG-TAB-RM] spezifiziert[4], dennoch wurden sie in der Abbildung 2.3 einzeln dargestellt.

TPM Command Transmission Interface

Für die Kommunikation mit dem TPM ist die Spezifikation TPM Command Transmission Interface (TCTI) interessant (zu finden unter [TCG-TCTI]). Mit diesem Protokoll wird nicht nur der TPM-Treiber angesprochen, sondern auch der TPM Access Broker (TAB) von den verschiedenen Anwendungen, da dieser eine Multi-Prozess Abstraktion des TPM darstellt.

[4] Für den Linux-Kernel vor Version 4.12 wurden ebenfalls noch beide Funktionen in einem zusätzlichen Dienst "TPM2 Access Broker & Resource Manager" (tpm2-abrmd) unter `https://github.com/tpm2-software/tpm2-abrmd/` entwickelt.

2.2.1 Application Programming Interfaces

Für die Kommunikation mit den TPM existieren drei Spezifikationen für Anwendungsschnittstelle in Form von Bibliotheken. Diese sollen eine klar definierte Nutzung des TPM vorgeben und verhindern, dass dieselben Funktionalitäten immer wieder und auf verschiedene Weise implementiert werden müssen. Hierfür wurden die drei aufeinander aufbauende Bibliotheken "System API (SAPI)", "Enhanced System API (ESAPI)" und "Feature API (FAPI)" spezifiziert.

System API

Die Spezifikation des System API (SAPI) entspricht dem Programmieren in der Sprache C. Sie sieht vor, die einzelnen Befehle (TPM Commands) direkt zu übersetzen und diese binär mit Hilfe des TPM Command Transmission Interface (TCTI) an das TPM zu übermitteln. Das SAPI ist so konstruiert, dass es bei der Verwendung selbst keinen Speicherplatz reserviert, diese Aufgabe muss der Entwickler selbst übernehmen. Durch die Komplexität der Spezifikation ist diese Anwendungsschnittstelle eher für Experten und nicht für unerfahrene Entwickler geeignet. (vgl. [ACG15, S. 85ff]) Für weitere Informationen siehe auch [TCG-SAPI], hier ist die Spezifikation zu finden.

Enhanced System API

Die Hauptaufgabe des Enhanced System API (ESAPI) ist es, die Komplexität des SAPI zu verringern und dafür verwendet es selbst das SAPI. Das ESAPI sieht vor, ebenfalls kryptografische Aufgaben zu übernehmen, um unter anderem die Sessions mit dem TPM abzusichern. Die Spezifikation benötigt immer noch ein tieferes Verständnis, trotz ihrer starken Vereinfachungen gegenüber dem SAPI und sie ist unter [TCG-ESAPI] zu finden. (vgl. [ACG15, S. 77])[5]

[5] Die ESAPI wird in der Arbeit deshalb verwendet, da hier alle benötigten Funktionen garantiert zur Verfügung stehen.

Feature API

Die Feature API (FAPI), hat nicht den Anspruch, alle Funktionalitäten des TPM auch zur Verfügung zu stellen. Das FAPI möchte mit der Umsetzung von ca. 80 % der Möglichkeiten des TPM dazu beitragen, dass der Benutzer dieser Bibliothek mit wenigen Parametern und Aufrufen zu seinem Ziel gelangt. Die restlichen 20 % werden als spezielle Fälle vernachlässigt, hierfür gibt es immer noch die ESAPI und SAPI. Die Spezifikation ist noch nicht abgeschlossen, doch ein erster Entwurf (Draft) ist unter [TCG-FAPI] zu finden. Daher gibt es auch noch keine vollständige Implementierung dieser API. (vgl. [ACG15, S. 79ff])

2.2.2 Implementierungen vom TPM Software Stack

Die Sammlung an Spezifikationen der erwähnten APIs werden unter dem Begriff "TPM Software Stack (TSS)" zusammengefasst. Es gibt verschiedenste Implementierungen dieser Spezifikationen, die in verschiedensten Programmiersprachen von verschiedenen Anbietern zur Verfügung gestellt werden.

Hier eine kleine Übersicht an Implementierungen:

TPM2-TSS ist eine unabhängige OpenSource Implementierung, an der sich neben den Mitgliedern der TCG auch andere interessierte Programmierer beteiligen. Von dieser Community wird ebenfalls ein Python Binding entwickelt und eine Bibliothek für Public-Keys (PKCS #11) bis zur OpenSSL Engine. Des Weiteren wird eine ganze Sammlung von Tools zum Verwalten des TPMs zur Verfügung gestellt. Eines dieser Tools ermöglicht den Bootvorgang durch die PCR-Bänke zu überwachen und gleichzeitig das Vertrauen in eine Person mit Hilfe eines "Time-based One-time Password" (TOTP) zu überprüfen.
Webseite: `https://tpm2-software.github.io`
Source Code: `https://github.com/tpm2-software`

IBM's TPM 2.0 TSS ist ebenfalls eine Implementierung in C, die sich allerdings nicht an die Spezifikation der TCG (ESAPI, SAPI und FAPI) hält. Der Umfang der Bibliothek ist groß und es sollten vermutlich die meisten Anwendungsfälle in ihren über 100 Beispiel-Tools des TPM umgesetzt sein.
Source Code: `https://sourceforge.net/projects/ibmtpm20tss`

TSS.MSR ist eine Sammlung von TSS Implementierungen von Microsoft.
Die C-Implementierung ist ein Fork vom *TPM2-TSS*. Doch hier werden autogenerierte Implementierungen der TSS Bibliothek für die Programmiersprachen C++, JavaScript (NodeJS), Java, .Net und Python zur Verfügung gestellt.
Source Code: `https://github.com/microsoft/TSS.MSR`

TPM.JS wurde von Google entwickelt und besteht unter anderem aus einem Binding vom *TPM2-TSS* für die Programmiersprache JavaScript. Doch im Browser wird kein Hardware TPM angesprochen, sondern der im WebAssembly befindliche "*IBM Software TPM 2.0 Simulator*".
Webseite: `https://google.github.io/tpm-js/`
Source Code: `https://github.com/google/tpm-js`

Rust-ESAPI ist ein Binding der ESAPI von *TPM2-TSS* in Rust. Diese API wird von dem Projekt ParSec ("**P**latform **A**bst**R**action for **SEC**urity") entwickelt. Das Ziel des Projektes ist es, dass Vertrauen in ein HSM und TPM auch in plattformunabhängige Dienste zur Verfügung zu stellen. So kann ein gewisses Vertrauen innerhalb von Containern auf unsicheren (Cloud-) Umgebungen gewonnen werden.
Source Code: `https://github.com/parallaxsecond/rust-tss-esapi`

Go-TPM ist eine Implementierung in Golang, die noch am Anfang ihrer Entwicklung steht.
Source Code: `https://github.com/google/go-tpm`

Bei der Suche nach einer geeigneten Implementierung für den eigenen Anwendungsfall begegnen einem immer wieder TSS Bibliotheken für den TPM 1.2 wie zum Beispiel **TrouSerS**, welcher von IBM entwickelt wurde.

*Im Rahmen dieser Arbeit wird das Fazit gezogen, dass es sinnvoll ist, auf Basis der C-Implementierung von **TPM2-TSS** zu arbeiten und sich so an die TSS Spezifikation zu halten. Da die Programmiersprache C dem Entwickler nur geringe Beschränkungen in der Art der Programmierung auferlegt, steigt mit ihrer Benutzung auch die Wahrscheinlichkeit Fehler zu machen. Aus diesem Grund werden die Anwendungen, die mit ein TPM kommunizieren sollen, im Rahmen dieser Arbeit in der Programmiersprache Rust geschrieben. Rust nutzt moderne Programmierparadigmen, die übliche Programmierfehler verhindern sollen. Der Compiler hilft dabei, dass Anwendungen sicher auch bei Nebenläufigkeit programmiert werden. Mehr als diese Zusammenfassung der Beschreibung ist im Buch "Why Rust?"[Bla15] selbst zu finden.*

Für die weitere Entwicklung wäre es ebenfalls interessant, wenn man seine eigene Software vielleicht sogar automatisiert gegen die eines Simulators testen könnte. Der Quellcode vom bekanntesten Simulator "IBM Software TPM 2.0 Simulator" wird vom IBM frei zur Verfügung gestellt:
`https://sourceforge.net/projects/ibmswtpm2/`

2.3 TAP Funktionsüberblick

Das “Trusted Attestation Protocol (TAP)” ist eine Spezifikation, die “Remote Attestation” umsetzt. Hierbei wird bei dem Computer von einer autorisierten Stelle aus überprüft wird, ob der Computer manipuliert wurde, indem zum Beispiel die Software verändert wurde. Das TAP wurde von der TCG entwickelt, unter anderem von der Workgroup “Trusted Network Communications” (TNC).

Bevor TAP entwickelt wurde, gab es bereits eine Sammlung an Spezifikationen für den TPM 1.2. Diese Spezifikationen bauten aufeinander auf, sodass der “Platform Trust Services (PTS)” [TCG-PTS] die Aufgaben hatte, welche nun durch das TAP mit geringerer Komplexität abgelöst werden soll. Neben der geringeren Komplexität und der TPM 2.0 Unterstützung ist ein wesentlicher Unterschied, dass in der Spezifikation selbst nicht definiert wird, wie die Nachrichten der Remote Attestation übermittelt werden. Das TAP definiert den Aufbau und Inhalt der Nachrichten, zwischen den zu überprüfenden Computer (Attester) und der autorisierte Stelle (Verifier) übermittelt werden. (vgl. [TCG-TAP1, S. 6])

2.3.1 Freshness / Nonce

Bevor ein paar mögliche Anwendungsfälle für das TAP vorgestellt werden, folgt eine Erläuterung des Freshness Konzeptes, welches in den verschiedenen Anwendungsfälle benötigt wird. Die Werte vom Attester werden in einem Quote vom TPM signiert, wodurch sichergestellt wird, dass die Werte von dem zu überprüfenden Computer kommen und bei der Übertragung nicht manipuliert wurden. Die Idee hinter Freshness ist es, die Aktualität der Werte sicher zu stellen, indem eine Nonce sich ebenfalls im Quote befindet. Das TAP sieht drei Methoden zur Herkunft einer solchen Nonce vor.

Nonce vom Verifier

Die einfachste Methode ist, wenn der Verifier eine Nonce generiert. Diese Nonce wird dann vom Attester erstellten Quote signiert, wie in Abbildung 2.4 skizziert.

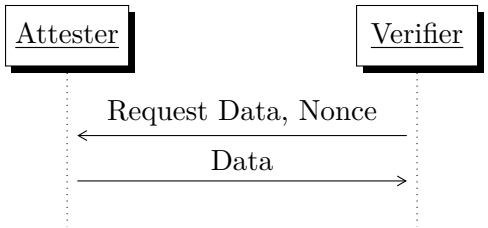

Abbildung 2.4: Sequenzdiagramm für Nonce vom Verifier (gekürzte Figure 1 aus [TCG-TAP1, S. 9])

Nonce von 3rd Party

Eine weitere Methode sieht vor, dass eine Nonce von einer dritten Partei zur Verfügung gestellt wird. Dieser Ablauf der Kommunikation wird in Abbildung 2.5 dargestellt. Die dritte Partei muss vor allem auch für den Verifier vertrauenswürdig sein, ein Time-Stamping Dienst wird hierfür vorgeschlagen.

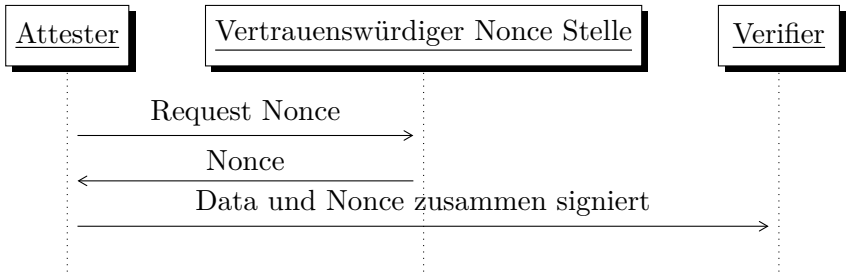

Abbildung 2.5: Sequenzdiagramm für Nonce vom 3rd Party (Figure 2 aus [TCG-TAP1, S. 10])

Clock-Based Attestation

Der Kommunikationsablauf einer Clock-Based Attestation entspricht der *Nonce vom 3rd Party*. Hierbei werden Werte wie die Counter des TPM Reset und Restart zu einem Time-Stamping Dienst geschickt. Der Time-Stamping Dienst signiert diese Werte mit der aktuellen Zeit. Auf diese Weise kann der Verifier auch die Counter ebenfalls mit in die Überprüfung einbeziehen.

2.3.2 Anwendungsfälle

Bei der Spezifikation von TAP wurden zunächst die möglichen Anwendungsfälle für das Protokoll festgehalten. Dadurch war es möglich diese Aspekte in den Details der Spezifikation zu berücksichtigen. Ein paar dieser Anwendungsfälle sollen hier zum Verständnis erwähnt werden.

