Dresden
Technische Universität Dresden
Fakultät Informatik
Angewandte Informatik

Entfernte Attestierung von Eigenschaften auf mikrokern-basierten Systemen

Diplomarbeit

vorgelegt von
Hans Marcus Krüger

Verantwortlicher Hoschschulprofessor
Prof. Dr. Hermann Härtig

2007

Diplomarbeit

Entfernte Attestierung von Eigenschaften auf mikrokern-basierten Systemen

©2007, Hans Marcus Krüger
hans@skbrasil.net

http://skbrasil.net/~hans/DA/

Verantwortlicher Hochschulprofessor:
Prof. Dr. Hermann Härtig

Betreuer:
Dipl. Inf. Carsten Weinold
Dr. Kai Martius

Technische Universität Dresden
Fakultät Informatik
Angewandte Informatik
Dresden

10. Mai 2007

Eid

Ich erkläre an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Ich versichere, dass ich dieses Diplomarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Dresden, den 10. Mai 2007

Hans Marcus Krüger

Aufgabenstellung

Entfernte Attestierung von Eigenschaften auf mikrokernbasierten Systemen

An der Professur Betriebssysteme wird intensiv im Bereich der „Trusted Computing“-Technologie geforscht. Diese Technologie ermöglicht es unter anderem einen Software-Stack auf einem entfernten Rechner in vertrauenswürdiger Art und Weise zu attestieren. Die direkte Attestierung des Binärcodes bedeutet dabei allerdings auch die Offenlegung der konkreten Software-Konfiguration.

Um diesen Nachteil in Bezug auf den Datenschutz zu vermeiden, sollen im Rahmen dieser Diplomarbeit Möglichkeiten zur Attestierung von abstrakten Eigenschaften verglichen und bewertet werden. Dabei sollen unter anderem die Konzepte „Property-based Attestation“, „Direct Anonymous Attestation“ und „Trusted Third Parties“ untersucht und ausgehend davon ein Ansatz zur Attestierung abstrakter Eigenschaften entwickelt werden. Die Beschreibung der Eigenschaften selbst sowie die Leistungsfähigkeit der formalen Beschreibungsmittel soll ebenfalls Bestandteil der Untersuchung sein.

Die Arbeit beinhaltet eine programmtechnische Umsetzung des Attestierungsansatzes als Prototyp im Kontext eines mikrokernbasierten Betriebssystems und einer minimalen Schnittstelle zum „Trusted Platform Module (TPM)“. Bei Design und Implementierung der zu entwickelnden Softwarekomponenten soll außerdem eine kleine „Trusted Computing Base“ und geringe Komplexität angestrebt werden.

Zusammenfassung

Das *trusted plattform module* (TPM) ist ein integrierter Schaltkreis, der moderne Rechnersysteme sicherer gestalten soll. Dieses Gerät erlaubt es, entfernt zu prüfen, in welchem Zustand sich ein System befindet. Diese Prüfung wird *entferntes Attestieren* (engl. *remote attestation* – RA) genannt.

Ein wesentliches Merkmal ist, dass die Anwendungen anhand ihrer Prüfsummen identifiziert werden. Diese Prüfsummen werden während des Ladevorgangs eines Programms ermittelt und in einem Protokoll, dem *Ladeprotokoll*, aufbewahrt. Somit stellt das Ladeprotokoll eine Chronologie über alle gestarteten Anwendungen dar. Um die Manipulation an diesem Ladeprotokoll auszuschließen wird es mit Hilfe des TPM-Geräts geführt und geschützt. Bei der Übertragung des Ladeprotokolls an einen anderen Rechner wird vom TPM-Gerät eine digitale Unterschrift erzeugt, mit dessen Hilfe die entfernte Instanz die Authentizität des Ladeprotokolls prüft.

Das Ladeprotokoll kann stets nur im Ganzen veröffentlicht werden. Welche Vor- und Nachteile daraus entstehen, hängt im wesentlichen vom Betrachter und vom Einsatz der Geräte ab. Werden elektronische Wahlurnen eingesetzt, so wird gewünscht, dass das System sehr transparent gestaltet ist. Doch entferntes Attestieren wird auch in Hinblick auf das *Digitales Rechtemanagement*, also für die Unterbindung unerlaubter Vervielfältigungen und Anwendung von urherberrechtlich geschützten Materialien, entwickelt. Inwiefern ein Anwender einen so großen Einblick in sein System gewähren muss, um lediglich eine Tageszeitung im Internet lesen zu können, sollte zurecht in Frage gestellt werden.

Auch für die Softwareentwickler birgt der Einsatz des Ladeprotokolls Nachteile, denn die Prüfsummen der Anwendungen unterscheiden sich bei jeder neuen Version einer Anwendung. Zudem enthalten die Prüfsummen keine Informationen über die Beziehungen zwischen den Programmen. Meistens beschränken sich die Messungen zudem auf die Anwendungen. Die Konfigurationsdateien werden gewöhnlicherweise nicht gemessen, da die daraus entstehenden Nachteile das System noch wesentlich komplizierter gestalten würden.

Als Ausweg bietet sich an, Anwendungen und Konfigurationsdaten mit Zertifikaten zu versehen, und anhand dieser Zertifikate das entfernte Attestieren durchzuführen. Diese Zertifikate können weit mehr Informationen aufnehmen als die Identität einer Anwendung. Vielmehr lassen sich die Daten im System um eine Beschreibung ihrer Eigenschaften erweitern und Programme anhand ihrer Eigenschaften messen (z.B. Besitzt das Programm x eine europäische Zulassung?).

Diese Arbeit stellt eine Methode vor, wie die Probleme von RA durch das Messen von Eigenschaften auf Mikrokern-Systemen umgesetzt werden kann. Sie befasst sich ebenfalls mit der Frage, wie die Privatsphäre der Anwender gewahrt werden kann. Insbesondere stellt diese Arbeit eine Methode vor, die *Trusted Computing Base* einer Anwendung gezielt zu messen. Die Arbeit erarbeitet sowohl die technischen Voraussetzungen am System, die Systemanwendungen, als auch das Modell und die Algorithmen für das Arbeiten und Messen mit und von Eigenschaften.

Diese Arbeit wurde in Form eines kurzen Abstrakts über 15 Seiten für den 10. deutschen IT-Sicherheitskongress im Mai 2007 angenommen.

Danksagung

Mein Danke geht insbesondere meinem Betreuer Carsten Weinhold, der sich immer sehr intensiv mit meinen Fragen und der Entwicklungen dieser Arbeit auseinander gesetzt hat. Seine Anregungen waren sehr wichtig für diese Arbeit.

Ich bedanke mich auch bei Dr. Kai Martius, Robert Dorn und Alex Senier, mit denen ich sehr interessante Diskussionen über diese Arbeit führen konnte. Mein Dank gilt auch der Firma *secunet Security Networks AG*, die mir großzügig einen sehr guten Arbeitsplatz für die Ausarbeitung dieser Arbeit gestellt hat.

Zusätzlich geht mein Dank an Sebastian Hegler und Natalia Krasowska für ihre endlose Geduld, die Fallen der deutschen Sprache in dieser Arbeit zu entdecken und zu beseitigen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AIK ..... Attestation Identity Key. *Seite 5*

BPRA ..... Boot protocol remote Attestation. *Seite 26*

CA ..... Certificate Authority. *Seite 14*

CRL ..... Certificate Revocation List. *Seite 16*

DAA ..... Direct Anonymous Attestattion. *Seite 16*

EK ..... Endorsment Key. *Seite 5*

IPC ..... Inter-Prozess-Kommunikation. *Seite 25*

MK ..... Manufakturer Key. *Seite 5*

PBRA ..... Property based remote Attestation. *Seite 47*

PCR ..... Platform Control Register. *Seite 6*

PKI ..... Public Key Infrastructure. *Seite 14*

RA ..... Remote Attestation. *Seite 21*

RAL ..... Remote Attestation Layer. *Seite 30*

RAQL ..... Remote Attestation Query Language. *Seite 44*

SAS ..... Sicheres Attestierungs-System. *Seite 21*

SQL ..... Structured Query Language. *Seite 44*

SRK ..... Root Storage Key. *Seite 5*

TAM ..... Trusted Attestation Manager. *Seite 63*

TCB ..... Trusted Computing Base. *Seite 13*

TCG ..... Trusted Computing Group. *Seite 5*

TNC ..... Trusted Network Connect. *Seite 17*

TPM ..... Trusted Platform Module. *Seite 5*

TTP ..... Trusted Third Party. *Seite 34*

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

1. Einleitung

In der *informationellen Gesellschaft* erlangt das Thema *entferntes Attestieren* einen hohen Stellenwert, da die Informationen und Dienstleistungen zunehmend über Netzwerke bereit gestellt werden. Beispiele hierfür sind Musikdateien, bei denen ein gewerbliches Interesse besteht, illegale Vervielfältigungen zu unterbinden. Doch die Informationstechnik macht auch vor den Grundpfeilern moderner Gesellschaften nicht halt, indem beispielsweise immer mehr Staaten zunehmend elektronische Wahlurnen einsetzen, um demokratische Wahlen durchzuführen.

