Masterarbeit

Verwendung von Remote Attestation in Trusted Network Connect

Mike Steinmetz (mbs)

Juli 2009

Fachhochschule Hannover
Fakultät IV, Abteilung Informatik

Erstprüfer
Prof. Dr. rer. nat. Josef von Helden
Fachhochschule Hannover (FHH)
Ricklinger Stadtweg 118
30459 Hannover
E-Mail: josef.vonhelden@fh-hannover.de

Zweitprüfer
Prof. Dr. rer. nat. Stefan Wohlfeil
Fachhochschule Hannover (FHH)
Ricklinger Stadtweg 118
30459 Hannover
E-Mail: stefan.wohlfeil@fh-hannover.de

Autor
Mike B. Steinmetz (mbs)
Am Dornbusch 14
31275 Lehrte/Ahlten
E-Mail: mike.steinmetz@fh-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Masterarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 1. Juli 2009

Mike Steinmetz

Inhaltsverzeichnis

1 Einleitung

Die homogenen Netzwerke weichen immer mehr den heterogenen Netzwerken. Aktuelle Schutzmaßnahmen wie Firewalls und demilitarisierte Zonen (DMZ) reichen heutzutage nicht mehr aus. Auch wenn diese Techniken sehr ausgereift sind, kaum Sicherheitslücken bieten und ihren Zweck hervorragend erfüllen, sind sie jedoch gegen die neuen Angriffsmethoden relativ machtlos. Für einen Angreifer der in ein Firmennetzwerk eindringen will, ist es teilweise einfacher sich physisch ins Netzwerk einzuklinken oder ein Endgerät (wie z. B. Laptops) des Netzwerkes zu kompromittieren als die Firewall zu durchbrechen. Auch wenn auf den meisten Endgeräten ein Virenscanner und/oder eine Personal Firewall installiert ist, sind diese Systeme jedoch meist unsicherer konfiguriert. Alleine die Tatsache, dass die meisten Benutzer auf ihren Endgeräten mit Administratorrechten arbeiten, macht diese Systeme potentiell gefährdeter. Schon das normale surfen im Internet wird zu einem Sicherheitsrisiko[1]. Auch versuchen Angreifer (nicht unbedingt technisch versierte) Mitarbeiter zu kontaktieren um Zugriff auf Daten/Geräte zu bekommen (Stichwort: Social Engeneering [1]). Ende 2007 z. B. fand ein gezielter Angriff auf die Schweizer Bundesverwaltung statt, bei denen einige Mitarbeiter per E-Mail direkt angesprochen wurden[2].

Das erfolgreiche Angriffe auf Endgeräten auch über tragbare Medien wie USB-Sticks erfolgen, zeigt die Untersagung von mobilen Datenträgern in der US-Armee[3]. Auch Vorfälle wie die Cyberangriffe auf das Bundeskanzleramt[4], Unternehmen wie Rolls-Royce[5] bzw. dem Weißen Haus[6] verdeutlichen wie wichtig der Schutz von Endgeräten ist.

1.1 Gegenstand und Ziele

Ziel dieser Arbeit ist, ein Protokoll zu entwickeln, dass die Messungen im Trusted Network Connect (TNC) fälschungssicher[7] attestiert werden können. Anhand von beste-

[1] `http://www.heise.de/security/Einfallstor-Browser--/artikel/115254`
[2] `http://www.heise.de/security/news/meldung/107715`
[3] `http://www.heise.de/security/news/meldung/119240`
[4] `http://www.heise.de/security/news/meldung/94932`
[5] `http://www.heise.de/security/news/meldung/99912`
[6] `http://www.heise.de/security/news/meldung/118561`
[7] Wenn Fälschungen auftreten, sollten diese erkannt werden.

henden Spezifikationen im TNC Kontext werden Konzepte ausgearbeitet, die eine vertrauenswürdige Attestierung ermöglichen. Anschließend wird ein Konzept exemplarisch umgesetzt. Der Schwerpunkt dabei liegt auf dem bisher noch nicht spezifizierten Übertragungsprotokoll, welches für die Informationen einer Attestierung zuständig ist.

Existierende Technologien, die zum Verständnis dieser Arbeit benötigt werden, sind in Kapitel 2 vorgestellt. Dazu zählt ein kurzer Einblick in das Thema Trusted Computing. Anschließend werden TNC und vergleichbare Techniken, die sich auf dem Markt befinden, vorgestellt.

In Kapitel 3 werden einige wesentliche Probleme der zuvor vorgestellten Techniken analysiert.

Auf Basis der Analyse und vorhandenen Techniken werden vier Konzepte in Kapitel 4 vorgestellt, die eine vertrauenswürdige Remote Attestation im TNC Umfeld ermöglichen.

Eine exemplarische Umsetzung eines Konzeptes findet sich in Kapitel 5. Dabei werden benutze Entwicklungsumgebungen und verwendete Bibliotheken kurz vorgestellt. Anschließend werden die einzelnen Projekte der Implementierung und die entwickelten Nachrichtenformate erläutert.

Im letzten Kapitel wird ein Resümee der Arbeit gezogen.

2 Stand der Technik

Dieses Kapitel gibt eine überblicksartige Darstellung zum Thema Trusted Computing und Network Access Control.

2.1 Trusted Computing und die Trusted Computing Group

Trusted Computing (TC) bedeutet übersetzt „vertrauenswürdige Datenverarbeitung“. Der Begriff Trusted Computing wird inzwischen maßgeblich von der Trusted Computing Group[1] (TCG) verbreitet. Die TCG ist ein Non-Profit-Konsortium aus mehr als 100 (teils namhaften) IT-Unternehmen[2], welche offene und herstellerunabhängige Industriestandards im Bereich TC spezifizieren.

Der Begriff Vertrauen wird i. d. R. gegenüber Menschen (bzw. anderen Lebewesen) verwendet. Da dieser Begriff sich (im ursprünglichen Sinne) nur schlecht auf Gegenstände abbilden lässt, hat die TCG den Begriff Vertrauen (engl. Trust) hierfür definiert:

> Trust is the expectation that a device will behave in a particular manner for a specific purpose. — [2]

Sinngemäß bedeutet dies, dass von einem (Teil-) System erwartet wird, dass es sich für einen bestimmten Zweck in einer definierten Art und Weise verhält. So wird z. B. erwartet, dass ein einfacher Taschenrechner mathematische Gleichungen löst und keine E-Mails verschickt. Dabei sind jedoch folgende Aspekte zu beachten:

1. Dass auch dieses Vertrauen letztendlich auf Menschen beruht, z. B. dem Hersteller bzw. Zertifizierer.

2. Dass die Definition eines Systems nicht zwingend den Anforderungen des „Anwenders“ entspricht. So kann z. B. ein Anwender dem einfachen Taschenrechner vertrauen, jedoch kann dieser Taschenrechner für diesen Anwender „unerwünscht“ sein, da dieser keine komplexen Funktionen berechnen kann.

[1] `http://www.trustedcomputinggroup.org/`
[2] `http://www.trustedcomputinggroup.org/about_tcg/tcg_members`

3. Vertrauen bedeutet nicht, dass das System frei von Fehlern bzw. Sicherheitslücken ist.

Systeme die ihren Zustand bzw. Vertrauen beweisen können, werden als Trusted Computing Platform (TP) bezeichnet und sind ein wesentlicher Bestandteil von TC.

2.1.1 Trusted Platform Module

Um TC in der Realität umsetzen zu können, hat die TCG einen passiven kryptografischen Hardwarechip, das Trusted Platform Module (TPM) [3] spezifiziert. Passiv bedeutet, dass der Chip selbstständig keine Aktionen ausführt bzw. auslöst. Ein TPM stellt verschiedene Funktionalitäten wie z. B. einen (True) Random Number Generator[3] (RNG) und eine Art Key-Storage[4] bereit. Das besondere an diesem Chip ist, dass ein TPM an eine bestimmte Plattform gebunden ist. Dies bedeutet, dass TPM und Plattform untrennbar miteinander verbunden sind. Da das TPM ein Chip ist, der i. d. R. auf das Mainboard gelötet ist, muss dieser gegen einige Hardwareattacken resistent sein. Dies soll verhindern, dass jemand unbefugten Zugriff auf ein TPM bzw. internen Daten enthält.

Im Folgenden sind Funktionalitäten aufgelistet, welche für diese Arbeit relevant sind:

- Einen Random Number Generator zum Erzeugen von Zufallszahlen,
- eine SHA-1 Engine [5] für Hashfunktionalitäten,
- eine RSA Engine [6] zum Ver- und Entschlüsseln von Daten,
- einen Key Generator zum Erzeugen von RSA Schlüsseln und
- mehrere Platform Configuration Register (siehe 2.1.2).

Schlüssel, die das TPM erzeugt bzw. nutzt, sind in Kapitel 2.1.4 näher erläutert.

2.1.2 Platform Configuration Register

Das TPM besitzt in der Version 1.2 mindestens 24 Platform Configuration Register (PCR) [3, Kapitel 39]. Die PCR können den Zustand einer Plattform wiedergeben. Dazu muss zunächst verstanden werden wie ein PCR funktioniert.

Das Register eines PCR ist 160-Bit groß (20-Byte) um exakt einen SHA-1 Hashwert zu speichern. Ein PCR enthält am Anfang, z. B. nach dem Einschalten des Computers,

[3] Ein Zufallsgenerator der „echte“ Zufallszahlen generiert.
[4] Schlüsselspeicher
[5] `http://de.wikipedia.org/wiki/Secure_Hash_Algorithm`
[6] `http://de.wikipedia.org/wiki/RSA-Kryptosystem`

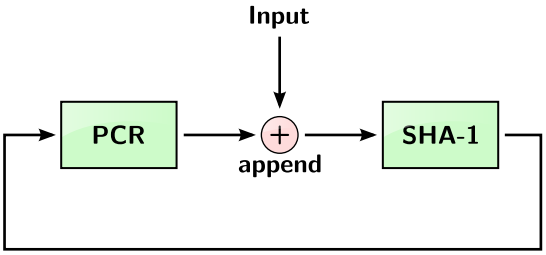


Abbildung 2.1: Algorithmus zum Verändern eines PCR

einen definierten Startwert. Eine notwendige Voraussetzung ist, dass das PCR sich nur über folgenden Algorithmus (siehe Abbildung 2.1) ändern lässt:

1. Der Input ist eine 20-Byte große Zahl (üblicherweise ein SHA-1 Hash).
2. Anschließend wird der Input mit dem PCR konkateniert und
3. der Hashalgorithmus (SHA-1) darauf angewendet.
4. Das Ergebnis des Hashalgorithmus aus dem letzten Schritt, wird dann ins PCR zurückgeschrieben.

Dieses Verfahren sorgt dafür, dass ein PCR nicht mit einem beliebigen Wert direkt beschrieben werden kann und wird als TPM-Extend bezeichnet.

2.1.3 Chain of Trust

Eine Chain of Trust ist eine Vertrauenskette, welche transitives Vertrauen gegenüber einer anderen Instanz ermöglicht. Eine Chain of Trust besteht aus einer Root of Trust Komponente, welcher vertraut werden muss. Wird dieser ersten Komponente nicht vertraut, ist die komplette Chain of Trust nicht vertrauenswürdig. Im Trusted Computing Kontext sind die Teile einer Vertrauenskette Hardware- bzw. Software-Komponenten. Der Aufbau einer Chain of Trust erfolgt nach folgendem Schema:

0. Als erstes wird die Root of Trust Komponente ausgeführt.
1. Die letzte ausgeführte Komponente misst die nächste Komponente (mit dem SHA-1 Algorithmus).
2. Das Messergebnis wird anschließend in ein PCR vom TPM geschrieben (siehe Kapitel 2.1.2).
3. Erst danach wird die gemessene Komponente ausgeführt.

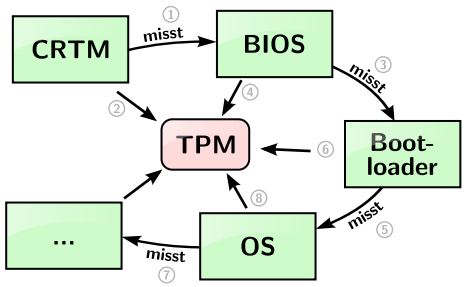


Abbildung 2.2: Chain of Trust Beispiel (Quelle: [4])

4. Der Schritt 1 bis 3 wiederholt sich bis zur letzten Komponente.

Das folgende vereinfachte Beispiel soll die Arbeitsweise einer Chain of Trust verdeutlichen. Auf einer TC Plattform existiert ein Core Root of Trust for Measurement (CRTM), welches auf dem Computer als erstes ausgeführt wird. Dieses misst anschließend den Code vom BIOS[7], speichert diesen Messwert im TPM (genauer im PCR) und führt das BIOS anschließend aus. Das BIOS misst, speichert und führt den Bootloader aus. Der Bootloader führt diese Prozedur mit dem Operating System (OS) durch etc. Wenn eine der Komponenten nicht vertrauenswürdig ist, können alle weiteren Komponenten ebenfalls nicht als vertrauenswürdig eingestuft werden.

2.1.4 Keys

Die TCG definiert eine Menge von asymmetrischen Schlüsselarten für verschiedene Aufgaben. Dabei wird zwischen Migratable- und Non-Migratable-Keys unterschieden.

Migratable-Keys sind Schlüssel, welche nicht an das TPM gebunden sind. Das heißt, dass diese Schlüssel auf ein anderes TPM portiert werden können.

Non-Migratable-Keys sind Schlüssel, die an das TPM gebunden sind. Das heißt, dass die Schlüssel nicht auf ein anderes TPM portiert werden können. Dazu verlässt der private Teil des Schlüssel das TPM niemals bzw. nur verschlüsselt.

Es existieren Schlüsselarten, die sowohl Migratable als Non-Migratable sein können. Jedoch kann ein Non-Migratable-Key nicht zu einem Migratable-Key umgewandelt werden und umgekehrt. Im Folgenden werden die für diese Arbeit wichtigen Schlüsselarten der TCG kurz erläutert.

[7]Basic Input Output System

Signing Key

Ein Signing Key ist ein Schlüssel, der zum Signieren von beliebigen Daten genutzt werden kann. Der Schlüssel kann sowohl ein Migratable- als auch ein Non-Migratable-Key sein.

Storage Key

Mit Hilfe von Storage Keys (SK) kann eine Schlüsselhierarchie [5, Kapitel 4.2.7] aufgebaut werden. Mit einem Storage Key können sowohl beliebige Daten als auch weitere Schlüssel verschlüsselt werden. Ein Storage Key kann sowohl ein Migratable- als auch ein Non-Migratable-Key sein. Eine Ausnahme bildet der **Storage Root Key** (SRK). Dieser Schlüssel bildet die Wurzel (Root of Trust for Storage[8]) der Schlüsselhierarchie und ist ein Non-Migratable-Key, der im TPM gespeichert wird.

Attestation Identity Key

Identity Keys oder auch Attestation Identity Keys (AIK) sind asymmetrische Schlüssel, die TPM spezifische Daten signieren. Zu diesen spezifischen Daten zählen u. a. die PCR-Werte aus Kapitel 2.1.2. Ein AIK ist immer ein Non-Migratable-Key. Ein AIK Credential ist ein Zertifizierter AIK (siehe [5]). Mit Hilfe von einem AIK Credential kann ein Dritter überprüfen, ob der AIK von einem zertifizierten TPM stammt.

Endorsement Key

Der Endorsement Key (EK) ist ein Schlüssel, welcher für jedes TPM nur einmal existiert. Der EK ist ein Non-Migratable-Key und wird im TPM gespeichert, wobei der private Schlüsselteil das TPM niemals verlässt. Durch den öffentlichen Teil kann deshalb ein TPM eindeutig identifiziert werden. Um den Datenschutz zu erhöhen benutzt das TPM den EK weder zum Verschlüsseln noch zum Signieren. Während der Herstellung des TPMs wir der EK erzeugt, jedoch kann ab der TPM Spezifikation 1.2 der EK nachträglich neu generiert werden. Wenn der EK neu generiert wird, werden alle für das TPM erstellten Zertifikate unbrauchbar, da der vorherige EK unwiderruflich gelöscht wird.

2.1.5 Direct Anonymous Attestation und Privacy CA

Um ein AIK zu zertifizieren existieren zwei Möglichkeiten:

1. Über eine Privacy Certification Authority (Privacy CA) und

[8] RTS

2. über das Direct Anonymous Attestation (DAA) Protokoll.

Bei der ersten Methode werden die AIKs von einer Privacy CA signiert. Wird ein neuer AIK benötigt, muss die Privacy CA den neuen Schlüssel signieren. Bei der zweiten Methode über das DAA Protokoll ist es möglich selber „zertifizierte“ AIKs zu erzeugen. Erst ab der TPM Spezifikation 1.2 wird das DAA Protokoll unterstützt. DAA bietet gegenüber der Privacy CA mehr Anonymität. Weitere Informationen zu DAA und Privacy CA finden sich unter anderem in [6].

2.1.6 Platform Trust Service

Der Platform Trust Service (PTS) ist ein Programm, der Komponenten verschiedene Funktionalitäten zu Verfügung stellt. Dieser ist eine Abstraktionsebene oberhalb des TSS (Trusted Software Stack) [7] angesiedelt, mit der andere Komponenten das TPM ansprechen können. Der PTS ermöglicht Komponenten ebenfalls ein Teil der transitiven Vertrauenskette zu werden. Für diese Arbeit wird der PTS u. a. zum Erstellen und Verifizieren des Zustandsreports (siehe Kapitel 4.1.5) genutzt.

Der PTS nimmt Funktionsaufrufe über eine Interprozesskommunikationsschnittstelle (IPC) von „beliebigen“ Prozessen entgegen. Die Funktionen sind in der Platform Trust Services Interface Specification (IF-PTS) [8] spezifiziert.

OpenPTS

Die einzige PTS Implementierung die bisher existiert ist Open Platform Trust Services (OpenPTS) in der Version 0.1.2. OpenPTS ist eine in Java geschriebene Open Source Implementierung eines PTS und kann unter `http://sourceforge.jp/projects/openpts/` heruntergeladen werden. Das Projekt selber bezeichnet sich als Proof of Concept und Referenzimplementierung eines PTS und ist daher nicht für den produktiven Einsatz gedacht. Zwar beherrscht OpenPTS schon Remote Attestation, jedoch wird dafür nicht die TNC-Architektur (siehe Kapitel 2.2.3) verwendet. Auf Grund der Tatsache, dass OpenPTS keine IPC-Schnittstelle bietet, kann die Software für diese Arbeit nicht ohne größeren Aufwand genutzt werden.

2.2 Network Access Control

Network Access Control (NAC) ist ein Ansatz, um die Sicherheit in heterogenen Computernetzwerken zu erhöhen. Dabei wird ein Netzwerk abgesichert, indem sich nur autorisierte Endgeräte zum Netzwerk verbinden können. Es wird neben der Benutzer- bzw. System-Authentisierung auch die Integrität des Endgerätes verifiziert. So kann z. B. überprüft werden, ob die Signatur des Virenscanners aktuell ist oder der Browser

nicht kompromittiert ist. Auf dem Markt existieren gegenwärtig drei NAC Lösungen, die in den folgenden Kapiteln vorgestellt werden.

2.2.1 Cisco Network Admission Control

Das Unternehmen Cisco Systems, Inc.[9] (Cisco) hat als einer der Größten im Netzwerksegment eine eigene proprietäre NAC Lösung im Angebot. Cisco Network Admission Control (Cisco NAC) wird schon seit mehreren Jahren vertrieben und ist bisher die wohl am meisten eingesetzte NAC Lösung. Cisco NAC setzt zwar teilweise auf offene Standards und Techniken wie z. B. 802.1X[10] und VPN[11], jedoch müssen für diese Lösung Hard- und Software Produkte von Cisco verwenden werden.

2.2.2 Microsoft Network Access Protection

Die Microsoft Corporation (Microsoft), eines der weltweit größten Softwareunternehmen, bietet ebenfalls eine NAC-Lösung an. Microsoft Network Access Protection (Microsoft NAP) ist in den Microsoft Produkten ab Windows Vista, Windows XP Service Pack 3 und Windows Server 2008 enthalten. Microsoft setzt dabei überwiegend auf offene Standards wie z. B. IPsec[12], VPN, DHCP[13] und 802.1X. Microsoft NAP setzt für die Übertragung der Nachrichten zwischen den Endgeräten (Client) und Server, auf das eigenentwickelte Statement of Health (SoH) Protokoll. Da Microsoft Mitglied der TCG ist, wurde bei der Interop 2007 in Las Vegas das SoH-Protokoll offengelegt und in die TNC-Spezifikation (siehe nächsten Abschnitt) beigesteuert [10; 11].

2.2.3 Trusted Network Connect

Die TCG spezifiziert ebenfalls eine NAC Lösung, das sogenannte Trusted Network Connect (TNC) [14]. Die TCG setzt dabei unter anderem auf den offenen 802.1X und EAP [15] Standard auf. Damit TNC mit SoH kompatibel ist, existiert seit Mai 2007 eine Alternative im TNC Standard [13].

Die TCG definiert ein Basismodell (siehe Abbildung 2.3) für eine Plattformauthentisierung. Die erste Domäne, **Requestor** genannt, will einen Dienst nutzen, welcher von der dritten Domäne **Relying Party** zur Verfügung gestellt wird. Die zweite Domäne

[9] `http://www.cisco.com`
[10] [9], `http://de.wikipedia.org/wiki/IEEE_802.1X`
[11] Virtual Private Network, `http://de.wikipedia.org/wiki/Virtual_Private_Network`
[12] Internet Protocol Security, `http://de.wikipedia.org/wiki/IPsec`
[13] Dynamic Host Configuration Protocol, `http://de.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol`
[14] `http://www.trustedcomputinggroup.org/developers/trusted_network_connect`
[15] [12], `http://de.wikipedia.org/wiki/Extensible_Authentication_Protocol`

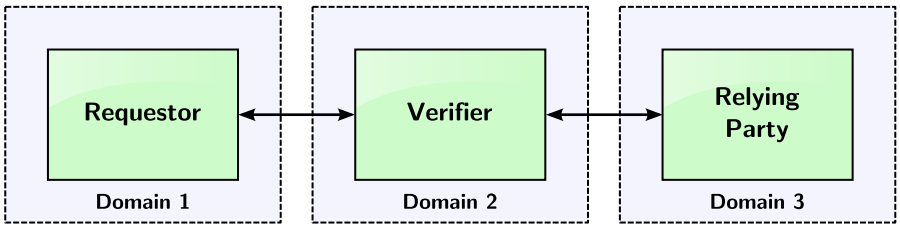


Abbildung 2.3: Basismodell einer Plattformauthentisierung (Quelle: [6])

Verifier führt die Plattformauthentisierung des Requestors durch und teilt der Relying Party das Ergebnis der Plattformauthentisierung mit. Die Relying Party erlaubt oder verwehrt dem Requestor daraufhin den Zugriff auf den angefragten Dienst.

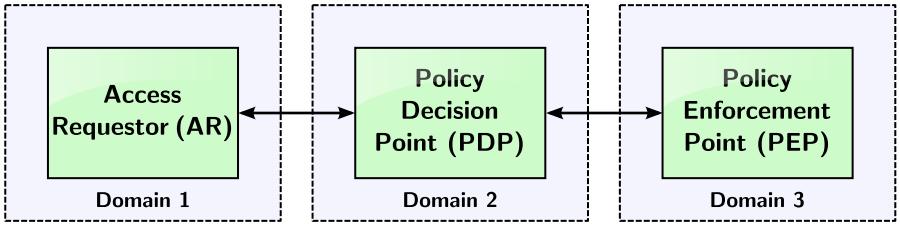


Abbildung 2.4: Basismodell einer Plattformauthentisierung im TNC Kontext (Quelle: [14])

Dieses Basismodell einer Plattformauthentisierung lässt sich im TNC Kontext wie in Abbildung 2.4 abbilden. Der Requestor heißt im TNC Kontext Access Requestor (AR), der Verifier ist der Policy Decision Point (PDP) und die Relying Party ist der Policy Enforcement Point (PEP).

Abbildung 2.5 zeigt die vereinfachte TNC Architektur. Der AR ist das Endgerät (Client), welches sich zum Netzwerk verbinden will. Der PDP ist der Server, der die Integrität und Authentizität überprüft. In einem 802.1X Netzwerk ist der PEP ein Switch und in einem VPN Netzwerk der VPN Server. Der AR besteht zum einen aus dem TNC Client (TNCC) und mehreren Integrity Measurement Collectors (IMC). Auf der Serverseite besteht der PDP zum einen aus dem TNC Server (TNCS) und mehreren Integrity Measurement Verifiers (IMV). Die TNC Architektur ist in drei Schichten aufgeteilt:

- Network Access Layer (NAL),
- Integrity Evaluation Layer (IEL) und
- Integrity Measurement Layer (IML).

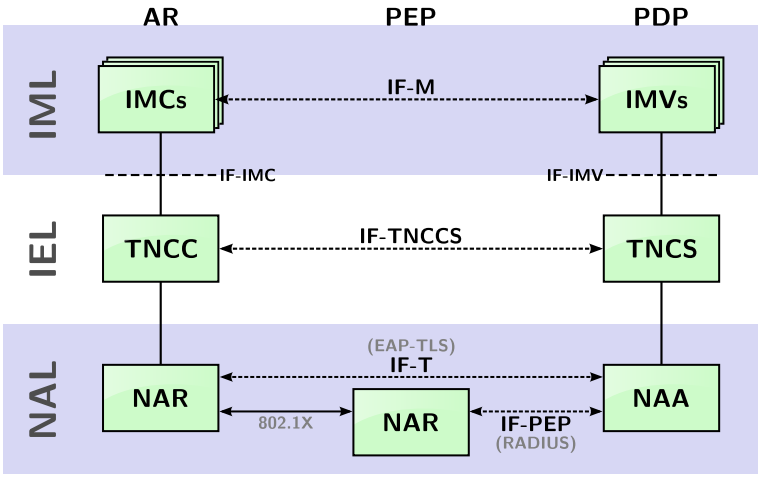


Abbildung 2.5: TNC Architektur (Vorlage: [14])

Die unterste Netzwerkschicht ist der NAL. In dieser Schicht übernimmt das Modul Network Access Requestor (NAR) die Kommunikation des AR. Die Kommunikation des PDP übernimmt das Modul Network Access Authority (NAA). Der NAR kommuniziert mit dem PEP üblicherweise über das 802.1X Protokoll. Der PEP kommuniziert mit den PDP über das IF-PEP (i. d. R. das RADIUS[16] Protokoll). Eine „halbe“ Schicht höher, kommuniziert der AR mit dem IF-T Protokoll [16; 17] (oftmals das EAP Protokoll durch einen TLS[17] Tunnel) über den PEP mit dem PDP.

In der mittleren Schicht IEL, kommuniziert der TNCC mit dem TNCS über das IF-TNCCS Protokoll [18]. Dieses Protokoll realisiert die eigentliche TNC Kommunikation zwischen Client und Server.

Die oberste Schicht ist der IML. Dort tauschen die IMCs mit den IMVs IMC-IMV-Nachrichten aus. Diese Nachrichten enthalten üblicherweise die Messungen des Clients. Da es beliebige viele IMCs bzw. IMVs von beliebigen Herstellern geben kann, hat die TCG die Schnittstellen IF-IMC [19] und IF-IMV [20] spezifiziert. Somit können die IMCs mit dem TNCC und die IMVs mit dem TNCS kommunizieren.

Da diese Arbeit im TNC Bereich auf den Arbeiten [21; 22] aufbaut, wird empfohlen diese Arbeiten zu lesen, da dort die TNC Architektur detaillierter erklärt wird.

[16] [15], `http://de.wikipedia.org/wiki/RADIUS`
[17] Transport Layer Security, `http://de.wikipedia.org/wiki/Transport_Layer_Security`

3 Analyse des Lying Endpoint Problems

Dieses Kapitel befasst sich im Abschnitt 3.1 mit dem Problem der Veränderbarkeit von Software und welche Auswirkung dies auf die NAC Lösungen haben (in den Abschnitten 3.2 und 3.3). Als letztes wird im Abschnitt 3.4 noch ein Angriffsvektor auf ein 802.1X Netzwerk erläutert.

3.1 Manipulationen von Software

Software ist eine Beschreibung von Arbeitsabläufen für einen Computerprozessor. Sie bildet damit den nichtphysischen Funktionsbestandteil eines Computers. Diese nichtphysischen Funktionsbestandteile müssen jedoch auf der Hardware gespeichert werden um zu existieren. Da Software immer auf eine Hardware angewiesen ist, kann Software sich selber nicht effektiv vor Manipulationen schützen. Software lässt sich nur mit Hilfe von Hardware schützen, indem z. B. die Hardware den Zugriff auf die Software für Außenstehende verhindert. Ist es jedoch möglich, die Daten, aus der eine Software besteht, beliebig zu verändern, kann diese Software manipuliert bzw. ersetzt werden. Liegt eine Software im Source Code vor, ist es verhältnismäßig einfach diese zu manipulieren bzw. zu ersetzen. Auch wenn eine Software nur im Bytecode (der so genannte Maschinencode) vorliegt, ist eine Manipulation möglich. Zwar ist dann eine sinnvolle Manipulation erheblich aufwändiger, jedoch nicht unmöglich. Dass solche Manipulationen nicht nur Theorie sind, zeigt eindrucksvoll die „Crackerszene“. Software, welche nur im Bytecode vorliegt, zu manipulieren ist in dieser Szene weit verbreitet um z. B. einen Kopierschutz zu umgehen. So wird kommerzielle Software, welche einen Lizenzschlüssel abfragt z. B. so manipuliert, dass eine manipulierte Version jeden beliebigen Lizenzschlüssel annimmt oder die Lizenzabfrage gar nicht mehr erscheint. Auch wenn sich diese Szene häufig im illegalen Bereich bewegt, existieren auch legale Zusammenschlüsse[1], die als „Hobby“ eigens für diesen Zweck geschriebene Programme „cracken“. Es existieren auch Schadprogramme, welche gecrackte Virenscanner einsetzten, um Konkurrenten auf einem infizierten System aufzuspüren und zu entfernen [23].