Attester Zugang zu Netzwerke oder Computer Cluster

Einer der Anwendungsfälle ist das Erlauben von Zugängen zu ein Netzwerk oder ein Computer Cluster. Hierbei wird die Kommunikation vom Attester initialisiert, welcher ein Zugang beim Verifier erfragen möchte.

Dieser kann zum Beispiel bei der Übermittlung eines Quotes vom TPM des Attesters, die Signatur, Nonce und PCR-Bänke überprüfen. Damit stellt er fest, dass der gewünschte TPM mit aktuellen Werten antwortet. Mit dem Abgleich von vorherigen Antworten überprüft der Verifier, ob der Attester sich in einem sicheren Zustand befindet, bevor der Verifier ihm Zugang gewährt.

Attester Remeasurement (Neuvermessung)

Ein Weitere Anwendungsfall ist, das der Verifier den Attester neu vermessen möchte. Dafür sendet der Verifier ein Nonce zum Attester, um die aktuellen Werte der PCR-Bänke zu erfahren. Dadurch ist es dem Verifier möglich, die von ihm aufbewahrten Werte aus einer vorherigen Messung zu aktualisieren.

Diese Methode ist von besonderem Interesse für diese Arbeit, da die Kommunikation vom Verifier aufgebaut wird. Dadurch würde der Verifier Teil des Monitoringsystems sein.

Unidirektionale Attestation

Der typische Anwendungsfall einer unidirektionalen Attestation ist eine Situation, in der nur eine Kommunikation vom Attester zum Verifier erlaubt ist. Diese Restriktion kann entstehen, wenn sich der Verifier in einem nicht vertrauenswürdigen Netzwerk befindet.

Hierbei verwendet der Attester das Freshness-Verfahren der *Clock-Based Attestation*, anstelle der klassischen Nonce. Zudem signiert der Attester das Quote mit einem Zertifikat, das mindestens aus einer Zertifikats-Kette besteht, welches vom Verifier vertraut wird.

Attestation ohne Sessionmanagement

Die Situation in der TAP über ein Protokoll ohne Verwaltung von Sessions übertragen werden müsste, ist ein Anwendungsfall für das sogenannte “Attestation ohne Sessionmanagement” Um dieses Problem zu lösen, wird die Kommunikation auf eine Nachricht über das TAP minimalisiert. Hierfür wird die Freshness nicht durch ein Nonce vom Verifier sichergestellt, sondern durch eine *Nonce vom 3rd Party* oder der *Clock-Based Attestation*.

2.4 Monitoring

Die Idee, Systeme zu überwachen, um so vor Ausfällen und dem Erschöpfen von Ressourcen rechtzeitig informiert zu werden, ist nicht neu. Insbesondere bei großen Unternehmen steigt die Komplexität durch die große Anzahl an Systemen, die für eine ausfallsichere Unternehmensstruktur stets im Blick behalten werden sollten. Im Laufe der Zeit sind viele Lösungen entwickelt worden, die sich an verschiedenen Einsatz-Szenarien orientieren. Diese Einsatz-Bereiche sollen im Weiteren kurz erwähnt werden, um anschließend auf die Struktur von Monitoringsystemen und deren Komponenten einzugehen.

2.4.1 grundsätzliche Einsatzmöglichkeiten

Ein Interesse an Monitoringsystemen besteht vor allem im operativen Bereich (wie in ein Rechenzentrum), um schnell reagieren zu können, wenn sich Veränderungen in den ihnen anvertrauten Rechnersystemen abzeichnen. Doch auch einzelne Geschäftsleitungen möchte gegebenenfalls einen Einblick in ihr jeweiliges Monitoringsystem haben, damit Sie auf Basis dieser Daten ggf. ökonomische Entscheidungen treffen können. Die aus einem Monitoringsystem ablesbaren, ökonomischen Werte und Kennzahlen können somit als “Key Performance Indicators” (KPI) genutzt werden. Ob ein Kunde die Unternehmensinfrastruktur effektiv nutzt, ob die Nutzungszahlen wachsen, zurückgehen oder stagnieren, sind lediglich exemplarische Fragen, denen mithilfe eines Monitoringsystems auf den Grund gegangen werden kann. Darüber hinaus kann ein Monitoringsystem sowohl für die Visualisierung der Zufriedenheit der Nutzer als auch zur Darstellung der gegenwärtigen Geschäftssituation (beispielsweise der Wirtschaftlichkeit) genutzt werden. (vgl. [Jul17, S. 57ff])

Eine weitere Einsatzmöglichkeit eines Monitoringsystems ist der Bereich der IT-Sicherheit, um IT-Infrastrukturen abzusichern. Hier können Informationen wie die Anzahl und Zeiten der Authentifizierungen oder auch die Ergebnisse von Anti-Virenscans zusammenlaufen, um das Erkennen verschiedener Angriffsmuster und damit das Ergreifen frühzeitiger Maßnahmen zu ermöglichen. Ein “Intrusion Detection System” (IDS) oder eine Erkennung von “Denial of Service” (DOS) - Angriffen sind Beispiele solcher Warnsysteme, die in ein Monitoringsystem eingebunden werden können. (vgl. [Jul17, S. 125ff])

Der Reifegrad eines Monitoring-Systems lässt sich laut [Tur16, S. 9ff] in drei Stufen unterteilen:

Nutzer-Initialisiertes Monitoring bezeichnet nicht automatisierte Prozesse. Falls überhaupt vorhanden, werden unter diese Art des Monitorings Checklisten bis hin zu Skripten verstanden, welche manuell vom Nutzer durchgearbeitet bzw. ausgeführt werden.

Reaktives Monitoring bezeichnet ein System, das die Verfügbarkeit der IT-Assets überprüft. Oftmals wird es im Nachhinein implementiert und dient zum Monitoring der Hardware (wie beispielsweise CPU, Speicherverbrauch). Alerts und Dashboards sind, falls vorhanden, nur für den operativen Gebrauch gedacht.

Proaktives Monitoring bezeichnet Systeme, die den Kern der IT darstellen. Beim Proaktiven Monitoring wird aus dem Konfigurationsmanagement automatisch das Monitoring generiert. Zudem liegt der Fokus auf dem Service und damit der Kundenerfahrung, sodass die Leistung der Anwendung überwacht wird und nicht mehr zwangsläufig die der Hardware. Des Weiteren können die Daten so aufbereitet sein, dass sogar die Geschäftsführung die Dashboards des Monitoringsystems für die Budgetplanung und die Überwachung der KPIs nutzen kann.

2.4.2 Komponenten eines Monitoringsystems

Trotz der verschiedenen Lösungen, die es an Monitoringsysteme gibt, lässt sich in allen Systemen eine Struktur erkennen. Diese Struktur besteht aus den folgenden bis zu fünf Komponenten, wobei mehrere Komponenten von einer Software abgedeckt werden können.

Datenerhebung: Die Datenerhebung ist dafür verantwortlich, dass die Werte der zu überwachenden Systeme an den Datenspeicher übermittelt werden. Diese Aufgabe lässt sich auf verschiedene Varianten lösen, mehr dazu später.

Speicher der Daten: Eine Speicherung der Daten, die während der Erhebung entstanden sind, ist notwendig für die Auswertung in den nächsten Schritten. Hierfür gibt es speziell entwickelte Datenbanken mit der Bezeichnung "Time-Series-Databases", mehr dazu später.

Visualisierung: Die Visualisierung der gespeicherten Daten kann dabei helfen, den Überblick zu behalten. Es gibt viele Darstellungsformen der Daten, die auf ein Dashboard zusammengefasst werden können. Diese reichen von einfachen Zahlenwerten, über Tabellen und Bar-Diagramme bis hin zu klassischen Liniendiagramme für den zeitlichen Verlauf und anderen Darstellungsformen.

Analytik und Reporting: Die Analytik und das Reporting sind als Auswertung außerhalb des IT-Bereiches gedacht. Diese Reports können für unternehmerische Entscheidungen oder als Grundlage für Verträge mit Kunden gelten. Die Frage danach, wie die KPIs aussehen oder ob die Verfügbarkeit des "Service-Level-Agreement" (SLA) mit den Kunden eingehalten wurde, können so beantwortet werden.

Alerting: Auf der anderen Seite ist das Alerting dagegen nur für die IT-Bereich relevant und soll bei kritischen Zuständen in der Infrastruktur die zuständigen Personen automatisch informieren, mehr dazu später.

Datenerhebung

In den Komponenten zur Datenerhebung gibt es grundlegende Unterschiede. Diese Unterschiede der Funktionsweise bestimmen die Struktur der Konfiguration des Monitoringsystems und die Art und Weise, wie neue zu überwachende Geräte oder Dienste ins Monitoringsystem zukünftig integriert werden.

Grundlegend ist hier beispielsweise die Unterschiedung in “Push” und “Pull” - getriebene Datenerbungssysteme. Im Folgenden sollen die Unterschiede kurz erläutert werden:

Push Im sogenannten “Push”-Verfahren werden die Daten selbstständig von dem zu überwachenden Geräten an die Datenerhebung übermittelt. Eine Hürde ist, dass jedes zu überwachende Gerät wissen muss, wie es die Datenerhebung des Monitoringsystems erreicht. Doch wenn sich die Erreichbarkeit des Monitoringsystems ändert, muss diese Einstellung auf alle Geräte verteilt werden. Je nach Funktionsweise des zu überwachenden Dienstes kann dieser die Veränderung sofort zu übermitteln, sodass eine sofortige Verarbeitung im Monitoringsystem erlaubt und braucht nicht auf das nächste Einsammeln der Werte zu warten. Beispiele hierfür sind Monitoringsysteme, die sogenannte “Logs” verarbeiten, bei der jede einzelne Aktion, vom zu überwachenden Dienst, an das Monitoringsystem übermittelt wird.

Pull Das “Pull”-Verfahren stellt die gegenteilige Option da, bei der die Daten von der Datenerhebung von dem zu überwachenden Gerät regelmäßig abgefragt werden. Dabei muss die Datenerhebung wissen, wie es sämtliche zu überwachende Geräte erreicht. Ein Vorteil ist, dass die Datenerhebung komplette Kontrolle über die Zeitpunkte hat, wann welches Gerät abgefragt wird. Auf diese Weise bleibt der Ressourcenverbrauch der Datenerhebung gleichmäßige, selbst wenn es zu größeren Problemen auf den zu überwachenden Systemen kommt. Durch die regelmäßigen Abfragen ist diese Methode optimal, um die Verfügbarkeit zu überwachen. Dabei kann, je nach Anbindung der Geräte am Monitoringsystem, ein Nachteil für die Sicherheit entstehen, da die überwachten Geräte eine eingehende Verbindung von der Datenerhebung erlauben müssen (vgl. [Tur16, S. 24].

(vgl. [Tur16, S. 23ff] und [Jul17, S. 16ff])

Des Weiteren gibt es noch das sogenannte **Blackbox**-Monitoring. Hierbei handelt es sich um eine Methode der Datenerhebung, bei ein Dienst von außerhalb überwacht wird. Da diese Überwachung den Dienst aus Sicht des Kunden testet, kann das damit versehende Monitoringsystem teilweise als "Proaktives Monitoring" eingeordnet werden. Ein solcher Test, der nur von außerhalb regelmäßig stattfinden kann, ist dem Pull-Verfahren zuzuordnen. Die Form des Blackbox-Monitorings kann auf verschiedenen Weisen stattfinden. Ein Variante ist die Überprüfung der Erreichbarkeit des Dienste von außerhalb. Eine weitere Variante ist das Testen sämtlicher Funktionen einer Schnittstelle, die ein Dienst dem Kunden zur Verfügung stellt. Die Möglichkeiten gehen soweit, dass die Software die beim Kunden läuft neben ihrer Hauptfunktion für den Kunden im Hintergrund auch den Dienst testet und Ergebnisse zum Monitoring übermittelt. Dieses "Real User Monitoring" spiegelt zwar am besten die Nutzererfahrung wieder, vor allem bei Performancefragen, doch es erhöht den Ressourcen-Verbrauch beim Kunden und wirft Fragen des Datenschutzes durch die Möglichkeit der Überwachung des genauen Verhaltens auf.[6]

Im Vorfeld dieser Arbeit fiel während der Entwicklung des Konzepts die Wahl auf die Software Prometheus. Diese Software für ein Monitoringsystem verwendet das oben genannte Pull-Verfahren.

[6] Die bekannteste Umsetzung des Blackbox-Monitorings in Form des Real User Monitorings ist wohl Google Analytics.