In beiden Fällen besteht ein großes Interesse, die Sicherheit der Systeme vertrauenswürdig messen zu können. Um diese Aufgabe zu lösen, muss feststehen, wie Sicherheit erreicht wird, und wie sie anschließend gemessen werden kann. Das entfernte Attestieren befasst sich mit diesen Fragen.

Das entfernte Attestieren hat mit der Einführung und der Verbreitung der TPM-Geräte an Bedeutung gewonnen. TPM-Geräte sind hoch-integrierte Schaltkreise, die in vielen modernen Rechnern verbaut sind und eine Reihe von besonderen Schutzmaßnahmen besitzen[1]. Doch die Markteinführung der TPM-Geräte steht in der Kritik nur den großen Konzernen zu dienen – die Belange der Anwender, wie Datenschutz und Anwendung der erworbenen Rechte würden nicht beachtet. Dementsprechend formte sich insbesonders Ende der 90er Jahre Widerstand gegen die Pläne der TCG, also dem Konsortium, dass die Entwicklung dieser Geräte vorantreibt[ant].

Die Nachteile für die Anwender, die aus der Anwendung von RA entstehen, werden in dieser Arbeit noch genauer betrachtet. Doch auch für die Unternehmen bestehen Nachteile, die vor allem auf die fehlende Flexibilität der entfernten Attestierung zurückzuführen sind. Höhere Entwicklungs- und Wartungskosten können die Folgen sein. Auch ist die Frage der Haftung für den Schaden den ein defektes TPM-Gerät verursacht, weder geklärt noch zu vernachlässigen. Zudem fürchten insbesondere kleinere Firmen und die *Open-Source-Gemeinde*, durch diese Technologie ausgesperrt zu werden.

Um einen besseren Datenschutz gewährleisten zu können, werden in dieser Arbeit neue Ansätze für das entfernte Attestieren vorgestellt. Insbesondere steht das Attestieren von Eigenschaften und von einzelnen Anwendungen im Vordergrund. Dieser Ansatz erlaubt mehr Schutz der Privatsphäre, mehr Transparenz und mehr Flexibilität in der Wartung und Gestaltung der Systeme.

1.1. Entferntes Attestieren anhand eines Fallbeispiels

Das Entfernte Attestieren soll anhand eines Fallbeispiels erläutert werden. Eine elektronische Wahlurne (Abbildung 1.1) soll so gestaltet werden, dass ein Wähler sich überzeugen kann, dass auf der Urne die Programme laufen, die zuvor von der Wahlleitung oder von seiner Partei überprüft und freigegeben wurden. Nachträgliche Manipulationen an dem Gerät müssen erkennbar sein. Abbildung 1.1 zeigt neben der elektronischen Wahlurne auch ein

[1] Das Kapitel 2, *Grundlagen*, führt TPM-Geräte ausführlich ein.

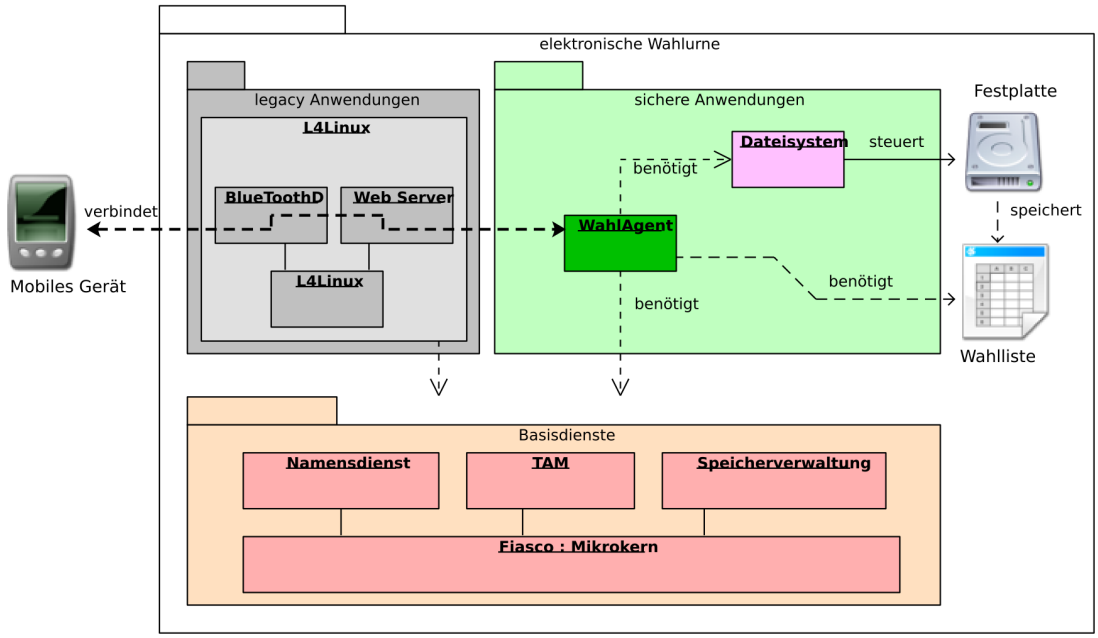

Abbildung 1.1.: Fallbeispiel: Die elektronische Wahlurne. Ein Wähler verbindet sich mit seinem mobilen Gerät zu der elektronischen Wahlurne, um ihre Sicherheit zu messen und anschließend zu wählen. Die Auszählung erfolgt in einem besonders gesicherten Programm `WahlAgent`, worauf das potenziell *unsichere*, virtualisierte Linux-Betriebssystem (L^4Linux) keinen Zugriff besitzt.

mobiles Gerät. Der Gedanke dahinter ist, dass der Wähler ein Gerät seines Vertrauens verwendet, um von der Wahlurne ein Prüfzertifikat zu erhalten und auszuwerten.

Die Wahllisten werden auf einer Festplatte vorgehalten. Jeder Wahlkreis besitzt seine eigene Wahlliste. Somit kommen für ein Bundesland mehrere Dutzend Wahllisten zusammen. Um die Kosten in der Entwicklung der Urnen zu senken, sollen alle Urnen einen identischen Softwarebestand besitzen, dass heißt jede Wahlurne kann mit jeder Wahlliste arbeiten. Die Korrektheit der Wahlurne ergibt sich mitunter aus der jeweils eingesetzten Wahlliste, die bestenfalls im Wahllokal bestimmt werden kann.

Die Anwendungen auf der Wahlurne werden in drei große Blöcke unterteilt: Basisdienste, sichere Anwendungen und sogenannte *legacy*-Anwendungen (oder *legacy*-Systeme). Die Basisdienste bilden das Betriebssystem. Die *legacy*-Anwendungen können ganze Betriebssysteme sein, die virtualisiert werden (z.B. `L4Linux` aus Abbildung 1.1). Diese sind zwar flexibel, jedoch wegen ihrem monolithischen und komplexen Aufbau nur schwer sicher zu gestalten (Abschnitt 2.3 geht genauer darauf ein). Deshalb werden kritische Anwendungen aus diesen *legacy*-Anwendungen herausgelöst und als sichere Anwendungen im System umgesetzt. Das Auszählen der Stimmen erfolgt in der Anwendung `WahlAgent`, das als sichere Anwendung umgesetzt wird. Es ist inbesondere gut von den anderen Anwendungen isoliert und besitzt einen geringen Funktionsumfang. Dadurch kann während der Entwicklung ein ganz besonderes Augenmerk auf die Sicherheit gelegt werden.

Diese Konstruktion verbindet die Flexibilität herkömmlicher Betriebssysteme mit der Sicherheit von Mikrokern-Systemen. Diese Architektur, auch *Nizza*-Architektur genannt, wird in Abschnitt 2.3 eingeführt.

Möchte der Wähler seine Stimme abgeben unternimmt er zwei Schritte. Zuerst prüft er das System mit seinem mobilen Endgerät. Kann er dem System vertrauen, so gibt er seine Stimme ab.

Die elektronische Wahlurne ist nicht unbedingt das geeignetste Beispiel für die Analysen von Datenschutz und Anonymität. Dennoch ist es ein sehr intuitives Beispiel für den Einsatz von entferntem Attestieren. Die Analysen zu Datenschutz und Anonymität werden somit hypothetisch am Beispiel der Wahlurne durchgeführt. Der Einsatz von Eigenschaften wiederum lässt sich sehr einfach nachvollziehen. Zum einem besitzen die Systeme dutzende von Wahllisten, jedoch ist jeweils nur eine für einen gegebenen Wahlkreis korrekt. Die mobilen Endgeräte können diese Listen vor Ort prüfen und die Sicherheit und Korrektheit ermitteln. Der weitere Vorteil dieses Fallbeispiels ist, dass es recht einfach aufgebaut ist, und somit für eine Reihe von Diskussionen, die im weiteren Verlauf stattfinden werden, herangezogen werden kann.