[1] `http://crackmes.de/`

3.1.1 Rootkit

Rootkits[2] sind eine Art von Software, die bestimmte Funktionalitäten gegenüber dem Anwender, Administrator oder anderer Software verbergen. Rootkits werden häufig verwendet um Schadprogramme vor Virenscannern zu verstecken oder um zukünftige Einbrüche in ein Computersystem zu verschleiern. Rootkits manipulieren das Betriebssystem oder andere Software um ihr Ziel zu erreichen und sind deshalb auf einem laufenden System relativ schwer zu entdecken. Auf der Black Hat[3] Konferenz 2006 in Europa wurde eine Möglichkeit vorgestellt [24] ein Rootkit im ACPI[4] BIOS zu realisieren, welches sogar eine komplette Neuinstallation des Betriebssystems übersteht.

Microsoft hat in ihren Betriebssystemen Windows Vista und Windows Server 2008 einen Code-Signing-Mechanismus[5] der unter anderem das Ausführen von nicht signiertem Code beim Booten verhindern soll. Auf der Black Hat Konferenz 2007 in Europa wurde jedoch dieser Mechanismus für Windows Vista RC1 und RC2 32-Bit ausgehebelt [25]. Dafür wird ein Programm (VBootkit) noch vor Windows Vista gestartet, welches den Bootprozess von Windows Vista anstößt und dort das ausführen von unsignierten Code erlaubt. Zwar wurde der Code von Windows Vista im Release so verändert, dass VBootkit nicht mehr funktioniert, jedoch ändert dies nicht an dem Grundproblem:

> Wer die Hardware kontrolliert, kontrolliert letztlich auch die Software, die darauf läuft. — [26]

3.2 Analyse von Cisco Network Admission Control

Die Architektur von Cisco NAC ist sehr ähnlich der TNC (siehe Kapitel 2.2.3) Architektur. Auf dem Client wird die Software Cisco Trust Agent (CTA) installiert, welche mit dem Cisco Secure Access Control Server (ACS) kommuniziert. Die Messungen werden beim CTA mit Hilfe von Plugins oder einem Scripting-Interface erstellt. Diese Messungen sollen den Integritätszustand des Clients darstellen.

Ein Problem an diesem Konzept ist, dass der CTA eine reine Software ist, welche manipuliert werden kann. Auf der Black Hat Konferenz 2007 in Europa wurde bereits ein erfolgreicher Angriff auf ein Cisco NAC System demonstriert [27]. Dabei unterscheiden die Autoren des Angriffes zwischen drei Angriffsvektoren:

- Das Ersetzen des CTA durch einen alternativen Client,

[2] `http://www.heise.de/security/Heimliche-Hintertueren--/artikel/38057/0`
[3] `http://www.blackhat.com/`
[4] Advanced Configuration and Power Interface
[5] `http://www.microsoft.com/whdc/winlogo/drvsign/drvsign.mspx`

- das Ersetzen der Plugins, welche die Messungen durchführen und
- das Missbrauchen des Scripting-Interfaces.

Alle drei Angriffstechniken zielen darauf ab, die Integritätsmessungen zu manipulieren um den ACS einen falschen und möglichst policykonformen Integritätszustand vorzutäuschen. Die Problematik hierbei ist, dass der ACS die Korrektheit der Integritätsmessungen nicht verifizieren kann.

Dieses Phänomen wobei der Client absichtlich falsche oder verfälschte Messungen verschickt, wird als Lying Endpoint (zu Deutsch „Lügendes Endgerät“) bezeichnet.

3.3 Analyse von Trusted Network Connect

Da TNC wie Cisco NAC (siehe Kapitel 2.2.1) und Microsoft NAP (siehe Kapitel 2.2.2) in erster Linie dieselben Ziele verfolgt, ist TNC wie Cisco NAC (siehe vorheriges Kapitel) und Microsoft NAP von der Lying Endpoint Problematik betroffen. Zwar existieren dazu in der Spezifikation Lösungsansätze (siehe nächste Kapitel), jedoch sind diese optional. Zudem sind dem Autor bisher keine Implementierung dieser optionalen Lösungsansätze bekannt.

Bei TNC (und Microsoft NAP) kommt noch die Eigenschaft hinzu, dass die Spezifikationen offen sind. Dadurch greift der ohnehin fragliche Sicherheitsgewinn „Security by Obscurity“[6], welcher bei Cisco NAC existiert, nicht. Auch existieren mehrere Open Source Programme, wie z. B. TNC@FHH[7], Wpa_supplicant[8] und XSupplicant[9] welche TNC nutzen. Die Tatsache, dass der Quellcode der Programme vorliegt, vereinfacht das Manipulieren oder Ersetzten dieser Software erheblich. Ein Einbruch in ein TNC gesichertes Netzwerk könnte z. B. wie folgt aussehen:

In ein TNC gesichertes Netzwerk dürfen sich z. B. nur Endgeräte verbinden, bei denen der SSH-Port TCP 22 geschlossen ist. Dafür überprüft der HostScannerIMC aus dem TNC@FHH Projekt, ob der Port 22 auf dem Client (AR) geschlossen ist und sendet anschließend das Ergebnis zum Server (PDP). Ist ein Endgerät mit einem Schadprogramm infiziert, welcher einen verstecken SSH-Server installiert, kann dieses Schadprogramm den HostScannerIMC so modifizieren, dass dieser immer den Zustand eines geschlossen TCP Ports 22 zum PDP sendet. Dadurch bekommt das Endgerät trotz offenen TCP Port 22 Zugang zum TNC gesicherten Netzwerk und der Einbrecher kann sich anschließend zum Endgerät verbinden.

[6] http://de.wikipedia.org/wiki/Security_through_obscurity
[7] http://trust.inform.fh-hannover.de/
[8] http://hostap.epitest.fi/wpa_supplicant/
[9] http://open1x.sourceforge.net/

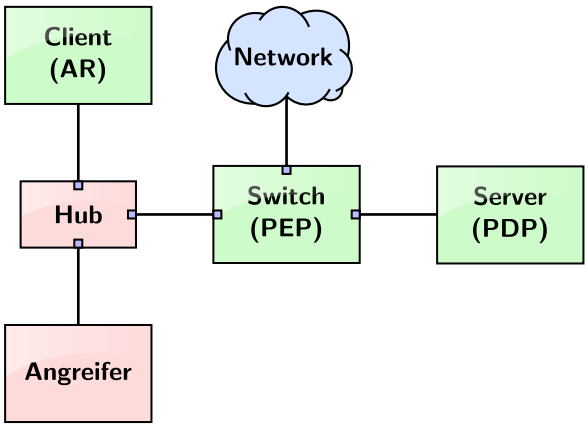


Abbildung 3.1: Angriff auf ein 802.1X Netzwerk.

3.4 Physischer Angriff auf 802.1X

TNC sieht in einem physikalischen Netzwerk standardmäßig 802.1X als Protokoll auf Netzwerkebene vor. Auch Cisco NAC und Microsoft NAP setzten 802.1X ein. Verbindet sich ein Endgerät (AR) physisch mit einem Switch (PEP), z. B. durch das Anstecken in einen Ethernetanschluss, kann der AR nur direkt mit dem PEP über das EAP Protokoll kommunizieren. Hat sich das Endgerät erfolgreich authentifiziert, wird der physische Port freigeschaltet, so dass der AR auch mit anderen Geräten „direkt“ kommunizieren kann.

Eine Schwachstelle ist jedoch, dass der Switch nicht erkennen kann, ob das Endgerät direkt am Port angeschlossen ist oder ein weiteres Gerät zwischen dem Switch und dem Endpoint geschaltet ist und umgekehrt. Abbildung 3.1 zeigt wie solch ein Angriff auf ein 802.1X gesichertes Netzwerk aussehen könnte. Der Angreifer platziert zwischen einem Policykonformen AR und einem PEP einen Hub[10], welcher die Ethernetpakete an alle weiteren am Hub angeschlossenen Geräte sendet. So können der AR und der PEP miteinander kommunizieren, ohne mitzubekommen, dass die Kommunikation über ein Hub geleitet wird. Der Angreifer, der ebenfalls am Hub angeschlossen ist, hat nach der Authentisierung des AR auch Zugriff auf das gesicherte 802.1X Netzwerk.

Da das Verhindern solcher Angriffe kein Teil dieser Arbeit ist, werden solche Arten von Angriffen nicht weiter betrachtet.

[10] http://de.wikipedia.org/wiki/Hub_(Netzwerk)

4 Konzept

In den folgenden Abschnitten werden vier Konzepte vorgestellt, welche Remote Attestation mit Hilfe eines TPM in einer TNC Umgebung ermöglicht. In Abschnitt 4.1 werden Grundlagen erläutert, welche für alle Konzepte zutreffen. In Abschnitt 4.2 bis 4.5 werden anschließend die vier Konzepte erklärt. Im letzten Abschnitt 4.6 werden die Konzepte kurz gegenübergestellt und verglichen.

4.1 Grundvoraussetzung der Konzepte

Die folgenden Abschnitte bilden die Grundlagen der vier Konzepte. Abschnitt 4.1.1 erläutert die notwendige Architektur der vier Konzepte. In Abschnitt 4.1.2 wird die theoretische einfachste Plattformauthentisierung beschrieben. Die Abschnitte 4.1.3 bis 4.1.5 befassen sich mit dem Aufbau und Speicherung des Nachrichtenformates der Plattformauthentisierung. Da einige Konzepte den Datenschutz verringern können, wird im letzten Abschnitt 4.1.6 ein Konzept vorgestellt, wodurch sich die Plattformauthentisierung auf der Clientseite besser kontrollieren lässt.

4.1.1 Ansatz zur Lösung des Lying Endpoint Problems

Die in Kapitel 2.2 vorgestellten NAC-Konzepte sind rein Software basierte Lösungen. Das heißt, dass die Integrität der Messungen auf der Seite des Requestors nicht durch eine Hardwarekomponente sichergestellt wird. Dadurch kann das in Kapitel 3 beschriebene Lying Endpoint Problem nicht effektiv verhindert werden. Ein Lösungsansatz für dieses Problem ist die Verwendung einer Hardwarekomponente, welche die Integrität der Messungen garantieren kann. Einen solchen Ansatz verfolgt die TCG mit dem TPM. Damit ist es mittels einer Chain of Trust (vgl. Kapitel 2.1.3) möglich das Lying Endpoint Problem effektiv zu lösen. Dafür muss beim Requestor die messende Software ein Teil der Chain of Trust sein, damit die Hardwarekomponente (das TPM) die Integrität der Vertrauenskette beweisen kann. Auf Basis dieser Hardware kann der Verifier eine (für ihn) nicht vertrauenswürdige Chain of Trust erkennen.

Für die Umsetzung eines solchen Konzeptes mit einem TPM hat die TCG den PTS (siehe Kapitel 2.1.6) spezifiziert. Um das Lying Endpoint Problem effektiv zu bewältigen, muss der PTS beim Requestor ein Teil der Chain of Trust sein. Das TPM kann durch Signieren der gespeicherten Messungen in den PCR die Integrität der Chain of

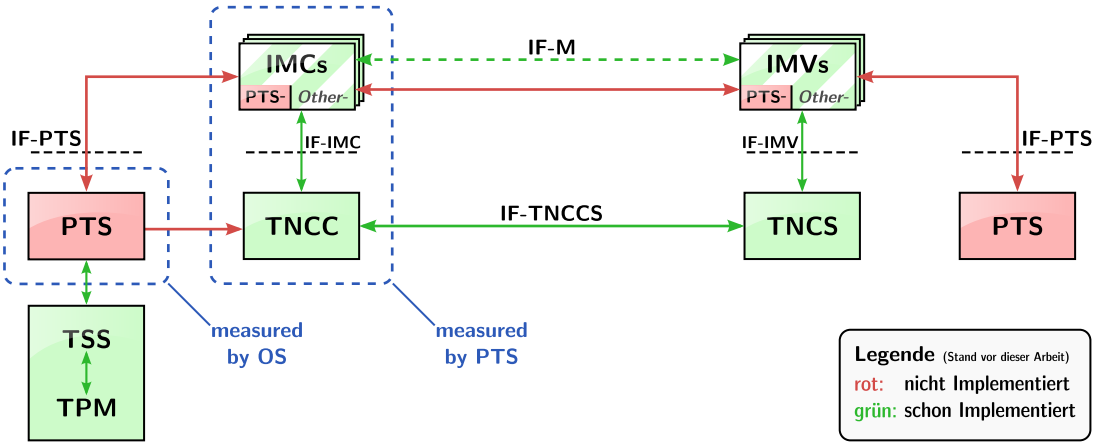


Abbildung 4.1: TNC Architektur mit PTS (Vergl.: [14])

Trust beweisen. Im Rahmen von TNC sieht die TCG eine (optionale) TPM Anbindung durch einen PTS vor. Abbildung 4.1 zeigt die erweiterte TNC-Architektur mit Unterstützung eines PTS.

Chain of Trust

Ein wichtiges Kriterium für eine sichere Vertrauenskette ist, dass jeder Prozess bzw. jedes Programm, welches nicht legitime Manipulationen an der Chain of Trust durchführen kann, ein Teil dieser Kette sein muss. Ist dies nicht der Fall, könnte ein „außenstehender“ Prozess die Chain of Trust nachträglich kompromittieren, indem dieser Prozess z. B. den Speicher eines anderen Prozesses modifiziert, der Teil der Chain of Trust ist. Somit könnte der Requestor dem Verifier einen falschen Zustand belegen. Üblicherweise sind es die Kernel- und Rootprozesse, die eine solche Kompromittierung vornehmen können, welche deshalb ein Teil der Chain of Trust sein müssen. Heutzutage gestaltet sich sowohl die Bildung als auch der Schutz dieser Vertrauenskette äußerst schwierig, zumal in modernen Betriebssystemen häufig eine Menge solcher Prozesse existieren. Ein Lösungsansatz dafür könnte ein gehärteter Kernel wie beispielsweise SELinux[1] sein oder eine Virtualisierungslösung wie sie Turaya[2] verwendet. SELinux z. B. könnte mittels Regeln jeglichen Zugriff auf die Chain of Trust unterbinden, während eine Virtualisierungslösung die Chain of Trust in einer eigenen Umgebung (Compartment) vor Zugriff schützen kann. Ein solches Betriebssystem wird Trusted OS genannt.

Bei einer Chain of Trust ist es ebenfalls wichtig, dass eine Komponente erst ausgeführt wird, nachdem diese gemessen wurde (siehe Kapitel 2.1.3). Auf Grund dieser

[1] Security-Enhanced Linux
[2] `http://www.emscb.de/content/pages/About-Turaya-de.htm`

Vorgehensweise ergibt es wenig Sinn, ein und dieselbe Komponente wiederholt zu messen. Wird ein Prozess bzw. Programm modifiziert, welches ein Teil der Chain of Trust ist, muss diese veränderte Komponente erneut gemessen und ausgeführt werden. Wichtig zu verstehen ist, dass sowohl die Alte als auch die neue Komponente damit ein Teil der Vertrauenskette sind. Sobald der Verifier der alten oder neuen Komponente nicht vertraut, ist für ihn die Chain of Trust nicht mehr vertrauenswürdig, denn:

Angenommen ein Systemdienst wird auf Grund einer Sicherheitslücke aktualisiert, gemessen und neu gestartet. Jedoch kann der Verifier dieser neuen Chain of Trust nicht vertrauen, da der Verifier der alten Version des Systemdienstes nicht mehr vertrauen kann. Es könnte nämlich sein, dass die Sicherheitslücke auf dem System schon vor der Aktualisierung ausgenutzt und die Vertrauenskette kompromittiert wurde.

Dies hat jedoch i. d. R. zur Folge, dass sobald ein Prozess bzw. Programm der Chain of Trust aktualisiert wird, die komplette Vertrauenskette vom Root of Trust erneut aufgebaut werden muss, damit nur noch die aktualisierte Version in der Chain of Trust existiert. In dem Anwendungsfall, dass einige Hardwarekomponenten (inkl. das TPM) die Root of Trust bilden, muss bei einer Manipulation der Chain of Trust das gesamte System neu gestartet werden. Diese Chain of Trust, die als Root of Trust Hardwarekomponenten besitzt, wird im Folgenden als Static Chain of Trust bezeichnet die auf der Static Root of Trust (SRoT) aufsetzt.

Die Static Chain of Trust ist für den TNC-Fall äußerst ungünstig, da sich z. B. durch Remediation[3] die statische Vertrauenskette ändern könnte. Das würde bedeuten, dass das System nach einem Remediation Vorgang ggf. komplett neu gestartet werden muss. Um diesen Effekt zu vermindern, wird eine weitere Chain of Trust gebildet. Diese neue Chain of Trust (im Folgenden Dynamic Chain of Trust) setzt auf der Static Chain of Trust auf. Das heißt, dass ein Teil der Static Chain of Trust die Root of Trust der Dynamischen Vertrauenskette bildet. Diese neue Root of Trust wird im Folgenden als Dynamic Root of Trust (DRoT) bezeichnet. Abbildung 4.2 zeigt den Zusammenhang

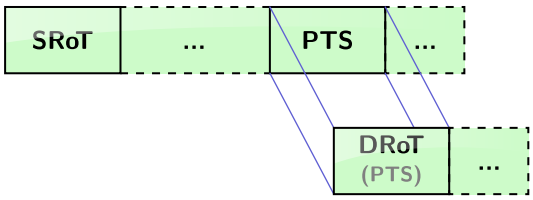


Abbildung 4.2: Eine Static Chain of Trust mit einer Dynamic Chain of Trust

der beiden Chain of Trusts. Die Aufteilung in zwei Chain of Trusts hat den Vorteil, dass bei einer Veränderung der Dynamic Chain of Trust nicht das komplette System neu gestartet werden muss, sondern nur ab der Dynamic Root of Trust.

[3] [14, Kapitel 5]

Damit der Verifier die Integrität der dynamischen Vertrauenskette verifizieren kann, muss er dem Dynamic Root of Trust vertrauen. Da die Dynamic Root of Trust ein Teil der statischen Vertrauenskette ist, genießt die Dynamic Root of Trust ein transitives Vertrauen. Das heißt, dass der Verifier der Dynamic Root of Trust nur Vertrauen schenken kann, wenn der Verifier der Static Chain of Trust vertraut. Vertraut der Verifier der Static Chain of Trust nicht, kann der Verifier (auf Grund des Lying Endpoint Problem) dem Dynamic Root of Trust nicht vertrauen.

Da die dynamische Vertrauenskette (ohne der Dynamic Root of Trust) kein Teil der statischen Vertrauenskette ist, muss der ungewollte Zugriff der Dynamic Chain of Trust auf die Static Chain of Trust verwehrt werden. Da z. B. sonst ein Programm der dynamischen Vertrauenskette die Static Chain of Trust durch eine kompromittierte Version ersetzen kann. Im Gegenzug ist die Static Chain of Trust ein Teil der Dynamic Chain of Trust und kann theoretisch auf die dynamische Vertrauenskette zugreifen.

PTS

Auf der Requestor Seite ist der PTS ein Teil der statischen Vertrauenskette und kann gleichzeitig die Dynamic Root of Trust bilden. Zusätzlich bietet der PTS den Prozessen der Dynamic Chain of Trust verschiedene Funktionalitäten an, z. B. einen beschränkten Zugriff auf das TPM oder um Komponenten zu messen bzw. zu verifizieren.

Der PTS wird üblicherweise vom Trusted OS gemessen und ausgeführt[4]. Es ist jedoch nicht klar spezifiziert, ob der PTS den TNCC misst, oder das Trusted OS diese Aufgabe übernimmt. Nach Meinung des Autors sollte der PTS den TNCC messen. Für diese Meinung spricht:

1. Dies trennt die Zuständigkeit der einzelnen Komponenten besser,
2. das Trusted OS muss nicht den Aufbau des TNCC kennen,
3. dadurch wird der PTS die Dynamic Root of Trust und
4. die Spezifikation deutet an[5], dass der PTS auch die *Measurement Collectors* messen kann.

Auch ist nicht spezifiziert, welche Komponente die Messung der IMCs durchführt bzw. in Auftrag gibt. Der TNCC könnte diese Aufgabe übernehmen da dieser die IMCs ausführt, jedoch wird dadurch der TNCC in die Pflicht genommen, die IMCs zu messen bzw. messen zu lassen. Führt der TNCC diese Messung nicht durch, so stellt dies einen Bruch der Dynamic Chain of Trust dar. Aus folgenden Gründen schlägt der Autor vor, dass der PTS ebenfalls diese Aufgabe übernimmt:

[4] [28, Kapitel 8.2]
[5] [28, Kapitel 8.2 Seite 44]

1. Da die Spezifikation Andeutung zum Messvorgang tätigt (siehe oben) und
2. vorhandene TNCCs müssen nicht angepasst werden, da der PTS diese Aufgabe übernimmt.

4.1.2 Simple Requestor Verification

Die PCR-Werte spiegeln den Zustand einer Plattform wieder (vgl. Chain of Trust in Kapitel 2.1.3). Systeme mit identischen Werten in den PCRs besitzen die gleiche Chain of Trust. Damit ein Verifier bei einem Requestor eine Plattformauthentisierung durchführen kann, benötigt der Verifier vom Requestor:

1. Die relevanten PCR-Werte der Chain of Trust,
2. eine digitale Signatur (erzeugt mit einem AIK) der PCR-Werte und
3. ein Zertifikat zum Überprüfen der Signatur (AIK Credential[6]).

Um Replay-Attacken zu verhindern, fließt in die digitale Signatur noch eine zufällige Zahl (Nonce) des Verifier ein. Damit der Verifier eine erfolgreiche Plattformauthentisierung des Requestors durchführen kann, müssen folgende Punkte erfolgreich überprüft werden:

1. Das Zertifikat (AIK Credential) ist vertrauenswürdig,
2. die Signatur der PCR-Werte ist korrekt und
3. die PCR-Werte stimmen mit einer Liste von vertrauenswürdigen Zuständen (Referenz-PCR-Werte) überein.

Aus diesen Informationen ergibt sich ein Kommunikationsablauf, den Abbildung 4.3 zeigt. Folgende Schritte sind dabei zu beachten:

1. Schritt (Requestor ← Verifier) Der Verifier generiert zum Schutz gegen Replay-Attacken eine Zufallszahl und sendet diese zum Requestor:

- zufällige Zahl (Nonce)

2. Schritt (Requestor → Verifier) Der Requestor lässt die PCR-Werte mit der zufälligen Zahl vom TPM signieren. Anschließend sendet der Requestor folgende Daten (*QuoteData*) zurück zum Verifier:

- Die PCR-Werte der Chain of Trust, mit jeweils:
  - Der PCR-Nummer und

[6] Attestation Identity Key Credential wurde in Kapitel 2.1.4 behandelt.

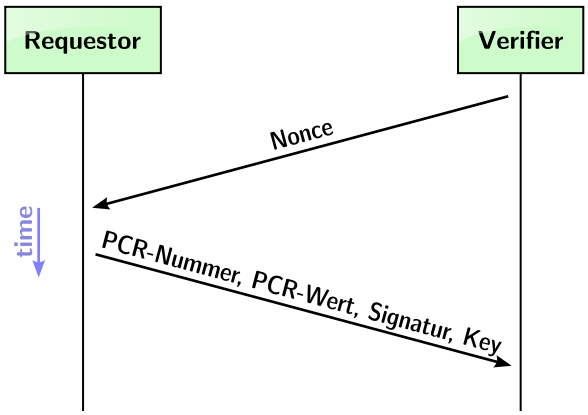


Abbildung 4.3: Kommunikation einer Verifizierung zwischen Requestor und Verifier

  - dem PCR-Wert.
- die digitale Signatur der PCR-Werte (inkl. Nonce),
- die Signaturmethode (z. B. RSA mit SHA-1),
- das AIK Credential in Form eines speziellen X.509 Zertifikats und
- *bei Verwendung von DAA (siehe Kapitel 2.1.5) auch das IKEY*[7]-Certificate.

3. Schritt Der Verifier führt anhand dieser Informationen eine Plattformauthentisierung durch (siehe oben).

4.1.3 Integrity Measurement Log

Das Integrity Measurement Log (IML), welches von der TCG spezifiziert wird [28], ist ein Log in dem die Veränderung der PCRs protokolliert werden. Dort wird auch gespeichert, auf was sich die Messungen im PCR beziehen, z. B. welche Komponenten dafür gemessenen wurden. Anhand dieses Protokolls ist es möglich, die PCR-Werte nachzuvollziehen.

Jedoch ist an keiner Stelle spezifiziert, wo und wie letztendlich das IML gespeichert wird. Da das IML nur für die Reproduktion der PCR-Werte benötigt wird, kann es ungeschützt abgespeichert werden. Wird das IML kompromittiert, ist das System nicht zwangsläufig kompromittiert. Im schlimmsten Fall kann der Verifier den Requestor nicht authentisieren. Dennoch sollte das IML vor unerlaubten Zugriff geschützt werden, schon alleine um einen DoS[8]-Angriff zu erschweren.

[7] Issuer Key
[8] Denial of Service

Die wohl einfachste Möglichkeit das IML zu speichern, ist das komplette IML in eine XML[9]-Datei zu schreiben, denn die TCG spezifiziert ihr Integrity Schema ebenfalls in XML[29]. Da es mehrere Prozesse geben kann, die zeitgleich schreibend auf das IML zugreifen, ist diese 'ein'-Datei Lösung denkbar ungeeignet. Um Komplikationen der Parallelität zu vermeiden, schlägt der Autor vor, dass ein PCR nur von einem Prozess beschrieben werden sollte. Dieser Vorschlag wird dadurch unterstützt, dass im Falle mehrerer PTS-Instanzen, jeder PTS einen exklusiven Zugriff auf ein anderes zurücksetzbares PCR erhält[10]. Dadurch kann das IML pro PCR in eine Datei abgelegt werden, ohne dass es zu Komplikationen kommt. Die Spezifikation definiert, dass der Messwert des PTS vorzugsweise in ein nicht zurücksetzbares PCR abgelegt wird, damit der PTS ein Teil der Static Chain of Trust wird. Gleichzeitig empfiehlt die Spezifikation[11], dass der gemessene PTS in das PCR#22 gespeichert werden sollte. Wohl wissentlich, dass das PCR#22 in der aktuellen PC Client Plattform Spezifikation als zurücksetzbar deklariert ist. Dieser Widerspruch könnte darin begründet sein, dass die TCG eine erste PTS-Implementierung nicht unnötig erschweren will. Auch empfiehlt die Spezifikation, dass der PTS selber den zurücksetzbaren PCR#23 (im folgenden PCR#PTS) für seine Messungen nutzt (welches nicht der Spezifikation widerspricht). Sobald ein PCR zurückgesetzt wird, muss auch das dazugehörige IML gelöscht werden. Das PCR#PTS wird nach der Beendigung des PTS i. d. R. nicht mehr benötigt und kann zurückgesetzt werden. Deshalb sollte das dazugehörige IML nach dem Zurücksetzen des PCR#PTS gelöscht werden.

Die Struktur des IML wird von der TCG nicht genauer spezifiziert. Da ein IML im Rahmen dieser Arbeit nicht umgesetzt wird, wird hier auch keine Struktur für das IML festgelegt.

4.1.4 Snapshot

In einem PCR werden die Ergebnisse der Messungen indirekt gespeichert (siehe Kapitel 2.1.3). Jeder PCR hat einen Startzustand *(starting state)*, ein oder mehrere Zwischenzustände *(intermediate states)* und einen Endzustand *(ending state)*. Ein Snapshot bildet i. d. R. eine Messung und somit einen solchen Übergang dieser Zustände ab. Dadurch können Snapshots den Verlauf der PCR-Werte als eine Kette wiedergeben (siehe Abbildung 4.4).

Die TCG spezifiziert ihre Komponenten in verschiedenen Spezifikationen. Häufig wird das Zusammenspiel der Komponenten in eigenen Spezifikationen definiert. Solche Spezifikationen beinhalten i. d. R. das Wort *Architecture* im Titel. In den anderen Spezifikationen werden implementierungstechnische Details spezifiziert. Diese Spezifikatio-

[9] Extensible Markup Language
[10] [8, Kapitel 3.9]
[11] [8, Kapitel 3.9]

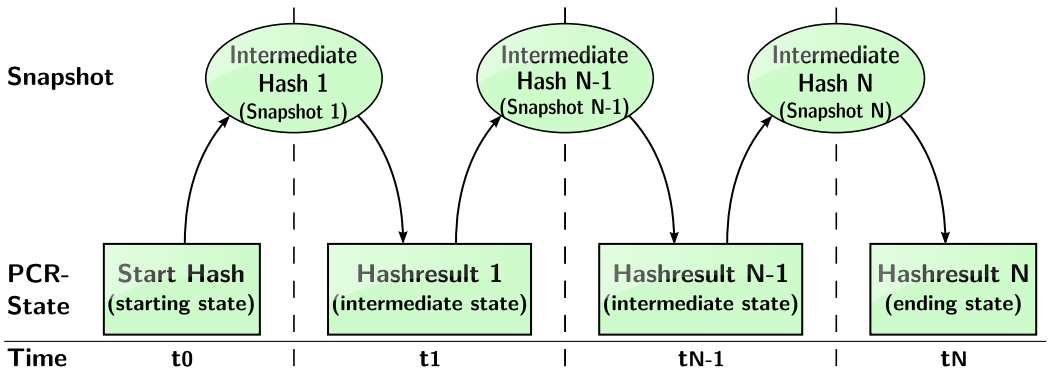


Abbildung 4.4: Verlauf eines PCRs anhand von Snapshots (Vorlage: [28, Seite 27])

nen weisen jedoch im Bereich der Snapshots teilweise unterschiedliche Granularität auf. In der Spezifikation [30] wird formal festgelegt (per XML-Schema[12]) wie ein Snapshot aufgebaut ist. Hingegen wird in der Spezifikation [28] die Nutzung der Snapshots im Rahmen von TC erklärt. Leider ist der Zusammenhang der Snapshots in den genannten Spezifikationen teilweise nicht ersichtlich. Beim XML-Schema, welche die Struktur eines Snapshots definiert, existieren viele optionale Elemente. Das führt dazu, dass es mehrere gültige[13] aber zueinander inkompatible Snapshots geben kann. In den Architektur Spezifikationen wird auch nicht darauf hingewiesen, wie eine Beziehung zwischen mehreren Snapshots aufgebaut werden sollte. Anhand einer genaueren Analyse der formalen Spezifikation [30] hat der Autor den folgenden Ansatz entwickelt. Aufgrund der Komplexität und fehlenden Übersicht einiger Spezifikationen, konnte dieser Ansatz nicht vollständig gegenüber der Spezifikation verifiziert werden.