Speicher der Daten

Zum Speichern der von Monitoringsystemen gesammelten Daten werden spezielle Datenbanken verwendet, sogenannte “Time-Series-Databases”, in der jeder Eintrag mit einem Zeitpunkt versehen wird. Diese Datenbanken haben daher einen Funktionsumfang und Aufbau, der optimal für das Monitoring ist. Im Vergleich zu den relationalen Datenbanken werden für diese zeitbasierten Datenbanken teilweise andere Begriffe verwendet. Beispielsweise werden “Werte” als “Datenpunkte” bezeichnet, eine “Auswahl” als “Serie” und die gleiche Art an Datenpunkten werden hier nicht in “Tabellen” sondern in “Metriken” zusammengefasst. (vgl. [Tur16, S. 27f])

Je nach Software der Time-Series-Database, haben sie verschiedenen Funktionalitäten, die sich mit der Dauer und der Granularität beschäftigen. Beschränkt man die Dauer der Speicherung, hat man die Vorteile, dass weniger Speicherressourcen benötigt werden und dass die Datenschutzrichtlinien besser eingehalten werden. Dem gegenüber steht die Frage nach dem Nutzen von alten Daten, die sich seit dem Aufbau des Monitoringsystems ansammeln und selten weiter Verwendung finden. Damit ist die über einen Zeitraum begrenzte Speicherung oftmals zu bevorzugen. Die Granularität wiederum wird zwar anfangs meist von der Datenerhebung vorgegeben, doch auch hier stellt sich die Frage nach dem Nutzen von einer kleinteiligen zeitlichen Auflösung, wenn größere Zeitspannen betrachtet werden. Eine hohe zeitliche Auflösung kann nach einer Weile zu einer niedrigeren Auflösung mit größeren Zeitabständen reduziert werden, sodass z.B. die im Sekundenbereich aufgenommenen Werte nach einem Tag nur noch im Minutenbereich vorliegen und nach einer Woche nur noch ein Datenpunkt in der Stunde existiert. Dadurch wird weniger Speicherplatz benötigt und das Anfragen, z.B. für die Visualisierung, können schneller beantwortet werden, weil weniger Datenpunkte zurückgeliefert werden müssen. (vgl. [Jul17, S. 19f])

Für die Arbeit gilt, dass die vorgegebene Software Prometheus die Daten im Standardfall nur 14 Tage speichert. Des Weiteren nimmt Prometheus, nach dem Einsammeln, keine Änderung an der Granularität vor. (Das vorgesehene Konzept von Prometheus ist, eine weitere Prometheus-Instanz dahinter kaskadiert zu installieren, die die Werte der vorherigen Prometheus-Instanz mit einer höheren zeitlichen Auflösung und für einen längeren Zeitraum speichert.)

Alerting

Ein Aspekt beim Alerting ist die Relevanz. Falls es zu Problemen kommt, wird wahrscheinlich nicht nur ein Alert stattfinden. Daher sollten diese beim Einrichten bereits klassifiziert werden. Eine automatische Klassifizierung der verschiedenen Alerts kann helfen, eine Reihfolge für die Abarbeitung festzulegen. [7] Hierbei wird mindestens die Nutzung von "kritischer Alert" und "Warnung" vorgeschlagen. Unter kritische Alerts werden komplette Ausfälle verstanden, bei denen gegebenenfalls auch IT-Personal im Bereitschaftsdienst durch Anrufe geweckt werden kann. Auf der anderen Seite machen Warnungen lediglich auf Misszustände aufmerksam, während die Infrastruktur selbst noch funktioniert. Ein Beispiel hierfür wäre: Ein vorgesehenes Backup hat in der Nacht nicht stattgefunden. (vgl. [Jul17, S. 31ff])

Die verschiedenen Metriken im Monitoringsystem ermöglichen es, auch präventive Warnungen durch die Alerting-Komponente erstellen zu lassen. Ein Beispiel hierfür ist die Ressource Speicherplatz, wo eine Warnung hilfreich ist, bevor die Festplatten voll sind, um rechtzeitig für eine Erweiterung zu sorgen. Dabei werden **Schwellwerte** definiert, ab wann eine Ressource als "knapp" gilt. Sobald die Metrik den Schwellwert erreicht, erstellt die Alerting-Komponente eine Warnung. Neben statische Schwellwerte wie "80% des Speichervolumens belegt" oder "Speicherplatz noch 10 GB" können auch statistische Methoden zum Einsatz kommen. Diese können Schwankungen, die z.B. durch die Erstellung eines Backups in der Nacht entstehen, herausgerechnet werden.[8] (vgl. [Jul17, S. 34])

Ein weiteres interessantes Themengebiet wäre die Selbstheilung von Alerts. Da eine vollständige Behandlung dieses Themas den Rahmen dieser Arbeit sprengen würde, sollen an dieser Stelle nur ein paar kleine Beispiele zur Erläuterung genannt werden. Eine Möglichkeit ist das Neustarten des Services oder des ganzen Systems, wenn diese nicht ordnungsgemäß funktionieren. Ein Erfolg des Neustarts bedeutet allerdings nicht, dass der Grund für die Auslösung des Alerts beseitigt ist. Des Weiteren können Ressourcen, die als ausgelastet oder nicht benötigt gelten, automatisiert erweitert oder gemindert werden. Hierfür bieten viele Rechenzentren APIs (Programmier-Schnittstellen), um beispielsweise virtuelle Maschinen dazu zu buchen oder den Speicher und die CPU-Cores zu erweitern.[9] (vgl. [Jul17, S. 36ff])

[7] Das Inzidenz Managment von ITIL kann dabei helfen die Alerts zu kategorisieren: `https://www.bsi.bund.de/DE/Publikationen/Studien/ITIL/index_htm.html`

[8] Für mehr Informationen zur statistischen Auswertung in Monitoringsytemen lohnt sich ein Blick in [Jul17, S. 45ff] und [Tur16, S. 42ff]

[9] Beispiele für eine API für Rechenzentren sind auch bei deutschen Anbietern, wie bei Hetzner, zu finden: `https://docs.hetzner.cloud/`

Für diese Arbeit wird nicht weiter auf das Alerting eingegangen. Wenn die erwarteten Werte nicht vom TPM durch das TAP übermittelt werden, ist dies als Sicherheitsvorfall zu bewerten und diesem muss nachgegangen werden. Ein solcher Sicherheitsvorfall kann nicht präventiv festgestellt werden. Zudem liegt die Kategorisierung und Priorisierung eines solchen Alerts außerhalb der Arbeit. Diese Einordnung müsste in einem Sicherheitskonzept stattfinden, der die hier entwickelte Möglichkeit verwendet.

3 Konzeption der TAP Implementierung

Um dem Spielraum an Innovation für die ganzen Anwendungsfälle zu ermöglichen, wie im Kapitel 2.3.2 und [TCG-TAP1] beschrieben, wurden keine Entscheidungen für diese Übertragung in den Spezifikationen [TCG-TAP2] vom TAP getroffen. Damit eine Übertragung zwischen Attester und Verifier stattfinden kann, sind die Entscheidungen für eine Implementierung nun notwendig und werden im Folgenden gefällt.

Die über das TAP übertragbaren Nachrichten sind zwar ebenfalls spezifiziert, allerdings nicht, in welchem Kontext diese verschiedenen Nachrichten verwendet werden sollen. Es fehlt eine Beschreibung, wie die verschiedenen Anwendungsfälle im Einzelnen ablaufen sollen, mit welchen Schritten und zu übertragenden Nachrichten die gewünschten Anwendungsfälle umgesetzt werden können.

Daher wird in diesem Kapitel eine genauere Ausarbeitung und Interpretation dieser Spezifikationen stattfinden. Hierbei wird zwar auf das TAP als Ganzes eingegangen, jedoch werden die notwendigen Entscheidungen nur in Hinblick auf den Anwendungsfall "Attester Remeasurement" getroffen. Dies ist notwendig, damit eine Implementierung des Protokolls möglich wird und somit das Ziel erreicht werden kann, ein vertrauenswürdiges Monitoringsystem zu entwickeln.

3.1 Übermittlung des Protokolls

Um im Rahmen des TAP-Protokolls eine Übertragung zu ermöglichen, ist ein Netzwerkprotokoll notwendig. An dieser Stelle sollte sich für ein Netzwerkprotokoll entschieden werden, welches das flexible einsetzbare Internet Protocol (IP) als Grundlage nutzt. Diese Entscheidung muss zwischen den meist verwendeten Netzwerkprotokollen UDP und TCP getroffen werden. Der Unterschied zwischen diesen Protokollen stellt die verbindungslose Kommunikation zwischen mehreren Geräten und die verbindungsorientierte Kommunikation mit Mechanismen zur automatischen Behebung von Datenverlust zwischen zwei Geräten dar.

Die Anwendungsfälle “Unidirektionale Attestation” und “Attestation ohne Sessionmanagement” würden für ein verbindungsloses Netzwerkprotokoll wie UDP sprechen. Falls alle Anwedungsfälle in Kombination ermöglicht werden sollen, wäre eine Festlegung auf allen Parametern notwendig. Für den in dieser Arbeit angestrebten Anwendungsfall des “Attester Remeasurement” würde das zum Beispiel bedeuten, dass Folgendes bereits festgelegt sein muss:

- die Auswahl der PCR-Bänke,
- die hierfür verwendeten Hash-Algorithmen,
- der Attester-Schlüssel für die Signierung des Quotes,
- der hierfür verwendeter Algorithmus
- und die hierfür ausgewählte Schlüsselgröße

Sollten zwischen dem Verifier und dem Attester keine Algorithmen ausgehandelt worden sein, würde dies im Zweifel bedeuten, dass für ein Konsens ohne Aushandeln die älteren Algorithmen genutzt werden. Da dies meist schwächere Algorithmen sind, kann dies zu Sicherheitsproblemen führen.

Für die Umsetzung des Monitoringsystems in dieser Arbeit und damit des “Attester Remeasurement”s wird der Nonce vom Verifier im Monitoringsystem gestellt. Dadurch kann mindestens “Unidirektionale Attestation” als Anwendungsfall nicht gleichzeitig umgesetzt werden und es ist somit eine Umsetzung aller Anwendungsfälle nicht mehr möglich. Mit dem Wegfallen des schwer umzusetzenden Anwendungsfalls des “Attestation ohne Sessionmanagement” durch die Verwendung eines Nonces vom Verifier wird es ermöglicht, dass der Parameter zwischen Attester und Verifier ausgehandelt wird.

Die erwähnten Netzwerkprotokolle stellen nur die Entscheidung für den grundlegenden Mechanismus in der Kommunikation da. Zwischen diesen Protokollen (TCP bzw. UDP) und dem TAP können weitere Protokolle untergebracht werden - dazu zählen unter anderem TLS [RFC8446] für TCP- bzw. DTLS [RFC6347] für UDP-Verbindung für eine verschlüsselte Kommunikation. *Hierauf wird bei der Implementierung während der Arbeit verzichtet, da im TAP keine sensiblen oder sicherheitsrelevanten Daten übertragen werden.*

3.2 Abläufe des Protokolles

In dieser Arbeit wird das in [TCG-TAP1, S. 11] vorgestellte "Attester Remeasurement" implementiert, bei dem die Aktualität (Freshness) vom Attester durch ein Nonce vom Verifier bewiesen werden soll.

Als Vorbedingung der Einsatzmöglichkeit des "Attester Remeasurement" wird vorausgesetzt, dass der Verifier alte PCR-Werte besitzt, die aktualisiert werden sollen. Für das vorgesehene Monitoring wird nun regelmäßig ein Remeasurement ausgeführt, bei dem im Normalzustand die gleichen alten PCR-Werte erwartet werden. Falls sich die PCR-Werte geändert haben, wird das Monitoringsystem einen Alert abgeben, da dieses unerwartete Verhalten einen Sicherheitsvorfall darstellt, dem nachgegangen werden muss.

Das Remeasurement sieht vor, dass der Verifier mit einer Anfrage an den Attester zum aktuellen Zustand des TPM befragt. Diese Anfrage soll dabei aus

- einer Liste von PCR-Bänken,
- einem Indikator für den letzten Stand der PCR-Werte
- und einem Nonce

bestehen [TCG-TAP1, S. 11].

In der Spezifikation [TCG-TAP2] wurden zwar vorgesehene, mögliche Nachrichten festgehalten, allerdings nicht die Abläufe des TAP Protokolls. Die Details, welche Nachrichten zu welchem Zeitpunkt möglich sind, um die verschiedenen Einsatzmöglichkeiten aus [TCG-TAP1] umzusetzen, sollen nachfolgend definiert werden.

In der Spezifikation wurde keine einzelne Nachricht definiert, die alle benötigten Informationen der Anfrage vom Verifier zum Attester übermittelt. Das Ziel der Arbeit ist es, herauszufinden, unter welchen Anforderungen ein vertrauenswürdiges Monitoring möglich ist. Das als Grundlage verwendende TAP-Protokoll ermöglicht mit den vorgesehenen Nachrichten keine direkte Umsetzung. Um eine Einschätzung zu erhalten oder gar Verbesserungen am Protokoll vorschlagen zu können, wird sich möglichst nahe an die Spezifikation der Nachrichten gehalten. Dabei werden die getroffenen Entscheidungen festgehalten, welche Nachrichten wie verwendet werden, um das Ziel zu erreichen.

Ein möglicher Ablauf der verschiedenen einzelnen Nachrichten wird in Abbildung 3.1 dargestellt.