1.2. Abgrenzung

Es ist nicht das Ziel dieser Arbeit, eine sichere elektronische Wahlurne zu entwerfen oder die elektronische Wahlurne zu bewerten. Auch betrachte ich in dieser Arbeit nicht die Beweggründe, warum so viele moderne Demokratien[2] die elektronische Wahlurne befürworten.

Diese Arbeit beschäftigt sich ausschließlich mit den Problemen des entfernten Attestierens. Das schließt das Attestieren, sicheres und überprüfbares Urladen, und das Messen der Sicherheit ein. Zudem beschränkt sich die Betrachtung auf das Gerät, das attestiert werden soll (die Wahlurne); das mobile Gerät, das die Urne überprüft, wird in dieser Arbeit nicht behandelt.

1.3. Gliederung

Diese Arbeit beruht auf den drei großen Themengebieten *Entferntes Attestieren*, *Mikrokern-Architekturen* und *Eigenschaften*. Zusätzlich sind Kenntnisse im Gebiet der Kryptographie und im Einsatz von TPM-Geräten vorausgesetzt.

Die Grundkenntnisse werden in Kapitel 2 vermittelt. Anschließend erfolgt eine Analyse der Themenbereiche *Entferntes Attestieren* und *Eigenschaften*. In Kapitel 5 werden aus den Ergebnissen dieser Studien das Modell und die Algorithmen für das entfernte Attestieren von Eigenschaften vorgestellt. Kapitel 6 ist eine kurze Überleitung zu der Referenzimplementierung. Zuletzt folgt die Auswertung der Ergebnisse auch in Bezug auf die gestellte Aufgabe und der Ausblick.

1.4. Notation

Korollare, Lemma und Co.

Die Anwendung von Definitionen, Theoremen, Korollaren und Lemmata sollen den Leser unterstützen. Definitionen drücken dabei eine möglichst eindeutige Festlegung oder Bedeutung eines Begriffes aus. Ein Theorem ist eine Aussage, die immer einen Beweis fordert. Lemmata sind Theoreme, die nicht bewiesen werden, da der Beweis in verwandten Arbeiten geführt wird.

Viele Theoreme, Beweise und Lemmata lassen einfache Schlussfolgerungen zu, bei denen auf einen expliziten Beweis verzichtet wird. Solche Schlussfolgerungen werden Korollare genannt.

[2] z.B. Deutschland, Vereinigte Staaten von Amerika, Estland und viele mehr.

Darüber hinaus besitzt diese Arbeit noch besondere Listen und Algorithmen, die allesamt durchnummeriert sind. Die Nummerierung folgt den Abschnitten: Algorithmus 3.6.2 ist in Abschnitt 3.6 zu finden. Für die dritte Stelle wird nicht zwischen Definitionen, Algorithmen usw. unterschieden, sondern sie wird fortlaufend inkrementiert.

Mengen-Referenz

Mengen

$\mathbb{N} := \{0, 1, 2, \cdots\}$	Menge der natürlichen Zahlen.
$\mathbb{B} := \{0, 1\}$	Binäre Menge. Wird auch als boolesche Menge eingesetzt, wobei 1 für *wahr* steht.
$H := \mathbb{B}^n$	Menge aller Prüfsummen. Wertebereich der Funktion $\mathcal{H}$. Für SHA1 gilt $n = 160$.
$K := \mathbb{B}^*$	Menge aller Schlüssel. Es wird nicht zwischen asymmetrischen und symmetrischen Schlüssel unterschieden.
$R := H^n$	Menge aller Zustände des PCR-Registersatzes. n ist die Anzahl der Register im System (vgl. Abschnitt 2.1.2).
$S := \mathbb{B}^*$	Menge aller Signaturen.

Symbole und Funktionen

Δ	Referenz für das entfernte Attestieren. Referenz enthält eine Liste aller sicheren Anwendungen. *Seite 23.*
$\mathcal{O}$	Das *Orakel* verwaltet die Referenz Δ und ist für das Prüfen eines Attests notwendig.
$\mathcal{H}$	Kollisionsresistente Einwegfunktion. Diese Arbeit setzt für $\mathcal{H}$ den Algorithmus SHA1 ein. *Abschnitt 2.1.2.*
$\mathcal{I}$	Die Interpretierungs-Funktion $\mathcal{I}$ kann aus einem gegebenen Zustand des Systems ermitteln, ob das System sicher ist, indem sie den Zustand mit der Referenz Δ vergleicht. *Seite 23.*
Γ	Funktion zur Berechnung des Soll-PCR-Registersatzes anhand eines Ladeprotokolls. *Seite 10.*
$\mathcal{X}$	Funktion die zu einem Programm die passende Objektbeschreibung (Menge der Eigenschaften) ermittelt. *Definition 5.2.7, Seite 50.*
$\mathcal{T}$	Instanz im System, die den Systemzustand kennt und das Ladeprotokoll führt. *Seite 53.*
$\mathbf{l}$	Das Ladeprotokoll. *Abschnitt 2.1.6.*
$\mathbf{r}$	PCR-Registersatz. *Abschnitt 2.1.2.*
$\|A\|$	Ermitteln der Mächtigkeit (Anzahl der Elemente) einer Menge A.
$a \| b$	Konkatenation der beiden Argumente a und b.

2. Grundlagen

2.1. Trusted Platform Module

Das *trusted platform module* (engl. für *Vertrauenenswürdiges-Plattform-Modul* – TPM) ist ein integrierter Schaltkreis, der mit einem Rechnersystem fest verbaut wird. Er soll modernen Rechnerplattformen zu mehr Sicherheit verhelfen. Das TPM wird von der *Trusted Computing Base* (TCG), einem internationalen Standardisierungs-Gremium [tcg] entwickelt. Dieser Arbeit liegt Version 1.2 der TPM-Spezifikationen zu Grunde [tpm06].

Das TPM-Gerät besteht aus unterschiedlichen Funktionseinheiten, die in Abbildung 2.1 gezeigt werden. Diese sollen kurz eingeführt werden.

Die Funktionseinheit `IO` verbindet den internen TPM-Bus mit dem Rechnersystem. Seit Version 1.2 steht mit der *TPM interface specification* (TIS) [tis06] ein Standard bereit, wie diese Funktionseinheit angesteuert werden soll.

Das TPM stellt sowohl persistenten als auch flüchtigen Speicher zur Verfügung. In dem flüchtigen Speicher werden die Operanden für die kryptographischen Operationen abgelegt. Der persistente Speicher sichert wichtige Schlüssel, die in Abschnitt 2.1.1 näher vorgestellt werden.

Das TPM besitzt eine generische Recheneinheit, die `execution engine`. Für viel verwendete Operationen verfügt es zudem über spezialisierte Co-Prozessoren, wie für die Prüfsummenbildung durch den Algorithmus SHA1[DEJ01], für die asymmetrische Kryptographie mit RSA[JK03], für das Bilden von Authentikationscode durch HMAC[KBC97] und einen kryptographischen Co-Prozessor für symmetrische Ver- und Entschlüsselung. Darüber hinaus besitzt das TPM eine Einheit, um einen Zufallszahlenstrom zu erzeugen, sowie einen Registersatz (engl. *platfom control register* – PCR) aus mindestens 16 je 20 Byte langen Registern.

Die Sicherheit eines TPM-Geräts gründet auf folgende Annahme:

Annahme 2.1.1: *Sicherheit des TPM-Geräts.* Ein TPM-Gerät ist sicher, da es schwierig ist, seine Sicherheitsmechanismen zu umgehen, da es sich um einen hochintegrierten, fest verbauten Schaltkreis mit geringem Funktionsumfang und besonderen Schutzmaßnahmen (*Anti-Tamper-Schutz*) handelt. Die Sicherheitsmechanismen unterbinden die unerlaubte Manipulation des Zustands der PCR-Register oder die unerlaubte Nutzung besonders geschützter Schlüssel oder Operationen. ◇

2.1.1. Besondere asymmetrische Schlüssel

Das TPM-Gerät besitzt unterschiedliche symmetrische und asymmetrische Schlüssel, von denen die wichtigsten hier erläutert werden sollen.

Endorsement Key – Jedes TPM wird vor seiner Auslieferung mit einem eindeutigen asymmetrischen Schlüssel, dem *endorsement key* (EK) initialisiert. Dieser asymmetrische Schlüssel ist dem TPM eindeutig zugeordnet. Der geheime EK darf das Gerät niemals verlassen.