Es gibt 2 Arten von Snapshots:

Normal Snapshot Ein normaler Snapshot (im folgenden Normal Snapshot genannt) bildet eine Messung ab. Er enthält Informationen über die Messungen und das Messergebnis in Form einer Hashsumme, die dort *CompositeHash* bezeichnet wird (siehe weiter unten).

Synchronized Snapshot Ein synchronisierter Snapshot (im Folgenden Sync Snapshot genannt) bildet den Verlauf eines PCRs ab. Es enthält sowohl den Initialisierungswert als auch den Endwert des PCRs. Auch sollte der Sync Snapshot eine Liste von Normal Snapshots enthalten, die zum Nachvollziehen des Verlaufes erforderlich sind. Diese genannten Informationen werden dort als *PcrHash* bezeichnet (siehe weiter unten).

[12] siehe `http://de.wikipedia.org/wiki/XML-Schema` und [31]
[13] Nach dem XML-Schema gültig.

Auf Basis der Spezifikation der Snapshots [30] und den hier vorgestellten Ansätzen, muss ein Snapshot unter anderem folgendes enthalten:

Id Eine eindeutige ID in dem XML-Dokument. Mit Hilfe dieser ID können andere Elemente des XML-Dokumentes auf diesen Snapshot verweisen.

RevLevel Ein Revisionszähler zur Unterscheidung von verschiedenen Revisionen mit derselben ID. Eine ID aber darf laut W3C XML-Schema-Spezifikation [31] [14] nur einmal in einem XML-Dokument vorkommen. Auch wird in der Architektur-Spezifikation kein solcher Zähler erwähnt. So konnte der Sinn dieses Attributs nicht nachvollzogen werden. Da dieses Attribut jedoch laut Schema-Spezifikation erforderlich ist, beinhaltet dieses Attribut i. d. R. "0".

UUID Eine allgemeine eindeutige ID (Universally Unique Identifier[15]). Der Aufbau einer UUID ist im RFC 4122 [32] spezifiziert. Diese UUID erlaubt es, einen Snapshot systemübergreifend zu referenzieren.

ComponentID Messungen von verschiedenen Softwarekomponenten müssen ggf. unterschiedliche Verifizierungsprozesse durchlaufen. Das heißt, dass Snapshots von unterschiedlichen Softwarekomponenten unterschiedlich behandelt werden (*management process*). Der Management Prozess wird an Hand dieser komplexen ID (*ComponentID*) ermittelt. Diese ID beinhaltet i. d. R. Informationen zu der Softwarekomponente, die diese Messung beauftragt hat. Wenn dieser Snapshot ein Sync Snapshot ist, enthält die ComponentID Informationen über die Software, die diesen Snapshot konstruiert hat (d. h. die ComponentID des PTS).

DigestMethod (optional $0 \ldots \infty$) Enthält Informationen zu einem Algorithmus zum erstellen von Hashsummen. Wenn irgendwo im XML-Dokument der hier beschrieben Algorithmus verwendet wird, kann diese Methode (*DigestMethod*) dort referenziert werden. Da laut TPM-Spezifikation 1.2 [3, Kapitel 4.1] der SHA-1 Algorithmus der einzige Hashalgorithmus ist, den ein TPM unterstützen muss, wird normalerweise nur ein DigestMethod Element für SHA-1 im XML-Dokument existieren.

Values (optional $0 \ldots \infty$) Dieses komplexe Element enthält beliebige zusätzliche Informationen dieser Messung. Dieses Element wird benutzt, wenn die ComponentID alleine zur Verifizierung nicht mehr ausreicht. So kann dieses Element z. B. die Namen der gemessenen Dateien beinhalten. Dieses Element kann eine beliebige XML-Struktur beinhalten und ist von der TCG nicht genauer spezifiziert.

[14] `http://www.w3.org/TR/2000/WD-xml-2e-20000814#NT-TokenizedType`
[15] `http://de.wikipedia.org/wiki/Universally_Unique_Identifier`

SubComponents (optional $0 \ldots \infty$) Es ist unter Umständen sogar sinnvoll einen Snapshot in mehrere Snapshots aufzuteilen. Daher kann ein Snapshot wiederum aus beliebigen Snapshots bestehen. Dieses Element *SubComponents* referenziert jeweils auf einen anderen Snapshot. Da die TCG den Umgang von SubComponents nicht spezifiziert, wird ein ähnliches Prinzip von SubComponents weiter unten definiert.

CompositeHash Um die Kette eines PCR nachzuvollziehen (siehe Abbildung 4.4), wird für einen Snapshot (nicht Sync Snapshot) die Hashsumme benötigt, mit der der PCR-Wert erweitert wurde. Unter anderem enthält dieses Element eine Referenz auf den verwendeten Hashalgorithmus (*DigestMethod*). *CompositeHash* ist kontravalent (xor) zu *PcrHash*.

PcrHash Dieses Element sollte nur in einem Sync Snapshot vorkommen, denn es enthält Informationen zur Kette (siehe Abbildung 4.4). Dazu gehört unter anderem der Start- und Endzustand der Kette, sowie die Referenz zum verwendeten Hashalgorithmus (*DigestMethod*) als auch ein Flag, welches aussagt, ob dieser PCR ein zurücksetzbares Register ist. Dieses Element kann auch die Reihenfolge der Snapshots zum Aufbau der Kette enthalten. Solch eine Zuordnung zwischen Sync Snapshots und Normal Snapshots ist laut der XML-Spezifikation optional. Das weglassen dieser Zuordnung ist jedoch nicht empfehlenswert, da eine solche Zuordnung ohne diese Information aufwendig ist. *PcrHash* ist kontravalent (xor) zu *CompositeHash*.

Eine alternative Interpretation zum Aufbau eines Snapshots, findet sich im Anhang A.

Sub-Snapshot

Um Snapshots fein granularer zu beschreiben, spezifiziert die TCG SubComponents (siehe weiter oben). Da die Spezifikationen [28; 29; 30] *SubComponents* nur unzureichend spezifizieren, werden diese für diese Arbeit nicht verwendet. Stattdessen wird ein Mechanismus verwendet, welcher

- ähnliche Funktionalität wie die SubComponents bietet,
- jedoch ohne das entsprechende SubComponent XML-Element auskommt,
- die Redundanz verringert und
- weiterhin Spezifikationskonform ist.

Die Spezifikationen lassen jedoch eine Menge verschiedener Interpretationen zu. Für diese Arbeit und Umsetzung wird im Folgenden der Ansatz definiert, welcher Spezifikationskonform ist. Dieser Ansatz wird im folgenden als Sub-Snapshots bezeichnet.

Ein Sub-Snapshot ist ein Normal Snapshot (Sub-Snapshot => Normal Snapshot), der einem Normal Snapshot untergeordnet ist.

Sync Snapshots enthalten keine Sub-Snapshots. Zwar wird in *PcrHash* beim Sync Snapshot auf andere Snapshots referenziert, jedoch werden diese nicht als Sub-Snapshots bezeichnet.

Normal Snapshots können Sub-Snapshots enthalten und werden dann Snapshot Parent genannt. Sobald ein Normal Snapshot Sub-Snapshots enthält, setzt sich der *CompositeHash* des Snapshot Parent, welches die Sub-Snapshots enthält, wie in Abbildung 4.5 zusammen.

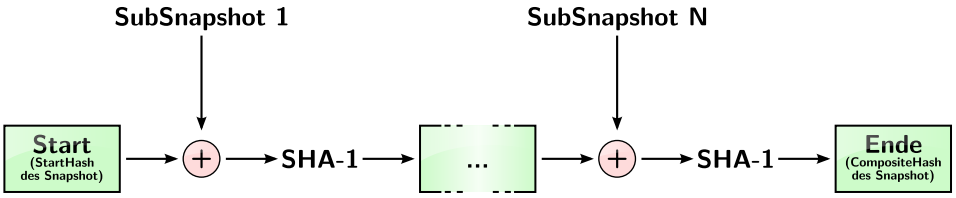


Abbildung 4.5: Zusammensetzung des CompositeHash eines Snapshot Parent.

Diese Berechnung basiert auf der Arbeitsweise eines PCRs:

1. Wie bei einem PCR fängt die Berechnung mit einem StartHash des Snapshot Parent an.
2. Anschließend wird der *CompositeHash* des nächsten Sub-Snapshots der aktuellen Hashsumme angehängt.
3. Anschließend wird daraus die neue Hashsumme berechnet.
4. Schritt 2 und 3 werden bis zum letzten Sub-Snapshots wiederholt.
5. Die daraus resultierende Hashsumme ist die Hashsumme (*CompositeHash*) des Snapshot Parent.

Das Snapshot Parent wird wie folgt erweitert:

- Das Element *CompositeHash* (siehe weiter oben) enthält ein Attribut, welches den StartHash enthält.
- Des weiteren enthält *CompositeHash* eine Referenzliste (*ExtendOrder*) der Sub-Snapshots. Die Liste definiert die Reihenfolge der Sub-Snapshots, mit dem der *CompositeHash* errechnet wurde.

4.1.5 Integrity Report

Im Kapitel 4.1.3 wurde beschrieben, welche Schritte für eine erfolgreiche Verifizierung notwendig sind, die ebenfalls für eine erfolgreiche Plattformauthentisierung benötigt werden. Der Verifier benötigt dafür eine Liste von vertrauenswürdigen Chain of Trusts, die aber mit zunehmender Anzahl von vertrauenswürdigen Komponenten exponentiell ansteigt. Das führt in der Praxis dazu, dass nahezu unendlich viele Hashsummen gespeichert werden müssten. Angenommen, es existiert eine Chain of Trust von vier Komponenten in der Reihenfolge: A, B, C und D. Komponente A misst Komponente B, dann misst Komponente B die Komponente C etc. Für jede Komponente existieren drei vertrauenswürdige Versionen von 0 bis 2. Alleine in diesem Beispiel, welches Abbildung 4.6 zeigt, existieren 81 (3^4) verschiedene Vertrauensketten und somit 81 vertrauenswürdige PCR-Endwerte. Ist die Reihenfolge der Komponenten beliebig, erhöhen sich die Möglichkeiten auf 1944 ($3^4 * 4!$).

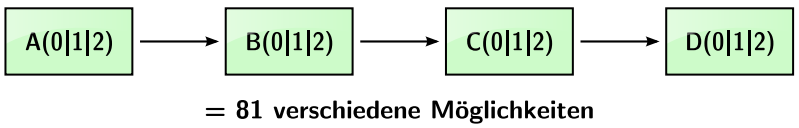


Abbildung 4.6: Eine Chain of Trust mit 4 Komponenten und jeweils 3 Versionen.

Besitzt der Verifier aber die einzelnen Hashsummen der gemessenen Komponenten, kann dieser die Vertrauenskette nachrechnen. Dazu muss der Verifier das Verhalten der PCR simulieren (siehe Kapitel 2.1.2). Sind bei dieser Simulation nur vertrauenswürdige Komponenten eingeflossen, ist der errechnete PCR-Wert vertrauenswürdig. Dieser errechnete Referenz-PCR-Wert kann zur Überprüfung für die Plattformauthentisierung (siehe Kapitel 4.1.2) genutzt werden.

Bei diesem Verfahren muss der Verifier nur so viele Hashsummen vorhalten, wie es vertrauenswürdige Komponenten existieren. Das sind in diesem Beispiel (Abbildung 4.6) nur 12 unterschiedliche Hashsummen, anstatt 81.

Anhand der Snapshots (Kapitel 4.1.4) die im IML (Kapitel 4.1.3) gespeichert werden, ist dieses Verfahren möglich. Damit der Verifier nach diesem Verfahren erfolgreich eine Plattformauthentisierung durchführen kann, benötigt dieser:

- die *QuoteData* (siehe Kapitel 4.1.2) und
- die nötigen Snapshots (siehe Kapitel 4.1.4).

Die TCG spezifiziert in [30] den Integrity Report (IR) als XML-Dokument, welcher für diesen Zweck gedacht ist. Damit die hier beschriebenen Konzepte funktionieren, muss ein IR die im letzten Absatz aufgelisteten Informationen enthalten.

Snapshot Signing

Gemäß der Spezifikation [30] ist es möglich, einzelne Snapshots zu signieren. Das ermöglicht die Integrität einer Dynamic Chain of Trust zu prüfen, ohne dass für diese Vertrauenskette PCRs genutzt werden. Das heißt, dass die Messungen der Dynamic Chain of Trust (ausgenommen der Dynamic Root of Trust) nicht in einem PCR gespeichert werden.

Eine Voraussetzung für dieses Verfahren ist, dass die Komponente, welche die signierten Snapshots erstellt, vertrauenswürdig ist. Diese Komponente ist der PTS, da dieser die Snapshots der Dynamic Chain of Trust erstellt. Die Static Chain of Trust besitzt als Vertrauensanker nur die Static Root of Trust. Da die Static Root of Trust jedoch den Zustand der statischen Vertrauenskette nur mit Hilfe der PCRs beglaubigen kann, werden für die sichere Überprüfung der Static Chain of Trust weiterhin die PCRs benötigt.

Damit der Verifier die Korrektheit der Dynamic Chain of Trust mittels signierter Snapshots verifizieren kann, sind folgende Schritte notwendig:

1. Der PTS erzeugt mit Hilfe des TPM einen Non-Migratable Signing Key[16] (PTS-Key). Dieser PTS-Key ist ein asymmetrischen Schlüssel, der nur zum Signieren von Daten verwendet werden kann. Da dieser Schlüssel ein Non-Migratable Key ist, ist dieser an das TPM und somit an die Plattform gebunden. Das heißt, dass nur das TPM Zugriff auf den privaten Teil des Schlüssels besitzt.

2. Der öffentliche Teil des PTS-Key wird vom TPM mit einem AIK[17] zertifiziert. Durch diese Zertifizierung kann der Verifier nachprüfen, ob der PTS-Key von dem besagten TPM erzeugt wurde. Ist der PTS vertrauenswürdig[18], so kann der Verifier sicher sein, dass der PTS-Key keine missbräuchliche Verwendung findet.

3. Der PTS bildet die Hashsummen der einzelnen Snapshots der Dynamic Chain of Trust.

4. Mit dem privaten Teil des PTS-Key werden dann die erzeugten Hashsummen signiert. Da nur das TPM Zugriff auf den privaten Teil des PTS-Key besitzt, muss das TPM diese Hashsummen signieren. Das Ergebnis ist dann eine Signatur der Snapshots.

5. Der Requestor integriert in dem IR zusätzlich:
   - die Signatur der Snapshots,
   - den zertifizierten öffentlichen Teil des PTS-Key,

[16] siehe Kapitel 2.1.4
[17] siehe Kapitel 2.1.4
[18] Der Vertrauensstatus des PTS muss vorher, mit Hilfe der Static Chain of Trust, überprüft werden.

- das AIK Credential und
- *bei Verwendung von DAA auch den IKEY.*

6. Der Verifier überprüft anhand des IR, zunächst den Vertrauensstatus der Static Chain of Trust und somit auch den des PTS.

7. Ist die Static Chain of Trust vertrauenswürdig, überprüft der Verifier die Korrektheit des zertifizierten PTS-Key. Das heißt, dass der verwendete PTS-Key an die Plattform des Requestors gebunden ist.

8. Ist die Zertifizierung des PTS-Key korrekt, werden die Signaturen der Snapshots überprüft. Bei korrekter Signatur der Snapshots, sind die Messungen in den Snapshots vertrauenswürdig.

Ist der AIK, welcher den PTS-Key zertifiziert, derselbe der die PCR-Werte signiert, sind AIK Credential und IKEY bereits im QuoteData enthalten. Ist dies der Fall, müssen diese Informationen nicht erneut im IR integriert werden. In der XML Spezifikation [29] sind AIK Credentials beim signieren von Snapshots nicht explizit vorgesehen, deshalb ist es vorteilhafter ein AIK Credential sowohl zum zertifizieren des PTS-Keys als auch zum signieren der PCR-Werte zu nutzen.

Der Vorteil von signierten Snapshots ist, dass keine Einschränkung durch die begrenzte Anzahl der PCRs existiert. Auch steigt die Performance, da das verhältnismäßig langsame TPM weniger Operationen durchführen muss. Jedoch vergrößert dieses Verfahren den IR, erhöht die Komplexität der Implementierung und bietet im TNC Umfeld keinen nennenswerten Sicherheitsgewinn. Deshalb wird in dieser Arbeit dieses Konzept mit dem Signieren der Snapshots nicht weiter betrachtet.

Es ist aber auch möglich den IR als ganzes zu Signieren. Dies ermöglicht wie bei dem Snapshot Signing, eine Verifikation der Dynamic Chain of Trust. Da dieses Verfahren größtenteils äquivalent mit dem vom Snapshot Signing ist und ebenfalls im TNC Umfeld keinen Mehrwert bietet, wird dieser Ansatz ebenso nicht weiter verfolgt.

Integrität eines Integrity Report überprüfen

Ein IR besteht aus mehreren Snapshots und den QuoteData, die den Status des Systems vom Requestor beschreiben. Anhand dieser QuoteData kann die Integrität und Authentizität eines IR überprüft werden. Die Integrität bezieht sich in diesem Fall nicht auf den gesamten IR im eigentlichen Sinne, sondern nur auf die wesentlichen Hashsummen im IR. Da aber die Hashsummen im Wesentlichen die Messungen repräsentieren, kann in diesem Fall von der Integrität eines Reports (IR) gesprochen werden. Für die Verifizierung der Integrität und Authentizität wird das Verfahren aus Kapitel 4.1.2 leicht modifiziert. Der Inhalt des IR wird nun folgt wie folgt überprüft:

1. Das Zertifikat (AIK Credential) ist vertrauenswürdig,
2. die Signatur der PCR-Werte ist korrekt und
3. die PCR-Werte lassen sich anhand der im IR enthaltenen Snapshots nachvollziehen. Dafür wird die TPM-Extend Funktionalität (siehe Kapitel 2.1.2) simuliert, indem die Hashsummen der Snapshots in der richtigen Reihenfolge[19] in das TPM-Extend „geschrieben“ werden. Die daraus resultierenden PCR-Werte müssen mit den PCR-Werten in den QuoteData übereinstimmen.

Treten bei der Überprüfung Unstimmigkeiten auf, ist der IR nicht vertrauenswürdig. Je nachdem wo diese Unstimmigkeit auftritt, hat es folgende zusätzliche Aussage:

Das AIK Credential ist nicht vertrauenswürdig: Es kann nicht sichergestellt werden, dass die PCR-Werte tatsächlich von einem vertrauenswürdigen TPM stammen.

Die Signatur ist nicht korrekt: Die PCR-Werte bzw. die Signatur wurde nachträglich manipuliert.

Die PCR-Werte lassen sich nicht rekonstruieren: Die Hashsumme von mindestens einem Snapshot wurde im Nachhinein verändert. Es kann jedoch nicht zweifelsfrei festgestellt werden, welche Snapshots genau von dieser Manipulation betroffen sind.

Nach dieser Verifikation müssen noch weitere Überprüfungen des IR vorgenommen werden. Diese werden in den jeweiligen Konzepten 1 bis 4 näher erläutert.

4.1.6 Clientside Policy

Ein Grundkonzept von TNC ist, dass die IMVs vorgeben, was genau die IMCs messen sollen. Aus den gesammelten Daten kann dann der Verifier (PDP) einen Benutzer identifizieren bzw. missbräuchliche Rückschlüsse ziehen, z. B. das Abfragen privater Daten. Um solch einen Missbrauch zu erschweren, können die auf der Requestor-Seite gemessenen Daten anonymisiert werden und/oder eine Policy (eine Clientside Policy) dafür erstellt werden. Diese Policy soll sensible Daten eines Benutzers bzw. einer Plattform schützen.

Im konkreten Fall des PTS-IMC bedeutet das, dass mit Hilfe der Clientside Policy bestimmte Dateien von der Messung (über den PTS-IMC) grundsätzlich ausgeschlossen werden können. So sollte die Policy z. B. unter Linux Systemen die Messung der Datei `/etc/passwd` verhindern, da sonst der Verifier den Requestor eindeutig identifizieren könnte, solange kein Benutzer hinzugefügt, geändert oder entfernt wird, weil sich die

[19] Die Reihenfolge der Snapshots stehen in den Sync Snapshots (siehe Kapitel 4.1.4).

Hashsumme (SHA-1) der Datei nicht ändert. Die Chance, dass die `passwd` Datei sich häufig ändert bzw. mehrere Systeme die gleiche `passwd` besitzen ist in der Praxis vergleichsweise gering. Auch sollte die Policy den Zugriff auf die Homeverzeichnisse unterbinden. Bei den Konzepten, die in Kapitel 4.2 und 4.4 beschrieben werden, wird eine Clientside Policy beim PTS-IMC nicht benötigt. Dort steht von vornherein fest, welche Messungen zum Verifier verschickt werden (nur die vom TNCC und den IMCs).

Die Clientside Policy wird vom PTS-IMC durchgesetzt und nicht direkt vom PTS. Es ist durchaus möglich, dass rein lokale Dienste beliebige Daten über den PTS messen. Da diese Dienste prinzipiell dazu berechtigt sind alle Daten zu messen, würde die Umsetzung der Policy im PTS keinen Sinn ergeben.

Für eine einfache Policy genügt es eine Liste von zu messenden erlaubten bzw. nicht erlaubten Dateien anzugeben. Dazu wird vorgeschlagen, eine positive und negative Liste von Regeln unter Verwendung von Regulären Ausdrücken[20] zu verwenden. Damit eine Datei gemessen werden kann, muss eine Regel der positiven Liste zutreffen aber gleichzeitig darf keine Regel der negativen Liste zutreffen. Als Standardregel enthält die positive Liste nur einen Eintrag, der es erlaubt jede beliebige Messung durchzuführen (`/.*/`). Sobald der Benutzer/Administrator eine neue Regel definiert, wird die Standardregel automatisch aus der Liste entfernt. Die negative Liste enthält standardmäßig keine Regeln.

[20] `http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck`

4.2 Konzept 1: Minimal

In diesem Kapitel wird das erste und einfachste Konzept dieser Arbeit vorgestellt. Abbildung 4.7 zeigt den Aufbau des Konzeptes. Die grundlegende Idee dabei ist, dass der TNCC und die IMCs vom PTS gemessen werden und somit ein Teil der Vertrauenskette sind. Dadurch müssen die Messungen der IMCs nicht weiter abgesichert werden.

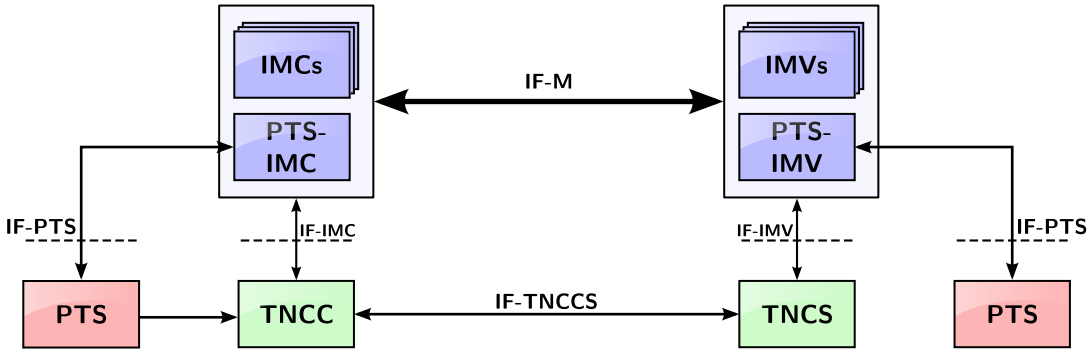


Abbildung 4.7: Aufbau des 1. und 2. Konzeptes

4.2.1 Beschreibung

Der PTS auf der Client-Seite (CPTS) misst nur den TNCC und die dazugehörigen IMCs. Der PTS-IMC sendet den IR, der die Messungen der Chain of Trust inkl. TNCC und IMCs enthält, über das TNC-Protokoll zum Server, welches in [18] spezifiziert wird. Dort empfängt der PTS-IMV den IR. Über den PTS auf der Server-Seite (SPTS), evaluiert der PTS-IMV anhand des empfangenen IR den Vertrauensstatus des Clients. Wird der Client (dessen Chain of Trust inkl. TNCC und IMCs) daraufhin als vertrauenswürdig eingestuft, können die übrigen IMVs den Messungen der IMCs ebenfalls vertrauen.

Der CPTS misst hier nur den TNCC inkl. IMCs. Wenn ein Hersteller beliebige Dateien bzw. Prozesse überprüfen will, muss dieser die Messungen und das Auswerten selber implementieren. Solange die Messungen direkt von einem IMC durchgeführt wird, benötigt der Verifier keinen weiteren Vertrauensanker, da die Integrität der IMCs schon verifiziert wird.

4.2.2 Transferprotokoll

Das Protokoll zwischen dem PTS-IMC und PTS-IMV wird wie folgt definiert:

1. Schritt (PTS-IMC → PTS-IMV) Als erstes müssen sich der PTS-IMC und der PTS-IMV auf eine Protokollversion einigen. Dafür sendet der PTS-IMC folgendes zum PTS-IMV:

- Die minimale unterstützte Protokollversion ($minProtocolVersion$) und
- die maximale unterstützte Protokollversion ($maxProtocolVersion$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq maxProtocolVersion$

2. Schritt (PTS-IMC ← PTS-IMV) Als nächstes schickt der PTS-IMV zum PTS-IMC:

- Die verwendete Protokollversion ($protocolVersion$) und
- eine Nonce ($nonce$) (siehe Kapitel 4.1.2).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq protocolVersion \leq maxProtocolVersion$ $\vee\, protocolVersion = 0$
- Wenn die $protocolVersion = 0$ ist, sind der PTS-IMC und der PTS-IMV inkompatibel zueinander und die Kommunikation ist zwangsläufig gescheitert.
- $nonce$ ist eine 20 Byte große Zahl (siehe SHA-1).

3. Schritt (PTS-IMC → PTS-IMV) Nachdem der PTS-IMC die Nonce empfangen hat, erstellt der CPTS mit Hilfe dieser Nonce einen IR (siehe Kapitel 4.1.5).

- Anschließend wird vom PTS-IMC der IR verschickt.

4. Schritt (PTS-IMC ← PTS-IMV) Im letzten Schritt überprüft der PTS-IMV mit Hilfe des SPTS den Vertrauensstatus des Clients, anhand des IR.

- Als letztes sendet der der PTS-IMV noch Informationen zur Entscheidungsfindung zum PTS-IMC ($resultInformation$).

4.2.3 Measurement

Auf der Client-Seite misst der CPTS den TNCC und die IMCs (siehe Kapitel 4.1.1) bevor diese ausgeführt werden. Wichtig ist, das alle Programmbibliotheken (Libraries[21]) ebenfalls mitgemessen werden, denn eine Library besitzt dieselben Rechte wie der Prozess in der sie ausgeführt wird. Während der TNCC (inkl. den IMCs) ausgeführt wird, sollte der PTS keine weiteren Messungen durchführen. Wird der TNCC mit den

[21] `http://de.wikipedia.org/wiki/Programmbibliothek`

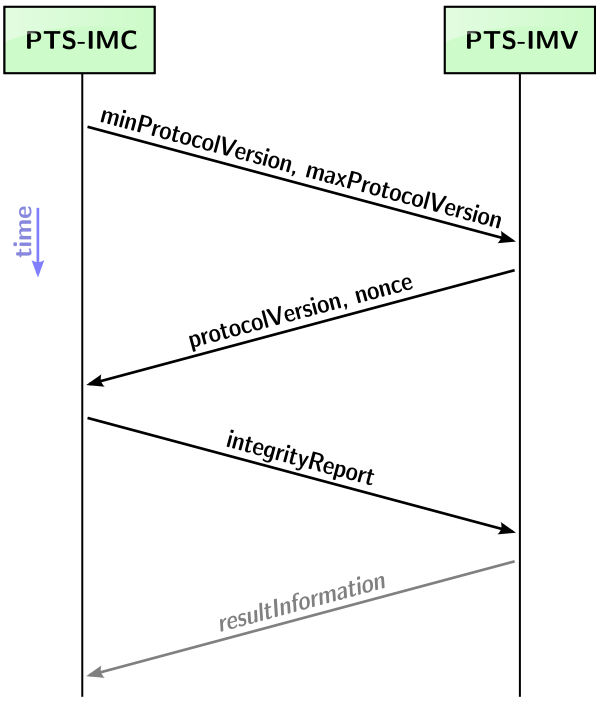


Abbildung 4.8: Sequenzdiagramm vom 1. Konzept

IMCs gestoppt, werden die Messungen gelöscht und der PCR#PTS zurückgesetzt. Der im Kapitel 4.2.2 Schritt 3 beschriebene IR enthält unter anderem diese Messergebnisse.