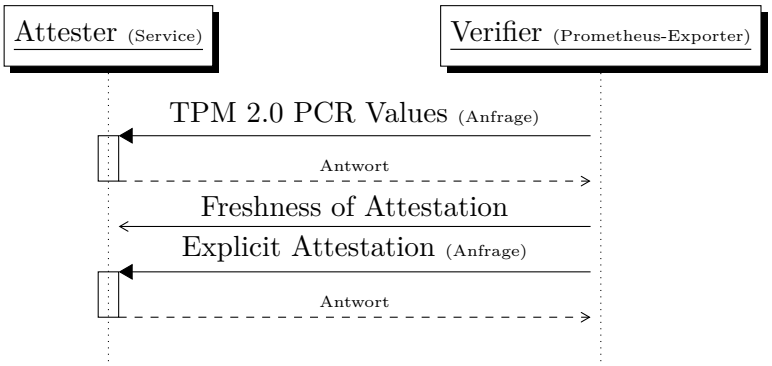

Abbildung 3.1: Detailliertes Sequenzdiagramm für die Einsatzmöglichkeit "Attester Remeasurement" des TAP

Die Nachricht "TPM 2.0 PCR Values" wird zur Übermittlung der Liste von PCR-Bänken mit dem Hashalgorithmus verwendet. Der dort enthaltende *pcrUpdateCounter* kann dabei als Indikator des letzten Standes verwendet werden. In der aktuellen (TCP-)Sitzung zwischen Verifier und Attester wird die Liste von PCR-Bänken für die spätere Erstellung des Quotes im Attester zwischengespeichert. Der Verifier erwartet vom Attester eine Antwort mit den Werten der PCR-Bänke. Der Grund hierfür ist, dass dem Attester zunächst nicht bekannt ist, ob der Verifier mit der Anfrage die Werte der PCR-Bänke erhalten möchte oder - wie hier benötigt - die Anfrage nach den Werten der PCR-Bänke nur für die spätere Verwendung gestellt wurde. Der Verifier hingegen erhält so die Werte der einzelnen PCR-Bänke und kann diese für eine detaillierte Verifikation nutzen. Betrachtet man hierzu im Vergleich eine Anfrage mit nur einem Quote, gibt es nicht die Möglichkeit für einen Rückschluss darauf, in welcher der PCR-Bänke eine Änderung stattgefunden hat. Der Grund hierfür ist, dass es sich bei dem Quote letztlich nur um einen Hash über alle Werte der ausgewählten PCR-Bänke handelt.

Der Verifier sendet daraufhin eine "Freshness of Attestation"-Nachricht an den Attester und teilt ihm hiermit den Nonce mit. Dieser wird in der aktuellen (TCP-)Verbindung beim Attester für die spätere Erstellung des Quotes zwischengespeichert. *Mithilfe dieser Nachricht muss dem Attester nicht direkt ein Nonce vorgegeben werden. Es könnten auch andere Freshness-Verfahren wie zum Beispiel durch 3rd-Party oder Clock-Based eingeleitet werden. Siehe hierzu auch die nähere Beschreibung von "Freshness / Nonce" in Abschnitt 2.3.1. Bei dem geplanten Monitoringsystem wird die Verwendung von weiteren Nachrichten, die zu einer ebenfalls vertrauenswürdigen Nonce führen, nicht näher betrachtet.*

Die Reihenfolge der Nachrichten von “TPM 2.0 PCR Values” und “Freshness of Attestation” mit dem Nonce ist dabei nicht vorgegeben. Dadurch können während der aktuellen Sitzung/Verbindung beide Werte unabhängig voneinander aktualisiert werden. Eine nützliche Variante kann das Auslesen aller PCR-Bänke sein. Da die Datenstruktur *TPML_DIGEST* nur acht PCR-Bänke übertragen kann, ist es möglich, die Liste der PCR-Bänke anzupassen, um sämtliche Bänke nacheinander während einer (TCP-)Sitzung zu erfragen.

Am Ende soll zur Überprüfung, dass keine Manipulation bei der Übertragung und der Aktualität der Werte aus dem PCR-Bänken stattgefunden hat, der Quote vom TPM erstellt werden. Für die Übertragung des Quotes wurde die “Explicit Attestation” Nachricht spezifiziert. Hier gibt es allerdings je nach Einsatz und TPM Version verschiedene “Explicit Attestation” Nachrichten. In der Umsetzung wird nun eine “Explicit Attestation” Nachricht mit nur dem Indikator für die Variante/Subtype verwendet, um den Einsatz “Attester Remeasurement” zu ermöglichen. Dies ist notwendig, da kein spezifiziertes Vorgehen bei der Auswahl der verschiedenen möglichen “Explicit Attestation” Nachrichten vorgesehen ist. Zudem wird auf diese Weise das Problem gelöst, ab welchem Zeitpunkt in der (TCP-)Sitzung der Quote erstellt werden soll. Außerdem lassen sich so innerhalb der (TCP-)Sitzung verschiedene PCR-Bänke jeweils mit einem Quote zum Verifizieren übertragen, ohne erneut ein Nonce auszuhandeln.

Im Folgenden werden die verschiedenen, verwendeten Nachrichten aus Abbildung 3.1, mit einer kurzen Beschreibung, der jeweiligen Übertragungsrichtung sowie des Formats und der Herkunft der Definition noch einmal im Detail aufgelistet. Die Datenstruktur und deren binäre Codierung ist Bestandteil der TSS-Bibliothek. Diese wurde in [TCG-TSS-MU] festgehalten und wird an dieser Stelle daher nicht weiter aufgeführt.

1. **TPM 2.0 PCR Values** - Request
   Beschreibung: Die PCR-Selection wird im Attester der aktuellen TCP-Verbindung zwischengespeichert. Auf diese Weise kann die PCR-Selection für die spätere Erstellung der Quote's verwendet werden.

   Richtung: Attester ← Verifier
   Format: 0x04 ‖ length ‖ (0x00000000 ‖ TPML_PCR_SELECTION)
   Definition: Siehe Response; um die aktuellen PCR-Werte zu erhalten, wird für den *pcrUpdateCounter* der Wert *0x00000000* verwendet. Zudem wird *TPML_DIGEST* hier nicht auf *0* gesetzt und überprüft, wie in der Spezifikation vorgesehen, die darauf passende *length* erleichtert die Erkennung des Request gegenüber dem Response.

2. **TPM 2.0 PCR Values** - Antwort
   Beschreibung: Die ausgelesenen PCR-Werte werden im *TPML_DIGEST* zum Verifier übertragen, der maximal für acht Werte ausgelegt ist.

   Richtung: Attester → Verifier
   Format: 0x04 ‖ length ‖ (pcrUpdateCounter ‖ TPML_PCR_SELECTION ‖ TPML_DIGEST)
   Definition: In [TCG-TAP2, S. 11f].

3. **Freshness of Attestation** (Nonce)
   Beschreibung: Durch diese Nachricht wird ein vom Verifier gewählter Nonce im Attester zwischengespeichert. Dieser wird in der aktuellen TCP-Verbindung für die spätere Erstellung des Quote's abgelegt.

   Richtung: Attester ← Verifier
   Format: 0x06 ‖ lenght ‖ 0x0000 ‖ (NONCE)
   Definition: In [TCG-TAP2, S. 12]

4. **Explicit Attestation** (TPM2-Quote) - Request
   Beschreibung: Die "Explicit Attestation" Nachricht dient zum Erfragen einer der verschiedenen Varianten / Subtypes.

   Richtung: Attester ← Verifier
   Format:: 0x09 ‖ length ‖ 0x04
   Definition: Siehe Response. Für die spätere Implementierung wird die Variante/-Subtype *0x04* verwendet, die in der Spezifikation für ein "TPM2_Quote" vorgesehen ist.

5. **Explicit Attestation** (TPM2-Quote) - Antwort
   Beschreibung: Die "Explicit Attestation" Nachricht für "TPM2_Quote" beinhaltet als Antwort einen Hash über alle ausgewählten PCR-Werte in "TPM2B_ATTEST". Des Weiteren wird mit der Nachricht im "TPMT_SIGNATURE" die Signatur vom "TPM2B_ATTEST" ebenfalls übertragen.

   Richtung: Attester → Verifier
   Format: 0x09 ‖ length ‖ 0x04 ‖ (TPM2B_ATTEST ‖ TPMT_SIGNATURE)
   Definition: In [TCG-TAP2, S. 23f], die Handhabung des Nonce wurde nicht spezifiziert. Die Lösung in der Implementierung ist, dass dieser als *extraData* im *TPM2B_ATTEST* zum Signieren hinterlegt wird.

4 Implementierung

Zur Beweisführung, ob das Trusted Attestation Protocol für ein Trusted Monitoring eingesetzt werden kann, ist geplant, die beiden Softwarekomponenten Attester und Verifier zu entwickeln. Diese beiden Komponenten verwenden zur Kommunikation untereinander das TAP. Der Attester ist dabei als ein Service für die Computer konzipiert und der Verifier als ein Prometheus-Exporter, der im Folgenden genauer beschrieben wird.

Konfiguration

Die Konfiguration der beiden Komponenten, des Attesters / Services und des Verifiers bzw. "TAP Prometheus Exporters" (siehe 4.3), findet durch Parameter auf der Kommandozeile oder mittels Umgebungsvariablen statt. Die Umgebungsvariablen können in einer Datei hinterlegt werden, während die Parameter eine kurzzeitige einfache Änderung der Einstellungen ermöglichen.

Zu den Einstellungen, die so konfiguriert werden können zählen beispielsweise:

- Adresse und Ports, auf denen eingehende Verbindungen erwartet werden.
- TCTI-Anbindung, auf welcher Weise das benötigte TPM erreicht werden kann.
- Wie detailliert die Ausgaben der Komponenten sind.
- Die Adresse bzw. der Speicherort des persistenten Speichers vom Verifier / Exporter.
- Ob Änderungen im persistenten Speicher des Verifiers / Exporters übernommen werden sollen.

4.1 Bibliothek

Als erster Schritt für eine Implementierung wird eine TAP-Bibliothek benötigt, die einzelne Nachrichten als Datenstruktur der Programmiersprache zur Verfügung stellt. Wie bereits in den Grundlagen zum TSS unter 2.2 erwähnt, soll dies in die Programmiersprache Rust umgesetzt werden.

Die für die Nachrichten verwendeten Datenstrukturen sind vom sogenannten Enumerated-Typ, wie es in Listing 4.1 als exemplarischer Auszug dargestellt wurde. Der Enumerated-Typ erleichtert es, die verschiedenen Nachrichten zu unterscheiden. Ein Beispiel für solch eine Fallunterscheidung ist die einfache Switch-Anweisung, die in der Programmiersprache Rust `match` genannt wird. Des Weiteren bietet Rust die Möglichkeit, ein Enumerated-Element mit Inhalt zu versehen.[1] So kann in der Bibliothek für jede TAP-Nachricht, ein Enumerated-Element definiert werden. Dabei wird das Format des Inhaltes der einzelnen TAP-Nachrichten ebenfalls definiert, diese können z.B. die Datenstrukturen aus der TSS-Bibliothek beinhalten.

```
48 pub enum TAPMessage {
49     Version(u8,u8),
       // ...
52     PcrTpm2(u32,TPML_PCR_SELECTION,Option<TPML_DIGEST>),
       // ...
64 }
```

Listing 4.1: Exemplarischer Auszug der TAP-Bibliothek (*src/tap/mod.rs*)

Ein weiterer wichtiger Bestandteil der Bibliothek ist die Serialisierung der TAP-Nachrichten, damit die Datenstrukturen der TAP-Nachrichten über das Netzwerk übertragen werden können. Bei der Serialisierung wird zunächst ein binäres Encoding für die Implementierung der prototypischen Bibliothek verwendet. Für die Serialisierung Traits für Read und Write verwendet, wodurch die Serialisierung der Nachrichten bereits unabhängig vom Übertragungsprotokoll ist. Diese Trait werden von den Rust eigenen Bibliotheken zum Beispiel bei der für das Dateisystem bis hin zu der für TCP-Sitzungen implementiert.

[1] Dieses Typenkonstrukt wird als “Algebraic Data Type” bezeichnet.

4.2 Attester (Service)

Danach wird die erste Komponente des Attesters implementiert, der als Service auf dem zu überwachenden Gerät die eingehenden Verbindungen vom Verifier annimmt und sie mithilfe seines TPM beantwortet. In Rahmen dieser Arbeit wird, wie bereits im 3.1 erörtert, das Netzwerkprotokoll TCP verwendet, bei dem der Service auf eine eingehende Verbindung (engl. Listen) wartet.[2]

Sobald von einem Verifier eine TCP-Verbindung beim Attester eingeht und aufgebaut wurde, wird eine Sitzung angelegt. Die nun eingehenden binären Daten werden durch die bereits erwähnte TAP-Bibliothek interpretiert, sodass die entsprechenden TAP-Nachrichten als Datenstrukturen vorliegen. Nach Vorliegen der Anfrage, in Form der Datenstruktur findet eine Unterscheidung der verschiedenen TAP-Nachricht statt, sodass daraufhin kurze Aktionen ausgeführt werden. Ein Beispiel für solch eine Aktion ist das Abspeichern des Nonce für die aktuelle Sitzung. Die Reihenfolge und Häufigkeit der verschiedenen Anfragen ist nicht vorgeschrieben, für den Attester nicht relevant aber für den Verifier wesentlich. Daher ist die Reihenfolge doch wieder relevant, um diese durch bestimmte Zustände im Attester zu erreichen. Ein Beispiel dafür wäre, dass ohne eine vorherige Übermittlung eines Nonce stattdessen kein Wert verwendet wird. Die meisten durch eine Anfrage ausgelösten Aktionen führen allerdings dazu, dass auch eine Antwort am Ende zurück zum Verifier übermittelt wird.