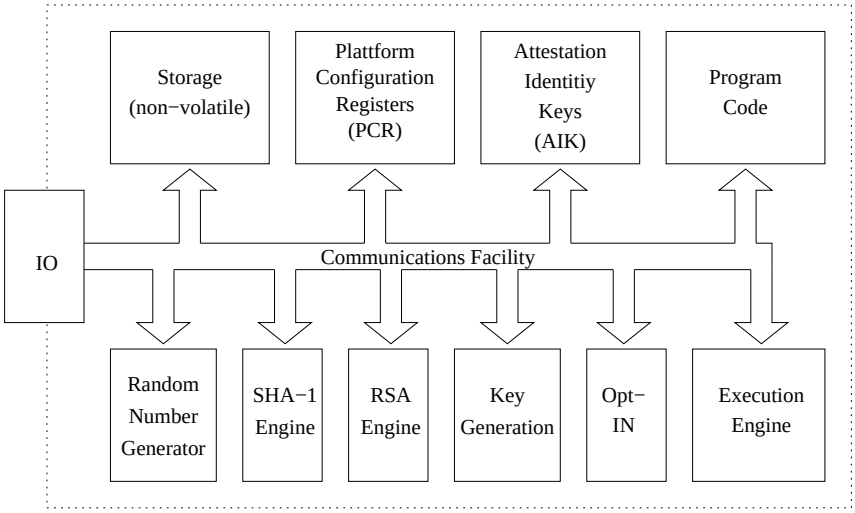

Abbildung 2.1.: Grundaufbau eines TPM-Geräts (nach [tpm06]). Das TPM-Gerät besteht aus mehreren Funktionseinheiten: Persistenten und flüchtigen Speichern, einem Registersatz, einer Kommunikationskomponente, einem Zufallszahlengenerator und unterschiedlichen kryptographischen Co-Prozessoren.

Manufacturer Key – Der *manufacturer key* (MK) ist ein asymmetrischer Schlüssel, der dem Hersteller der ausgelieferten Plattform zugeordnet wird. Bevor das TPM ausgeliefert wird, wird der EK durch den MK signiert.

Attestation Identity Key – Für den Attestierungsprozess (Abschnitt 2.1.7) wird gewöhnlicherweise nicht der EK verwendet. Stattdessen werden für jede Sitzung eine Menge von *attestation identity keys* (AIK) erstellt und für das Signieren verwendet.

Storage Root Key – Ein EK ist einer Plattform fest zugeordnet. Der *storage root key* (SRK) hingegen wird einem Anwender zugeordnet. Wechselt der Anwender, wird der alte SRK verworfen und ein neuer angelegt.

Gewöhnlicherweise wird der SRK für das Verschlüsseln der Daten, mit Hilfe der Operation `Seal` verwendet. Dadurch werden die Daten an einen Anwender gekoppelt. Erzeugt der Anwender einen neuen SRK, dann gehen alle bisherigen Daten verloren, wenn sie vorher nicht dechiffriert und anschließend unter dem neuen SRK wieder chiffriert werden.

2.1.2. Plattform Control Register

Definition 2.1.2: *PCR.* Das *platform control register* (PCR) ist ein 20 Byte langes Register im TPM, das zum Startzeitpunkt des Rechners auf Null zurückgesetzt wird und anschließend nicht mehr beliebig gesetzt werden kann. Die einzige Operation, die ein PCR manipulieren kann, ist `Extend` (Siehe Abschnitt 2.1.3.1). ◇

Sei $\mathcal{H}$ eine kryptographisch sichere Einwegfunktion, die vom TPM für die Prüfsummenbildung eingesetzt wird. Sie bildet eine Eingabe beliebiger Länge auf eine Ausgabe fester Länge n ab. $\mathcal{H}$ ist gegeben durch

$$\mathcal{H} : \mathbb{B}^* \longrightarrow \mathbb{B}^n$$

Die Spezifikation [tpm06] setzt für diese Funktion den Algorithmus SHA1[DEJ01] voraus. Für SHA1 entspricht $n = 160$, und somit 20 Bytes. Für $\mathbb{B}^n$ wird im weiteren Verlauf $H := \mathbb{B}^n$ verwendet.

Das TPM verfügt über mindestens 16 PCR-Register (nach [tpm06]). Die Menge aller Register wird mit $\mathbf{r} = (r_1, \cdots, r_m)$ bezeichnet, wobei m die Anzahl der Register auf dem System ist. $\mathbf{r}$ ist geordnet. Die Register selbst können jeweils einen Rückgabewert der Hash-Funktion $\mathcal{H}$ speichern: $r_i \in H$. Es gilt $\mathbf{r} \in R$ wobei R ein m-dimensionaler Raum aller Zustände ist, die der PCR-Registersatz annehmen kann: $R := H^m$.

Ist $\mathbf{r}^t$ mit dem hochgestellten Index t versehen, so deutet dies auf einen bestimmten Zeitpunkt t hin. Wird t weggelassen ist immer der aktuelle Zustand gemeint. Analog bedeutet r_i^t, das Register an i-ter Stelle aus $\mathbf{r}^t$. $\mathbf{r}^0$ ist der initiale Zustand des Registersatzes.

2.1.3. Die wichtigsten Operationen

Das TPM bietet eine Vielzahl von Operationen, von denen einige wenige für diese Arbeit vorgestellt werden sollen.

2.1.3.1. Extend

Die Operation `Extend` ist die einzige Operation, die ein PCR-Register manipulieren kann. Sie nimmt den gegenwärtigen Zustand r_i^t eines Registers mit Index i und eine Eingabe $d \in H$ und berechnet daraus einen neuen Zustand $r_i^{(t+1)}$ mit der folgenden Formel:

$$\forall r \in \mathbf{r}, r_j^{(t+1)} \quad = \quad \begin{cases} \mathcal{H}(r_j^t \| d) & \text{für } i = j \\ r_j^{(t)} & sonst \end{cases} \tag{2.1}$$

Die Operation `Extend` ist definiert durch:

$$\begin{aligned} \texttt{Extend} \quad &: \quad \mathbb{N} \times R \times H \longrightarrow R \\ \mathbf{r}^{t+1} \quad &= \quad \texttt{Extend}(i, \mathbf{r}^t, d) \end{aligned} \tag{2.2}$$

Im TPM-Gerät kann die Operation `Extend` immer nur auf den aktuellen Zustand des PCR-Registersatz angewendet werden. Der Inhalt des ausgewählten PCR-Registers wird durch das Ergebnis der Operation ersetzt.

Die sukzessive Anwendung der Operation `Extend` bildet eine Prüfsummen-Kette (engl. *hash-chain*), die insbesondere für das Absichern des Ladeprotokolls eingesetzt wird. Dieses Verfahren wird in Kürze erläutert.

2.1.3.2. Quote

Die Operation `Quote` (engl. für notieren, zitieren) erlaubt den Anwendern eine kryptographisch signierte Kopie des PCR-Registersatzes zu erstellen. Sei k ein Schlüssel aus der Schlüsselmenge K.[1] Sei S die Menge aller gültigen Signaturen. Die Operation `Quote` lässt sich vereinfacht folgendermaßen definieren. Sei h ein zufälliger Wert:

[1] Um die Notation möglichst einfach zu gestalten, verwende ich die Schlüsselmenge K sowohl für symmetrische als auch für asymmetrische Schlüssel. Bei letzteren ergibt sich die Tatsache, ob k ein öffentlicher oder privater Schlüssel ist, aus dem Kontext (Operation) seiner Anwendung (vgl. Abschnitt 2.5.1).

$$\begin{aligned} \texttt{Quote} &: H \times R \times K \longrightarrow S \times H \times R \\ (s, h, \mathbf{r}) &= \texttt{Quote}(h, \mathbf{r}, k) \end{aligned} \tag{2.3}$$

Die Ausgabe dieser Operation wird im weiteren Verlauf *Quote* benannt. Der zufällig gewählte Wert h wird in der Signatur berücksichtigt. Dies erlaubt einem Anwender ein eindeutiges *Quote* anzufordern, indem er h bestimmt.

Die Operation `Quote` kann zudem mit einer Maske versehen werden, wodurch sich ausgewählte PCR-Register ausblenden lassen. Die Anwendung dieser Technik wird in Abschnitt 3.4.6 verdeutlicht.

2.1.3.3. Sealing

Die Operationen `Seal` und `Unseal` erlauben es, Daten an eine Plattform und sogar an einen Wunsch- oder Soll-PCR-Registersatz-Zustand $\mathbf{r}'$ zu koppeln.

Führt der Anwender die Operation `Seal` aus, so bestimmt er eine Zielplattform, indem er einen entsprechenden öffentlichen Schlüssel auswählt und die Daten damit verschlüsselt. Die Zielplattform besitzt den passenden privaten Schlüssel – die verschlüsselten Daten sind fortan in dieser Plattform *gefangen*. Das Entschlüsseln der Daten erfolgt über die Operation `Unseal`.

Werden die Daten auf der Zielplattform entschlüsselt, wird der mitgelieferte Soll-PCR-Registersatz-Zustand $\mathbf{r}'$ mit dem gegenwärtigen PCR-Registersatz-Zustand $\mathbf{r}$ der Zielplattform verglichen. Stimmen beide Registersätze nicht überein, so werden die entschlüsselten Daten verworfen und die Operation `Unseal` bricht mit einer Fehlermeldung ab.