4.2.4 Serverside Policy

Um eine Plattformauthentisierung durchzuführen, müssen mehrere Komponenten des Verifier eine Entscheidung zum (Vertrauens-)Status treffen. In diesem Konzept treffen folgende Komponenten diese Entscheidung:

SPTS Der SPTS überprüft anhand des IR, ob der Verifier vertrauenswürdig ist. Dies entscheidet der PTS anhand von Regeln. Die TCG definiert einfache Regeln in Form von XML [33]. Solch eine Regel enthält genau einen Referenzsnapshot, den der Verifier als vertrauenswürdig erachtet. Es ist aber auch möglich, eigene Regeln zu definieren. Die Struktur bzw. Interpretation der Regeln wird jedoch von der TCG nicht vorgegeben. Für dieses Konzept wird die Interpretation der Regeln im Kapitel 4.2.5 definiert.

PTS-IMV Der SPTS bekommt die Regeln für die Verifizierung des Requestors vom PTS-IMV. Dieser IMV erstellt diese Regeln oder liest diese, z. B. aus einer Datenbank, aus. Dadurch entscheidet der PTS-IMV, welche Regeln der PTS zur Ver-

ifizierung des Clients benutzen soll. Damit solch eine Verifizierung erfolgreich verlaufen kann, muss der PTS anhand dieser Regeln sowohl die Static- als auch die Dynamic Chain of Trust erfolgreich verifizieren können.

Im Betracht einer vollständigen Plattformauthentisierung ist dieses Ergebnis nur eine partielle Entscheidung, denn das Resultat liefert nur eine Aussage über die Static- und Dynamic Chain of Trust, jedoch nicht über Komponenten außerhalb der beiden Vertrauensketten. Der PTS-IMV liefert das Ergebnis an den TNCS zurück.

IMVs Die anderen IMVs treffen ebenfalls nur partielle Entscheidungen. Dabei handelt es sich i. d. R. um Aussagen über andere Komponenten bzw. Konfiguration (z. B. über Antivirus-Software, Personal-Firewall oder Browser) des Requestors. Die Ergebnisse werden dem TNCS zurückgeliefert.

TNCS Der TNCS sammelt die Entscheidung der IMVs (inkl. PTS-IMV) und trifft daraufhin selbst eine Entscheidung. Diese Entscheidung des TNCS ist damit das Ergebnis der Plattformauthentisierung. Wichtig bei diesem Konzept ist, dass die Plattformauthentisierung in jedem Fall fehlschlägt, wenn der PTS-IMV kein vertrauenswürdiges Ergebnis liefert. Wie der TNCS die weiteren Ergebnissen der IMVs in die Plattformauthentisierung mit einbezieht, ist dem TNCS und seiner internen Policy überlassen. Auf Grund des Ergebnis der Plattformauthentisierung kann der Verifier dem Requestor den Zugriff auf den angefragten Dienst gewähren bzw. verweigern.

4.2.5 Regeln

Wie im letzten Kapitel beschrieben entscheidet der SPTS anhand von Regeln, ob ein Requestor vertrauenswürdig ist oder nicht. Dazu wendet der SPTS die Regeln auf den IR des Requestors an. Da die TCG die Struktur bzw. Interpretation der Regeln nicht spezifiziert, wird nun beschrieben, wie der PTS den IR anhand der Regeln verifiziert.

Abbildung 4.9 Abschnitt 1 zeigt als Beispiel einen vereinfachten IR. Dieser einfache IR enthält:

- einen Sync Snapshot,
- 3 Normal Snapshots und
- QuoteData für ein PCR.

Am Anfang muss die Integrität und Authentizität des IR überprüft werden. Dazu wird die im Kapitel 4.1.5 auf Seite 30 beschriebene Verifizierung der Integrität vorgenommen. Bei fehlgeschlagener Überprüfung dieser Integrität müssen keine Regeln angewen-

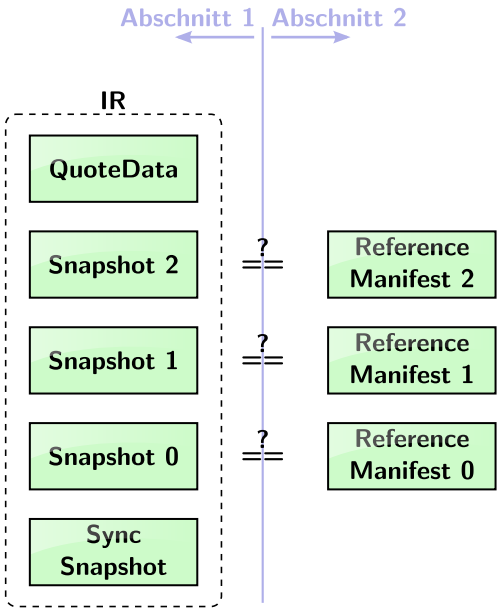


Abbildung 4.9: Beispiel von Regeln im 1. Konzept

det werden, da der Client dabei schon als nicht vertrauenswürdig eingestuft wird. Jedoch müssen bei erfolgreicher Verifizierung der Integrität noch die Regeln des SPTS angewendet werden, damit der Client als vertrauenswürdig gilt.

Auf alle Snapshots, die für die Integritätsmessung des IR benötigt wurden, werden die Regeln angewendet. Eine Regel kann einen Snapshot als vertrauenswürdig markieren. Dafür muss der Snapshot mit einem Referenzsnapshot (RIMM[22]) übereinstimmen. Die TCG spezifiziert eine solche (XML)-Regel im Reference Manifest (RM) Schema [33]. Da ein RIMM nur einen Referenzsnapshot enthalten kann, bekommt der SPTS vom PTS-IMV eine Liste von RIMMs (Regeln).

Um einen Angriff zu verhindern, bei dem der Angreifer die Snapshots und die dazugehörigen PCRs einer kompromittierten Komponente aus dem IR entfernt, muss jeder Referenzsnapshot (Regel) einen Snapshot im IR als vertrauenswürdig markieren.

Sind alle Snapshots als vertrauenswürdig markiert, liefert der SPTS zurück, das die Vertrauensketten (sowohl die Static Chain of Trust als auch die Dynamic Chain of Trust) des Clients vertrauenswürdig sind. Ansonsten liefert der SPTS einen nicht vertrauenswürdigen Status des Clients zurück.

[22]Reference Integrity Measurement Manifest

4.3 Konzept 2: Data Extension

Im zweiten hier vorgestelltem Konzept ist es vorgesehen, dass der PTS neben den TNCC (inkl. IMCs) auch beliebige Messungen durchführen kann. Dazu gibt der Server, vor welche weiteren Messungen durchgeführt werden sollen. Dadurch ist es möglich, die Messungen mit Hilfe vom TPM zu signieren. Des Weiteren sind der TNCC und die IMCs ein Teil der Chain of Trust und erfordern, wie auch im Minimal Konzept (Kapitel 4.2), keine weitere Absicherung. Abbildung 4.7 zeigt den Aufbau des Konzeptes.

4.3.1 Beschreibung

Der CPTS misst wie auch im Minimal Konzept (siehe 4.2) den TNCC und die dazugehörigen IMCs. Der Server, konkret der PTS-IMV, schickt dem Client eine Liste der zusätzlich durchzuführenden Messungen zu. Diese zusätzlichen Messungen führt ebenfalls der CPTS durch. Anschließend sendet der PTS-IMC den IR, welches die Messungen der Chain of Trust (inkl. TNCC und IMCs) und die zusätzlichen Messungen enthält, über das TNC-Protokoll zurück zum Server. Dort empfängt der PTS-IMV den IR und ermittelt mit Hilfe des SPTS den Vertrauensstatus des Clients. Wird der Client (dessen Chain of Trust) als vertrauenswürdig erachtet und die übrigen Messungen entsprechen einer bestimmten Policy, dann wird dem Client der Zugriff auf den angefragten Dienst gewährt. Ist der Client hingegen nicht vertrauenswürdig und/oder die übrigen Messungen nicht Policykonform, so enthält der Client auch keinen Zugriff auf den angefragten Dienst.

Wie auch im ersten Konzept sind die separaten Messungen der IMCs vertrauenswürdig, wenn die Vertrauensketten des Client es ebenfalls sind. Ein Hersteller kann nach wie vor Messungen und Überprüfungen selbstständig durchführen, jedoch ermöglicht dieses Konzept dem Hersteller von IMC/IMV-Paaren vorhandene Funktionalitäten zum messen und verifizieren von Daten zu nutzen. Wenn z. B. ein Hersteller eine Datei messen und verifizieren will, dann kann das PTS-IMC/IMV-Paar diese Aufgabe übernehmen.

4.3.2 Transferprotokoll

Das Protokoll zwischen dem PTS-IMC und PTS-IMV wird wie folgt definiert:

1. Schritt (PTS-IMC → PTS-IMV) Als erstes müssen sich der PTS-IMC und der PTS-IMV auf eine Protokollversion einigen. Dafür sendet der PTS-IMC folgendes zum PTS-IMV:

- Die minimale unterstützte Protokollversion ($minProtocolVersion$) und
- die maximale unterstützte Protokollversion ($maxProtocolVersion$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq maxProtocolVersion$

2. Schritt (PTS-IMC ← PTS-IMV) Als nächstes schickt der PTS-IMV zum PTS-IMC:

- Die verwendete Protokollversion ($protocolVersion$),
- die Nonce ($nonce$) (siehe Kapitel 4.1.2) und
- eine Liste der zusätzlich zu messenden Daten ($listOfMeasure$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq protocolVersion \leq maxProtocolVersion \vee protocolVersion = 0$
- Wenn die $protocolVersion = 0$ ist, sind der PTS-IMC und der PTS-IMV inkompatibel zueinander und die Kommunikation ist zwangsläufig gescheitert.
- *nonce* ist eine 20 Byte große Zahl (siehe SHA-1).

3. Schritt (PTS-IMC → PTS-IMV) Der PTS-IMC misst über den CPTS die zusätzlichen zu messenden Daten, welche dieser vom PTS-IMV empfangen hat (*listOfMeasure*). Danach erstellt der CPTS mit Hilfe der empfangenen Nonce einen IR (siehe Kapitel 4.1.5).

- Anschließend wird vom PTS-IMC der IR ($integrityReport$) verschickt.

4. Schritt (PTS-IMC ← PTS-IMV) Im letzten Schritt überprüft der PTS-IMV mit Hilfe des SPTS den Vertrauensstatus des Clients, anhand des IR.

- Als letztes sendet der PTS-IMV noch Informationen zur Entscheidungsfindung zum Client ($resultInformation$).

4.3.3 Measurement

Der CPTS misst wie im ersten Konzept (siehe Kapitel 4.2.3) den TNCC und die IMCs bevor sie ausgeführt werden. Jedoch kann der CPTS auch während einer Plattformauthentisierung beliebige Daten zusätzlich messen.

1. Dafür empfängt der PTS-IMC eine Liste von zu messenden Daten (*listOfMeasure*) vom Server (siehe Schritt 3 des Kapitel 4.3.2).

2. Um die Privatsphäre des Client zu schützen, wird die Clientside Policy (siehe Kapitel 4.1.6) auf die Liste angewendet. Die Policy filtert nicht erwünschte Messungen (aus Sicht des Clients) heraus.

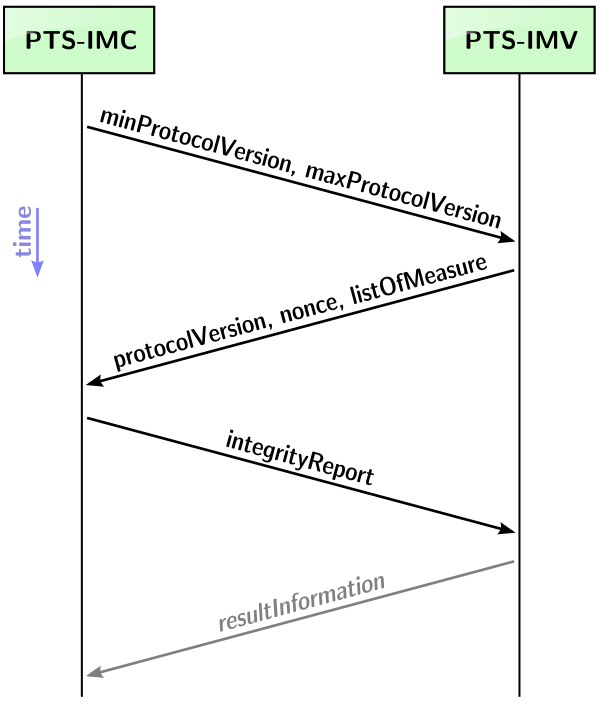


Abbildung 4.10: Sequenzdiagramm vom 2. Konzept

3. Anschließend lässt der PTS-IMC die zusätzlichen Daten vom CPTS messen.

4. Die aus den Messungen resultierenden Informationen werden zuletzt im IR zusammengefasst und zum Server zurückgesendet (siehe Schritt 3 des Kapitels 4.3.2).

Die TCG sieht vor, dass sowohl Dateien auf der Festplatte als auch Speicherbereiche des Arbeitsspeichers gemessen werden können. Jedoch werden keine näheren Angaben getätigt, wie solch eine Speichermessung aussehen könnte. Da das Messen des Arbeitsspeichers den Rahmen dieser Arbeit sprengen würde, wird auf eine genauere Definition zum Messen des Speichers verzichtet. Stattdessen beschränkt sich diese Arbeit auf das Messen von Dateien, da die Art der Messungen keine nennenswerten Änderungen an dem hier spezifizierte Protokoll mit sich bringen sollte.

Auch im diesem Konzept werden bei der Beendigung des TNCC die Messungen gelöscht und der PCR#PTS zurückgesetzt.

4.3.4 Serverside Policy

Um eine Plattformauthentisierung durchzuführen, müssen wie auch im ersten Konzept 4.2.4 mehrere Komponenten des Verifier eine Entscheidung zum (Vertrauens-)Status treffen. In diesem Konzept treffen folgende Komponenten diese Entscheidung:

SPTS Der SPTS überprüft anhand des IR, ob die Chain of Trust des Verifier vertrauenswürdig ist. Des Weiteren werden die zusätzlichen Messungen im IR überprüft, welche im Kapitel 4.3.3 beschrieben wurde. Der SPTS entscheidet anhand von Regeln, wie auch im ersten Konzept (4.2.4), ob ein Client vertrauenswürdig ist und ob die zusätzlichen Messungen korrekt sind. Die Bedeutung und Interpretation der Regeln dieses Konzeptes wird in Kapitel 4.3.5 erläutert.

PTS-IMV Wie auch im ersten Konzept erhält der SPTS die Regeln zur Verifizierung des Clients ausschließlich vom PTS-IMV. Dieser IMV liest die Regeln z. B. aus einer Datenbank aus. Der PTS-IMV entscheidet dadurch, welche Chain of Trust bzw. welche Ergebnisse die zusätzlichen Messungen im IR enthalten sein müssen.

Für eine vollständige Plattformauthentisierung ist das Ergebnis ebenfalls eine partielle Entscheidung, dessen Resultat eine Aussage über die Static-, Dynamic Chain of Trust und der zusätzlichen Messungen außerhalb der Vertrauensketten trifft. Da die Schnittstellen, sowohl IF-PTS [8] als auch IF-IMV [20], keine detaillierten Resultate liefern können, liefert der PTS-IMV dem TNCS nur ein Access Allow wenn alle Messungen des IR erfolgreich verifiziert wurden.

IMVs Die anderen IMVs treffen ebenfalls nur partielle Entscheidungen. Diese Entscheidungen treffen i. d. R. Aussagen über andere Komponenten bzw. Konfiguration (z. B. über Antivirus-Software, Personal-Firewall oder Browser) des Requestors. Da nicht vorgesehen ist, dass IMVs miteinander kommunizieren, gibt es für IMVs keine standardisierte Methode abzufragen, zu welchem Resultat der PTS-IMV gekommen ist. Dadurch kann unter Umständen die Policy für eine Komponente aufgeteilt werden. Wenn als Beispiel ein Browser gemessen werden soll (sowohl Programmdateien als auch die Konfiguration), enthält der PTS-IMV eine Policy (in Form von Regeln) für die Dateien und ein spezieller IMV eine Policy für die Konfiguration.

Die Ergebnisse werden dem TNCS zurückgeliefert.

TNCS Der TNCS sammelt die Entscheidung der IMVs (inkl. PTS-IMV) und trifft daraufhin selber eine Entscheidung. Diese Entscheidung des TNCS ist damit das Ergebnis der Plattformauthentisierung. Ebenfalls wichtig bei diesem Konzept ist, dass die Plattformauthentisierung in jedem Fall fehlschlägt, wenn der PTS-IMV kein Access Allow liefert. Wie der TNCS die weiteren Ergebnissen der IMVs in die Plattformauthentisierung mit einbezieht, ist dem TNCS und seiner internen Policy überlassen. Auf Grund des Ergebnis der Plattformauthentisierung kann der Server dem Client den Zugriff auf den angefragten Dienst gewähren bzw. verweigern.

4.3.5 Regeln

Auch in diesem Konzept wird der SPTS nur vom PTS-IMV aufgerufen und enthält somit die Regeln zur Verifizierung des IR ebenfalls vom PTS-IMV. Da die TCG die Struktur bzw. Interpretation der Regeln nicht spezifiziert, wird hier definiert, wie der PTS den IR anhand der Regeln verifiziert.

Bevor ein der SPTS die Regeln anwendet, wird wie schon im Minimal Konzept die Integrität und Authentizität, des IR überprüft. War diese Überprüfung korrekt, werden die Regeln vom PTS-IMV angewendet. Die Anwendung der Regeln erfolgt wie in Kapitel 4.2.5 beschrieben.

Auch muss jede Regel (Referenzsnapshot) zu einem Snapshot aus dem IR gehören, denn ein Angreifer könnte die kompromittierten Snapshots aus dem IR entfernen, bzw. die Messung wurden auf Grund der Clientside Policy (siehe Kapitel 4.1.6) nicht durchgeführt.

Sind alle Snapshots als vertrauenswürdig markiert, liefert der SPTS zurück, das die Vertrauensketten (sowohl die Static Chain of Trust als auch die Dynamic Chain of Trust) und die zusätzlichen Messungen des Clients vertrauenswürdig sind. Ansonsten liefert der SPTS einen nicht vertrauenswürdigen Status des Clients zurück.

4.4 Konzept 3: Seperate

Im dritten Konzept, welches in diesem Kapitel vorgestellt wird, kann der CPTS ebenfalls beliebige zusätzliche Messungen durchführen. Jedoch werden diese zusätzlichen Messungen nicht vom PTS-IMC in Auftrag gegeben, wie im zweiten Data Extension Konzept (siehe Kapitel 4.3), sondern von den anderen IMCs. Auf der Serverseite wird die Aufgabe der Verifizierung des IR auf die IMVs (nicht nur PTS-IMV) aufgeteilt. Abbildung 4.11 zeigt den Aufbau des Konzeptes.

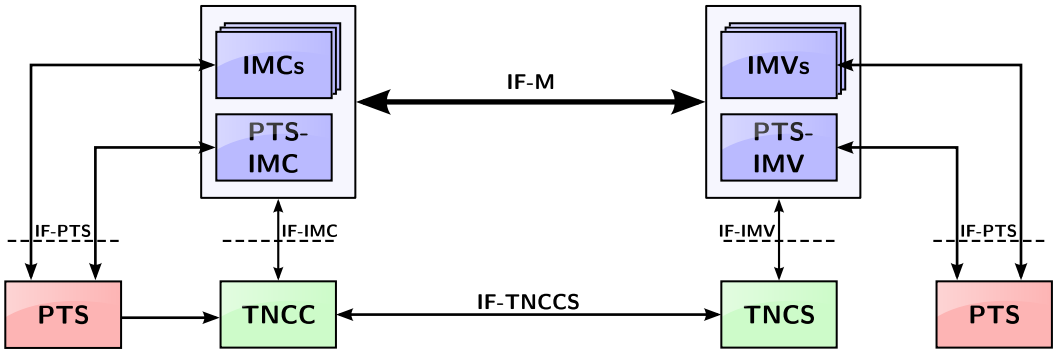


Abbildung 4.11: Aufbau des 3. Konzeptes

4.4.1 Beschreibung

Wie in den vorherigen Konzepten misst der CPTS den TNCC mit allen dazugehörigen IMCs. Andere IMCs können zusätzlich mit Hilfe des CPTS beliebige Daten messen. Jeder IMC sendet die Messergebnisse als IR an den jeweiligen passenden IMV auf der Serverseite. Das heißt, dass der PTS-IMC die Messung vom TNCC und IMCs zum PTS-IMV und z. B. ein „Browser-IMC“ seine Messungen zum „Browser-IMV“ übermittelt. Die IMVs empfangen die IRs und überprüfen den Teil der Messungen, für den sie zuständig sind. Der PTS-IMV z. B. verifiziert die Static- und Dynamic Chain of Trust, während der „Browser-IMV“ nur die Messungen vom Browser überprüft. Die anderen IMC/IMV-Paare können neben den IR aber auch noch herstellerspezifische Daten verschicken. Damit kann dann z. B. der „Browser-IMV“ auch die Konfiguration des Browsers überprüfen.

Da der PTS-IMV den Vertrauensstatus der wichtigsten Komponenten sicherstellt (die Verifizierung der Vertrauensketten), sind die Messungen der anderen IMCs ebenfalls vertrauenswürdig. Dieses Konzept bietet, wie beim zweiten Konzept (siehe Kapitel 4.3), die Möglichkeit vorhandene Funktionalitäten zum Messen und Verifizieren mit TPM-Unterstützung zu nutzen.

4.4.2 Transferprotokoll

In diesem Konzept kommuniziert jedes IMC/IMV-Paar nur untereinander. Diese Kommunikation kann daher von dem jeweiligen Hersteller selbst definiert werden. Im Folgenden wird für diese Kommunikation exemplarisch ein Ansatz vorgestellt, der exakt auf dem Protokoll vom PTS-IMC/IMV im Kapitel 4.2.2 basiert.

1. Schritt (IMC → IMV) Als erstes sollten sich ein IMC und der dazugehörige IMV auf eine Protokollversion einigen. Dafür sendet der IMC folgendes zum IMV:

- Die minimale unterstützte Protokollversion ($minProtocolVersion$) und
- die maximale unterstützte Protokollversion ($maxProtocolVersion$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq maxProtocolVersion$

2. Schritt (IMC ← IMV) Als nächstes schickt der IMV zum IMC:

- Die verwendete Protokollversion ($protocolVersion$) und
- eine Nonce ($nonce$) (siehe Kapitel 4.1.2).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq protocolVersion \leq maxProtocolVersion$
  $\vee\, protocolVersion = 0$
- Wenn die $protocolVersion = 0$ ist, sind der IMC und der IMV inkompatibel zueinander und die Kommunikation ist zwangsläufig gescheitert.
- $nonce$ ist eine 20 Byte große Zahl (siehe SHA-1).

3. Schritt (IMC → IMV) Nachdem der IMC die Nonce empfangen hat, erstellt der CPTS mit Hilfe dieser Nonce einen IR (siehe Kapitel 4.1.5).

- Anschließend wird vom IMC der IR ($integrityReport$) verschickt.

4. Schritt (IMC ← IMV) Im letzten Schritt überprüft der IMV mit Hilfe des SPTS die Messungen des IMVs, anhand des IR.

- Als letztes kann der der IMV noch Informationen zur Entscheidungsfindung zum IMC ($resultInformation$) senden.

4.4.3 Measurement

Der PTS-IMC misst wie im Minimal Konzept nur den TNCC inkl. aller IMCs, bevor diese aufgerufen werden. Jedoch wird die komplette Messung (TNCC und IMCs) unter einem Snapshot als einzelne Sub-Snapshots vereint, welches im Kapitel 4.1.4 auf Seite

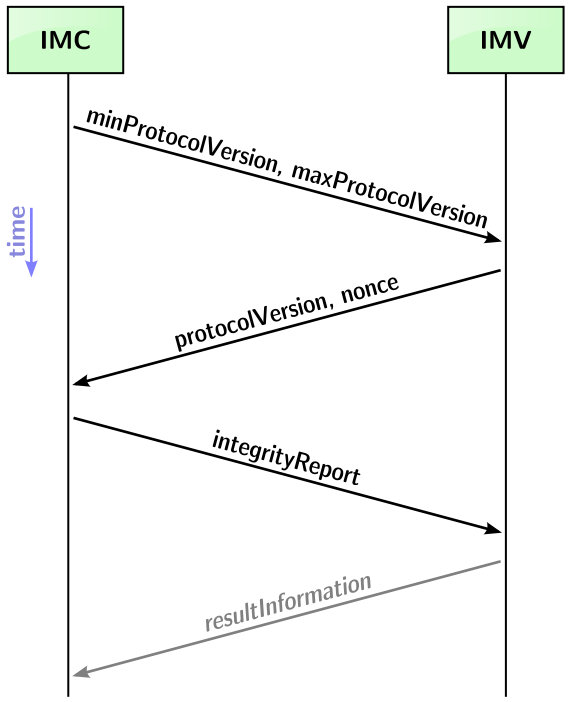


Abbildung 4.12: Sequenzdiagramm vom 3. Konzept

26 definiert wurde. Abbildung 4.13 zeigt ein Beispiel einer TNCC (inkl. IMCs) Messung. Dies wird getan, damit der SPTS diese Messung (inkl. aller Unterkomponenten) eindeutig identifizieren kann (siehe Kapitel 4.4.5).

Wenn andere IMCs Messungen von beliebigen Daten durchführen wollen, nutzen sie wie der PTS-IMC den CPTS. Auch hier wird die komplette Messung unter einem Snapshot als einzelne Sub-Snapshots integriert, um diese Messung auf der Serverseite eindeutig zu identifizieren.

4.4.4 Serverside Policy

Auf der Serverseite entscheiden folgende Komponenten, ob der Client Zugriff auf den angefragten Dienst enthält:

SPTS Der PTS auf der Serverseite ist nur für generische Überprüfung des IR zuständig. Das heißt, der SPTS verifiziert wie im vorherigen Konzepten den IR anhand von Regeln. Die Definition und Interpretation wird im Kapitel 4.4.5 beschrieben.

Der Unterschied zu den vorherigen Konzepten besteht darin, dass der SPTS nicht nur vom PTS-IMV aufgerufen wird, sondern von allen IMVs aufgerufen werden kann.

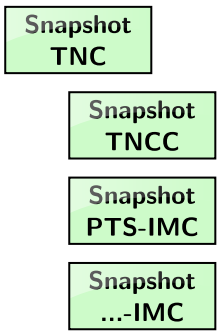


Abbildung 4.13: Beispiel einer TNC Messung

PTS-IMV Der PTS-IMV ist wie im ersten Konzept nur für die Verifikation der Static- und Dynamic Chain of Trust zuständig, auch wenn der IR noch andere Messungen enthält.

IMVs Alle anderen IMVs treffen wie in den vorherigen Konzepten i. d. R. eine Aussage über bestimmte Komponenten auf dem Client. Jedoch können die IMVs den SPTS zur Überprüfung der für sie relevanten Messungen im IR nutzen. Dafür übergibt der IMV dem SPTS nur die Regeln für die relevanten Messungen.

TNCS Der TNCS sammelt wie in den vorherigen Konzepten die Entscheidung aller IMVs und trifft daraufhin die Entscheidung ob der Client Zugriff auf den angefragten Dienst bekommt. Auch in diesem Konzept müssen die Chain of Trusts vertrauenswürdig sein. Das heißt, dass die Plattformauthentisierung in jedem Fall fehlschlägt, wenn der PTS-IMV kein Access Allow liefert.

4.4.5 Regeln

Der SPTS kann, im Gegensatz zu den vorherigen Konzepten, von jedem IMV aufgerufen werden. Da i. d. R. ein IMV jedoch nur Teile des IR überprüfen will, sind Regeln notwendig welche eine partielle Verifikation erlauben.

Bevor die Regeln auf einem IR angewendet können, muss die die Integrität und Authentizität des IR sichergestellt werden. Dies geschieht wie im ersten Konzept im Kapitel 4.2.5 beschrieben. Bei korrekter Integrität werden anschließend die übergebenen Regeln angewendet.

Abbildung 4.14 Abschnitt 1 zeigt ein Beispiel IR für das Data Extension Konzept. Der Sync-Snapshot (siehe Kapitel 4.1.4) enthält die Reihenfolge der obersten Snapshots (Snapshot 0 und Snapshot 1). Die eigentlichen Messungen (Snapshot 00, 10 und 11) liegen als Sub-Snapshots unter den oberen Snapshots 0 und 1 (siehe Kapitel 4.2.3).

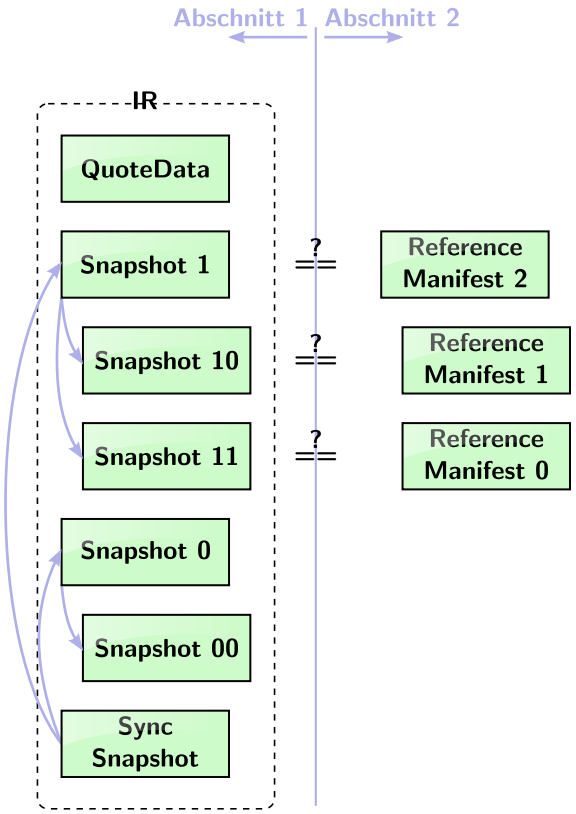


Abbildung 4.14: Beispiel von Regeln im 3. Konzept

Will ein IMV nur die Messungen im Snapshot 1 Verifizieren, so übergibt dieser nur Regeln für den Snapshot 1, so dass der Snapshot 0 nicht geprüft wird. Sind alle Snapshots Policykonform, für den ein Referenzsnapshot (Regel) existiert, liefert der SPTS dem IMV ein positives Ergebnis zurück.