Am Ende wird die Verbindung und damit die Sitzung durch den Verifier beendet, sobald dieser alle nötigen Information eingesammelt hat.

Im Anhang unter Listing 1 befindet sich die entstandene Hilfeausgabe des Services.

[2] Hier am Rande erwähnt, wurde im Attester-Service die Unterstützung für systemd's "socket-based activation" implementiert, um zumindest beim Booten von kleineren Systemen eine geringere Last zu erzeugen. Die Aufgabe den ListenPort anzulegen übernimmt dabei das Betriebssystem/systemd und der Service wird erst dann gestartet, wenn tatsächlich eine Verbindung eingeht. Im Gegensatz zum "Super-Server" wie in [TS07, S. 89] erklärt, beendet sich der Attester-Service nicht von selbst wieder, sondern wartet auf weitere eingehende Verbindungen.

4.3 Verifier

Die zweite entwickelte Komponente ist der Verifier. Neben den im Folgenden erklärten "TAP Verifier Exporter" wurde zum Testen ein Verifier-Client entwickelt. Dieser Client kann beim Attester ebenfalls ein kleines "Attester Remeasurement" durchführen und gibt seine Ergebnisse auf der Kommandozeile aus.

Hierfür wurde prototypisch eine Verifier-Bibliothek für TAP entwickelt, die von beiden Anwendungen genutzt wird. Im Vergleich mit der TAP-Bibliothek, die hier auch genutzt wird, setzt diese Verifier-Bibliothek einige Einschränkungen. Eine der größten Einschränkungen ist, dass zu Beginn eine etablierte TCP-Verbindung erwartet wird. Des Weiteren benötigt die Verifier-Bibliothek eine TCTI-Anbindung zum TPM, um Nonces zu generieren und um dessen Hash-Funktion zu nutzen.

Prometheus Exporter

Der "TAP Verifier Exporter" bietet Prometheus eine HTTP-Schnittstelle. Diese HTTP-Schnittstelle wird regelmäßig von Prometheus mit der Adresse der verschiedenen Attester aufgerufen. Die vom Exporter übermittelten Daten werden von Prometheus für eine definierte Zeitspanne gespeichert und für eine grafische Aufbereitung zur Verfügung gestellt. Zudem werden die Daten von Prometheus für Alerts ausgewertet und ggf. an einen Alertmanager übermittelt. Den genauen Ablauf zeigt das Sequenzdiagramm in Abbildung 4.1 und wird im Folgende erläutert.

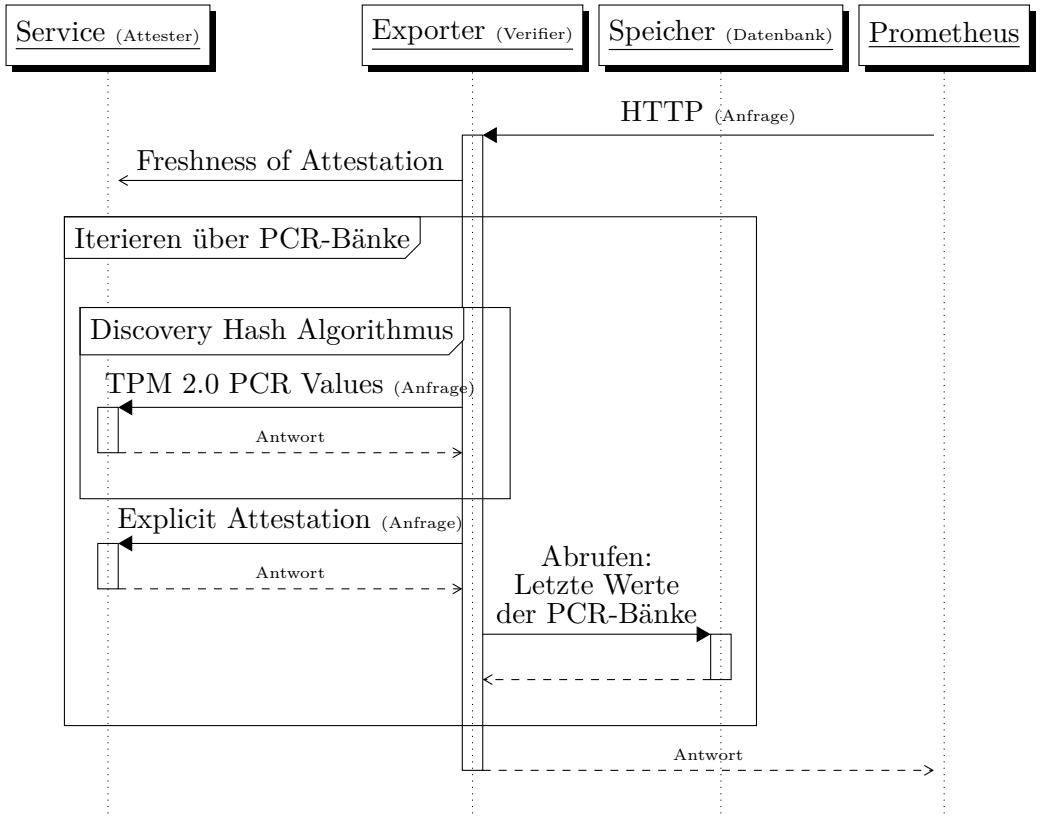

Abbildung 4.1: Detailliertes Sequenzdiagramm des Monitoringsystem

Wenn die HTTP-Anfrage vom Prometheus beim Exporter eingeht, fängt der Exporter an, den Attester zu kontaktieren und seine Antworten auszuwerten. Hierzu wird eine TCP-Verbindung aufgebaut, in der die TAP-Nachrichten ausgetauscht werden. Als Erstes überträgt der Exporter / Verifier mit der TAP-Nachricht “Freshness of Attestation” zum “TAP Service” / Attester. Die “Freshness of Attestation” beinhaltet eine Nonce, der vom TPM des Verifier’s generiert wurde. Diese Nonce wird im späteren Verlauf der TCP-Sitzung für alle Quote’s des “Explicit Attestation” verwendet.

Im nächsten Schritt wird ermittelt, welche Hashalgorithmen das TPM des Attesters unterstützt. Hierzu werden die ersten PCR-Bänke mit der "TPM 2.0 PCR Values"-Nachricht solange mit den verschieden Hashalgorithmen angefragt, bis der Exporter keine leere Antwort sondern eine mit Werten der PCR-Bänke erhält. Um den besten vom Attester unterstützen Hashalgorithmus zu ermitteln, wurde eine Reihenfolge der Algorithmen anhand der Version und Bitlänge zugunsten der Sicherheit festgelegt. Da nicht alle, sondern nur acht PCR-Bänke auf einmal erfragt werden können, muss das Erfragen der Werte und die folgenden Schritte für die restlichen PCR-Bänke wiederholt werden. Für die restlichen PCR-Bänke kann der inzwischen bekannte Hashalgorithmus verwendet werden, sodass dieses Vorgehen zum Finden des Hashalgorithmus nicht mehr notwendig ist.

Nachdem der Verifier die Werte der erfragten PCR-Bänke erhalten hat, kontaktiert er den Attester noch einmal, um nun ein Quote vom Attester zu erfragen. Dabei wird die bestehende TCP-Verbindung weiter genutzt, sodass dem Attester der gewünschte Nonce und die dafür zu verwendenden PCR-Bänke bekannt sind. Hierzu sendet der Verifier, die entsprechende leere "Explicit Attestation"-Nachricht zum Attester, der durch die gleiche Nachricht mit einem Quote und einer Signatur gefüllt vom Attester beantwortet wird.

Nun besitzt der Verifier vom Attester alle Informationen, um den ersten Satz an Werten von PCR-Bänken auszuwerten. Die Auswertung beginnt mit der Überprüfung des Nonce, ob das vom Verifier Erzeugte auch dem im Quote vom Attester entspricht. Daraufhin findet eine Überprüfung der Signatur von der "Explicit Attestation"-Nachricht statt und ob das Quote mit den bekannten öffentlichen Schlüssel signiert wurde. Als Nächstes findet eine Überprüfung der einzelnen Werte aus dem PCR-Bänken statt. Hierzu wird ein Hash mit dem bekannten Hashalgorithmus über die, in der "TPM 2.0 PCR Values"-Nachricht, übermittelten Werte erstellt. Ein Vergleich dieses Hashs mit dem, welcher im Quote enthaltenen ist, gibt Aufschluss, ob den einzelnen Werte ebenfalls vertraut werden kann.

Zuletzt werden die nun vertrauenswürdigen Werte der PCR-Bänke mit den vorherigen im Verifier abgelegten Werten abgeglichen. Auf diese Weise kann bei einer Änderung die genaue PCR-Bank identifiziert werden. Somit wird es gegebenenfalls möglich herauszufinden, warum das System sich anders verhält. Ein Prometheus-Exporter besitzt selbst keinen persistenten Speicher, sodass hierfür eine Datenbank verwendet wird. In unserem Prototyp handelt es sich um eine SQLite-Datenbank, da für die Anbindung der Datenbank ein "Object-relational mapping" (ORM) - Bibliothek[3] verwendet wird, ist eine Erweiterung für weitere Datenbanken leicht realisierbar.

[3]Die verwendete Rust ORM-Bibliothek ist `https://diesel.rs`.

Nachdem dieser Vorgang, wie eingangs erwähnt, für die restlichen PCR-Bänke wiederholt wurde, erstellt der Exporter einen Bericht. Mit diesem Bericht wird die HTTP-Anfrage beantwortet, welches die Überprüfung ausgelöst hat.

Bei der Implementierung wird es eine Einstellungsoption im Exporter geben, die es verhindert bzw. ermöglicht, bei der Feststellung einer Änderung der Werte in den PCR-Bänken, die neuen Werte in den persistenten Speicher zu übernehmen. Durch diese Option entsteht der folgende Vor- und Nachteil. Ein Vorteil ist, dass zukünftige weitere Änderungen ebenfalls bemerkt werden, ohne das eine manuelle Aktualisierung im persistenten Speicher vorgenommen wird. Ein Nachteil dabei liegt in der Struktur eines Prometheus-Monitoringsystems, sodass der einzelne Bericht über eine Änderung der Werte PCR-Bänke nicht mit verarbeitet wird. Der Verlust eines Berichtes vom Exporter kann dadurch entstehen, dass dieser bereits von einem anderen Prometheus-Server eingesammelt wurde oder mit einem Browser abgerufen wurde.

Im Anhang unter Listing 5 befindet sich die entstandene Hilfeausgabe des Exporters.

5 Prototypische Realisierung

Da nun alle Komponenten vorliegen, findet jetzt eine prototypische Integration statt. Damit wird aufgezeigt, das es tatsächlich möglich ist ein Trusted Monitoring mithilfe des TPM durch das TAP aufzubauen. In Abbildung 5.1 wird die entstandene Struktur des Gesamtsystems mit allen Komponenten und die Kommunikation zwischen diesen Komponenten dargestellt. Im Folgenden wird die Abbildung und das Zusammenspiel der Komponenten erläutert.

5.1 Aufbau

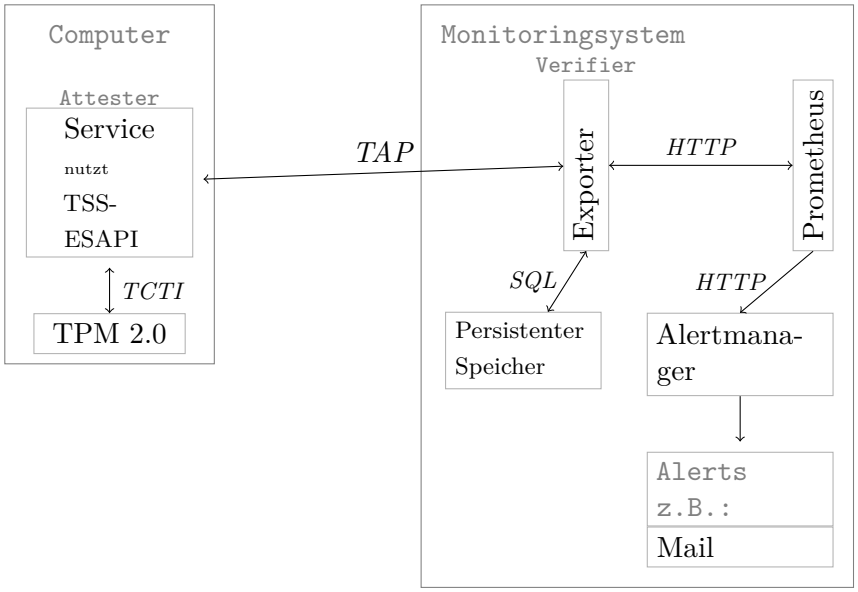

Abbildung 5.1: Die umgesetze Struktur des Monitoringsystems für Computer mit einem integrieren TPM 2.0

Im Folgenden findet eine Auflistung und Erläuterung aller der in Abbildung 5.1 sichtbaren Komponenten statt:

Attester-Service

Der Attester-Service bietet die Möglichkeit eines "Attester Remeasurement" des TPM. Diese Möglichkeit wird durch das TAP über TCP und IP zur Verfügung gestellt. Dafür wird das TPM mittels TCTI angesprochen.