Die Operationen sind folgendermaßen definiert:

$$\begin{aligned} \texttt{Seal} &: \mathbb{B}^n \times R \times K \longrightarrow \mathbb{B}^m \\ c &= \texttt{Seal}(d, \mathbf{r}', k) \end{aligned} \tag{2.4}$$

$$\begin{aligned} \texttt{Unseal} &: \mathbb{B}^m \times K \longrightarrow \mathbb{B}^n \\ d &= \begin{cases} \texttt{Unseal}(c, k) & \text{für } \mathbf{r}' = \mathbf{r} \\ \text{Fehler} & \text{sonst.} \end{cases} \end{aligned} \tag{2.5}$$

n und m sind die Längen der Eingabenachricht und der Ausgabe der Operation `Seal`.

2.1.4. Sicheres Urladen

Moderne Rechnersysteme haben das Problem, dass die Software beliebig ausgetauscht werden kann. Diese Tatsache machen sich Hacker zunutze. Moderne Betriebssysteme versuchen dies zu verhindern, worauf die Hacker damit reagieren, dass Teile des Betriebssystems ausgetauscht werden. Um dieses Problem zu beseitigen, entstand das Konzept des sicheren Urladens [Gro91, AFS97].

Arbaugh *et al.* formulieren in [AFS97] die Voraussetzungen für einen sicheren Ladevorgang moderner Rechnersysteme. Ihrer Aussage nach besteht ein Rechnersystem aus mehreren Schichten, die den Startprozess einer Maschine abbilden. Die nächste Aufzählung zeigt einen stark vereinfachten Ladeprozess:

Aufzählung 2.1.3: *IBM PC Startprozess (nach [AFS97]).*

1. *POST* – Ein initialer Zustand, den der Prozessor nach dem Einschalten einnimmt.
2. *System BIOS* – Das BIOS übernimmt die Kontrolle des Rechners. Es initialisiert die Ein- und Ausgabe und alle vorhandenen Geräte im System. Anschließend ermittelt es den Bootsektor und führt das dort hinterlegte Programm, den *bootloader*, aus (z.B. Grub).
3. Der *bootloader* ermittelt das Betriebssystem, lädt dessen Kern in den Speicher und führt ihn aus.
4. Der Kern startet, lädt alle Treiber und re-initialisiert Ein- und Ausgabe.
5. Der Kern startet die Anwendungen.

Jeder einzelne Eintrag in der obigen Aufzählung entspricht einer Schicht. Arbaugh *et al.* behaupten:

Definition 2.1.4: *Sichere Schichten.* Jede Schicht s_i im System kann genau dann als *sicher* angesehen werden, wenn

1. die untere Schicht s_{i-1} bereits als sicher angesehen wird, und
2. der Übergang von Schicht s_{i-1} zu s_i ebenfalls als sicher angesehen wird.

◇

Für den Ladevorgang bedeutet dies, dass wenn ein sicheres BIOS (Schicht 2) vorhanden ist und anschließend ein sicherer *Bootloader* gestartet wird, dann ist das System in der Schicht 3 ebenfalls sicher. Dieser Ansatz setzt sich fort bis zu den letzten Schichten, wo die Anwendungen angesiedelt sind. Dieses Verfahren nennt sich *sicheres Urladen* (engl. *secure booting*).

Aus der Definition 2.1.4 leitet sich die Definition für eine sichere Plattform ab:

Korollar 2.1.5: *Sichere Plattform.* Eine Plattform ist genau dann sicher, wenn alle Schichten im System sicher sind. ◇

Arbaugh *et al.* behandeln in ihren Studien die Probleme, wie die erste Schicht abgesichert werden soll und wie die Messungen der neu geladenen Schichten erfolgen kann. Eine Folgerung ist, dass der Ladeprozess abgebrochen wird, wenn der Übergang in eine neue Schicht das System unsicher machen würde – das System kommt zum Stillstand.

Aus der Definition und aus den Arbeiten von Arbaugh *et al.* lassen sich zwei wichtige Schlüsse ziehen:

Lemma 2.1.6: *Sicherer Übergang.* Ein Übergang von Schicht s_i in eine neue Schicht s_{i+1} ist genau dann *sicher*, wenn die Schicht s_{i+1} als *sicher* angesehen wird. ◇

Lemma 2.1.7: *Sichere erste Schicht – Schicht-0.* Das System kann nur dann *sicher* sein, wenn es über eine erste Schicht (*Schicht-0*) verfügt, die bedingungslos als *sicher* angesehen wird. ◇

Eine Schicht s_{i+1} wird dann als *sicher* angesehen, wenn beispielsweise die Schicht s_i eine Prüfsumme über die Schicht s_{i+1} ermittelt hat und diese Prüfsumme mit einer Datenbank vergleicht.

Für den Einsatz eines TPM-Geräts ist Lemma 2.1.7 wegen der Annahmen 2.1.1 implizit gegeben.

Für Schicht-0 gilt jedoch:

Theorem 2.1.8: *Absicherung der Schicht-0.* Die Sicherheit der Schicht-0 kann selbst nicht durch die Algorithmen und Verfahren vertrauenswürdig gemessen werden, die durch diese Schicht zur Verfügung gestellt werden. ◇

Der Grund für diese Aussage liegt darin, dass wenn ein Angreifer erfolgreich die Schicht-0 in seine Gewalt gebracht hat, dieser auch in der Lage ist, jede Antwort nach belieben zu beantworten. Am Beispiel des TPM-Geräts würde das bedeuten, dass der Angreifer Zugriff auf die internen Zustände und Schlüssel besitzt (insbesondere den EK-Schlüssel) und den PCR-Zustand nach belieben setzen kann. Der Angreifer kann alle Zustände und Signaturen so gestalten, dass das System den Anschein erweckt, es sei sicher. War ein Angreifer erfolgreich, so ist die einzige Möglichkeit, die dem Anwender bleibt, den EK-Schlüssel als ungültig zu erklären.

2.1.5. Überprüfbares Urladen

Das überprüfbare Urladen ist eine Abwandlung des sicheren Urladens. Das Problem mit des sicheren Urladens ist, dass jede Schicht s_i alle Schichten s_{i+1} kennen muss, die ausgeführt werden dürfen. Das überprüfbare Urladen (engl. *authenticated booting*) hingegen beschränkt sich darauf, den Ladevorgang sicher zu protokollieren. Das heißt, die Messungen werden an einem sicheren Ort, vor Manipulationen geschützt, abgelegt.

Die Überprüfung dieses Protokolls kann zu einem späteren Zeitpunkt erfolgen, wenn mehr Ressourcen (z.B. Datenbanken, Netzwerkverbindung und Algorithmen) zur Verfügung stehen. Zudem kann das Protokoll auch auf andere Rechnerplattformen übertragen und überprüft werden. Das ermöglicht erst das entfernte Attestieren.

2.1.6. Ladeprotokoll

Das TPM-Gerät erlaubt, ein sicheres Ladeprotokoll zu erstellen und überprüfbares Urladen zu ermöglichen. Die Prüfsumme d über den Programmtext jeder neu gestarteten Anwendung wird zuvor in das Ladeprotokoll eingetragen. Das Ladeprotokoll wird im Speicher abgelegt. Um es vor Manipulationen zu schützen, wird mit Hilfe der Operation `Extend` eine Prüfsummen-Kette (engl. *hash-chain*) über alle Einträge des Ladeprotokolls im TPM-Gerät hinterlegt. Mit diesem Protokoll kann der Soll-Zustand $\mathbf{r}^t$ des PCR-Registersatzes errechnet werden. Stimmt dieser mit dem Ist-Zustand des Registersatzes überein, so ist das Ladeprotokoll authentisch.

Das Ladeprotokoll $\mathbf{l}$ ist eine geordnete Liste: $\mathbf{l}^t := (l^1, l^2, \cdots, l^t) \in L$. L ist dabei die Menge aller möglichen Ladeprotokolle. Jeder Eintrag ist gegeben durch

$$\begin{aligned} l^t &\in H \times \mathbb{N} \\ l^t &= (d, i). \end{aligned} \tag{2.6}$$

Der Index i deutet auf das PCR-Register, das verwendet wurde (vgl. Operation `Extend`). Der Index t ist die Position dieses Ladeprotokoll-Eintrags im Ladeprotokoll.