4.5 Konzept 4: Cross Over

In diesem Kapitel wird das vierte und letzte Konzept dieser Arbeit vorgestellt. Es ist eine Kombination und Erweiterung der vorherigen Konzepte. Auf der Clientseite wird die Komplexität verringert, in dem nur der PTS-IMC für Messungen der zusätzlichen Daten verantwortlich ist. Auf der Serverseite dagegen wird die Verifikation dieser Daten, zu Gunsten der Policy, auf die IMVs verteilt.

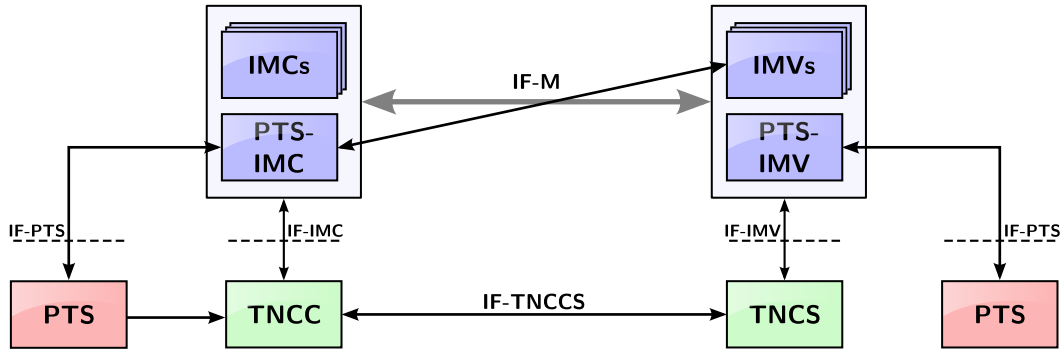


Abbildung 4.15: Aufbau des 4. Konzeptes

4.5.1 Beschreibung

Der CPTS misst wie in allen anderen Konzepten den TNCC inkl. IMCs, bevor diese ausgeführt werden. Die IMVs auf der Serverseite geben dem PTS-IMC vor, welche Daten dieser zusätzlich messen soll. Dazu senden die IMVs dem PTS-IMC eine Liste der zu messenden Daten zu. Der PTS-IMC führt die Messungen mit Hilfe des CPTS durch und sendet anschließend den IR zurück zu den IMVs. Jeder IMV verifiziert daraufhin einen Teil des IR.

Als Beispiel existieren auf der Serverseite mehrere IMVs:

- Browser-IMV
- Mailclient-IMV
- PTS-IMV

Der Browser-IMV sendet dem PTS-IMC eine Liste von Dateien zu, die für die Überprüfung der Integrität des Internetbrowsers benötigt werden. Äquivalent verhält sich der Mailclient-IMV, mit den Dateien des Mailclients. Da auf der Clientseite die Static- und Dynamic Chain of Trust implizit gemessen wurden, sendet der PTS-IMV keine Liste der zu messenden Daten zum PTS-IMC. Der PTS-IMC lässt den Browser und

den Mailclient über den CPTS messen. Anschließend wird der IR zurück zu den IMVs (Browser-, Mailclient- und PTS-IMV) gesendet. Der Browser-IMV überprüft mit Hilfe des SPTS die Messungen des Browser im IR. Der Mailclient-IMV verifiziert nach dem gleichen Verfahren die Messungen des Mailclient. Der PTS-IMV hingegen überprüft mit Hilfe des SPTS die Vertrauensketten des Clients.

4.5.2 Transferprotokoll

Die IMC/IMV Paare können, wie in den anderen Konzepten, untereinander kommunizieren. Jedoch sieht dieses Konzept vor, dass IMVs ebenfalls mit dem PTS-IMC kommunizieren können. Im Folgenden wird das Protokoll vorgestellt, welches die Kommunikation zwischen den IMVs und den PTS-IMC spezifiziert. IMVs, die keine Kommunikation mit dem PTS-IMC beabsichtigen, nutzen dieses Protokoll nicht.

1. Schritt (PTS-IMC → IMV) Zuerst einigt sich der PTS-IMC mit jedem IMV auf eine Protokollversion. Dazu sendet der PTS-IMC eine Nachricht an alle IMVs welche folgenden Inhalt enthält:

- Die minimale unterstützte Protokollversion ($minProtocolVersion$) und
- die maximale unterstützte Protokollversion ($maxProtocolVersion$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq maxProtocolVersion$

2. Schritt (PTS-IMC ← IMV) Als nächstes schicken die IMVs jeweils zum PTS-IMC:

- Die verwendete Protokollversion ($protocolVersion$),
- eine Komponenten-ID ($ComponentID$) mit der der PTS-IMC den IMV bzw. der IMV die gewünschten Messungen identifizieren kann,
- eine Nonce ($nonce$) (siehe Kapitel 4.1.2) und
- eine Liste der zusätzlich zu messenden Daten ($listOfMeasure$).

Wobei folgende Bedingungen gelten:

- $minProtocolVersion \leq protocolVersion \leq maxProtocolVersion \vee protocolVersion = 0$
- Wenn die $protocolVersion = 0$ ist, sind der IMC und der IMV inkompatibel zueinander und die Kommunikation ist zwangsläufig gescheitert.
- $nonce$ ist eine 20 Byte große Zahl (siehe SHA-1).
- Die $nonce$ ist für einen Integrity Check Handshake[23] bei allen IMVs identisch.

[23] [14, Kapitel 11]

- Der PTS-IMV sendet eine leere Liste ($listOfMeasure$) zum PTS-IMC, da die nötige Vertrauenskette (TNCC und IMCs) bereits gemessen wurden.

3. Schritt (PTS-IMC → IMV) Anhand von Listen der zu messenden Daten (*listOfMeasure*) der IMVs, lässt der PTS-IMC diese zusätzlichen Daten vom CPTS messen. Anschließend sendet der PTS-IMC für jede verwendete Protokollversion folgendes zurück zu den IMVs:

- Die verwendete Protokollversion ($protocolVersion$) für diese Nachricht und
- den IR ($integrityReport$).

Gesetzt den Fall, zwei IMVs kommunizieren mit dem PTS-IMC über die Protokollversion 1.0 und drei IMVs über die Protokollversion 2.0. Dann schickt der PTS-IMC zwei Nachrichten, wobei eine Nachricht mit der Protokollversion 1.0 und die andere mit der Protokollversion 2.0 anfängt.

IMVs müssen Nachrichten ignorieren, wenn die Protokollversion nicht mit der verwendeten Protokollversion (in Schritt 2) übereinstimmt.

4. Schritt (PTS-IMC ← IMV) Im letzten Schritt überprüfen die IMVs mit Hilfe des SPTS die Messungen im IR. Als letztes können die IMVs noch Informationen zur Entscheidungsfindung zum PTS-IMC senden, indem sie folgendes zum PTS-IMC senden:

- Die Komponenten-ID ($ComponentID$) aus Schritt 2 und
- die Informationen zur Entscheidungsfindung ($resultInformation$).

4.5.3 Measurement

Der CPTS misst wie in den anderen Konzepten die Dynamic Chain of Trust (den TNCC inkl. IMCs). Auch wie im Seperate Konzept (Kapitel 4.4) werden Messungen, die der CPTS durchführt, als Sub-Snapshots unter einem Snapshot zusammengefasst. Die durchgeführten Messungen können anhand einer $ComponentID$, welche im Kapitel 4.5.2 Schritt 2 versendet wurde, identifiziert werden.

Nachdem alle Messungen durchgeführt wurden, wird der IR erstellt und zum Server geschickt (siehe Kapitel 4.5.2 Schritt 3).

4.5.4 Serverside Policy

SPTS Auf der Serverseite ist der PTS, wie im dritten Konzept, nur für generische Überprüfung des IR zuständig und kann daher von allen IMVs aufgerufen werden. Im Kapitel 4.5.5 werden die Regeln für dieses Konzept Definiert.

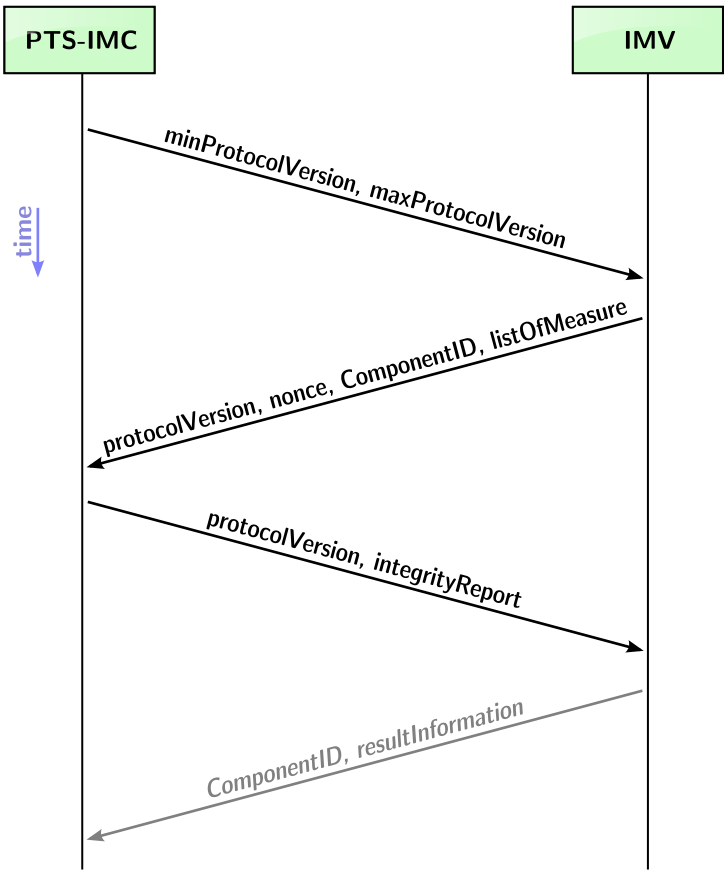


Abbildung 4.16: Sequenzdiagramm vom 4. Konzept

PTS-IMV Der PTS-IMV ist wie im vorherigen dritten Konzept nur für die Überprüfung der zwei Vertrauensketten zuständig.

IMVs Alle weiteren IMVs treffen wie in den anderen Konzepten Aussagen über bestimmte Komponenten und können wie im Seperate Konzept (siehe Kapitel 4.4.4) den SPTS zur Überprüfung des IR nutzen.

TNCS Der TNCS ist für die finale Entscheidungsfindung einer Plattformauthentisierung zuständig und arbeitet wie beim Seperate Konzept (siehe Kapitel 4.4.4).

4.5.5 Regeln

Da der SPTS wie im dritten Konzept (siehe Kapitel 4.4.5) von jedem IMV aufgerufen werden kann, sind auch hier Regeln nötig, die eine partielle Verifikation erlauben.

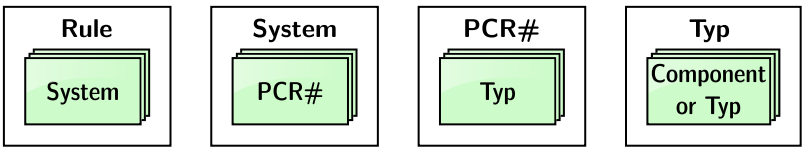


Abbildung 4.17: Aufbau einer erweiterten Regel

Da die von der TCG spezifizierten RIMMs [33] alleine zu statisch sind, werden zusätzliche Regeln definiert, die es ermöglichen mehrere Anwendungsfälle abzudecken. Den Aufbau solch einer Regel zeigt Abbildung 4.17.

System Ein „System“ stellt eine Referenzsystem dar, welches vertrauenswürdig ist. Ein „System“ enthält eine Liste von „PCR#Nr“ die für das Referenzsystem wichtig sind.

PCR#Nr Ein „PCR#Nr“ gibt an wie ein bestimmtes Platform Configuration Register aufgebaut sein muss. Es besteht aus einer Liste von „Typen“ die den Aufbau des PCR beschreiben. Alle „Typen“ müssen in der gegeben Reihenfolge im PCR vorkommen.

Typ Ein „Typ“ stellt eine Komponentenkategorie dar. Solch ein „Typ“ besteht aus einem oder mehreren Kindern. Ein Kind ist ein „Component“ oder „Typ“. Die Kinder können als und- bzw. oder-Verknüpft werden. Bei der und-Verknüpfung müssen alle Kinder im PCR vorkommen und bei der oder-Verknüpfung darf nur ein Kind im PCR vorkommen.

Component Ein „Component“ beschreibt einen Referenzsnapshot. Es ist entweder ein RIMM[24] oder ein Verweis auf ein RIMM.

Abbildung 4.18 zeigt ein Beispiel wie solch eine erweiterte Regel aussehen kann. Dort sind verschiedene Systeme gelistet (Linux, Windows, MacOS, etc.), die auf dem Client zulässig sind. Bei dem Linux-System z. B. muss sich der PCR Nummer 8 aus einem Bootloader (Grub[25] oder LILO[26]) und einem Kernel (Version 2.6.29 oder 2.6.30) bestehen. Solch ein Kernel setzt sich wiederum aus dem eigentlichen Kernel (vmlinuz[27]) und dem Initial RAM Disk[28] zusammen. Während der PCR Nummer 22 nur den PTS enthalten muss. Bei einem Windows Vista System dagegen, muss der PCR Nummer 8 nur den Windows-Bootloader enthalten. Jedoch muss der PCR Nummer 9 aus dem Windows-Kernel und der PCR Nummer 22 aus dem PTS bestehen.

[24] siehe [33]
[25] siehe `http://de.wikipedia.org/wiki/Grand_Unified_Bootloader`
[26] siehe `http://de.wikipedia.org/wiki/Linux_Loader`
[27] siehe `http://en.wikipedia.org/wiki/Vmlinux`
[28] siehe `http://en.wikipedia.org/wiki/Initrd`

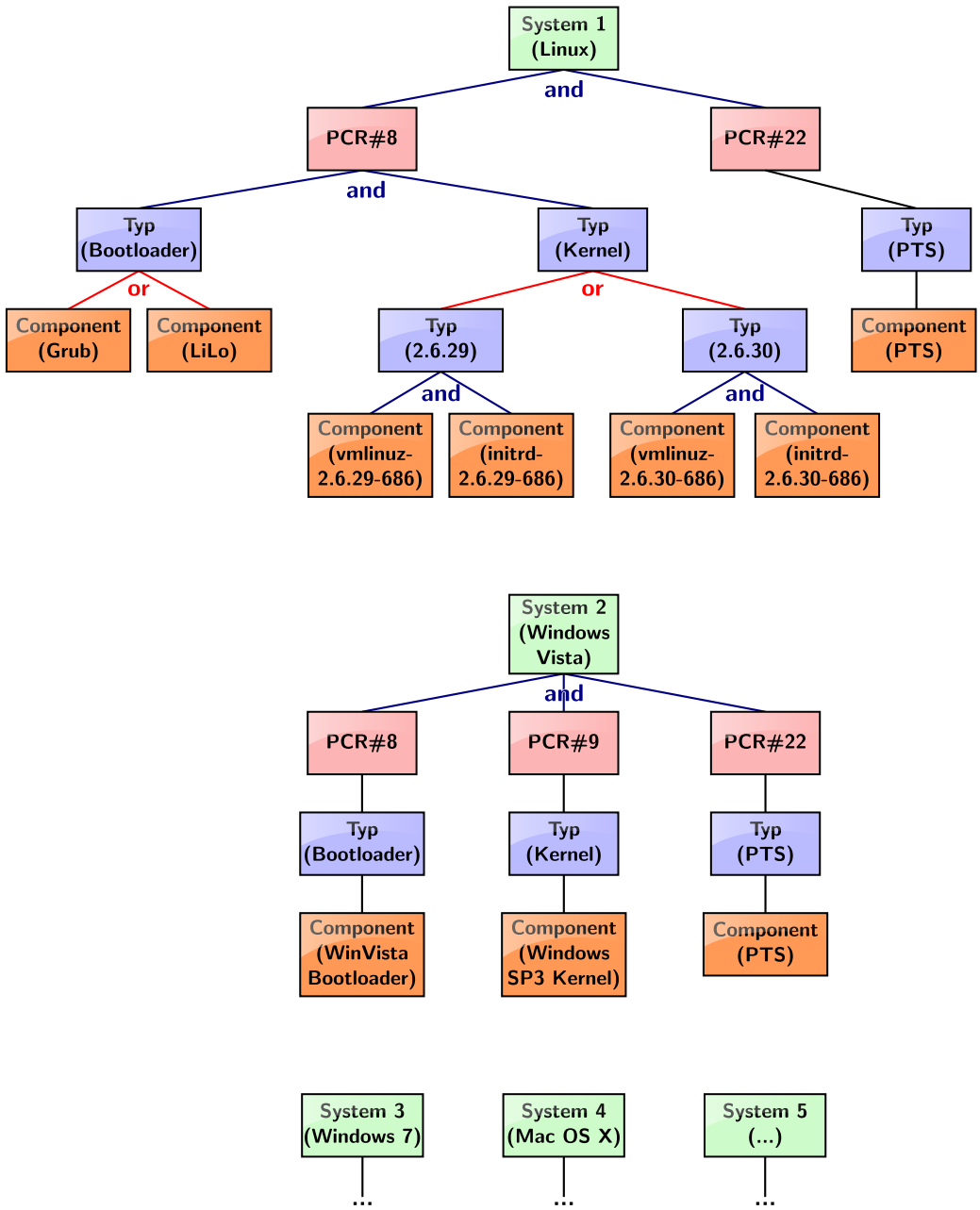


Abbildung 4.18: Ein Beispiel einer erweiterten Regel

Das Beispiel in Abbildung 4.18 ist ein stark vereinfachtes Beispiel, welches nur für das Verstehen der erweiterten Regeln gedacht ist und in der Praxis so keine Verwendung findet. Welche Messungen die PCRs (Nummer 1 bis 8) auf der Clientseite enthalten müssen, findet sich in [34].

Es ist jedoch auch möglich, dass IMVs nur die normalen Regeln (siehe Kapitel 4.4.5) verwenden. Bevor jedoch diese erweiterten Regeln auf den IR angewendet werden können, muss die Authentizität und Integrität, wie im ersten Minimal Konzept (siehe Kapitel 4.2.5), sichergestellt werden.

4.6 Zusammenfassung der vier Konzepte

Tabelle 4.1 zeigt einen Überblick der einzelnen Konzepte.

	Konzept 1 Minimal	Konzept 2 Data Extension	Konzept 3 Seperate	Konzept 4 Cross Over
Dynamic Chain of Trust	TNCC & IMCs	TNCC, IMCs & *Daten*	TNCC, IMCs & *Daten*	TNCC, IMCs & *Daten*
IF-PTS	PTS-IMC & PTS-IMV	PTS-IMC & PTS-IMV	IMCs & IMVs	PTS-IMC & IMVs
IF-M-PTS-IMC-IMV	PTS-IMC ↔ PTS-IMV	PTS-IMC ↔ PTS-IMV	IMCs ↔ IMVs	PTS-IMC ↔ IMVs
Komplexität Client	gering	mittel	hoch	mittel bis gering
Komplexität Server	gering	mittel	hoch	hoch
Größe IR	gering	mittel	hoch	mittel
Serverside Policy	einfach	komplex	einfach	einfach
Regeln	unflexibel	unflexibel	mittel flexibel	flexibel

Tabelle 4.1: Eigenschaften der Konzepte im Überblick.

Dynamic Chain of Trust zeigt, welche Komponenten neben den PTS noch ein Teil der Dynamic Chain of Trust sind (kursiv ist Optional).

IF-PTS gibt an, welche Komponenten mit einem PTS kommunizieren.

IF-M-PTS-IMC-IMV zeigt, welche IMCs mit welchen IMVs über das IF-M-PTS-IMC-IMV kommunizieren.

Komplexität Client gibt den Funktionsumfang und Anzahl der IMCs an.

Komplexität Server gibt den Funktionsumfang und Anzahl der IMVs an.

Größe IR Gibt die gesamte Größe der IRs an, die für die Übertragung notwendig sind.

Serverside Policy gibt an, wie verteilt die Policy einer Komponente ist. Wird die Policy für eine Komponente auf mehrere Komponenten verteilt, ist die Verteilung der Policy komplex.

Regeln geben die Flexibilität der Regeln an.

Das erste Minimal Konzept, ist das einfachste Konzept. Jedoch enthält die Dynamic Chain of Trust nur den TNCC und die IMCs. Eine Clientseite Policy existiert nicht. Die Regeln auf der Serverseite sind unflexibel und die Policy ist nicht verteilt.

Das zweite Data Extension Konzept, erweitert das erste Minimal Konzept, um beliebige Daten messen zu können. Hier existiert eine Clientside Policy für den PTS-IMC. Die Regeln auf der Serverseite sind ebenfalls unflexibel und die Policy für eine Komponente ist auf mehrere Komponenten verteilt.

Das dritte Seperate Konzept, verteilt die PTS Funktionalität auf die einzelnen IMC/IMV-Paare auf. Da der PTS-IMC keine weiteren Daten misst, benötigt dieser keine Clientside Policy. Durch das aufteilen auf die einzelnen IMC/IMV-Paare müssen die Regeln ein wenig flexibler sein. Jedoch ist die Policy nicht mehr verteilt.

Das vierte und letzte Cross Over Konzept, minimiert auf der Clientseite die Komplexität und maximiert auf der Serverseite die Flexibilität. So kommuniziert der PTS-IMC mit allen IMVs, welche die PTS Funktionalität nutzen wollen. Auf der Clientseite existiert eine Policy und auf der Serverseite ist die Policy nicht verteilt.

5 Implementierung

Diese Arbeit enthält eine prototypische Implementierung des im Abschnitt 4.5 vorgestellten Konzeptes. Das Ziel dieser Implementierung ist, zu einem das Konzept auf der Implementierungsebene zu definieren, als auch eine Grundlage für die Weiterentwicklung im Rahmen des tNAC Projektes[1] zu schaffen. Diese Implementierung setzt auf das TNC-Projekt der FHH (TNC@FHH)[2] auf. Da das TNC@FHH-Projekt ausschließlich in C++[3] entwickelt ist, wird auch diese Implementierung ausschließlich in C++ entwickelt.

Die Implementierung der einzelnen Projekte aus Abschnitt 5.3 befinden sich auf der beiliegenden CD. Eine Beschreibung zum Kompilieren befindet sich ebenfalls auf der CD.

5.1 Entwicklungsumgebung

Da das TNC@FHH-Projekt intern die Entwicklungsumgebung (IDE) Eclipse[4] verwendet, bietet es sich an, diese Implementierung ebenfalls in Eclipse zu entwickeln.

Damit unter Eclipse komfortabel mit C++ entwickeln werden kann, wird neben der Standard Eclipse-Software auch das C/C++ Development Tooling (CDT)[5] benötigt. Diese beiden Komponenten können auf der Eclipse-Homepage gemeinsam heruntergeladen werden[6].

Wird mit Eclipse standardmäßig ein C++ Projekt erstellt, erzeugt Eclipse ein Verzeichnis mit Makefiles[7], welche zum Kompilieren und Linken genutzt werden. Diese Makefiles sind jedoch nur für das aktuelle Projekt (abhängig vom Projekt-Pfad und Konfiguration des Systems) geeignet, so dass sie auf einem anderen System nicht korrekt arbeiten. Dies hat zur Folge, dass ohne Eclipse das Projekt nicht ohne weiteres Kompiliert werden kann.

[1] http://www.tnac-project.org/
[2] http://trust.inform.fh-hannover.de/
[3] http://de.wikipedia.org/wiki/C%2B%2B
[4] http://www.eclipse.org/, http://de.wikipedia.org/wiki/Eclipse_(IDE)
[5] http://www.eclipse.org/projects/project_summary.php?projectid=tools.cdt
[6] http://www.eclipse.org/downloads/
[7] http://de.wikipedia.org/wiki/Make

Das TNC@FHH-Projekt hatte bisher für dieses Problem zusätzliche Makefiles (auf Basis der Eclipse Makefiles) erstellt, die mit Hilfe eines für dieses Projekt entwickelten Configure-Scriptes konfiguriert werden konnten. Da diese Lösung jedoch recht fehleranfällig und wenig dynamisch ist, wird dies nur als Übergangslösung verwendet, bis eine bessere Lösung gefunden und umgesetzt wird.

Ein weiterer Ansatz ist, ein C++ Projekt komplett mit Hilfe eines Build-Tools (wie z. B. GNU build system[8], auch Autotools genannt) zu verwalten. Dies hat den Vorteil, dass das Projekt auch ohne Eclipse entwickelt bzw. kompiliert werden kann.

Für die Implementierung dieser Arbeit hat der Autor sich für die zweite Lösung entschieden, da solche Build-Tools flexibler und ausgereifter sind. Als Build-Tool kommt CMake[9] zum Einsatz. CMake ist gegenüber Autotools einfacher und nicht so fehleranfällig und wird mittlerweile auch in großen Projekten wie KDE (ab Version 4.0)[10] eingesetzt.

5.1.1 CMake

Bei CMake existiert in jedem Verzeichnis, in der eine Aktion ausgeführt werden soll, eine Datei namens „CMakeLists.txt“[11]. In dieser Datei stehen Aktionen, die für dieses Verzeichnis ausgeführt werden sollen. Da die Projekte relativ klein sind, befindet sich nur eine CMakeLists.txt in jedem Projektverzeichnis.

Um ein CMake Projekt zu kompilieren, wird zunächst CMake aufgerufen, welche Makefiles im aktuellen Verzeichnis erstellt. Dies ist vergleichbar mit dem Configure-Skript bei Autotools. CMake hat jedoch den Vorteil, dass mehrere Makefiles für verschiedene Konfiguration eines Projektes parallel erstellt werden können. Anschließend wird wie gewohnt make[12] für die Makefiles aufgerufen und das Projekt damit kompiliert und gelinkt.

CMake überprüft beim Erstellen der Makefiles, ob die nötige Software zum Kompilieren, wie z. B. der C++-Compiler, installiert ist. Auch kann überprüft werden, ob benötige Bibliotheken bzw. die dazugehörigen Headerfiles existieren. CMake enthält dafür standardmäßig Funktionen für einige weitverbreitete Bibliotheken, wie z. B. Boost. Funktionen zum Prüfen von weiteren Bibliotheken können hinzugefügt werden und befinden sich im Projektverzeichnis unter „cmake_modules“.

Durch CMake ist es möglich, die Projekte auch ohne Eclipse zu entwickeln und zu kompilieren. Ein weiterer Vorteil von CMake ist, dass alle Projekte auf einmal kompiliert werden können.

[8] `http://en.wikipedia.org/wiki/GNU_build_system`
[9] `http://www.cmake.org/`, `http://de.wikipedia.org/wiki/CMake`
[10] Quelle: `http://dot.kde.org/2006/06/30/kde-switches-cmake`
[11] Groß- und Kleinschreibung beachten
[12] `http://de.wikipedia.org/wiki/Make`

5.1.2 Eclipse

Zwar beherrscht CMake ab der Version 2.6.0[13] auch das erstellen von Eclipse CDT-Projekten, aber die ersten Versuche haben gezeigt, das solche CDT-Projekte unter Eclipse Probleme mit der Syntaxvervollständigung haben. Da einer der Stärken von Eclipse die Syntaxvervollständigung ist, ist es wünschenswert, dass Eclipse mit CMake in diesem Punkt problemlos arbeitet.

Die Projekte werden zunächst als Standard CDT-Projekt erstellt. Anschließend wird Eclipse die Aufgabe entzogen, die Makefiles des Projektes zu verwalten. Bevor über Eclipse das Projekt kompiliert werden kann, müssen die Makefiles für Eclipse mit CMake erzeugt werden. Dies kann erreicht werden, indem CMake mit entsprechenden Parametern manuell in der Konsole aufgerufen wird. Eclipse bietet auch an, definierte Befehle auszuführen (sogenannte „Make Targets“). Diese Targets können dann z. B. über einen Mausklick ausgeführt werden. Dadurch kann das Ausführen von CMake halb automatisiert werden. Wird das Projekt z. B. von einem Subversion (SVN)[14] Repository neu heruntergeladen, führt man einmal das entsprechende Target aus und kann wie gewohnt mit Eclipse arbeiten.

Im Normalfall erzeugt Eclipse für jede Build-Konfiguration ein Verzeichnis direkt im Projektverzeichnis, indem die Makefiles liegen. Um die Übersichtlichkeit zu erhöhen, werden diese Verzeichnisse jedoch in einem extra Verzeichnis namens „build“ zusammengefasst. Dies trennt die computerspezifische Konfiguration besser von der übrigen Konfiguration.

5.2 Externe Bibliotheken

Einige Implementierungen dieser Arbeit benötigen neben den Standardbibliotheken noch weitere Bibliotheken. Im Folgenden werden die für die prototypische Implementierung verwendeten zusätzlichen Bibliotheken vorgestellt.