Die dafür verwendete Konfiguration, also die Datei mit Umgebungsvariablen (Listing 2) und die Service-Datei für Linux (Listing 3) befinden sich im Anhang.[1]

Exporter / Verifier

Der Exporter stellt einen HTTP-Webserver dar, der bei einer Anfrage den entsprechenden Attester-Service kontaktiert. Eine solche Anfrage kann wie folgt aussehen: `http://localhost:30910/metrics?target=geno-notebook:30271`. Der Exporter baut anhand der als *target*- Parameter angegeben IP-Adresse und Port eine TCP-Verbindung zum entsprechenden Attester-Service auf. Ein "Attester Remeasurement" findet durch das TAP über die TCP-Verbindung statt, sodass mithilfe des persistenten Speichers die erhaltenen Daten vom Attester ausgewertet werden können. Der detaillierte Ablauf ist bei der Implementierung unter 4.3 erläutert. Am Ende beantwortet der Exporter die HTTP-Anfrage mit der Auswertung in einem für Prometheus verständlichen Format, wie es in Listing 5.1 gekürzt dargestellt ist.

```
1  # HELP tap_up TPM Service is up
2  # TYPE tap_up gauge
3  tap_up{attester="geno-notebook:30271"} 1
4  # HELP tap_version Version of requested tpm
5  # TYPE tap_version gauge
6  tap_version{attester="geno-notebook:30271"} 1.0
7  # HELP tap_quote_nonce validate if quote contain correct nonce (by pcr)
8  # TYPE tap_quote_nonce gauge
9  tap_quote_nonce{attester="geno-notebook:30271",pcr="04",hash_algo="Sha1"} 1
10 # HELP tap_pcr_in_quote validate if pcr values are in quote (by pcr)
11 # TYPE tap_pcr_in_quote gauge
12 tap_pcr_in_quote{attester="geno-notebook:30271",pcr="04",hash_algo="Sha1"} 1
```

[1] Die Datei für die "socket-based activation" befindet sich ebenfalls im Anhang unter Listing 4.

```
13 # HELP tap_pcr_storage_changed TPM Service is up
14 # TYPE tap_pcr_storage_changed gauge
15 tap_pcr_storage_changed{attester="geno-notebook:30271",pcr="04",hash_algo="Sha1"} 0
```

Listing 5.1: Ausgabe des TAP-Verifier Exporters (gekürzt, nur für die PCR-Bank Nr. 04)

Der Bericht beinhaltet verschiedenen Metriken, die meisten liegen durch die Interpretation vom Exporter bereits in binärem Format vor.

Die für den Exporter verwendete Konfiguration, also die Datei mit Umgebungsvariablen (Listing 6) und die Service-Datei für Linux (Listing 6) befinden sich im Anhang.

Persistenter Speicher

Der Exporter greift auf den persistenten Speicher zu, damit dieser die Werte der PCR-Bänke aus vorherigen Attester Remeasurement's verwenden kann. Als persistenter Speicher wird eine Datenbank verwendet, die durch "Structured Query Language" (SQL) angesprochen wird. In der prototypischen Umsetzung wurde dafür die Datenbank-Software die SQLite benutzt.

In der Datenbank werden vom Exporter die folgenden zwei Tabellen benötigt.

Eine Tabelle für den Attester, dem eine ID und PublicKey für die Überprüfung der Signatur des Quotes zugeordnet wird, hier exemplarisch dargestellt.

ID	Attester-Addresse	PublicKey
2	geno-notebook:30271	76B6543B0D1E6A3EC008D...

Eine zweite Tabelle, in der anhand der Attester-ID, der PCR-Bank-Nummer und des Hashalgorithmus, der letzte Wert der PCR-Bank nachgeschlagen werden kann. Im Folgenden wird ein exemplarischer Auszug dieser Tabelle dargestellt.

ID	Attester-ID	PCR-Bank NR	Hashalgorithmus	Data
25	2	0	Sha1	BD5D1207AF2827149E0...
		...		
29	2	4	Sha1	D5BB2656A3440B17200...
		...		

Falls der Exporter unerwartet einen neuen Hashalgorithmus vom Attester erhält, wird dieser vom Exporter in dieser Datenbank hinterlegt. Der neue Hashalgorithmus wird als eine Änderung der PCR-Bank in dem Bericht vom Exporter durch die Metrik tap_pcr_storage_changed (wie in Listing 5.1 erwähnt) Prometheus mitgeteilt.

Prometheus

Die Monitoring-Software Prometheus ist der zentrale Bestandteil des Monitoringsystems. Die Prometheus-Software verwendet das Pull-Verfahren, wodurch ein gleichzeitiger Betrieb auf mehreren unabhängigen Servern möglich wird, was die Redundanz und damit die Ausfallsicherheit des Systems erhöht.

Das Monitoringsystem benötigt einen Zugriff auf eine Liste von Exportern, mit dem die einzelnen Services überwacht werden. Dafür bietet Prometheus verschiedene "Service Discovery"-Methoden, wie beispielsweise das Auslesen eines Kubernetes-Clusters[2]. Bei der Integration für diese Arbeit wurde lediglich eine statische Liste wie in Zeile 37 vom Listing 8 verwendet.

Der TAP-Attester-Service kann von Prometheus nicht wie ein Exporter abgefragt werden. Hierfür wurde der erwähnte TAP-Verifier-Exporter entwickelt. Indem Prometheus intern eine passende URL zusammensetzt, kann er beim Aufruf dieser URL den Service indirekt erreichen. Damit Prometheus die Liste der Attester-Services über diesen Exporter auf diese Weise abarbeiten kann, muss er passend konfiguriert werden. Bei der exemplarischen Konfiguration von Prometheus (Listing 8) wurde dies in den Zeilen 29 bis 35 umgesetzt, wobei in der Zeile 35 die Adresse des Exporters hinterlegt ist.

Des Weiteren ist der Pfad in Zeile 22-23 hinterlegt, unter dem die Dateien für die Auswertung von Alerts hinterlegt sind. Eine solche Datei wird im nächsten Abschnitt zum Alertmanager erläutert. Die Anbindung von Alertmanagern und eine Liste an Adressen wird in der Prometheus-Konfiguration Listing 8 in Zeile 13-19, dargestellt.

Zuletzt soll erwähnt werden, dass es ebenfalls möglich ist, das zeitliche Verhalten von Prometheus zu konfigurieren, wie es in Zeile 2 bis 10 von Listing 8 zu sehen ist. Dabei wird unter anderem festgelegt, wie oft ein Exporter abgefragt wird und wie oft die eingesammelten Daten mit den Regeln für Alerts analysiert werden.

[2]Kubernetes ist eine Software zum automatisierten Bereitstellen, Skalieren und Verwalten von Container-Anwendungen.

Alertmanager

Die Konfigurationen von Aktionen durch Alerts liegen außerhalb des Zieles dieser Arbeit. Daher soll nur der Weg von Alerts zum Alertmanager beschrieben werden und anschließend soll ein kleiner Einblick in die mögliche Handhabung von Alerts erwähnt werden.

Die eingesammelten Metriken des Exporters, werden von Prometheus anhand von Regeln wie beispielsweise den Folgenden analysiert:

```
1 groups:
2 - name: tap.alerts
3   rules:
4   - alert: "[TAP] PCR Value changed (with nr 0-7)"
5     expr: tap_pcr_storage_changed{pcr=~"0[0-7]"} == 1
6     labels:
7       severity: critical
8     annotations:
9       summary: "pcr nr {{ $labels.pcr }} on {{ $labels.attester }} changed"
```

Listing 5.2: Konfiguration von Prometheus für Alerts (gekürzt für nur eine Metrik, für die ersten acht PCR-Bänke)

Die Regel in Zeile 5 besagt, dass die Metrik für die Änderung von Werten der PCR-Bänke 0 bis 7 von sämtlichen Attestern, einen Alert wirft, sobald dort eine Änderung vorliegt. Hierbei wird dieser Alert durch Zeile 7 mit dem Label *critical* und durch Zeile 9 bereits mit einem Text versehen, der daraufhin an den Alertmanager übermittelt wird. Der Alertmanager zeigt den Alert nach Erhalt auf seinem Webfrontend an.

Nun ist es möglich, den Alertmananger so zu konfigurieren, dass er bei Erhalt von Alerts weitere Aktionen ausführt. Die Labels der Alerts können in den Einstellungen genutzt werden, um Unterscheidungen vorzunehmen und so unterschiedliche Aktionen auszuführen. Weitere konfigurierte Aktionen können ausgelöst werden, wenn ein Alert nach einer definierten Zeitdauer immer noch vorliegt. Ein Beispiel: Falls die Ursache eines Alerts nicht zeitnah von einen Bereitschaftsdienst behoben wird, könnte so eine Eskalation dieser Alerts an weitere Personen umgesetzt werden.

5.2 Auswertung

Zur Auswertung lässt sich grundsätzlich sagen, dass die prototypische Integration den Erwartungen entsprechend funktioniert hat. Es wurden Änderungen an den PCR-Bänken, wie in Abbildung 5.2 zu sehen, festgestellt.

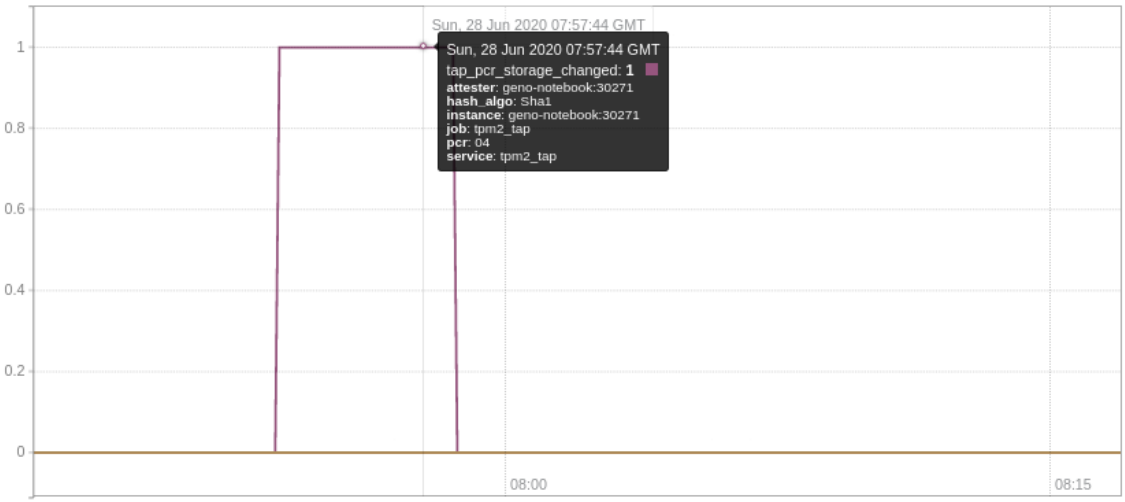

Abbildung 5.2: Ergebnis des Monitoringsystem durch Prometheus visualisiert (Änderung des Wertes einer PCR-Bank)

Das Monitoringsystem hat daraufhin wie erwartet einen Alert gemeldet, dieser ist in Abbildung 5.3 zu sehen.

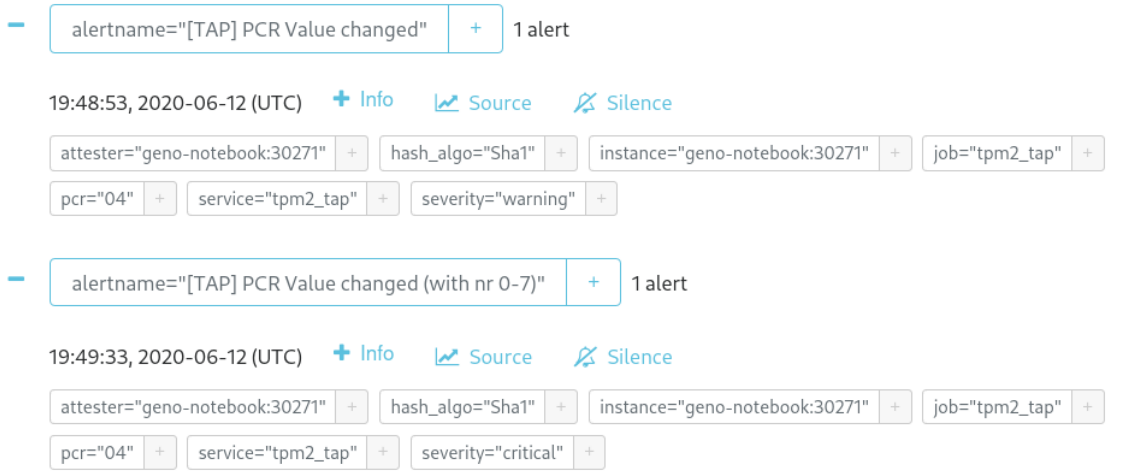

Abbildung 5.3: Alert des Monitoringsystem im Alertmanager, durch Änderung eines Wertes einer PCR-Bank

Beim Auswerten der erhobenen Daten von Prometheus, während der Integration und des Test-Betriebes über 14 Tage im Heimnetzwerk konnte folgendes festgestellt werden:

Es konnten die Ursachen zu verschiedenen Änderungen der Werte in PCR-Bänken rekonstruiert werden. Die Meldung der PCR-Bank Nummer 4 konnte mehrfach festgestellt werden, wie es in Abbildung 5.2 zu sehen ist. Diese Änderung wurde entweder durch ein Update des Linux-Kernels oder das Update des Init-Systems von Linux (hier die Software systemd) verursacht.