Der Soll-PCR-Registersatz-Zustand lässt sich errechnen, indem er vom initialen Zustand $\mathbf{r}^0$, der von der TCG festgesetzt ist und durch die wiederholte Anwendung von `Extend`, gemäß der Anleitung aus dem Ladeprotokoll errechnet wird. Sei die Operation γ die Operation, die den Nachfolgezustand $\mathbf{r}^t$ anhand von $\mathbf{r}^{t-1}$ und dem Eintrag $l^{t-1}i$ des Ladeprotokolls errechnet. Sei Γ die Operation, die γ sukzessiv anwendet, um den Soll-Zustand $\mathbf{r}^t$

der PCR-Register zu errechnen:

$$\begin{aligned} \mathbf{r}^t &= \gamma(\mathbf{r}^{t-1}, l^t) \\ &= \texttt{Extend}(i, \mathbf{r}^{t-1}, d) \qquad (2.7) \\ \Gamma &: L \longrightarrow R \\ \mathbf{r}^t &= \Gamma(\mathbf{l}^t) \qquad (2.8) \\ &= \gamma(\gamma(\gamma(\cdots\gamma(\mathbf{r}^0, l^1)\cdots, l^{t-2}), l^{t-1}), l^t) \end{aligned}$$

Soll das Ladeprotokoll ausgewertet werden, muss auch der Ist-Zustand der PCR-Register bekannt sein. Dieser lässt sich durch die Operation `Quote` ermitteln. Das TPM-Gerät kann sich alte Zustände des Registersatzes nicht merken und auch nicht zurückrechnen. Deshalb kann die Operation `Quote` nur den aktuellen Registerzustand berücksichtigen. Zudem muss das Ladeprotokoll vollständig vorhanden sein, damit die Berechnungen durch die Operation Γ korrekt ausgeführt werden können. Dies hat zur Folge, dass das Ladeprotokoll nicht partiell ausgewertet werden kann. Lediglich die Anwendung der Maske in der Operation `Quote` kann bewirken, dass bestimmte Einträge im Ladeprotokoll nicht berücksichtigt werden müssen und deshalb ausgelassen werden können.

Das Ladeprotokoll muss sich nicht auf die Prüfsumme einer Anwendung beschränken. Sollen noch weitere Informationen im Ladeprotokoll aufbewahrt werden, so kann wie folgt vorgegangen werden: Das Ladeprotokoll besitzt neben den Informationen d und i noch die Information a, die beispielsweise den Namen der Anwendung ausdrückt. Daraus folgt der neue Aufbau eines Ladeprotokolls-Eintrag: $l = (a, d, i)$. Die Eingabe d für die Operation `Extend` (vgl. Gleichung 2.7) wird durch die Prüfsumme über l ersetzt: $\mathcal{H}(a\|d)$, bzw. $\mathbf{r}^t = \texttt{Extend}(i, \mathbf{r}^{t-1}, \mathcal{H}(d\|a))$. Somit lässt sich das Ladeprotokoll nahezu beliebig erweitern. Lediglich Informationen, die sich im Verlauf der Zeit ändern, wie der Zustand eines Prozess (Aktiv, Blockiert, Beendet etc.) lassen sich nicht durch diesen Ansatz mit aufnehmen.

2.1.7. Attestieren und entferntes Attestieren durch das Ladeprotokoll

Mit Hilfe des Ladeprotokolls und der Operation `Quote` lässt sich der Zustand einer Plattform sicher nachweisen. Dazu ist noch ein *Orakel* notwendig, das die Prüfsummen aller Programme kennt, die als sicher angesehen werden können. Darüber hinaus kennt das Orakel alle TPM-Geräte, die durch ihre EK-Schlüssel identifiziert werden, und kann feststellen, ob eine Signatur von einem solchen (sicheren) TPM-Gerät stammt.

Sind das Ladeprotokoll und die Ausgabe der Operation `Quote` bekannt, kann das Ladeprotkoll wie folgt geprüft werden. Die Operation Γ errechnet anhand des Ladeprotokolls den Soll-Zustand der PCR-Register, die mit der Ausgabe der Operation `Quote` auf Gleichheit getestet werden.

Nun muss noch festgestellt werden, ob die Ausgabe von der Operation `Quote` wirklich durch ein vertrauenswürdiges TPM-Gerät erstellt wurde. Diese Ausgabe wird durch das TPM-Gerät mit dem Schlüssel EK unterzeichnet. Zudem wurde der Schlüssel EK durch den Schlüssel MK signiert. Diese Signatur s_{EK}^{MK} muss ebenfalls vorliegen, sowie der öffentliche Schlüssel EK.

Die Ausgabe der Operation `Quote` wird mit dem öffentlichen Schlüssel EK geprüft. Anschließend wird EK durch s_{EK}^{MK} geprüft. Kennt das Orakel den Schlüssel MK, so stammt das Quote aus einem gültigen TPM-Gerät.

Um sicherzustellen, dass das System sicher ist, besitzt das Orakel eine Liste der Prüfsummen aller sicheren Anwendungen. Jeder Eintrag im Ladeprotokoll muss in dieser Referenz vorhanden sein. Erst dann kann das System als sicher angenommen werden.

Das Entfernte Attestieren mit Hilfe des Ladeprotokolls wird in Abschnitt 3.4 gründlich vorgestellt.

2.2. Mikrokern-Systeme und L4

Ein Mikrokern bezeichnet einen Betriebssystem-Kern, der im Gegensatz zu den monolithischen Kernen nur sehr wenige Funktionen anbietet. Üblicherweise besitzt ein Mikrokern nur Funktionen für die Prozess- und Speicherverwaltung sowie elementare Mechanismen zur Synchronisation und Kommunikation[Lie95].

Damit Mikrokern-Systeme den Funktionsumfang monolithischer Kerne anbieten können, sind sie auf zusätzliche Komponenten angewiesen. Solche Komponenten können Gerätetreiber, komplexe Speicherverwaltung usw. sein. Im Unterschied zu monolithischen Systemen sind diese zusätzlichen Komponenten eigenständige Prozesse mit eigenständigen Adressräumen und eingeschränktem auf die Hardware. Durch diese Isolation zwischen den Komponenten und durch die Tatsache, dass jede Komponente relativ einfach gehalten werden kann und eine definierte Schnittstelle besitzt, soll die Zuverlässigkeit des Systems erhöht werden[2].

L4 ist eine ursprünglich von Jochen Liedtke entwickelte Mikrokern-Plattform[Lie95]. Fiasco ist ein L4-Mikrokern, der an der Technischen Universität Dresden entwickelt wird[tud]. Es besitzt weniger als 15.000 Zeilen Code (nach [HHF+05], Abschnitt 4.2). Im Vergleich dazu besitzt der Linux-Kern in Version 2.6.17 über 5 Millionen Zeilen Code[3].

2.3. Nizza

Die Nizza-Architektur[HHF+05] vereint die Vorteile monolithischer Betriebssysteme mit der Mikrokern-Architektur. Die Nizza-Architektur besteht aus einem Mikrokern und einer Menge von Basisdiensten. Darüber hinaus bietet sie die Möglichkeit, monolithische Betriebssysteme in sogenannten Partitionen (engl. *legacy container* oder *legacy partition*) auszuführen.

Monolithische Systeme wie *Linux* oder *Windows* besitzen die wesentlichen Vorteile, dass durch ihre große Verbreitung die meisten Anwender im Umgang mit ihnen vertraut sind und diese Systeme ein sehr großes Software-Repertoire besitzen. Doch durch ihre monolithische Architektur und ihre Komplexität sind diese Systeme sehr anfällig für Fehler und können somit schnell die Sicherheit des Systems gefährden.

Nizza bietet die Möglichkeit, bestimmte Funktionalitäten in andere Adressräume auszulagern. Dadurch können besonders sensible Bereiche aus den monolithischen Betriebssystemen ausgelagert werden, die dann durch Addressraumgrenzen und den Mikrokern geschützt werden. Solche ausgelagerten Teile könnten z.B. das Schlüsselmaterial aufbewahren. Ein Angreifer, der die Sicherheitslücken des monolithischen Systems ausnutzt und dort vollen Zugriff erlangt, bleibt in dieser Partition *gefangen* und besitzt keinen direkten Zugriff auf das Schlüsselmaterial.

[2] Im Gegensatz zu Mikrokern-Systemen besitzt jeder Treiber in monolithischen Systemen einen privilegierten Zugriff auf die Hardware und kann jedes beliebige Programm beeinflussen.

[3] Gemessen mit der Anwendung `sloccount` von David A. Wheeler, `http://www.dwheeler.com/sloccount`

Auf die gleiche Weise können unsichere Anwendungen mit einer sicheren Hülle verpackt werden, die Ein- und Ausgabe steuert. Damit lässt sich Linux beispielsweise zu einem *Treiber-Linux* umrüsten. Dieses *Treiber-Linux* betreibt somit die Hardware und leitet die Daten an andere Partitionen oder Komponenten weiter, wo die Daten anschließend bearbeitet werden. Diese Kapselung unsicherer Komponenten ist unter Nizza als *trusted wrappers* [HHF+05] bekannt.

2.4. Trusted Computing Base

Die meisten Anwendungen sind auf zusätzliche Ressourcen, z.B. Speicher, Algorithmen oder Dienstleistungen angewiesen. Diese Ressourcen werden von weiteren Programmen verwaltet. Über diese Ressourcen besitzen diese Programme Einfluss auf die Sicherheit der Anwendung. Beispielsweise könnte der Festplattentreiber den Inhalt von Dateien frei diktieren.