5.2.1 Apache Log4cxx

Im TNC@FHH-Projekt wurden alle Logausgaben über ein eigens entwickeltes (siehe [22, Kapitel 3.2.3]) Loggingframework getätigt. Dieses Framework bietet jedoch in einigen Fällen zu wenig Flexibilität. So ist es z. B. nicht möglich, die Logausgaben parallel auf mehrere Streams auszugeben oder das Ausgabeformat zu ändern. Auf Grund dieser Nachteile ist es wünschenswert, ein Loggingframework zu verwenden, welches mehr Flexibilität bietet.

[13] Quelle: `http://www.cmake.org/Wiki/Eclipse_CDT4_Generator`

[14] `http://subversion.tigris.org/`, `http://de.wikipedia.org/wiki/Subversion_(Software)`

Das TNC@FHH-Projekt will demnächst auf ein anderes Loggingframework umschwenken. Der wohl am vielversprechendste Kandidat ist das log4cxx-Framework[15] von der Apache Software Foundation. Log4cxx ist eine C++-Portierung des weitverbreiteten Loggingframework Log4j[16], welches ebenfalls von der Apache Software Foundation entwickelt wird.

Log4cxx bietet die Möglichkeit, sämtliche Einstellungen in einer separaten Konfigurationsdatei auszulagern. Die Konfigurationsdatei ist kompatibel zu der Konfigurationsdatei von Log4j. Für die prototypische Implementierung wird Log4cxx in der Version 0.9 benötigt.

5.2.2 Boost-Thread

Die Spezifikation [8] lässt asynchrone Funktionsaufrufe zu. Deswegen werden Threads benötigt. Die Boost-Thread[17] Bibliothek ist unter C++ die geeignetste Methode für Threadprogrammierung. Ein wesentlicher Punkt, der dafür spricht ist, dass Boost-Thread z. Zt. im C++ Draft Standard [35] aufgenommen wurde und in der nächsten C++-Version enthalten sein wird. Für diese Arbeit werden Boost-Threads ab der Version 1.35 benötigt.

5.3 Projekte

Die Implementierung dieser Arbeit besteht aus mehreren Programmen. Auf der Clientseite existieren zu einem ein PTS-IMC und PTS (CPTS). Auf der Serverseite existieren zudem zwei IMVs, ein PTS-IMV und BrowserIMV, und ein PTS (SPTS). Jedes dieser Programme wird in einem eigenen Projekt entwickelt. Softwarekomponenten, die in mehreren Programm genutzt werden können, wurden als Bibliotheken in eigene Projekte ausgelagert. Im Folgenden werden die einzelnen Projekte beschrieben. Die Implementierungen der Projekte sind in der beigelegten CD enthalten.

5.3.1 TNCUtil

Das TNCUtil-Framework wird im Rahmen von TNC@FHH entwickelt und ist ein wesentlicher Bestandteil vieler TNC@FHH Komponenten. So nutzen z. B. bisher alle von der FHH entwickelten IMVs und IMCs dieses Framework.

Das TNCUtil-Framework konnte bisher nur mit Hilfe von Eclipse entwickelt werden. Damit dieses Framework den Anforderungen im Kapitel 5.1 entspricht, wird das

[15] `http://logging.apache.org/log4cxx/`
[16] `http://logging.apache.org/log4j/`
[17] `http://www.boost.org/`

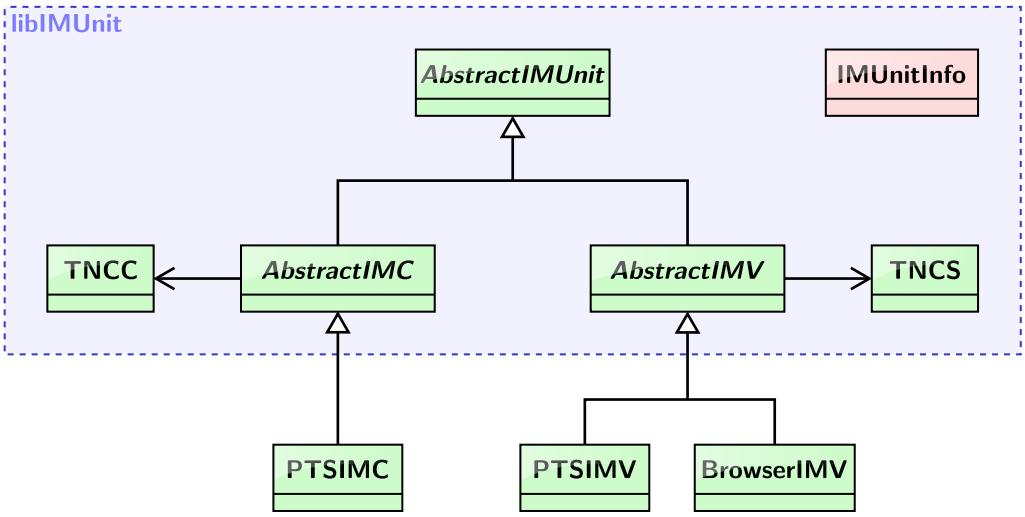


Abbildung 5.1: Klassendiagramm vom IMUnit-Projekt

TNCUtil-Framework für diese Arbeit auf CMake umgestellt, da im Framework einige Veränderungen vorgenommen wurden.

Das TNCUtil-Framework wurde bisher als statische Bibliothek (Static Library) erzeugt. Da dieses Framework jedoch von mehreren Projekten genutzt wird, bietet sich für das TNCUtil-Framework eine dynamische Bibliothek (Shared Library) an. Einige Codefragmente lassen sich jedoch nicht sinnvoll als Shared Library kompilieren. Da diese Fragmente ausschließlich den IML (Integrity Measurement Layer) Bereich betreffen, wird dieser Bereich in ein neues Projekt 5.3.2 ausgelagert.

Das TNCUtil-Framework wurde, bis auf wenige Bugfixes, nicht weiter angepasst. So benutzt das Framework noch das alte Loggingframework anstatt Log4cxx 5.2.1.

5.3.2 IMUnit

IMUnit war ursprünglich ein Bestandteil vom TNCUtil-Framework (siehe vorheriges Kapitel) und enthält eine abstrakte Implementierung der IF-IMC und IF-IMV Schnittstelle. Dieses Projekt wird als statische Bibliothek erzeugt und soll die Implementierung eines IMC/IMV vereinfachen. Für das IMUnit-Projekt wird weiterhin die TNCUtil (Kapitel 5.3.1) Bibliothek benötigt.

Der Gedanke hinter IMUnit ist ein objektorientiertes Konzept. Das vereinfachte Klassendiagramm des IMUnit-Projektes stellt Abbildung 5.1 dar. Für jede Connection-ID[18] wird eine Instanz des IMC bzw. IMV erstellt, welche von der Klasse AbstractIMC

[18]Repräsentiert eine Netzwerkverbindung zwischen einem TNCC und TNCS Paar (siehe [19, Kapitel 3.4.2.2] bzw. [20, Kapitel 3.4.2.2]).

bzw. AbstractIMV erben muss. Die IMUnit Library nimmt die Funktionsaufrufe des TNCC bzw. TNCS entgegen und koordiniert allgemeine Funktionen des IMC/IMV. Zu diesen Funktionen zählen:

- Zuweisen der Funktionszeiger (Dynamic Function Binding) vom TNCC bzw. TNCS [19],
- automatische Synchronisation der IMC/IMV-Funktionsaufrufe zur Threadsicherheit (Thread Safety),
- Erzeugen und Verwalten von IMC bzw. IMV Instanzen für jede gültige ConnectionID,
- abbilden der C-Funktionen auf C++-Methoden und umgekehrt und
- Exceptionhandling vor der C-Schnittstelle.

Die IF-IMC/IMV-Spezifikationen [19] und [20] definieren C-Funktionen, welche für jede IMC/IMV-Library einzeln implementiert werden müssen. Da diese C-Funktionen im IMUnit-Projekt implementiert sind, müssen diese Funktionen direkt in den IMC/-IMV mit einkompiliert werden. Da die IMUnit Library globale Variablen nutzt, die bei jedem IMC/IMV separat genutzt werden müssen, wird das IMUnit-Projekt als Static Library kompiliert. Damit die globalen Funktionen und Variablen des IMUnit-Projektes für jeden IMC/IMV mehrfach in den Speicher geladen werden, müssen die IMCs/IMVs korrekt geladen werden. Unter Linux z. B. müssen die IMCs und IMVs bei `dlopen()` mit dem Flag `RTLD_LOCAL` geladen werden. Wenn zum Laden von IMCs/IMVs Libtool[20] in der Version <= 1.5 verwendet wird, ist es von der Linux-Distribution abhängig, ob Libtool (über `dlopen`) eine Library mit dem dem Flag `RTLD_LOCAL` lädt. Ab der Libtool Version >= 2.2 ist es jedoch möglich, dieses Standardverhalten zu ändern [21].

Da der TNCS der TNC@FHH Implementierung in der Version 0.5 Libtool 1.5 nutzt, funktioniert IMUnit damit nicht auf jeder Linux-Distribution. So lässt sich diese Kombination z. B. unter Debian[22] 5.0 (Lenny) und Ubuntu[23] 9.04 (Jaunty Jackalope) nutzten, jedoch nicht unter Fedora 11[24] (Leonidas). Dieser Bug wird erst in der nächsten Version IMUnit 0.6.0 behoben sein.

Für eine Implementierung eines IMC/IMV welche die IMUnit-Library benutzt sind folgende Punkte zu beachten:

[19] siehe [19, Kapitel 4] bzw. [20, Kapitel 4]
[20] `http://www.gnu.org/software/libtool/`
[21] `http://www.gnu.org/software/libtool/manual/html_node/Libltdl-interface.html`
[22] `http://www.debian.org/`
[23] `http://www.ubuntu.com/`
[24] `http://fedoraproject.org/`

- Die Funktion `TNCUtil::iml::getIMUnitInfo()` muss implementiert werden und liefert wichtige Informationen zum aktuellen IMC bzw. IMV.

- Auch muss die Funktion `TNCUtil::iml::createNewIMCInstance()` beziehungsweise `TNCUtil::iml::createNewIMVInstance()` implementiert werden. Diese Funktionen liefern beim Aufruf eine neue Instanz eines konkreten IMC bzw. IMV.

- Die Datei `imc/IFIMCImpl.cpp` (beim IMC) oder `imv/IFIMVImpl.cpp` (beim IMV) müssen beim IMC/IMV mit hineingelinkt werden. Dies kann am einfachsten erreicht werden, indem im IMC `#include <imc/IFIMCImpl.cpp>` bzw. im IMV `#include <imv/IFIMVImpl.cpp>` im Quelltext einfügt wird.

Die Klasse AbstractIMUnit wurde um eine Variable `round` erweitert, welche die Anzahl der verschickten TNCCS-Messages[25] während des aktuellen Handshakes liefert. Dadurch kann z. B. ein IMC/IMV feststellen welcher Kommunikationsschritt als nächstes an der Reihe ist.

5.3.3 Libif-pts

Die IF-PTS Spezifikation [8] definiert die Kommunikation zwischen einem PTS und einem Client (z. B. ein PTS-IMC) welcher die Funktionalitäten des PTS nutzen will. In dieser Spezifikation wird festgelegt, dass die Kommunikation über eine Interprozesskommunikation (IPC) stattfinden muss. Die IF-PTS Schnittstelle besteht aus 43 Funktionen, welche über die IPC aufgerufen werden. Die Parameter und Rückgabewerte der Funktionen enthalten Teils komplexe Datentypen. Die Funktionsaufrufe und Datentypen müssen zur Interprozesskommunikation zuerst serialisiert und anschließend wieder geparst werden. Da diese Funktionalität sehr ähnlich und Teilweise sehr umfangreich ist, wurde die Libif-pts Bibliothek entwickelt, welche automatisch die Kommunikation über die IPC Schnittstelle realisiert. Diese Bibliothek nutzt als Loggingframework Log4cxx (siehe Kapitel 5.2.1) und Threads aus der Boost-Library (siehe Kapitel 5.2.2).

Die Spezifikation sieht als Standard IPC, Named Pipe[26] vor. Unter Windows ist es möglich, mit Hilfe von Named Pipe eine bidirektionale Kommunikation zwischen zwei verschiedenen Prozessen aufzubauen. Unter Unix/Linux jedoch sind Named Pipes ausschließlich unidirektional. Das Pendant zu Named Pipes unter Windows in Unix/Linux sind Unix Domain Sockets[27]. Auf Grund der Verwendung von Named Pipes in der Spezifikation nimmt der Autor an, dass bei Unix/Linux Unix Domain Sockets anstatt Named Pipes gemeint sind.

[25] [18]
[26] `http://en.wikipedia.org/wiki/Named_pipe`
[27] `http://en.wikipedia.org/wiki/Unix_domain_socket`

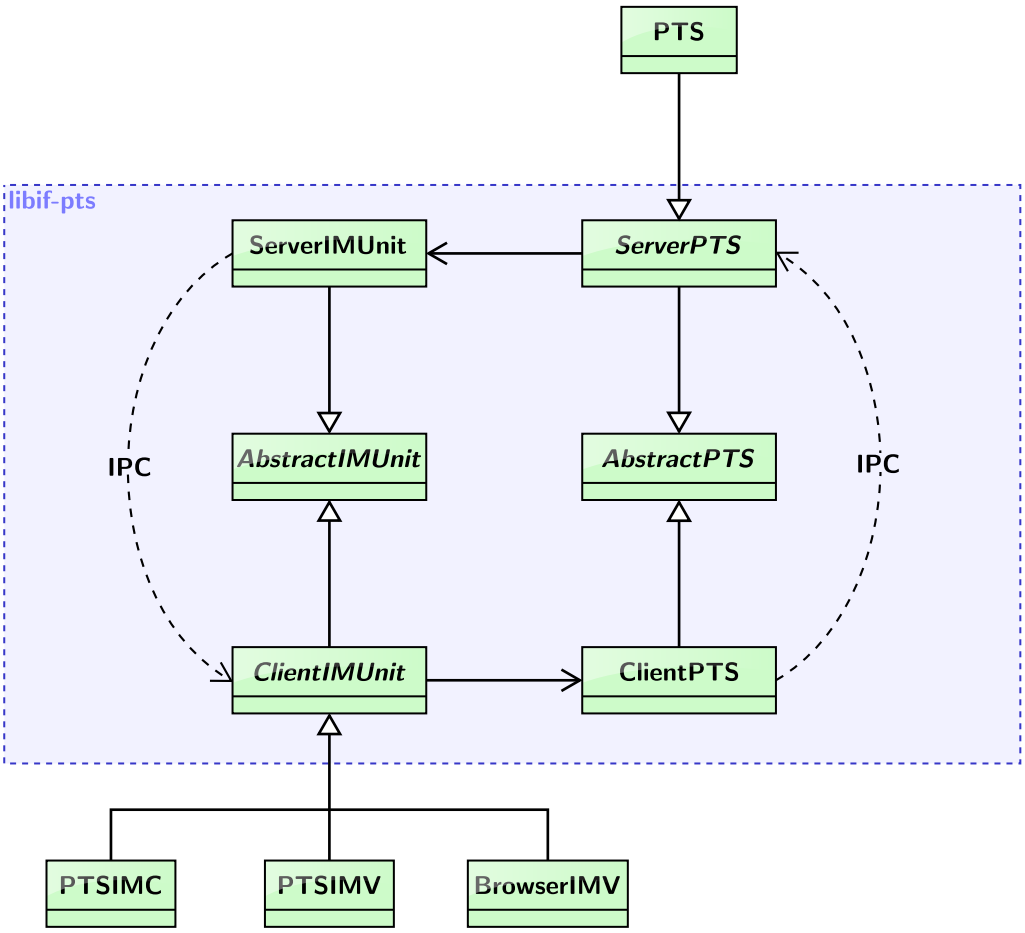


Abbildung 5.2: Klassendiagramm vom Libif-pts

Die Bibliothek Libif-pts wurde komplett in C++ implementiert, da ein objektorientierter Ansatz eine einfache und elegante Umsetzung ermöglicht. Auf der Clientseite muss nur von der Klasse `ifpts::client::ClientIMUnit` geerbt werden. Mit der Methode `connect()` wird eine Verbindung über einen IPC Kanal zum PTS aufgebaut. Anschließend kann der PTS über die Methode `getPts()` direkt angesprochen werden. Ruft der PTS eine asynchrone Funktion beim Client auf, wird der Aufruf automatisch zu den dazugehörige abstrakten Methode im `ifpts::client::ClientIMUnit` weitergeleitet.

Auf der PTS-Seite muss von der abstrakten Klasse `ifpts::server::ServerPTS` geerbt werden. Wird die Funktion `ifpts::server::listenOnIpc` aufgerufen, wartet diese Funktion in einer Endlosschleife auf neue Clients die einen neuen IPC Kanal zum PTS aufbauen. Verbindet sich ein Client, wird mit Hilfe einer Funktion (zweiter Parameter der `listenOnIpc`-Funktion) eine neue Instanz des `ServerPTS` erzeugt und

alle Clientanfragen in einem neuen Thread automatisch an die passende Methode im `ServerPTS` weitergeleitet. Es wird also für jeden Client eine eigene PTS-Instanz erzeugt.

Abbildung 5.2 zeigt das Klassendiagramm vom Libif-pts.

Serialisieren und Parsen

Um Funktionen über IPC aufzurufen, müssen die Funktionsaufrufe serialisiert und auf der Gegenstelle geparst werden. Die IF-PTS Spezifikation [8], unterscheidet sich ein wenig von den klassischen Funktionsaufrufen in der Programmierwelt. So sind die Rückgabewerte auch wieder Funktionsaufrufe. Funktionsaufrufe beim PTS werden immer Request und beim Client Response genannt. Zu jedem Request-Aufruf gibt es ein passenden Response-Aufruf. So wird nach jedem Request-Aufruf den dazugehörigen Response-Aufruf getätigt. Ein Response ist somit vergleichbar mit den Rückgabewerten einer klassischen Funktion. Es gibt jedoch einige Response-Aufrufe, die zu keinem Request-Aufruf gehören. Diese werden dann als asynchrone Requests bezeichnet. Die Responses und Requests können als Parameter einfache und/oder komplexe Datentypen enthalten. Die Bibliothek Libif-pts bringt verschiedene Funktionen mit (`conversion.h`), die verschiedene Operationen auf diese Datentypen ausführen können:

ifpts::size_of liefert die Größe eines komplexen Datentyp in Bytes, welche für die Serialisierung laut PTS-Spezifikation benötigt werden.

ifpts::write_in_buffer schreibt einen PTS-Datentyp in einen Byte-Buffer und gibt die Anzahl der geschriebenen Bytes zurück (siehe `sizeof` bzw. `ifpts::size_of`). Diese Funktionen reserviert keinen Speicher, so dass der Aufrufer für genügend Platz im Buffer sorgen muss. Es wird automatisch von Host Byte Order in Network Byte Order konvertiert.

ifpts::read_from_buffer ließt aus einem Byte-Buffer einen speziellen Datentyp und gibt die gelesenen Bytes im Buffer zurück. Diese Funktion überprüft auch ob die Länge des Byte-Buffer für diesen Datentyp ausreicht und wirft eine `std::runtime_error`-Exception falls dem nicht so ist. Bei Datentypen mit variabler Länge (die Zeiger auf einen Byte-Buffer beinhalten) wird neuer Speicher mit `new[]` reserviert. Tritt ein Fehler nach dem reservieren des Speichers auf, so wird der dort reservierte Speicher automatisch freigegeben und der Zeiger auf `NULL` gesetzt. Tritt ein Fehler jedoch vor dem `new[]` auf, so wird der Zeiger nicht verändert. Es ist daher empfehlenswert alle Zeiger in der Zielvariable vor dem Aufruf der `read_from_buffer`-Funktion mit `NULL` zu initialisieren. Da Speicher der mit `new[]` in dieser Funktionen reserviert wurde nicht automatisch wieder freigegeben wird (ausgenommen im Fehlerfall), muss der Aufrufer selber dafür

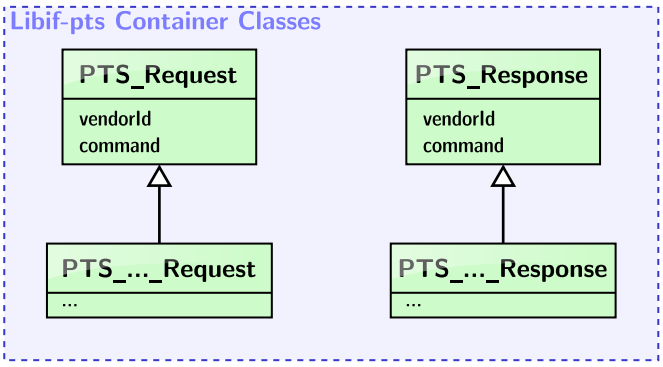


Abbildung 5.3: Libif-pts Container Klassen zum Serialisieren und Parsen

sorge tragen, dass dies geschieht. Diese Funktionen konvertieren automatisch von Network Byte Order in Host Byte Order.

ifpts::create_... Zusätzlich existieren auch Funktionen die das Erzeugen von komplexen Datentypen erleichtern.

Das Serialisieren und Parsen der Requests bzw. Responses wird über Klassen realisiert. Dafür existieren zwei abstrakte Klassen `PTS_Request` und `PTS_Response` (siehe Abbildung 5.3). Zum Identifizieren des Kommandos, enthalten die Klassen eine Vendor ID (`vendorId`) und ein Command Ordinal (`command`) [28]. Diese Klassen enthalten zudem eine send-Methode (`send_request` bzw. `send_response`), welche den Request bzw. Response über einen IPC Kanal[29] verschickt.

Konkrete Requests erben von `PTS_Request` und konkrete Response erben von `PTS_Response`. Diese Unterklassen serialisieren bzw. parsen nur noch die Parameter, die für jede Unterklasse verschieden sind. Einzige Ausnahme bilden `PTS_Generic_Request` und `PTS_Generic_Response`. Diese beiden Klassen werden für alle Request- bzw. Response-Aufrufe ohne Parameter genutzt. Sie können jedoch anhand der Vendor ID und Command Ordinal identifiziert werden. Alle Unterklassen von `PTS_Request` und `PTS_Response` implementieren die Methode `send`, welche die Parameter serialisiert und anschließend die Methode `send_request` bzw. `send_response` aufruft. Diese Unterklassen erhalten neben den Standardkonstruktoren einen Konstruktor, der aus einem Byte-Stream die Parameter (und damit den Inhalt der Klasse) parst.

Aus Zeitgründen konnten jedoch nicht alle Request- bzw. Response-Klassen implementiert werden. Zwar existieren alle Klassen, jedoch enthalten diese keine Funktionalität. Deshalb wurden diese Klassen noch nicht komplett ins Projekt integriert. Diese

[28] siehe [8, Kapitel 6.1 & 6.2]
[29] Der IPC Kanal wird in der Methode als File Descriptor (`fd`) dargestellt.

Klassen zu implementieren und zu integrieren ist auf Grund des Aufbau des Projektes nur noch eine „Fleißarbeit“.

Abweichung der Spezifikation

Die Libif-pts Implementierung weicht an einer Stelle von der IF-PTS Spezifikation ab. Der Datentyp (struct[30]) `PTS_String` besteht wiederum aus zwei Variablen, `size` und `stringData`. Laut Spezifikation enthält `size` die Anzahl der in `stringData` enthalten UTF-8[31] Zeichen. Da ein UTF-8 Zeichen 1 bis (Theoretisch) 8 Byte lang sein kann, ist es nicht möglich nur anhand von `size` den benötigten Speicherplatz zu ermitteln. Da diese Definition äußerst ungünstig ist, enthält `size` in Libif-pts, entgegen der Spezifikation, die Anzahl von Bytes in `stringData`. Der Inhalt von `stringData` ist jedoch weiter in UTF-8 kodiert.

FHH Erweiterungen

Die Spezifikation erlaubt es, die IF-PTS Schnittstelle um eigene Funktionen zu erweitern. Da die Konzepte im Kapitel 4 teilweise Funktionalität benötigen, die die aktuelle Spezifikation nicht bietet, wird die Schnittstelle um weitere Funktionen erweitert. Der Klassenname für Request und Response ist wie folgt aufgebaut: „`FHH_..._Request`“ oder „`FHH_..._Response`“. Die Vendor ID dieser IPC-Aufrufe bekommen die Nummer 32939 (in Hexadezimal 0x80AB) [32]. Für das hier umgesetzte Konzept in Kapitel 4.5 wird die Schnittstelle um folgende IPC-Aufrufe erweitert:

FHH_RandomizeNonce (Command Ordinal: 0) Diese Funktion randomisiert die Nonce für eine ConnectionID.

FHH_GetNonce (Command Ordinal: 1) Diese Funktion gibt die randomisierte Nonce für eine bestimmte ConnectionID zurück. Diese Funktion liefert bei allen PTS-Instanzen für eine ConnectionID immer die gleiche Nonce zurück, bis zum nächsten FHH_RandomizeNonce Aufruf (siehe Kapitel 4.5.2 Schritt 2).

5.3.4 PTS-IMC

Auf der Clientseite (TNCC) existiert der PTS-IMC, welcher den IR zum Server (TNCS) schickt (siehe Kapitel 4.5). Das Projekt nutzt für das Logging Log4cxx (siehe Kapitel 5.2.1). Der PTS-IMC muss sowohl das IF-IMC [19] Protokoll als auch das IF-PTS [8] Protokoll unterstützen. Das IF-IMC Segment übernimmt die IMUnit-Bibliothek aus Kapitel 5.3.2 und das IF-PTS Segment die Libif-pts-Bibliothek aus Kapitel 5.3.3.

[30] `http://de.wikipedia.org/wiki/Verbund_(Datentyp)`
[31] `http://de.wikipedia.org/wiki/UTF-8`
[32] Quelle: `http://www.iana.org/assignments/enterprise-numbers`

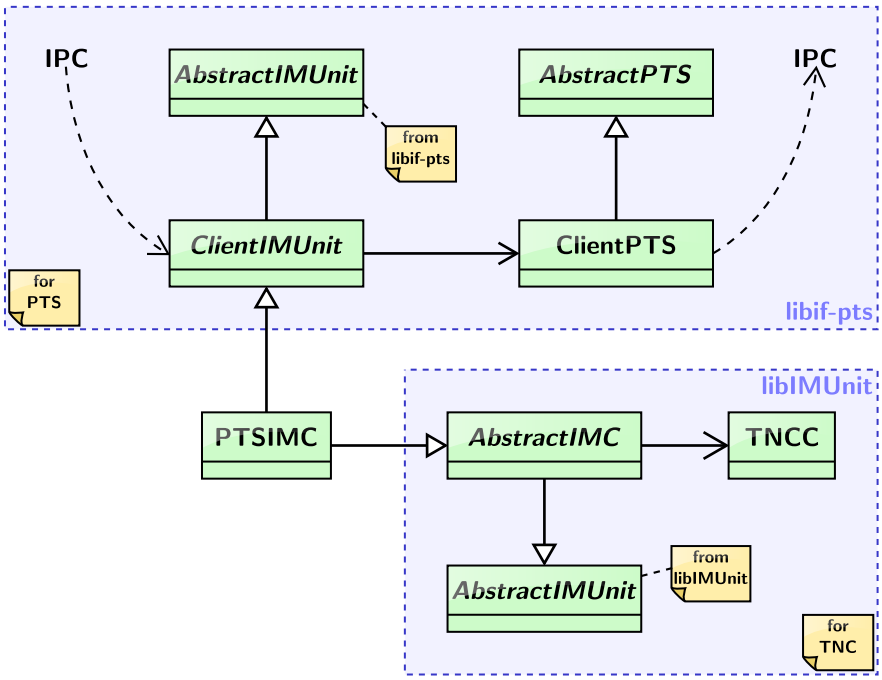


Abbildung 5.4: Klassendiagramm vom PTS-IMC

Projektaufbau

Dieses Projekt besteht aus nur einer Klasse, dem `PTSIMC`. Diese Klasse wiederum erbt von zwei Klassen:

- Einmal von der Klasse `TNCUtil::iml::AbstractIMC` aus dem IMUnit Projekt und
- von der Klasse `ifpts::client::ClientIMUnit` aus dem Libif-pts Projekt.

Damit das IMUnit-Projekt die IF-IMC Aufrufe entgegennehmen kann, wird die `IFIMCImpl.cpp` aus dem IMUnit Projekt Kapitel 5.3.2 inkludiert mit:

```
#include <imc/IFIMCImpl.cpp>
```

Zusätzlich werden die zwei Methoden `TNCUtil::iml::getIMUnitInfo()` und `TNCUtil-::iml::createNewIMCInstance()` implementiert (siehe Kapitel 5.3.2).

Der konkrete Aufbau der Nachrichten zwischen den PTS-IMC und IMVs (siehe Kapitel 4.5.2), ist im Kapitel 5.4 definiert.