Das Ändern von Einstellungen im BIOS führt zu einer Änderung der Werte in den PCR-Bänken Nummer 1 und 2. Während die Verwendung einer anderen Root-Partition zu einer Änderung der Werte in der PCR-Bank Nummer 8 führte. Dabei konnten jeweils andere Ursachen ausgeschlossen werden, wie beispielsweise eine andere Boot-Partition oder andere Versionen vom Bootloader, Linux-Kernel bzw. Init-System.

Diese Erkenntnisse entsprechen den Regeln zur Verwendung der PCR-Bänke, wie im [ACG15, S. 152] beschrieben. Dadurch ist es möglich, die Alerts des Monitoringsystems für einen Anwendungsfall anzupassen. Indem zum Beispiel Kernelupdates nur als Warnungen gewertet werden, während eine Änderung am BIOS als kritisch betrachtet wird und im Bedarfsfall passende Aktionen ausgelöst werden.

Des Weiteren wurde festgestellt, dass Prometheus zum Einsammeln der Daten von einem TPM durchschnittlich 12,7 Sekunden mit einer Standardabweichung von $0,7s$ benötigt, vorausgesetzt es liegt keine Störung im Netzwerk vor. Diese lange Dauer konnte dem TPM und der wiederholten Generierung des Schlüssels für die Signierung des Quote's zugeschrieben werden. In der Implementierung des TAP-Attester-Services können hierfür einige Optimierungen vorgenommen werden.

6 Fazit und Ausblick

In dieser Arbeit wurde demonstriert, dass die Implementierung eines Trusted Monitoring durch das TAP möglich ist. Allerdings ist die Performance des aktuellen Attesters für den produktiven Einsatz zu langsam, um ihn in einem Unternehmen mit einer großen Anzahl an Computern zu verwenden. Die Tatsache des langsamen Attesters ändert jedoch nichts an der Funktionalität und der Sicherheit der Implementierung. Das Vertrauen in das Monitoringsystem entstand dadurch, dass es bereits in das TPM 2.0 gelegt wurde. Durch das TAP reicht nun das Vertrauen vom Attester mit integriertem TPM über den Verifier bis in das Monitoringsystem.

Nachdem im Kapitel 2 zunächst die grundlegenden Begriffe und Funktionsweisen des TPM, TAP sowie vom Monitoring dargestellt wurden, lag ab dem 3. Kapitel der Fokus auf dem konkreten Anwendungsszenario, das dieser Arbeit zugrunde liegt. Dazu wurde ein Konzept zur Implementierung des TAP zur Nutzung des TPM für ein Trusted Monitoring entwickelt. Um die Umsetzung dieses Konzeptes zu demonstrieren, wurde in Kapitel 4 gezeigt, wie die einzelnen Komponenten des Monitoringsystems - der Attester-Service 4.2 und der Verifier-Exporter 4.3 - entwickelt wurden. Besondere Aufmerksamkeit wurde auf die Implementierung einer TAP-Bibliothek gleich zu Beginn des Kapitels 4.1 gelegt. Diese Bibliothek wurde unabhängig von der, für diese Arbeit vorgenommene Eingrenzung implementiert, sodass sie auch für andere spezifizierte Anwendungsfälle [TCG-TAP1] des TAP genutzt werden kann.

Im Versuchsaufbau der prototypischen Realisierung eines Trusted Monitoring mit Hilfe des TPM durch das TAP (Kapitel 5) konnte das Vertrauen exemplarisch für den vom TPM aufgezeichneten Bootvorgang von sämtlichen überwachten Geräten im Monitoringsystem kontrolliert werden.

Dazu mussten kleine Anpassungen an den Nachrichten des TAP, welche in [TCG-TAP2] spezifiziert sind, vorgenommen werden (siehe Kapitel 3.2). Diese Anpassungen erlauben es nun dem Monitoringsystem, den vom Attester unterstützten Hashalgorithmus herauszufinden und “TPM2_Quote”s für die verschieden PCR-Bänke zu erfragen.

Trotz der abstrakten Definition des Anwendungsfalles “Attester Remeasurement” in [TCG-TAP1] von der Trusted Computing Group (TCG) für das TAP, ergab sich nicht genug Spielraum für die Innovation zum Trusted Monitoring ohne die o.g. Anpassungen. Das TAP bietet in seiner Spezifikation keine Möglichkeit, herauszufinden, welche Algorithmen bei einer Anfrage an den Attester verwendet werden könnten. Die verwendbaren Algorithmen hängen dabei nicht vom Attester ab, sondern davon, welche vom TPM unterstützt werden. Damit der Verifier beispielsweise den richtigen Algorithmus bei einer “TPM 2.0 PCR Values“-Anfrage an den Attester verwenden kann, benötigt der Verifier dieses Wissen.

Allgemein fehlt in der Spezifikation für das Protokoll eine Aussage, wie mit unerwarteten Zuständen des TAP umgegangen werden soll. Als Beispiel könnte die Frage auftreten, wie der Attester reagieren muss, wenn Werte aus PCR-Bänken mit einem nicht unterstützen Hashalgorithmus angefragt werden.

In dieser Arbeit wurde dieses Problem in Kapitel 3.2 gelöst, indem das Verhalten des TAP wie folgt definiert wurde: Falls bei der Bearbeitung einer Anfrage ein Fehler auftritt, soll diese Nachricht ohne Inhalt beantwortet werden. Durch diese Definition war eine Anpassung an die Spezifikation für einige Nachrichten notwendig, sodass sie ohne Inhalt versendet werden konnten. Des Weiteren ist es mit dieser Definition gelungen, eine Möglichkeit zu entwickeln, um die vom Attester unterstützten Algorithmen herauszufinden. Beides wird in Kapitel 3.2 detailliert beschrieben.

Anforderungen an die Monitoring-Komponenten

Ein weiteres Ziel dieser Arbeit war es, herauszuarbeiten, welche Anforderungen an die einzelnen Komponenten gestellt werden, damit ein Trusted Monitoring mit dem TPM durch das TAP funktioniert. Bei der Entwicklung des Trusted Monitoringsystems mittels TAP konnten die folgenden Anforderungen an die einzelnen Komponenten ermittelt werden:

Für den Verifier, der in Form eines Prometheus-Exporters entwickelt wurde, wird ein persistenter Speicher benötigt. Er wird für die Speicherung der Werte der PCR-Bänke des Attesters und seinem öffentlichen Schlüssel benötigt. Dadurch kann der Verifier die neu erhaltenen Werte der PCR-Bänke mit den zuletzt erfassten Werten abgleichen, sodass er eine mögliche Manipulation am Computer des Attesters detektieren kann. Der öffentliche Schlüssel des Attesters hingegen wird vom Verifier benötigt, um die vom Attester erhaltene Signatur der Quote's zu überprüfen. Auf diese Weise kann der Verifier feststellen, ob eine Manipulation der Übertragung vom TPM zu ihm selbst stattgefunden hat. Als zweite Anforderung an den Verifier lässt sich festhalten, dass er einen sicheren Zufallszahlengenerator für Nonces benötigt, wofür ebenfalls ein TPM aufseiten des Verifiers verwendet werden kann.

Der Attester benötigt selbstverständlich den Zugriff auf das TPM, um die Anfragen über das TAP beantworten zu können. Des Weiteren benötigt er eine Adresse (und TCP-Port-Nummer), damit er überhaupt vom Verifier kontaktiert werden kann.

Eine weitere wichtige Anforderung an das Netzwerkprotokoll, welches zur Übertragung des TAP genutzt wird, betrifft die beiden Komponenten Attester und Verifier. Damit das Erfragen der einzelnen Werte aus den PCR-Bänken beim "Attester Remeasurement" ermöglicht wird, muss das Netzwerkprotokoll - wie in Kapitel 3.1 festgestellt - verbindungsorientiert sein. Bei der prototypischen Realisierung wurde dafür das Transmission Control Protocol (TCP) verwendet.

Ausblick

Als Ausblick sollen zunächst Ideen für den produktiven Einsatz der Erkenntnisse dieser Arbeit erwähnt werden, bevor auf weitere Einsatzmöglichkeiten der in Kapitel 4.1 entwickelten TAP-Bibliothek eingegangen werden soll.

Für den produktiven Einsatz

Obwohl das Trusted Monitoring funktioniert, kann die aktuell noch schlechte Performance des Attesters bei einer großen Anzahl von zu überwachenden Systemen zu einer erheblichen Belastung des Monitoringsystems führen. Daher ist eine Optimierung zugunsten der Performance für den produktiven Einsatz des Attesters nötig. Einige Vorschläge zur Optimierung sind bereits in Kapitel 5.2 erwähnt worden. Prinzipielle sollte es möglich sein, das TPM effektiver zu nutzen und so einem schnellen, produktiv einsetzbaren Attester zu entwickeln. Während der Recherche für diese Arbeit ist dafür ein ähnliches Softwareprojekt Keylime[1] aufgefallen. Keylime bietet ebenfalls die Möglichkeit von Remote Boot Attestation. Allerdings wurde dabei die gesamte Software des Monitoringsystems selbst entwickelt, ohne das standardisierte Trusted Attestation Protocol (TAP) der TCG zu verwenden. Im Whitepaper zu diesem Projekt [Sch+16] werden unter 5.1 “TPM Operations” Angaben zur Performance bei der Erstellung von Quotes gemacht, die die These untermauern, dass das TPM effektiver genutzt werden kann. Hier könnte sich eine weitere Forschungsarbeit anschließen.

Eine weitere Erkenntnis dieser Arbeit ist, dass die Pflege aller öffentlichen Schlüssel der Attester beim Verifier einen Aufwand darstellt, der im produktiven Einsatz möglicherweise vermieden werden kann. Eine Lösung dafür könnte sein, dass der Verifier bei einem neuen Attester den öffentlichen Schlüssel zunächst erfragt und diesen Schlüssel für spätere Überprüfungen im persistenten Speicher selbstständig ablegt. Dann sollte es durch das TAP möglich sein, den öffentlichen Schlüssel des Attesters mithilfe der TAP Nachricht für Zertifikats-Ketten [TCG-TAP2, S. 9] zu erhalten.

[1] `https://keylime.dev/`

Als letzter Verbesserungsvorschlag soll nun auch ein kritischer Blick auf die Festlegung des Monitoringsystems - Prometheus - geworfen werden. Im Prozess dieser Arbeit kam zutage, dass ein solches System mit einem Pull-Verfahren nicht nur Vorteile hat. Diese Entscheidung ist zwar im Vorfeld der Arbeit getroffen worden, sollte hier im Fazit dennoch kritisch reflektiert werden. Da ein Teil dieser Arbeit sich mit dem Aspekt der Sicherheit beschäftigt, sollte erwähnt werden, dass vor einer Implementierung grundsätzlich eine Risikoabschätzung empfehlenswert gewesen wäre. Dabei hätten beispielsweise die Pull- und Push-Verfahren bezüglich des Risikos von eingehenden Verbindungen gegenübergestellt und gegeneinander abgewogen werden sollen. Das Pull-Verfahren setzt voraus, dass auf allen zu überwachenden Geräten eingehende Verbindungen erlaubt sind. Das Push-Verfahren hingegen benötigt nur beim Monitoringsystem, das potenziell aus einer geringeren Anzahl an Geräten besteht, die Erlaubnis für eingehende Verbindungen. Allerdings könnte das Monitoringsystem als besonders schützenswert gelten, da dort sensiblere Daten von allen Geräten gespeichert werden, sollte daher aus Sicherheitsgründen ggf. nicht direkt kontaktierbar sein. Wiederum sprechen die technischen Gegebenheiten für das Pull-Verfahren. Das TPM bietet nicht die Möglichkeit, über Änderungen durch Interrupts oder andere Benachrichtigungsmechanismen eine Anwendung bspw. die des Attesters zu informieren. Ein Beispiel für eine solche Änderung wäre, dass neue Werte in den PCR-Bänken hinterlegt werden. Daher müssen sowieso regelmäßige Abfragen vorgenommen werden und ist somit eine vollständige Implementierung des Push-Verfahrens nicht möglich, sodass es sinnvoll ist, die Aufgabe - das regelmäßige Abfragen - bis zum Pull-Verfahren auszulagern. Dadurch dass mit dem TPM kein Push-Verfahren für ein Monitoring mit Echtzeitwerten implementiert werden kann, entsteht ein temporärer Verlust an Vertrauen. Der erwähnte Vertrauensverlust besteht nur kurzzeitig zwischen den regelmäßigen Abfragen. Das Vertrauen wird allerdings durch das Abfragen des aktuellen Zustandes rückwirkend für den Zeitraum zur letzten Abfrage wiederhergestellt.