Aus diesem Grund besitzt jede Anwendung eine Menge von Diensten, auf die sie angewiesen ist und denen sie trauen muss. Diese Menge vertraulicher Basisdienste ist unter dem Namen *Trusted Computing Base* (TCB) bekannt. Jede Anwendung besitzt ihr eigenes TCB.

Definition 2.4.1: *Trusted Computing Base.* Die *Trusted Computing Base* (engl. für *sichere Programm-Basis* – TCB) einer Anwendung besteht aus

1. der Menge aller privilegierten Anwendungen, die einen bevorzugten Zugriff auf die Ressourcen des Systems (z.B. der Kern) oder auf die Ressourcen der Anwendung (z.B. Debugger) besitzen.

2. der Menge aller Anwendungen, die dem Programm Ressourcen zur Verfügung stellen (Dateien, Algorithmen etc).

◇

Um nochmals den Unterschied zwischen monolithischen Systemen und Mikrokernen zu verdeutlichen, ist die Menge der privilegierten Anwendungen (Punkt 1) bei Mikrokernen sehr gering und beinhaltet insbesondere nicht zwangsläufig irgendwelche Gerätetreiber.

2.5. Kryptographie

Diese Arbeit muss auf eine gründliche Einführung im Gebiet der Kryptographie verzichten und auf Fachliteratur [Sch95, MVO96] verweisen. Die nachfolgende Einführung beschränkt sich auf das Wesentlichste.

2.5.1. RSA

RSA ist ein asymmetrisches Authentikations- und Konzellationssystem. Es besitzt ein Schlüsselpaar, das aus öffentlichem und geheimem Schlüssel besteht. Der geheime Schlüssel lässt sich nicht effizient aus dem öffentlichen Schlüssel errechnen.

RSA erlaubt digitale Signaturen mit Hilfe des geheimen Schlüssels anzufertigen, die anschließend mit dem öffentlichen Schlüssel überprüft werden können. Die Operation `Sign`

erstellt eine Signatur s über einen geheimen Schlüssel k^{sec} zu einem Datum d. Die Operation `Verify` prüft die Signatur mit einem gegebenen öffentlichen Schlüssel k^{pub}:

$$\begin{aligned} \texttt{Sign} &: K \times B^* \longrightarrow S \\ s &= \texttt{Sign}_{k^{\text{sec}}}(m) \qquad (2.9) \\ \texttt{Verify} &: K \times S \longrightarrow \mathbb{B} \\ b &= \texttt{Verify}_{k^{\text{pub}}}(s) \qquad (2.10) \end{aligned}$$

Somit kann ein Anwender, Bob, der den privaten Schlüssel k^{sec} besitzt, eine Nachricht d unterschreiben. Vorausgesetzt Alice weiss, dass der öffentliche Schlüssel k^{pub} auch wirklich Bob gehört, kann Alice prüfen, dass Bob die Nachricht d unterschrieben hat.

Für eine ausführliche Einleitung in RSA wird auf [JK03, Sch95, MVO96] verwiesen.

2.5.2. Vertrauen im Internet messen

Die asymmetrische Kryptographie bietet eine Möglichkeit, Vertrauen im Internet effizient zu messen. Angenommen Bob traut Alice. Bob besitzt ein asymmetrisches Schlüsselpaar $b = (b^{\text{pub}}, b^{\text{sec}})$. Auch Alice besitzt so ein Schlüsselpaar $a = (a^{\text{pub}}, a^{\text{sec}})$.

Alice kann Bob ihr Vertrauen aussprechen, indem sie seinen öffentlichen Schlüssel signiert: $s_b^a = \texttt{Sign}_{a^{\text{sec}}}(b^{\text{pub}})$. Bob kann mit dieser Signatur Dritte überzeugen, dass Alice ihm traut.

Christoph besitzt ebenfalls ein Schlüsselpaar c. Zudem traut Bob Christoph. Wird Vertrauen als transitive Eigenschaft gewertet, so folgt: Wenn Bob Christoph traut, und Alice traut Bob, dann traut Alice auch Christoph.

Durch dieses Verfahren lässt sich eine transitive Hülle aufspannen, die das Vertrauen im Internet messbar macht. Die *public key infrastructure* (PKI) [HPFS02] ist ein Beispiel, wo dieses Prinzip zum Einsatz kommt.

Das PKI ist für diese Arbeit interessant, denn es spannt einen Baum mit mehreren Wurzelknoten, den *certificate authorities* (CA) auf. Die Signaturen werden in Zertifikaten mit zusätzlichen Informationen wie Eigentümer und Adresse aufbewahrt.

Auch das TPM-Gerät nutzt eine PKI, wobei die MK-Schlüssel die Wurzelknoten (oder CA) des PKI-Baums bilden. Wie bereits in Abschnitt 2.1.7 dargestellt, ist das Vertrauen in ein Ladeprotokoll auf eine Ausgabe der Operation `Quote` und durch die transitive Hülle über alle MK, alle EK und alle Signaturen von EK durch MK begründet.

2.5.3. Anonymität und Schutz der Privatsphäre

Ein TPM-Gerät besitzt durch seinen eigenen Signaturschlüssel EK nicht nur eine eindeutige Identifizierung, sondern das Ladeprotokoll offenbart alle gestarteten Programme im System.

In dieser Arbeit sollen zwei Aspekte untersucht werden. Zum einem soll die Frage erörtert werden, inwiefern Anonymität umgesetzt werden kann. Gibt der Anwender seine Identität bekannt, so stellt sich noch die Frage nach dem Informationsfluss und inwiefern der Anwender darauf Einfluss nehmen kann.

Anonymität ist folgendermaßen definiert (nach [Pfi00]):

Definition 2.5.1: *Anonymität.* Sei r eine Rolle, die ein Individuum in einer Kommunikation einnimmt. Sei b die Beobachtung eines Ereignisses e.

Ein Individuum in der Rolle r ist *anonym*, wenn ein Angreifer anhand der Beobachtung b des Ereignisses e schließen kann, dass die Wahrscheinlichkeit, dass e und r kooperieren (Operation $\sim$) größer Null, jedoch echt kleiner Eins ist:

$$0 < P(r \sim e \mid b) < 1$$

◇

Es besteht immer die Möglichkeit, dass die Anonymität aufgedeckt wird. Deshalb ist die Wahrscheinlichkeit immer echt größer Null. Mathematisch gesehen ist die Aussage *echt kleiner Eins* ebenfalls korrekt, obwohl in der Praxis dieser Wert durch eine sehr viel kleinere Zahl ϵ begrenzt sein sollte.

Die Wahrscheinlichkeit ϵ ist durch die Anzahl n der Individuen die als Urheber in Frage kommen, nach unten hin durch $1/n$ beschränkt: $1/n \leq \epsilon < 1$. Erst wenn diese Anzahl hinreichend groß ist, kann überhaupt von *Anonymität* gesprochen werden.

Die obige Definition betrachtet nicht den Inhalt des Ereignisses e. Doch besonders in modernen Kommunikationsnetzwerken kann dieser Inhalt eindeutige Adressen besitzen, die zumindest die Nachrichten miteinander verketten lassen – Nachrichten gleichen Urhebers besitzen meistens auch die gleiche Absenderadresse.

Der Datenschutz verwendet unterschiedliche Modelle für die theoretischen Betrachtungen. Eines dieser Modelle ist das *Mosaik-Modell*, das besagt, dass durch Verkettung einzelner Informationen ein Gesamtbild des Individuums entsteht. Ein weiteres Modell, das *Rollen-Modell*, besagt, dass ein Individuum im Leben unterschiedliche Rollen besitzt (z.B. Vater, Versicherter, Patient usw.). Die Ausübung einzelner Rollen erfordert die Anwendung unterschiedlicher personenbezogener Daten, die zusammengeführt ein gesamtes Bild ergeben können.

Die Definition der Anonymität ist somit unzureichend. Vielmehr ist die Verkettbarkeit der Daten das Problem (nach [Pfi00]):

Definition 2.5.2: *Unverkettbarkeit.* Zwei Ereignisse e und f sind bezüglich eines Merkmals m unverkettbar, wenn ein Angreifer aus der Beobachtung von e und f mit einer Wahrscheinlichkeit echt größer Null, aber echt kleiner Eins, sagen kann, dass beide Ereignisse das Merkmal m besitzen:

$$0 < P(e \sim_m f \mid e, f) < 1$$

◇

Das Merkmal m kann sich sowohl auf die Identität als auch auf den Inhalt der Ereignisse beziehen. Somit kann die Definition von Unverkettbarkeit auch dazu verwendet werden, Anonymität auszudrücken.