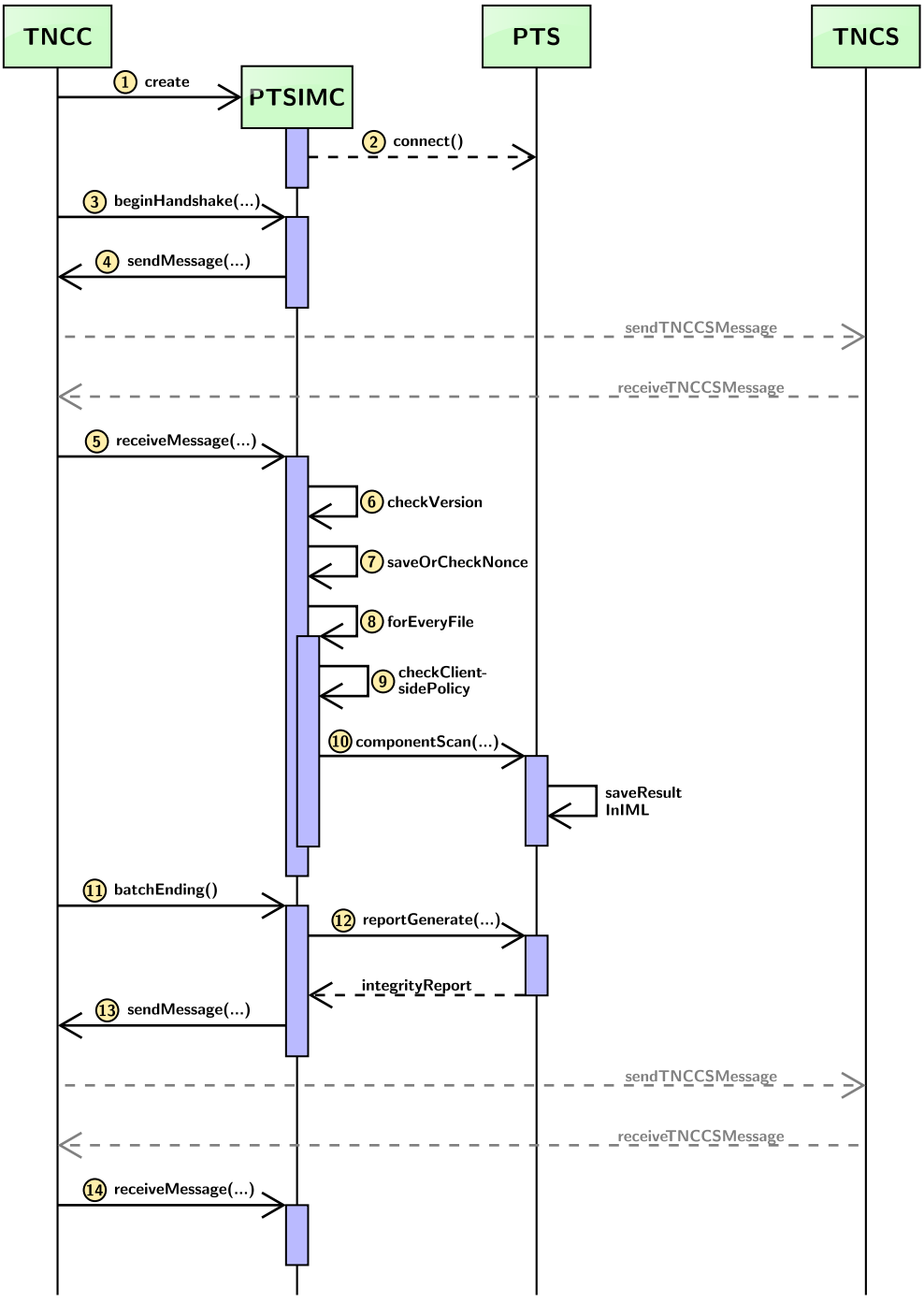


Abbildung 5.5: Sequenzdiagramm des PTS-IMCs

Programmablauf

Abbildung 5.5 zeigt den Ablauf eines Integrityhandshakes aus der Sicht der PTSIMC Klasse. Wird eine neue Verbindung zu einem TNC-Port (z. B, ein 802.1X Port) erstellt, so wird eine neue PTS-IMC Instanz erstellt (1). Dabei wird eine IPC-Verbindung zum PTS hergestellt (2).

Beim Beginnen eines neuen Handshakes, wird der PTS-IMC komplett zurückgesetzt (3) und die erste Nachricht (Kapitel 4.5.2 Schritt 1) zu den IMVs verschickt (4). Anschließend ist die erste Runde zu Ende.

In der zweiten Runde Empfängt der PTS-IMC die Nachrichten von den IMVs (Kapitel 4.5.2 Schritt 2) (5). Anschließend wird überprüft, ob die Version der empfangenen Nachricht unterstützt wird (6). Wurde für diesen Handshake schon eine Nonce empfangen, so wird diese Nonce gespeichert. Ansonsten wird die empfangene Nonce mit der gespeicherten Nonce verglichen (7). Als letztes in dieser Runde wird die Liste der zu messenden Dateien ausgewertet. Diesbezüglich wird für jede zu messende Datei folgendes ausgeführt (8):

- Die Überprüfung der Clientside Policy (9) [33] und
- das Messen der Datei mit Hilfe des PTS (10).

Nachdem alle Nachrichten von den IMVs empfangen wurde (11), fordert der PTS-IMC den IR vom PTS an (12). Anschließend sendet der PTS-IMC den IR zu den IMVs (siehe Kapitel 4.5.2 Schritt 3) (13).

Im letzten Schritt empfängt der PTS-IMC Informationen über die Entscheidungsfindung der IMVs (14). Da diese letzte Nachricht von den IMVs optional ist, werden diese Nachrichten im Rahmen der prototypischen Implementierung ignoriert.

5.3.5 PTS-IMV

Auf der Serverseite (TNCS) existiert der PTS-IMV, welcher die Static Chain of Trust (siehe Kapitel 4.1.1) und den TNCC inkl. IMCs (welche Teile der Dynamic Chain of Trust bilden) verifiziert. Diese Verifikation wird anhand des IR durchgeführt, der von der Clientseite (TNCC), konkret vom PTS-IMC, versendet wird. Dieses Projekt nutzt für das Logging Log4cxx (siehe Kapitel 5.2.1). Der PTS-IMV muss sowohl das IF-IMV [20] Protokoll als auch das IF-PTS [8] Protokoll unterstützen. Das IF-IMV Segment übernimmt, die IMUnit-Bibliothek aus Kapitel 5.3.2 und das IF-PTS Segment die Libif-pts-Bibliothek aus Kapitel 5.3.3.

[33] Die Clientside Policy wurde aus Zeitgründen im Rahmen dieser Arbeit nicht Implementiert.

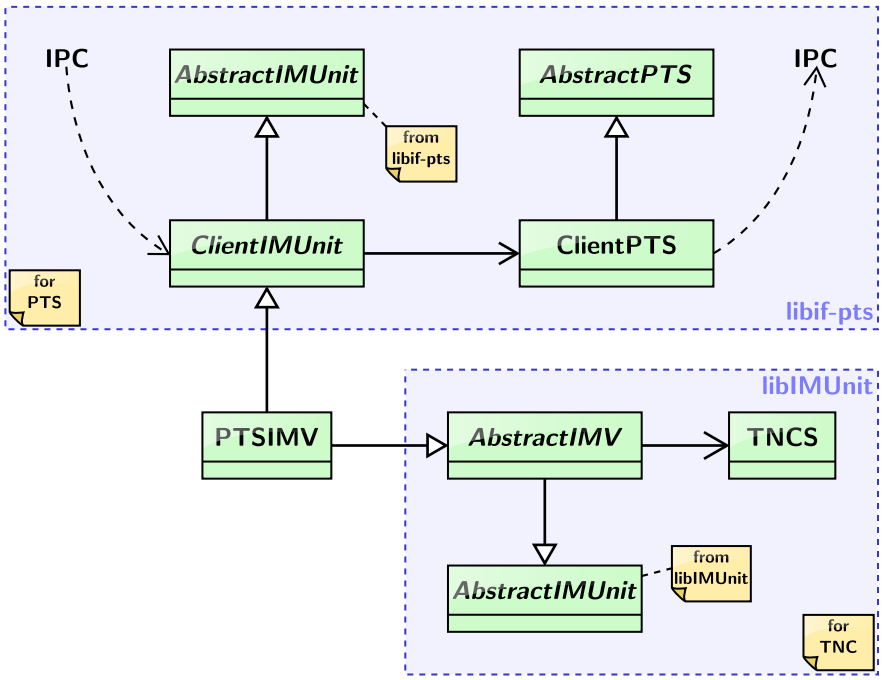


Abbildung 5.6: Klassendiagramm vom PTS-IMV

Projektaufbau

Wie der PTS-IMC aus Kapitel 5.3.4, besteht auch der PTS-IMV aus nur einer Klasse, dem `PTSIMV`. Diese Klasse wiederum erbt von zwei Klassen:

- Einmal von der Klasse `TNCUtil::iml::AbstractIMV` aus dem IMUnit Projekt und
- von der Klasse `ifpts::client::ClientIMUnit` aus dem Libif-pts Projekt.

Damit das IMUnit-Projekt die IF-IMV Aufrufe entgegennehmen kann, wird die `IFIMVImpl.cpp` aus dem IMUnit Projekt Kapitel 5.3.2 inkludiert mit:

```
#include <imv/IFIMVImpl.cpp>
```

Zusätzlich werden die zwei Methoden `TNCUtil::iml::getIMUnitInfo()` und `TNCUtil-::iml::createNewIMVInstance()` implementiert (siehe Kapitel 5.3.2).

Der konkrete Aufbau der Nachrichten zwischen den PTS-IMC und IMVs (siehe Kapitel 4.5.2), ist im Kapitel 5.4 definiert.

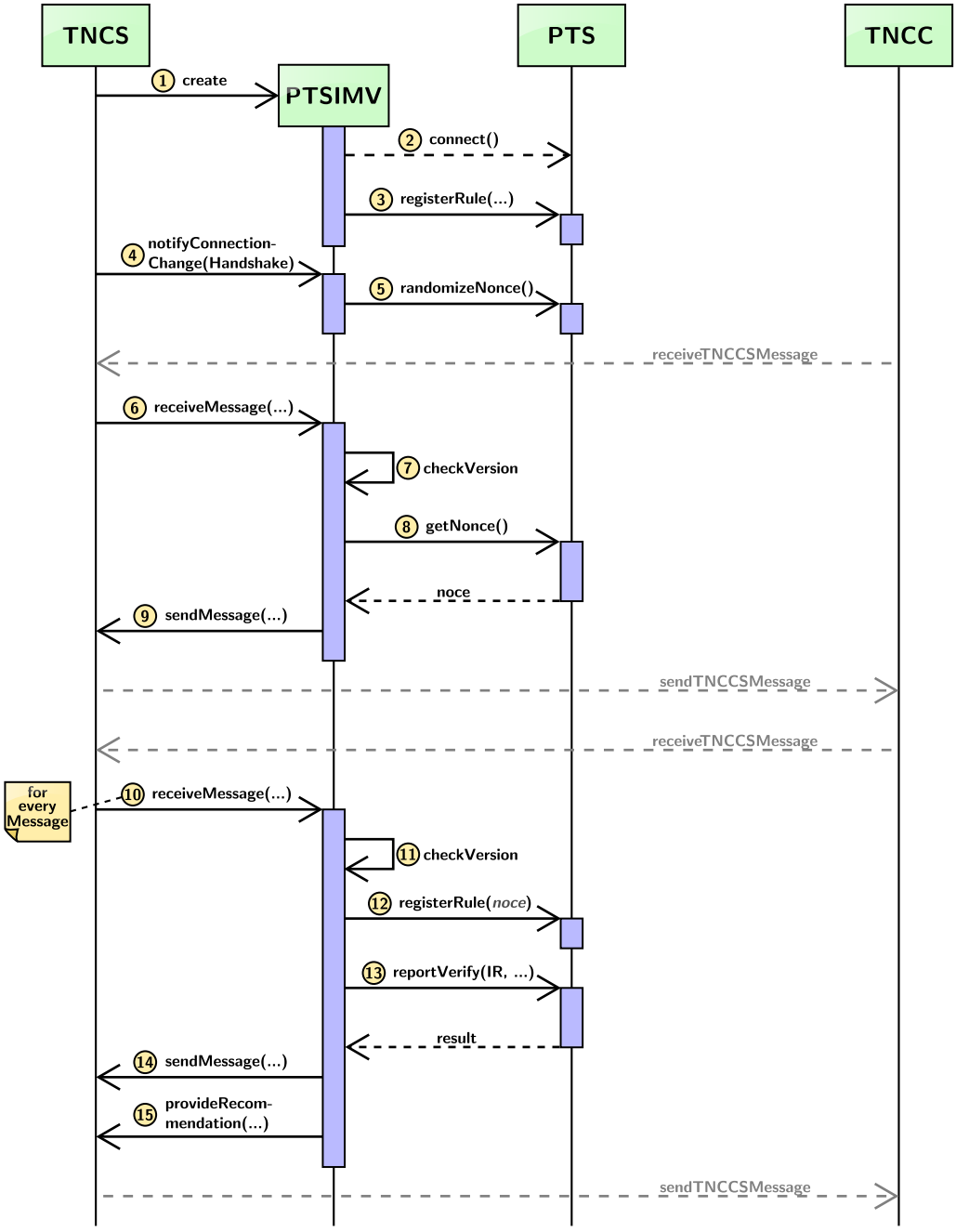


Abbildung 5.7: Sequenzdiagramm des PTS-IMVs

Programmablauf

Abbildung 5.7 zeigt den Ablauf eines Integrityhandshakes aus der Sicht der PTSIMV Klasse. Wird eine neue Verbindung zu einem TNC-Port (z. B, ein 802.1X Port) erstellt, so wird eine neue PTS-IMV Instanz erstellt (1). Dabei wird eine IPC-Verbindung zum PTS hergestellt (2). Auch werden hier einige Regeln beim PTS registriert (siehe [8, Kapitel 6.7.1]) (3) die zum Überprüfen der Vertrauenskette notwendig sind.

Beim Beginnen eines neuen Handshakes (4), wird der PTS-IMV komplett zurückgesetzt. Auch wird die Nonce der ConnectionID beim PTS mit der Funktion `FHH_RandomizeNonce` (siehe Kapitel 5.3.3) erneut randomisiert (5).

Anschließend empfängt der PTS-IMV die Nachricht vom PTS-IMC aus Kapitel 4.5.2 Schritt 1 (6). Zuerst wird Version überprüft, ob die Nachricht kompatibel mit dem PTS-IMV ist (7). Ist die Version kompatibel mit dem PTS-IMV, ruft dieser die `FHH_GetNonce` Funktion beim PTS auf (8) (siehe Kapitel 5.3.3). Danach wird eine Nachricht (siehe Kapitel 4.5.2 Schritt 2) zum PTS-IMC verschickt (9), welches

1. die verwendete Versionsnummer,
2. die ComponentID des PTS-IMV,
3. die Nonce vom PTS und
4. keine weiteren zu messenden Daten enthält.

In der nächsten Runde empfängt der PTS-IMV die Nachrichten vom PTS-IMC aus Kapitel 4.5.2 Schritt 3 (10). Für jede dieser Nachrichten wird die Versionsnummer überprüft (11). Nachrichten dessen Versionsnummer nicht mit der verwendeten Versionsnummer übereinstimmen, werden ignoriert. Stimmen die Versionsnummern überein, wird der IR aus dieser Nachricht gelesen. Anschließend wird die Nonce als weitere Regel hinzugefügt (12), da die PTS Funktionen zum überprüfen eines IR (13) keinen Parameter für diese Nonce enthält. Danach könnte der PTS-IMV Informationen zur Entscheidungsfindung zum PTS-IMC zurücksenden (siehe Kapitel 4.5.2 Schritt 4) (14). Da diese Nachricht optional ist, wurde dies im Rahmen dieser Arbeit nicht Implementiert. Als letztes wird dem TNCS das Resultat der Entscheidungsfindung mitgeteilt (15).

5.3.6 BrowserIMV

Im Rahmen dieser Arbeit wurde noch ein IMV entwickelt, welche den Browser auf dem Client mit Hilfe eines PTS verifiziert. Der Aufbau des BrowserIMV (siehe Abbildung 5.8) ähnelt sehr dem PTS-IMV aus Kapitel 5.3.5. Aus diesem Grund wird eine wiederholte Beschreibung des Projektaufbaus verzichtet.

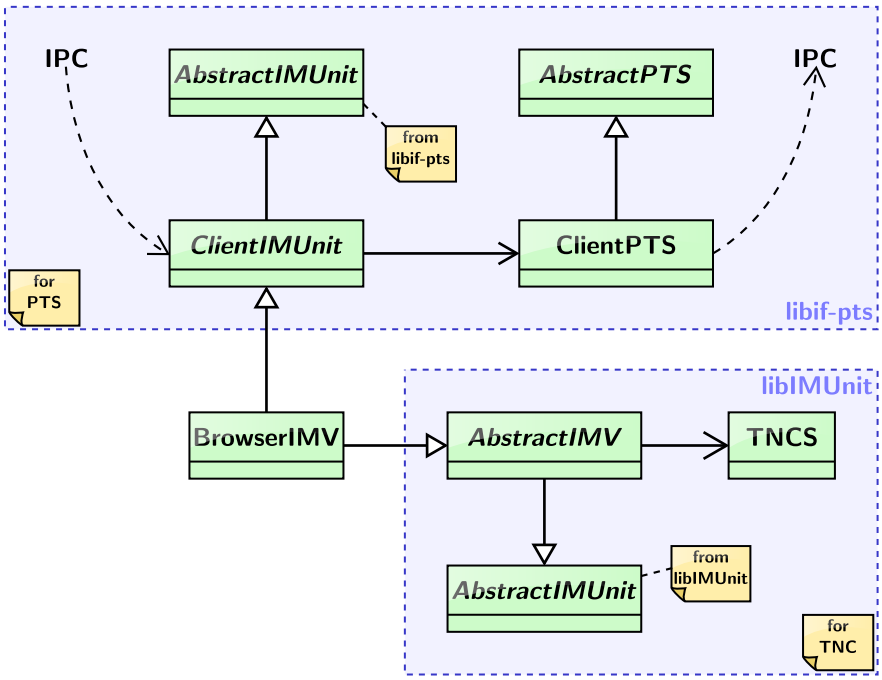


Abbildung 5.8: Klassendiagramm vom BrowserIMV

Programmablauf

Auch der Programmablauf ähnelt dem PTS-IMV aus Kapitel 5.3.5, jedoch enthält dieser kleine aber dennoch wichtige Änderungen. Ein verkürzter Ablauf aus der Sicht des BrowserIMVs zeigt Abbildung 5.9.

Beim Erstellen eines neuen BrowserIMV, werden andere Regeln beim PTS registriert (1). Diese Regeln werden nur zur Überprüfung des Browsers im IR benötigt.

Beim Beginn eines neuen Handshakes, wird nur der BrowserIMV zurückgesetzt (2) und **nicht** die Funktion `FHH_RandomizeNonce` beim PTS aufgerufen, da der PTS-IMV dies bei einem neuen Handshake bereits durchführt. Wird der PTS-IMV nicht ausgeführt, sind die Messungen des BrowserIMV in jedem Fall nicht vertrauenswürdig.

Als weiteren Unterschied schickt der BrowserIMV im zweiten Schritt aus Kapitel 4.5.2 (3) eine Liste der zusätzliche zu messenden Daten. In diesem Beispiel IMV besteht die Liste aus einer Datei, die der PTS auf der Clientseite messen soll.

5.3.7 DummyPTS

Da zum Zeitpunkt dieser Arbeit keine brauchbare Implementierung eine PTS bekannt sind, wird im Rahmen dieser Arbeit ein PTS implementiert. Dieser PTS (DummyPTS)

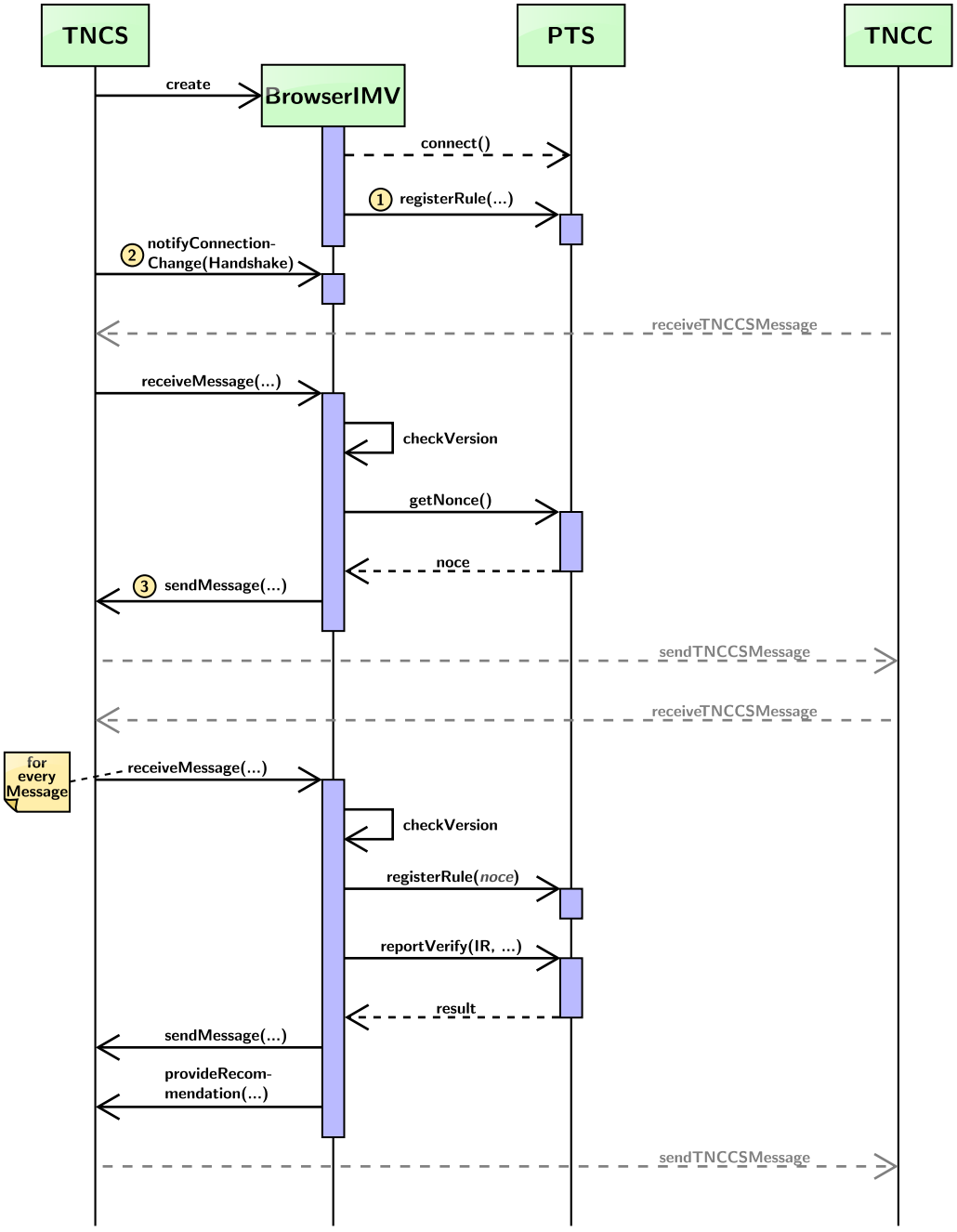


Abbildung 5.9: Sequenzdiagramm des BrowserIMVs

enthält keine Logik und simuliert die PTS-Funktionalität. Der DummyPTS kommuniziert nicht mit einem TPM, misst und überprüft keine Daten und ist kein Teil einer Static und Dynamic Chain of Trust. Der vom DummyPTS erzeugte IR z. B. ist immer derselbe IR. Es sind auch nur die PTS Funktionen implementiert, damit die IMCs und IMVs, die in dieser Arbeit entwickelt wurden, korrekt arbeiten.

5.3.8 Tncsim

Um die IMCs und IMVs auf einem Computer komfortabel zu entwickeln und zu testen, wurde zusätzlich das Programm Tncsim entwickelt, welche einen TNCC und TNCS inklusiver Kommunikation simuliert. Dabei nutzt Tncsim als TNCC entweder Libtnc[34] oder die Netzwerkschnittstelle. Im letzten Fall werden die TNCCS-Nachrichten über das EAP-Protokoll [12] (siehe auch 802.1X [9]) verschickt und empfangen. Als TNCS nutzt Tncsim entweder Libtnc oder die TNCS-Implementierung der TNC@FHH.

Für diese Arbeit wurde die neuste Libtnc Version 1.19 verwendet. Da Libtnc in dieser Version ein paar gravierende Bugs enthält, muss diese Libtnc Version noch gepatcht werden. Der nötige Patch liegt bei Tncsim dabei. Tncsim wurde mit der TNC@FHH Version 0.5.0 getestet.

5.4 Message Structure

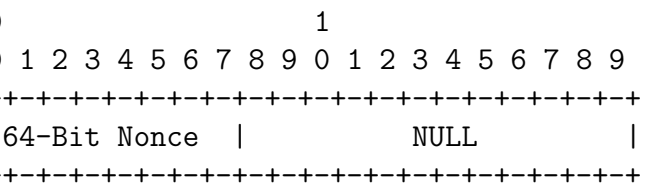
In diesem Kapitel werden die Nachrichten des IF-M-PTS-IMC-IMV Protokolls formal definiert, welche zwischen den PTS-IMC und den IMVs ausgetauscht werden (siehe Kapitel 4.5.2).

5.4.1 Nonce

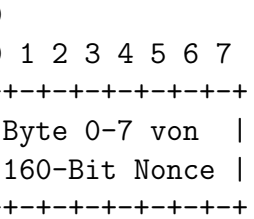
In der IF-PTS Spezifikation [8, Kapitel 6.6.4] wird die Nonce als 8-Byte (64-Bit) lange Zahl spezifiziert, jedoch in der TPM Spezifikation [36, Kapitel 5.5] als 20-Byte (160-Bit) lange Zahl. Die Umwandlung wird in der Spezifikation nicht definiert. Deshalb wird im Rahmen dieser Arbeit die Umwandlung wie folgt definiert:

64-Bit Nonce → 160-Bit Nonce Die 64-Bit Nonce wird zuerst in Network Byte Order gewandelt. Anschließend wird die 64-Bit Nonce in die ersten 8-Byte des 160-Bit Nonce kopiert. Die restlichen Bytes in der 64-Bit Nonce werden auf NULL (0x00) gesetzt. Die 160-Bit Nonce setzt sich also wie folgt zusammen (Network Byte Order):

[34] `http://libtnc.sourceforge.net/`



160-Bit Nonce → 64-Bit Nonce Aus der 160-Bit Nonce werden nur die ersten 64-Bit verwendet wobei die restlichen 12-Byte (96-Bit) aus NULL (0x00) bestehen müssen. Die ersten 64-Bit der 16-Bit Nonce werden in Host Byte Order umgewandelt welches dann die 64-Nonce ergibt. Die 64-Bit Nonce setzt sich also wie folgt zusammen (Network Byte Order):



5.4.2 PTS-IMC-IMV-Message

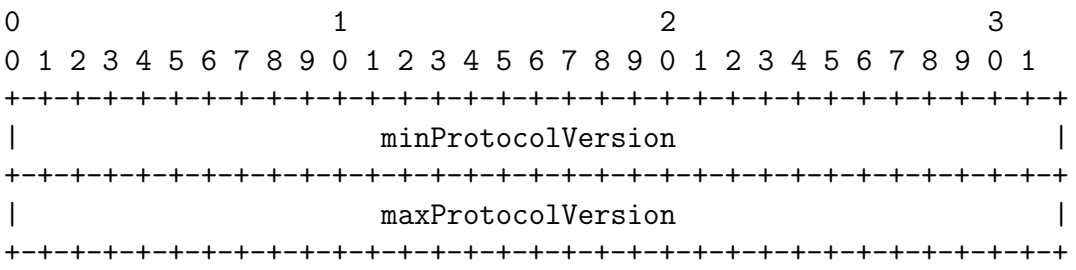
Der PTS-IMC und die IMVs kommunizieren mit Hilfe des TNCC und TNCS welche für jede Nachricht einen Message Type (siehe [19, Kapitel 3.4.2.5] bzw. [20, Kapitel 3.4.2.7]) benötigen. Der Message Type setzt sich aus einer Vendor ID und einem Message Sub Type zusammen. Die Vendor ID der FHH ist 32939 (in Hexadezimal 0x80AB) [35]. Da die Message Sub Types in der FHH nicht zentral verwaltet werden, wird für im Rahmen dieser Arbeit der Message Sub Type 17 (in Hexadezimal 0x11) verwendet. Daraus ergibt sich ein Message Type 8432401 (in Hexadezimal 0x80AB11) der für die Kommunikation zwischen den PTS-IMC und den IMVs genutzt wird.

Der Datentyp PTS_String wird wie bei Libif-pts genutzt. Siehe dazu „Abweichung der Spezifikation“ auf Seite 67.