Einsatzmöglichkeit der TAP-Bibliothek

Die anderen Anwendungsfälle aus der Spezifikation des TAP sollten durch die in der Arbeit entwickelte TAP-Softwarebibliothek möglich sein, da in der Bibliothek eine generische Serialisierung genutzt wurde. Einer dieser Anwendungsfälle ist die Authentifikation des Attesters, um diesem den Zugang zu einem Netzwerk "Attester access to a network or compute cluster" [TCG-TAP1, S. 8ff] zu ermöglichen. In diesem Themenbereich gibt es bereits erste wissenschaftliche Arbeiten, deren Ziel es ist, mit Hilfe des Extensible Authentication Protocol (EAP) und dem TPM - TSS den Zugang zu einem Netzwerk abzusichern (siehe [Upp10]). Die Verwendung von TAP über EAP wäre ein realistisches Szenario, das mit der für diese Arbeit entwickelten Bibliothek (siehe Kapitel 4.1) machbar sein sollte.

Literaturverzeichnis

[ACG15] Will Arthur, David Challener und Kenneth Goldman. *A practical guide to TPM 2.0 : using the trusted platform module in the new age of security*. Berkeley, CA: APressOpen, 2015. ISBN: 978-1-4302-6583-2. URL: https://link.springer.com/book/10.1007/978-1-4302-6584-9.

[Bla15] Jim Blandy. *Why Rust?* O'Reilly Media, Inc., 2015. ISBN: 978-1-4920-4858-9. URL: https://www.oreilly.com/library/view/why-rust/9781492048589/.

[Eck18] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Berlin: De Gruyter Oldenbourg, 2018. ISBN: 978-3-11-055158-7.

[Jul17] Mike Julian. *Practical monitoring: effective strategies for the real world*. O'Reilly, 2017. ISBN: 978-1-491-95735-6. URL: http://oreil.ly/2y3s5AB.

[Sch+16] Nabil Schear u. a. *Bootstrapping and Maintaining Trust in the Cloud*. Dez. 2016. DOI: 10.1145/2991079.2991104. URL: https://raw.githubusercontent.com/keylime/keylime/master/doc/tci-acm.pdf (besucht am 27. 04. 2020).

[TS07] Andrew S. Tanenbaum und Maarten van. Steen. *Distributed systems: principles and paradigms*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. ISBN: 0-13-239227-5.

[Tur16] James Turnbull. *The Art of Monitoring*. 11. Juni 2016. ISBN: 978-0-9888202-4-1. URL: https://artofmonitoring.com/.

[Upp10] Hardeep Uppal. *Enabling Trusted Distributed Control with Remote Attestation*. Juni 2010. URL: https://pdfs.semanticscholar.org/f34e/5fb1431b024009c8350c1326f227ef5956e5.pdf (besucht am 14. 02. 2020).

Normenverzeichnis

[ISO 11889-1] ISO/IEC. *11889-1 — Information technology — Trusted Platform Module. Part 1: Overview*. ISO. International Organization for Standardization, 15. Mai 2009. URL: `https://www.iso.org/standard/50970.html` (besucht am 20. 06. 2020).

[RFC3748] RFC. *3748. Extensible Authentication Protocol (EAP)*. RFC 3748. Internet Engineering Task Force (IETF) - Network Working Group, 2004. URL: `https://tools.ietf.org/html/rfc3748` (besucht am 14. 02. 2020).

[RFC6347] RFC. *6347. Datagram Transport Layer Security Version 1.2*. RFC 6347. Internet Engineering Task Force (IETF), Jan. 2012. URL: `https://tools.ietf.org/html/rfc6347` (besucht am 06. 03. 2020).

[RFC7323] RFC. *7323. TCP Extensions for High Performance*. RFC 7323. Internet Engineering Task Force (IETF), Sep. 2014. URL: `https://tools.ietf.org/html/rfc7323` (besucht am 06. 03. 2020).

[RFC768] RFC. *768. User Datagram Protocol*. RFC 768. Internet Engineering Task Force (IETF), 28. Aug. 1980. URL: `https://tools.ietf.org/html/rfc768` (besucht am 06. 05. 2020).

[RFC8200] RFC. *8200. Internet Protocol, Version 6 (IPv6)*. RFC 8200. Internet Engineering Task Force (IETF), Juli 2017. URL: `https://tools.ietf.org/html/rfc8200` (besucht am 06. 05. 2020).

[RFC8446] RFC. *8446. The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Internet Engineering Task Force (IETF), Aug. 2018. URL: `https://tools.ietf.org/html/rfc8446` (besucht am 06. 03. 2020).

[SP 800-108] Elaine Barker u. a. *Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography*. NIST SP 800-108. Apr. 2018. DOI: `10.6028/nist.sp.800-56ar3`. URL: `https://csrc.nist.gov/publications/detail/sp/800-108/final` (besucht am 05. 03. 2020).

[SP 800-56A] L Chen. *Recommendation for key derivation using pseudorandom functions (revised)*. NIST SP 800-56A. Version r3. 2009. DOI: `10.6028/nist.sp.800-108`. URL: `https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final` (besucht am 05. 03. 2020).

[TCG-Algo] TCG. *Algorithm Registry (Family 2.0)*. Techn. Ber. Version v00-r1.27. 7. Feb. 2018. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG-_Algorithm_Registry_Rev_1.27_FinalPublication.pdf (besucht am 28. 02. 2020).

[TCG-Arch] TCG. *Trusted Platform Module Library (Family 2.0) - Part 1: Architecture*. Techn. Ber. Version v00-r1.38. 29. Sep. 2016. URL: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf (besucht am 28. 02. 2020).

[TCG-Brief] TCG. *TPM 2.0 - A Brief Introduction*. Techn. Ber. Juni 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf (besucht am 29. 02. 2020).

[TCG-ESAPI] TCG. *TSS 2.0 Enhanced System Level API (ESAPI) Specification*. Techn. Ber. Version v1.00-r05. 11. Sep. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TSS_ESAPI_v1p00_r05_published.pdf (besucht am 10. 03. 2020).

[TCG-FAPI] TCG. *Feature API (FAPI) Specification - Draft*. Techn. Ber. Version v0.94-r04. 25. Sep. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TSS_FAPI_v0.94_r04_pubrev.pdf (besucht am 15. 03. 2020).

[TCG-PTS] TCG - Infrastructure Working Group. *Platform Trust Services Interface Specification (IF-PTS)*. Techn. Ber. Version v1.0-r1.0. 17. Nov. 2006. URL: https://trustedcomputinggroup.org/wp-content/uploads/IWG-IF-PTS_v1.pdf (besucht am 25. 03. 2020).

[TCG-SAPI] TCG. *TSS 2.0 System Level API (SAPI) Specification*. Techn. Ber. Version v1.1-r29. 6. Aug. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TSS_SAPI_v1p1_r29_pub_20190806.pdf (besucht am 10. 03. 2020).

[TCG-TAB-RM] TCG. *TSS 2.0 TAB and Resource Manager - Specification*. Techn. Ber. Version v1.0-r18. Apr. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TSS_2p0_TAB_ResourceManager_v1p0_r18_04082019_pub.pdf (besucht am 20. 03. 2020).

[TCG-TAP1] TCG. *Trusted Attestation Protocol (TAP) Use Cases - for TPM Families 1.2 and 2.0 and DICE*. Techn. Ber. Version v1.00-r0.35. 5. Nov. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_TNC_TAP_Use_Cases_v1r0p35_published.pdf (besucht am 14. 02. 2020).

[TCG-TAP2] TCG. *Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0*. Techn. Ber. Version v1.00-r0.36. 3. Sep. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TNC_TAP_Information_Model_v1.00_r0.36-FINAL.pdf (besucht am 14. 02. 2020).

[TCG-TCTI] TCG. *TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification*. Techn. Ber. Version v1.0-r18. 24. Jan. 2020. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_TSS_TCTI_v1p0_r18_pub.pdf (besucht am 10. 03. 2020).

[TCG-TSS-MU] TCG - Infrastructure Working Group. *TSS 2.0 Marshaling/Unmarshaling API Specification*. Techn. Ber. Version v1.0-r0.7. 10. März 2020. URL: https://trustedcomputinggroup.org/wp-content/uploads/TCG_TSS_Marshaling_Unmarshaling_API_v1p0_r07_pub.pdf (besucht am 25. 06. 2020).

Anhang

TAP-Attester-Service

Listing 1: Hilfeausgabe des TAP-Attester-Service

```
tap_service 0.1.0
genofire <geno+dev@fireorbit.de>
It is a tap service, which should be run on every monitored

USAGE:
    tap_service [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -l <addr>            listen tap address
                         [env: ADDRESS=]  [default: ::1]
    -p <port>            listen tap port
                          [env: PORT=]  [default: 30271]
    -T, --tcti <tcti>    The TCTI or 'Transmission Interface' is the
                          communication mechanism with the TPM.
                          [env: TPM2TOOLS_TCTI=]
                          [possible values: device, mssim, tabrmd]
```

Listing 2: Datei mit Umgebungsvariablen/Konfiguration für den TAP-Attester-Service

```
1 RUST_LOG=debug
2 ADDRESS=::
3 # PORT=30271
4 # TPM2TOOLS_TCTI=tabrmd
```

Listing 3: Systemd-Service-Datei für das Starten des TAP-Attester-Service

```
[Unit]
Description=TPM2 TAP Service for Remeasurement

[Service]
Type=simple
ExecStart=/usr/local/bin/tap_service
Restart=always
RestartSec=5s
EnvironmentFile=/etc/tpm2-tap.env
Environment=PATH=/usr/bin:/usr/local/bin

[Install]
WantedBy=multi-user.target
```

Listing 4: Systemd-Socket-Datei für "socket-based activation" für den TAP-Attester-Service

```
[Socket]
ListenStream=30271

[Install]
WantedBy=sockets.target
```

TAP-Verifier-Exporter

Listing 5: Hilfeausgabe des TAP-Verifier-Exporters

```
tap_verifier_exporter 0.1.0
genofire <geno+dev@fireorbit.de>
a prometheus exporter as verifier for tap

USAGE:
    tap_verifier_exporter [FLAGS] [OPTIONS]

FLAGS:
    -h, --help                  Prints help information
        --update-changed        update pcr values in storage/database -
                                 exporter would once a change just on time
                                 [env: UPDATE_CHANGED=]
    -V, --version               Prints version information

OPTIONS:
    -l <addr>                                   exporter address
                                                [env: ADDRESS=]  [default: ::]
    -D, --database-url <database-url>           database url to store and detect changes
                                                in the pcr values
                                                [env: DATABASE_URL=]  [default: verifier.db]

    -p <port>                                   exporter port
                                                 [env: PORT=]  [default: 30910]
    -T, --tcti <tcti>                           The TCTI or 'Transmission Interface' is the
                                                communication mechanism with the TPM.
                                                (used random generator and hashing)
                                                [env: TPM2TOOLS_TCTI=]
                                                [possible values: device, mssim, tabrmd]
```

Listing 6: Datei mit Umgebungsvariablen/Konfiguration für den TAP-Verifier-Exporter

```
RUST_LOG=debug
ADDRESS=::1
# PORT=30910
# TPM2TOOLS_TCTI=tabrmd

DATABASE_URL=/var/lib/tap_verifier.db
UPDATE_CHANGED=true
```

Listing 7: Systemd-Service-Datei für das Starten des TAP-Verifier-Exporters

```
[Unit]
Description=TPM2 TAP Exporter

[Service]
Type=simple
ExecStart=/usr/local/bin/tap_verifier_exporter
Restart=always
RestartSec=5s
EnvironmentFile=/etc/tap_verifier_exporter.env
Environment=PATH=/usr/bin:/usr/local/bin

[Install]
WantedBy=multi-user.target
```

Prometheus

Listing 8: Exemplarische Konfiguration von Prometheus (mit statischer Liste von “Targets” statt “Service Discovery”)

```
global:
  # Intervall, bei dem Daten von Exportern regelmäßig eingesammelt werden.
  scrape_interval:     5m

  # Zeitüberschreitung nach der das Einsammeln von Daten
  # eines Exports abgebrochen wird.
  scrape_timeout:      3m

  # Intervall, bei dem die Alert-Regeln regelmäßig überprüft werden.
  evaluation_interval: 10s


# Verbindung zum Alertmanager
alerting:
  alertmanagers:
    - scheme: http
      static_configs:
      - targets:
          - localhost:9093

# Ort, wo Regeln für Alerts gespeichert werden.
rule_files:
  - '/etc/prometheus/alerts/*.yml'

scrape_configs:
  - job_name: 'tpm2_tap'
    # Übersetzung, da der Attester nicht direkt,
    # wie ein Exporter angesprochen werden kann.
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:30910 # Adresse des Verifier-Exporter's
    static_configs:
      - targets: # Liste an Attester-Service's
        - "geno-notebook:30271"
```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht.

Diese Erklärung erstreckt sich auch auf in der Arbeit enthaltene Grafiken, Skizzen, bildliche Darstellungen sowie auf Quellen aus dem Internet. Die Arbeit habe ich in gleicher oder ähnlicher Form auch auszugsweise noch nicht als Bestandteil einer Prüfungs- oder Studienleistung vorgelegt.

Ich versichere, dass die eingereichte elektronische Version der Arbeit vollständig mit der Druckversion übereinstimmt.

Ort, Datum	Unterschrift