[35] Quelle: `http://www.iana.org/assignments/enterprise-numbers`

Die Nachrichten werden wie folgt definiert:

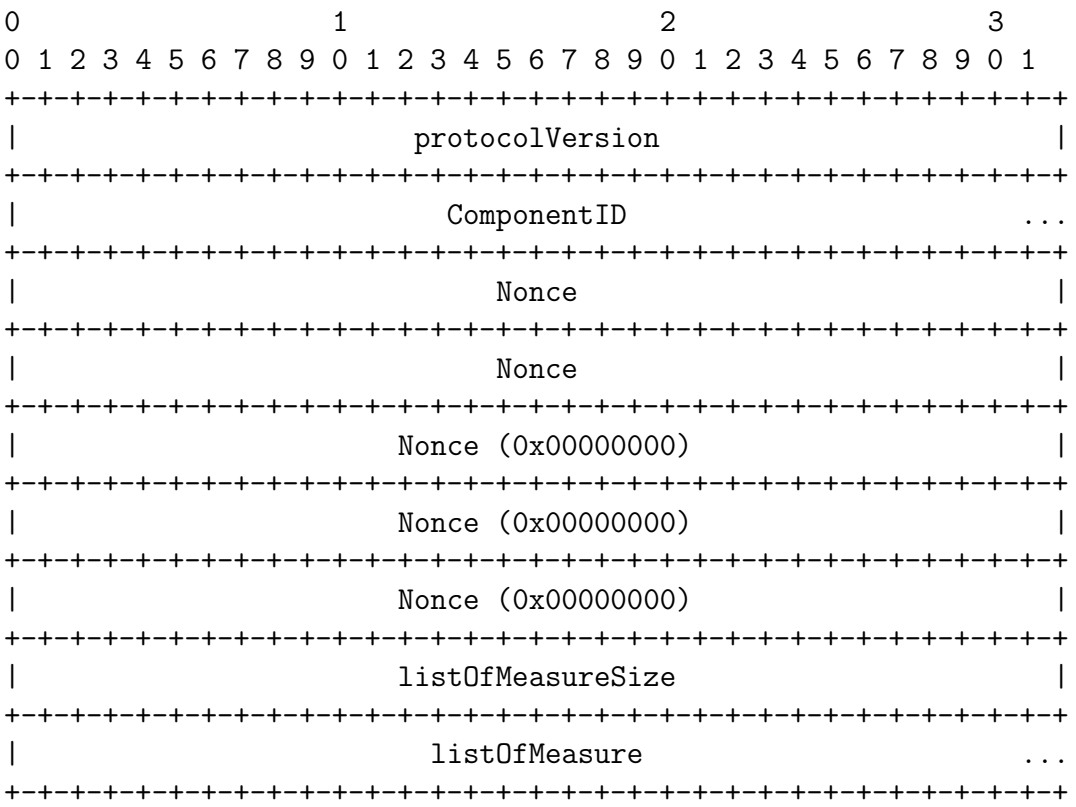
Nachricht aus Kapitel 4.5.2 Schritt 1 :



minProtocolVersion
0x00000001

maxProtocolVersion
0x00000001

Nachricht aus Kapitel 4.5.2 Schritt 2 :



protocolVersion
0x00000001

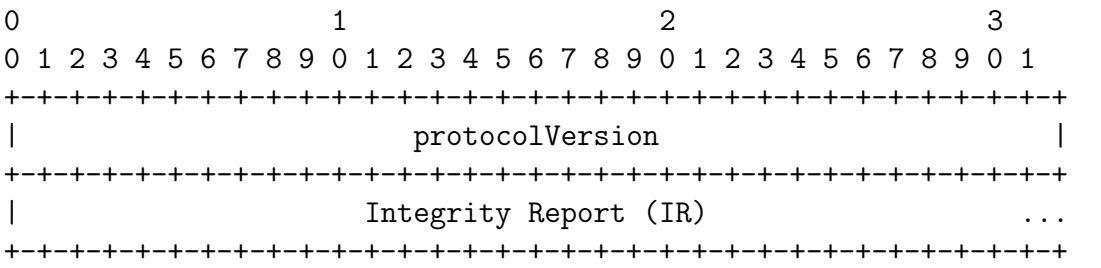
ComponentID
Die Länge der ComponentID ist variabel und wird in [8, Kapitel 5.3.8] unter PTS_ComponentId definiert.

Nonce
Zwar ist die Nonce Formal 20 Byte lang, jedoch werden effektiv nur die ersten 8 Byte genutzt (siehe Kapitel 5.4.1).

listOfMeasureSize
Die Anzahl der Elemente in der nachfolgenden Liste listOfMeasure.

listOfMeasure
Enthält eine Liste von Dateien die der PTS-IMC messen soll. Eine Datei wird als PTS_String dargestellt, welche eine variable Länge besitzt. Der Aufbau von PTS_String ist in [8, Kapitel 5.3.18] spezifiziert.

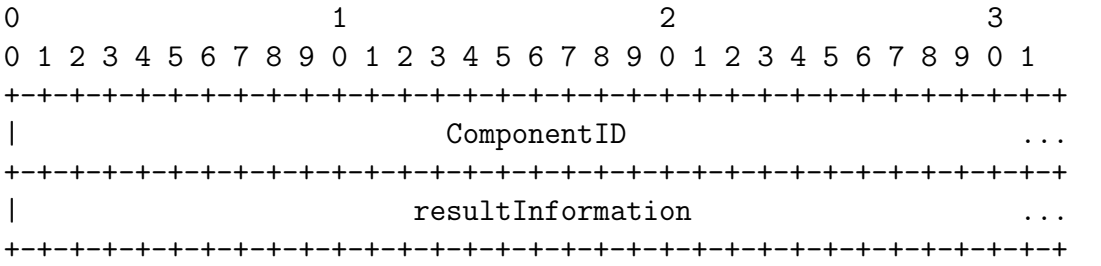
Nachricht aus Kapitel 4.5.2 Schritt 3 :



protocolVersion
0x00000001

IR
Der IR wird als PTS_String mit variabler Länge dargestellt und ist in [8, Kapitel 5.3.18] spezifiziert.

Nachricht aus Kapitel 4.5.2 Schritt 4 (optional):



ComponentID
Die Länge der ComponentID ist variabel und wird in [8, Kapitel 5.3.8] unter PTS_ComponentId definiert.

resultInformation
ResultInformation wird als variabler PTS_String dargestellt und ist in [8, Kapitel 5.3.18] spezifiziert.

6 Fazit

6.1 Spezifikationen der TCG

Die TCG hat zahlreiche (ca. 66[1]) Spezifikationen zum Thema Trusted Computing veröffentlicht. Jedoch schwankt die Qualität der Spezifikationen stark. So pflegt und aktualisiert die TNC Work Group[2] ihre Spezifikationen im TNC Umfeld regelmäßig. Die Infrastructure Work Group[3] hingegen, welche unter anderem die IF-PTS [8] und den IR [30] spezifizieren, pflegen und aktualisieren ihre Spezifikationen äußerst selten.

6.1.1 IF-PTS Spezifikation

Ein wichtiger Teil dieser Arbeit war die Umsetzung der IF-PTS Schnittstelle, welche im Projekt Libif-pts, aus Kapitel 5.3.3, umgesetzt wurde. In der IF-PTS Spezifikation werden Datentypen, wie z. B. `PTS_String`, definiert, welche eine variable Länge besitzen. Für den Transfer über die IPC Schnittstelle werden diese Datentypen ab und zu in einem extra Container gespeichert und manchmal nicht. Diese inkonsistente Umsetzung erhöht den Aufwand des Serialisierens und Parsens unnötig. Auch die Definition des `PTS_String` in der Spezifikation (siehe Kapitel 5.3.3) erhöhen die möglichen Fehlerquellen unnötig.

Die IF-PTS Spezifikation beinhaltet neben der formalen syntaktischen Definition auch die semantische Definition. Der Zusammenhang dieser Definitionen ist teilweise nur schwer Verständlich. Z. B. enthält die Semantik nicht existierende Verweise auf die Syntax, wie die Funktion PTS_Configure_SigningKey().

Auch logisch gesehen weist die Spezifikation Mängel auf. So ist es z. B. laut Spezifikation nicht vorgesehen, dass bei der Verifikation eines IR die Nonce mit angegeben wird. Ein Workaround dazu findet sich in Kapitel 5.3.5.

Auf Grund der Mängel und Komplexität der IF-PTS Spezifikation, ist eine interoperable Implementation eines PTS in dieser Version nahezu unmöglich.

[1] `http://www.trustedcomputinggroup.org/?e=resource.list&resource_type_id=2&category_type_id=1`
[2] `http://www.trustedcomputinggroup.org/developers/trusted_network_connect`
[3] `http://www.trustedcomputinggroup.org/developers/infrastructure`

6.1.2 XML Spezifikationen

Für diese Arbeit waren die XML Spezifikationen [29; 30; 33] von Bedeutung. In diesen Spezifikationen wird die Syntax des IR und RM formal definiert. Jedoch wird die Semantik des IR und RM von der TCG nur unzureichend spezifiziert. Das erhöht die Gefahr, dass es zueinander inkompatible IR und RM gibt, die syntaktisch gleich aber semantisch unterschiedlich sind. Hinzu kommt die Tatsache, dass diese XML Spezifikationen relativ komplex sind und mehrere verschiedene Anwendungsfälle abdecken.

Dies sind einige Gründe, weshalb auch hier eine interoperable Implementation der XML Spezifikationen in dieser Version nahezu unmöglich ist. Beiliegend findet sich ein Poster, welches das XML-Schema des IR und RM grafisch darstellt.

6.1.3 TNC Spezifikationen

Die TNC Spezifikationen IF-IMC/IF-IMV [19; 20] dienen der Implementierung eines IMCs bzw. IMVs und spezifizieren die Schnittstelle zwischen TNCC und IMC bzw. TNCS und IMV. Diese Spezifikationen definieren sowohl die formale Syntax als auch die Semantik relativ gut, so dass sie wenig Interpretationen zulassen. Das erhöht die Interoperabilität erheblich. So existieren schon verschiedene Projekte, wie TNC@FHH, XSupplicant, Wpa_supplicant und Libtnc, welche untereinander kompatible IF-IMC/IF-IMV Schnittstellen enthalten.

6.2 Konzepte

In diesem Abschnitt wird ein kurzes Fazit über die Kernpunkte der Konzepte gezogen.

6.2.1 Lying Endpoint Problem

Der Ansatz zum Lösen des Lying Endpoint Problems in Kapitel 4.1.1 ist in den aktuellen Betriebssystemen relativ aufwendig umzusetzen. Jedoch existiert bislang keine einfachere Möglichkeit, die mindestens genau so effektiv wie die vorgestellte Lösung ist.

Die Aufteilung in mehrere Chain of Trusts (eine Static Chain of Trust und mindestens eine Dynamic Chain of Trust) erhöht zwar die Komplexität, jedoch erhöht es auch zugleich den Komfort beim Neustart der TNC Komponenten.

Die heutigen TNCC müssen häufig als Rootprozess laufen, um die Nachrichten über das 802.1X Protokoll zu versenden. Rootprozesse können i. d. R. eine Manipulation an der Static Chain of Trust vornehmen. Da der TNCC jedoch die Static Chain of Trust nicht manipulieren können darf, muss noch ein konkretes Konzept entwickelt werden, welche eine Manipulation der Static Chain of Trust verhindert.

6.2.2 Clientside Policy

Das Thema Clientside Policy hat für den Datenschutz eine relativ hohe Relevanz. Eine Clientside Policy ist für eine prototypische Implementierung von Remote Attestation nicht erforderlich. Jedoch sollte diese Thema, zu Gunsten des Datenschutzes, weiter verfolgt werden.

In dieser Arbeit wurde die Clientside Policy nur für den PTS-IMC betrachtet. Eine Clientside Policy bei anderen IMCs kann durchaus sinnvoll sein um festlegen zu können, welches Netzwerk was messen darf.

6.2.3 Konzept 4: Cross Over

Das vierte und letzte Cross Over Konzept (siehe Kapitel 4.5) wurde in der Implementierung größtenteils umgesetzt. Da eine funktionierende PTS Implementierung fehlt, konnte das Konzept nicht vollständig getestet werden. Abgesehen davon ließ sich das beschriebene Konzept erfolgreich umsetzten.

Die TCG spezifiziert nicht, wie eine Speichermessung aussehen könnte. Da die Überlegungen zur Speichermessung den Rahmen dieser Arbeit sprengen würden, kann die aktuelle Implementierung nur zusätzliche Dateien messen. Die Liste der zu messenden Daten (listOfMeasure) besteht aus einer Liste von Strings. Jeder String entspricht genau einer Datei. Um später die zu messenden Dateien von dem zu messenden Speicher unterscheiden zu können, könnte ein Präfix wie `file://` vor jeder Datei stehen. Auch das Messen von Verzeichnissen könnte sinnvoll sein, wurde aber im Rahmen dieser Arbeit nicht betrachtet.

Ein wenig suboptimal ist, dass die Nonce bei allen IMVs in einem Handshake gleich sein muss. An dieser Stelle sollte vielleicht nach einer eleganteren Möglichkeit gesucht werden.

6.3 Projekte

In diesem Abschnitt wird ein Resümee der einzelnen Projekte aus Kapitel 5.3 gezogen. Zusätzlich wird auch ein allgemeines Fazit über TNC@FHH gezogen.

6.3.1 TNC@FHH

Im Rahmen dieser Arbeit wurden in den TNC@FHH Projekten einige Änderungen getätigt, welche gute Aussichten haben, offiziell im TNC@FHH aufgenommen zu werden.

CMake

TNC@FHH Projekte benutzen bisher ein selbstgeschriebenes Build-Tool. Da dieses Tool zu unflexibel und fehleranfällig ist wurden die hier benutzen TNC@FHH Projekte auf CMake umgestellt. Der Aufwand, die TNC@FHH Projekt auf CMake umzustellen, war relativ gering. CMake bietet im Gegenzug zum alten Build-Tool viel mehr Flexibilität und Komfort.

Aufgrund der Erkenntnisse die hier in CMake gewonnen wurden, wird das offizielle TNC@FHH Projekt auf CMake umgestellt. Die Umstellung wird voraussichtlich in der TNC@FHH Version 0.6.0 abgeschlossen sein.

Log4cxx

Bisher wurden in den TNC@FHH Projekten ein selbstentwickeltes Loggingframework verwendet. Da dieses Loggingframework nicht flexibel genug ist, wurden schon länger Überlegungen getätigt, das TNC@FHH Projekt auf das Loggingframework Log4cxx umzustellen. Für diese Arbeit wurde das alte Loggingframework nicht ersetzt. Nur neugeschriebener Quelltext benutzt das neue Loggingframework Log4cxx.

Um die Übersichtlichkeit der Loggingausgaben zu erhöhen sollte ein Konzept entwickelt werden, welches die Benutzung des Loggings definiert. So enthalten nahezu alle Funktionen im TNC@FHH Projekt eine Loggingausgabe. Diese Ausgaben sind für einen Benutzer oder einen Entwickler häufig unnötig und teilweise redundant. So sind z. B. für einen IMC/IMV Entwickler die logischen Abläufe wichtiger.

Das TNC@FHH Projekt wird voraussichtlich in der Version 0.6.0 nur noch auf das Loggingframework Log4cxx setzen. Bleibt zu hoffen, dass für das TNC@FHH Projekt ein Konzept entwickelt/genutzt wird, welches die Übersichtlichkeit der Logausgaben erhöht.

6.3.2 TNCUtil

Da das TNCUtil Projekt aus Kapitel 5.3.1 nach der Entfernung des IMUnit Codes kaum noch logischen Code enthält, lohnt sich dessen Weiterentwicklung nicht mehr. TNCUtil wird voraussichtlich in der TNC@FHH Version 0.6.0 nicht mehr enthalten sein.

6.3.3 IMUnit

IMUnit implementiert die IF-IMC und IF-IMV Schnittstelle und stellt diese den IMC und IMV Implementierungen zu Verfügung. Ein IMC/IMV Entwickler der IMUnit nutzt, sollte die Semantik der IF-IMC und IF-IMV Spezifikationen [19; 20] kennen, um IMUnit korrekt nutzen zu können.

Die Idee, IMUnit als eine extra Bibliothek anzubieten, wurde vom TNC@FHH Projekt aufgenommen und wird ein offizieller Bestandteil von TNC@FHH. Da IMUnit einen gravierenden Fehler enthält (siehe Kapitel 5.3.2), wird IMUnit für die TNC@FHH Version 0.6.0 komplett redesignt und als Shared Library entwickelt.

6.3.4 Libif-pts

Das Projekt Libif-pts hat den größten Teil der Implementierung ausgemacht. Jedoch erleichtert diese Bibliothek den Entwicklungsaufwand eines PTS oder IMC/IMV mit PTS Funktionalität erheblich. Aus Zeitgründen konnten nicht alle Containerklassen für die PTS Funktionen implementiert werden. Dies sollte im Rahmen einer Weiterentwicklung noch umgesetzt werden.

Die komplexen Datentypen aus der IF-PTS Spezifikation [8, Kapitel 3.5], werden in Libif-pts als reine `struct` implementiert. Für den Entwickler ist es jedoch komfortabler, wenn diese Datentypen als Klasse implementiert werden. Dann z. B. könnte der Inhalt dieser Container beim Erzeugen mit einigermaßen sinnvollen Daten gefüllt werden.

Da diese Bibliothek dem Entwickler eines PTS, IMC oder IMV das aufwendige Serialisieren und Parsen erheblich erleichtert, hat diese Library großes Potenzial bei solchen Implementierungen erfolgreich zum Einsatz zu kommen.

6.3.5 PTS-IMC, PTS-IMV und BrowserIMV

Der PTS-IMC, PTS-IMV und BrowserIMV sind nur prototypische Implementierungen, welche sich noch in einem hoch experimentellen Status befinden. Auch sind noch nicht alle Funktionalitäten umgesetzt. Da noch keine passende PTS Implementierung existiert, wurden die Clientside Policy und die Regelerzeugung noch nicht implementiert. Dies sollte mit einem PTS implementiert werden, um das Konzept und Implementation gleich auf Fehler überprüfen zu können.

6.3.6 DummyPTS

Der DummyPTS ist ein PTS, welcher keine Funktionalität besitzt und sollte nicht weiterentwickelt werden. Stattdessen sollten die Ressourcen für eine korrekt funktionierende PTS Implementierung eingesetzt werden.

6.3.7 Tncsim

Ursprünglich wurde Tncsim geschrieben, um die Funktionsweise des PTS-IMCs bzw. PTS-IMVs auch ohne ein TNC Netzwerk testen zu können. Inzwischen unterstützt

Tncsim jeweils zwei verschiedene TNCC und TNCS Engines zum testen der Interoperabilität.

Bisher kann Tncsim bei einer Simulation nur einen TNC Handshake durchführen. Da die Resonanz dieses Projektes bisher am größten war, hat dieses Projekt, welches aus der Not heraus entstanden ist, sehr gute Chancen eigenständig weiterentwickelt zu werden.

A Alternative Interpretation von Snapshots

In diesem Anhang wird eine weitere Möglichkeit vorgestellt, wie ein Snapshot aufgebaut sein kann ohne gegen die Spezifikation zu verstoßen. Da diese Interpretation nicht so elegant und flexibel ist, wird dieser Ansatz in der restlichen Arbeit ignoriert.

A.1 Aufbau eines Snapshots

Hier wird nicht zwischen Normal und Sync Snapshots unterschieden. Der Aufbau eines Snapshots setzt sich wie in Kapitel 4.1.4 zusammen, nur dass jeder Snapshot das *PcrHash* Element besitzt. Dieses Element besitzt einen *StartHash*, welches den PCR-Wert vor der Snapshotmessung enthält. Der Wert von *PcrHash* enthält den PCR-Wert nach der Snapshotmessung. Eine Reihenfolge der Snapshots wird nicht gespeichert.

A.2 Zuordnung der Snapshots zu den PCRs

Zwar enthält ein Snapshot, wie oben beschrieben unter anderem einen *StartHash* und einen *PcrHash*, aber die PCR-Nummer ist laut Spezifikation optional. Das heißt, dass ein Snapshot nicht sofort einem PCR zugeordnet werden kann, wenn die PCR-Nummer fehlt. Jedoch ist diese Zuordnung möglich, wenn alle Snapshots eines PCR vorhanden sind. Anhand eines Beispiels wird erklärt, wie diese Zuordnung erfolgen kann:

Es sind 7 Snapshots gegeben $S_x\langle StartHash, PcrHash\rangle$ mit $x \in \{0 \dots 6\}$ und 3 PCR $PCR_y\langle Hash\rangle$ mit $y \in \{0 \dots 2\}$ (siehe Abbildung A.1). Dazu werden die Snapshots in umgekehrter Reihenfolge, vom Endzustand über die Zwischenzustände bis hin zum Startzustand, gesucht und zugeordnet. Im Beispiel bildet der Snapshot S_4 den Übergang zum Endzustand vom PCR PCR_0 ab, da der $PcrHash$ vom Snapshot äquivalent mit dem $Hash$ vom PCR ist. Damit gehört der Snapshot S_4 zu dem PCR PCR_0. Als nächstes wird der Snapshot gesucht der den vorherigen Zustandsübergang abbildet. Dafür muss der $StartHash$ vom letzten gefundenen Snapshot (hier S_4) identisch mit dem $PcrHash$ vom nächsten gesuchten Snapshot (hier S_6) sein. Dieser letzte Schritt wird so lange wiederholt, bis kein weiterer passender Snapshot gefunden wird. Der Letzte gefundene Snapshot bildet dann den 'Startzustand' ab (in diesem Beispiel ist es

$$
\begin{array}{lll}
 & S_0\langle H_8, H_9\rangle & \left.\begin{array}{l} S_3\langle H_0, H_1\rangle \\ S_6\langle H_1, H_2\rangle \\ S_4\langle H_2, H_3\rangle \end{array}\right\} K_0 \rightarrow PCR_0\langle H_3\rangle \\
 & S_1\langle H_4, H_5\rangle & \\
PCR_0\langle H_3\rangle & S_2\langle H_7, H_8\rangle & \\
PCR_1\langle H_6\rangle & S_3\langle H_0, H_1\rangle \quad \Longrightarrow & \left.\begin{array}{l} S_1\langle H_4, H_5\rangle \\ S_5\langle H_5, H_6\rangle \end{array}\right\} K_1 \rightarrow PCR_1\langle H_6\rangle \\
PCR_2\langle H_9\rangle & S_4\langle H_2, H_3\rangle & \\
 & S_5\langle H_5, H_6\rangle & \\
 & S_6\langle H_1, H_2\rangle & \left.\begin{array}{l} S_2\langle H_7, H_8\rangle \\ S_0\langle H_8, H_9\rangle \end{array}\right\} K_2 \rightarrow PCR_2\langle H_9\rangle
\end{array}
$$

Abbildung A.1: Beispiel einer Zuordnung von Snapshots zu PCRs

S_3). Dieser Algorithmus wird für jedes PCR erneut angewendet. Die Snapshots eines PCR bilden zusammen eine Kette von Zuständen (K_y). In diesem Beispiel ergibt es eine Zuordnung wie in Abbildung A.1.

Zu Komplikationen kann es kommen, wenn mehrere Ketten am Anfang identisch sind, weil z. B. die ersten Snapshots dieselbe Messungen enthalten. Dies sollte jedoch bei der Policy keinen Unterschied machen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Mitnick, Kevin D. ; Simon, William: *Die Kunst der Täuschung: Risikofaktor Mensch*. Mitp-Verlag, 2006. – ISBN-13 978–3826615696

[2] Trusted Computing Group: *Glossary*. – `http://www.trustedcomputinggroup.org/developers/glossary`

[3] Trusted Computing Group: *TPM Main - Part 1 Design Principles*. Juli 2007. – Specification Version 1.2, Level 2 Revision 103
`http://www.trustedcomputinggroup.org/resources/tpm_specification_version_12_revision_103_part_1__3`

[4] Helden, Josef v. ; Pohlmann, Norbert ; Steinmetz, Mike B. ; Bente, Ingo ; Jungbauer, Marian ; Linnemann, Markus: *T-NAC - Vertrauenswürdige Zugriffssteuerung in Netzwerken auf Basis von Trusted Computing*. November 2007. – Vorhabenbeschreibung des Antrages für die Förderrunde 2008 für das Förderprogramm FHprofUnd (nicht Öffentlich)

[5] Trusted Computing Group: *TCG Specification Architecture Overview*. August 2007. – Specification Revision 1.4
`http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14`

[6] Trusted Computing Group: *Reference Architecture for Interoperability (Part I)*. Juni 2005. – Specification Version 1.0, Revision 1
`http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_reference_architecture_for_interoperability_specification_part_1_version_10`

[7] Trusted Computing Group: *TCG Software Stack (TSS)*. März 2007. – Specification Version 1.2, Level 1, Errata A
`http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification__version_12_errata_a`

[8] Trusted Computing Group: *Platform Trust Services Interface Specification (IF-PTS)*. November 2006. – Specification Version 1.0, Revision 1.0
`http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_platform_trust_services_interface_specification_version_10`

[9] Institute of Electrical and Electronics Engineers (IEEE): *IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control.* 2004. – IEEE Std 802.1X-2004
http://standards.ieee.org/getieee802/download/802.1X-2004.pdf

[10] Trusted Computing Group: *Trusted Computing Group (TCG) Timeline.* April 2009. – http://www.trustedcomputinggroup.org/resources/tcg_timeline

[11] Trusted Computing Group & Microsoft: *Standardizing Network Access Control: TNC and Microsoft NAP to Interoperate.* Mai 2007. –
http://www.trustedcomputinggroup.org/resources/standardizing_network_access_control_tnc_and_microsoft_nap_to_interoperate

[12] Aboba, Bernard ; Blunk, Larry J. ; Vollbrecht, John R. ; Carlson, James ; Levkowetz, Henrik: *RFC 3748: Extensible Authentication Protocol (EAP).* Juni 2004. – IETF - Network Working Group
http://www.rfc-editor.org/rfc/rfc3748.txt

[13] Trusted Computing Group: *TNC IF-TNCCS: Protocol Bindings for SoH.* Mai 2007. – Specification Version 1.0, Revision 0.08
http://www.trustedcomputinggroup.org/resources/tnc_iftnccs_protocol_bindings_for_soh_version_10

[14] Trusted Computing Group: *TNC Architecture for Interoperability.* April 2008. – Specification Version 1.3, Revision 6
http://www.trustedcomputinggroup.org/resources/tnc_architecture_for_interoperability_version_13

[15] Rigney, Carl ; Rubens, Allan C. ; Simpson, William A. ; Willens, Steve: *RFC 2865: Remote Authentication Dial In User Service (RADIUS).* Juni 2000. – IETF - Network Working Group, http://www.rfc-editor.org/rfc/rfc2865.txt

[16] Trusted Computing Group: *TNC IF-T: Protocol Bindings for Tunneled EAP Methods.* Mai 2007. – Specification Version 1.1, Revision 10
http://www.trustedcomputinggroup.org/resources/tnc_ift_protocol_bindings_for_tunneled_eap_methods_version_11

[17] Trusted Computing Group: *TNC IF-T: Binding to TLS.* Mai 2009. – Specification Version 1.0, Revision 16
http://www.trustedcomputinggroup.org/resources/tnc_ift_binding_to_tls_version_10_revision_16

[18] Trusted Computing Group: *TNC IF-TNCCS*. Mai 2009. – Specification Version 1.2, Revision 6.00
http://www.trustedcomputinggroup.org/resources/tnc_iftnccs_specification_version_12_revision_6

[19] Trusted Computing Group: *TNC IF-IMC*. Februar 2007. – Specification Version 1.2, Revision 8
http://www.trustedcomputinggroup.org/resources/tnc_ifimc_specification_version_12

[20] Trusted Computing Group: *TNC IF-IMV*. Februar 2007. – Specification Version 1.2, Revision 8
http://www.trustedcomputinggroup.org/resources/tnc_ifimv_specification_version_12

[21] Schmiedel, Martin: *Entwicklung einer Client-/Server-basierten Software für die Prüfung der Vertrauenswürdigkeit von Netzwerkkomponenten*, Fachhochschule Hannover, Masterarbeit, 2006. – http://tnc.inform.fh-hannover.de/tnc/wiki/media/9/90/Master_Thesis_Martin_Schmiedel_%28german%29.pdf

[22] Wuttke, Daniel: *Erweiterung sicherheitsrelevanter Software für die automatische Integritätsprüfung von Endgeräten*, Fachhochschule Hannover, Masterarbeit, 2006. – http://tnc.inform.fh-hannover.de/tnc/wiki/media/d/de/Master_Thesis_Daniel_Wuttke_%28german%29.pdf

[23] Benz, Benjamin: Trojaner benutzt Virenscanner. In: *Heise Online* (2006), Oktober. – http://www.heise.de/newsticker/meldung/79836

[24] Heasman, John: Implementing and Detecting An ACPI BIOS Rootkit / Black Hat Europe. 2006. – Technischer Report. –
http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf

[25] Kumar, Nitin ; Kumar, Vipin: Vboot Kit: Compromising Windows Vista Security / Black Hat Europe. 2007. – Technischer Report. –
http://www.blackhat.com/presentations/bh-europe-07/Kumar/Whitepaper/bh-eu-07-Kumar-WP-apr19.pdf

[26] Ruetten, Christiane: Vom Rootkit zum Bootkit: Vistas Code-Signierung ausgehebelt. In: *Heise Online* (2007), März. – http://www.heise.de/newsticker/meldung/87697

[27] Roecher, Dror-John ; Thumann, Michael: NACATTACK - Hacking the Cisco NAC Framework / Black Hat Europe. 2007. – Technischer Report. –

http://www.blackhat.com/presentations/bh-europe-07/Dror-Thumann/Whitepaper/bh-eu-07-dror-WP.pdf

[28] Trusted Computing Group: *Architecture Part II - Integrity Management*. November 2006. – Specification Version 1.0, Revision 1.0
http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_architecture_part_ii__integrity_management_version_10

[29] Trusted Computing Group: *Core Integrity Schema*. November 2006. – Specification Version 1.0.1, Revision 1.0
http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_core_integrity_schema_specification_version_101

[30] Trusted Computing Group: *Integrity Report Schema*. November 2006. – Specification Version 1.0, Revision 1.0
http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_integrity_report_schema_specification_version_10

[31] Thompson, Henry S. ; Beech, David ; Maloney, Murray ; Mendelsohn, Noah: *XML Schema Part 1: Structures Second Edition*. http://www.w3.org/TR/xmlschema-1/. Version: Oktober 2004. – World Wide Web Consortium (W3C)

[32] Leach, Paul J. ; Mealling, Michael ; Salz, Rich: *RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace*. Juli 2005. – IETF - Network Working Group, http://www.rfc-editor.org/rfc/rfc4122.txt

[33] Trusted Computing Group: *Reference Manifest (RM) Schema*. November 2006. – Specification Version 1.0, Revision 1.0
http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_reference_manifest_rm_schema_specification_version_10

[34] Trusted Computing Group: *TCG PC Client Specific Implementation Specification For Conventional BIOS*. Juli 2005. – Version 1.2 FINAL, Revision 1.00
http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios_specification_version_12

[35] ISO/IEC JTC1/SC22/WG21 - C++ Standards Committee: *Working Draft, Standard for Programming Language C++*. März 2009. – Doc No: N2857=09-0047
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1745.pdf

[36] Trusted Computing Group: *TPM Main - Part 2 TPM Structures*. Oktober 2006. – Specification Version 1.2, Level 2 Revision 103
http://www.trustedcomputinggroup.org/resources/tpm_specification_version_12_revision_103_part_1__3