Beispiel. Der Ferrari-Fahrer eines kleinen Dorfes in Deutschland kann sich nicht anonym bezeichnen, wenn in dem Dorf ein Ferrari gesichtet wird, da alle Ferrari-Besitzer aus dem Dorf bekannt sind und die Wahrscheinlichkeit, dass es mehr als einen Ferrari-Besitzer gibt, sehr gering ist.

In diesem Beispiel wird der Fahrer nicht direkt identifiziert. Die Verkettung erlaubt jedoch eine relativ sichere, wenn gleich nicht eindeutige Bestimmung des Fahrers.

Die Kryptographie bietet jedoch Verfahren, in denen die Anonymität wesentlich besser geschützt ist. Eines dieser Verfahren ist die Gruppensignatur.

In dieser Arbeit soll die Frage nach dem Datenschutz durch eine Betrachtung der Verkettbarkeit erfolgen. Grundsätzlich gilt es, Verkettbarkeit zu verhindern. Dies beinhaltet insbesonders, dass die Identität des Anwenders oder der eingesetzten Programme nicht preisgegeben werden.

2.5.4. Gruppensignaturen

Definition 2.5.3: *Gruppenanonymität.* Sei G eine Gruppe von Individuen $g \in G$. g ist genau dann *anonym bezüglich eines Ereignisses* e, wenn der Urheber von e zwar eindeutig der Gruppe G, jedoch nicht einem speziellen Individuum g dieser Gruppe zugeordnet werden kann. ◇

Gruppensignaturen gibt es in der Kryptographie in unterschiedlichen Varianten. Ihr Ursprung liegt in der Veröffentlichung von D. Chaum [Cha03]. Alle besitzen jedoch den gleichen Aufbau: Ein Individuum wendet sich an eine Instanz, die die Gruppenmitgliedschaft bestimmt. Das Individuum weist sich aus und erhält eine Identität innerhalb der Gruppe, die es fortan einsetzen kann.

Seit der Veröffentlichung 1991 ist das Konzept mehrfach weiter entwickelt worden. Die einzelnen Verfahren auf ihre Tauglichkeit für RA zu untersuchen, ist nicht Gegenstand der Arbeit. Zudem besitzt das TPM-Gerät bereits einen solchen Algorithmus, der kurz DAA genannt und in Folge vorgestellt wird.

2.5.5. Direct Anonymous Attestation

Direct anonymous attestation (DAA) [BCC04] kann als eine Form von Gruppensignaturen angesehen werden. Es wurde von Brickel, Camenisch und Chen entwickelt. Mit *idemix*[CH02] existiert ein digitales anonymes System, Berechtigungen nachzuweisen (engl. *anonymous credential system*), das auf DAA beruht.

DAA setzt das Prinzip *zero-knowledge-proof* [Cha03, FFS87] um. Das bedeutet, dass die eingesetzten Signaturen, die die Gruppenmitgliedschaft belegen, beliebig oft eingesetzt werden können, ohne das dabei Informationen über die Signaturen preisgegeben werden. Werden zwei unterschiedliche Nachrichten durch eine Gruppenidentität unterzeichnet, so kann dennoch nicht ermittelt werden, dass beide Nachrichten den selben Urheber besitzen (Unverkettbarkeit der Ereignisse bezüglich *einer* Gruppenidentität).

Das TPM-Gerät verfügt seit Spezifikation 1.2 über diesen Algorithmus.

2.5.6. Certificate Revocation List

Unter bestimmen Gegebenheiten kann es notwendig sein, die Gruppenmitgliedschaft nachträglich aufkündigen zu müssen. Dafür werden sogenannte *certificate revocation list* (CRL) [HPFS02], die in regelmäßigen Abständen oder je nach Notwendigkeit veröffentlicht werden, eingesetzt. Die CRL besitzt eine Liste aller (Gruppen-) Identitäten, die nachträglich aus der Gruppe entfernt wurden. Ähnliches gilt auch für TPM-Geräte, die nicht mehr sicher sind, aber über einen signierten EK-Schlüssel verfügen.

Die CRL besitzen einige große Nachteile:

Aufzählung 2.5.4: *Nachteile einer CRL.*

1. Die Liste wird immer länger, da jedes ungültige Zertifikat bzw. Gruppenmitglied auf ewig gespeichert werden muss.
2. Systeme sind nur dann sicher, wenn sie über eine aktuelle CRL-Liste verfügen.
3. Jeder Teilnehmer muss die (Gruppen-)Identität ermitteln können, um sie mit der CRL-Liste vergleichen zu können[4].

[4] DAA gibt zwar die Gruppenidentität nicht preis, es verwendet jedoch Techniken, um unsichere TPM-Geräte dennoch identifizieren zu können.

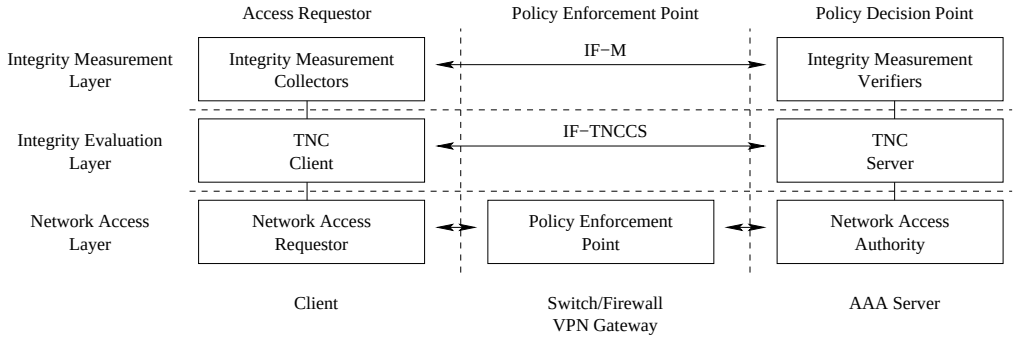

Abbildung 2.2.: TNC-Architektur nach [tnc06]. Die TNC-Architektur besitzt drei Rollen und drei Abstraktionsebenen. Der *access requestor* (AR) möchte sich über den *policy enforcement point* (PEP) zu einem Netzwerk (hier nicht abgebildet) verbinden. Der PEP lässt den AR durch den *policy decision point* (PDP) prüfen. Die Messung der Sicherheit geschieht in der *integrity measurement layer* (IML) z.B. durch das Auswerten des Ladeprotokolls.

Eine Alternative zu den CRL ist, die Gruppe vollständig aufzulösen und eine neue Gruppe zu erstellen. Dieser Aufwand ist jedoch größer als die CRL-Listen zu verwalten. Auch Zeitstempel, die den Verfall der Gruppenmitgliedschaft bewirken, helfen nicht weiter, wenn das System für die Langzeitarchivierung eingesetzt wird.

Die Ursache für die Existenz der CRL-Listen ist, dass die Sicherheit aufgrund einer Annahme ausgesprochen wird. Stellt sich im Nachhinein heraus, das diese Annahme zu unrecht getroffen wurde, so geschieht das aufgrund neuer Informationen. Diese Informationen müssen dem System hinzugefügt werden. Diese Tatsache hat zur Folge, dass CRL-Listen oder äquivalente Lösungen unverzichtbar sind und somit der zweite Kritikpunkt auf der obigen Liste bei jeder Lösung in irgendeiner Weise vorhanden ist.

2.6. Das Trusted Network Connect

Das *trusted network connect* (TNC) [tnc06] versteht sich als Rahmenwerk für die entfernte Attestierung. Es definiert Rollen und Schnittstellen, die für RA notwendig sind, doch es beschreibt keine konkrete Form für das entfernte Attestieren, wie z.B. das entfernte Attestieren durch das Ladeprotokoll. Die Architektur der TNC ist in Abbildung 2.2 dargestellt. Die TNC wird von der TCG[tcg] entwickelt.

2.6.1. Rollenwahl der TNC

Das TNC definiert die drei Rollen *access requestor* (AR), *policy enforcement point* (PEP) und *policy decision point* (PDP).

Der *access requestor* erbittet Zugriff auf das System und muss sich zuvor als sicher ausweisen. Demnach entspricht er dem Dienstleister, wobei PEP und PDP dem *Herausforderer*, also dem mobilen Gerät aus dem Fallbeispiel entsprechen. Der PEP ist der Teil in der Anwendung im mobilen Gerät, der aufgrund des Attests entscheidet, ob die Wahl ausgeführt wird. Der PDP wertet das Attest aus und übermittelt dem PEP seine Entscheidung.

2.6.2. Abstraktionsschichten

Die TNC definiert zudem drei Abstraktionsschichten, die *network access layer* (NAL), *integrity evaluation layer* (IEL) und die *integrity measurement layer* (IML). Die *unterste* Schicht NAL befasst sich mit dem Transport der Informationen, die für RA notwendig sind. Diese Schicht soll insbesondere den Transport über die gängigsten Technologien wie TCP/IP, HTTP, Ethernet usw. ermöglichen.

Diese Daten gelangen dann in die *integrity evaluation layer* (IEL). Diese Schicht wertet das Resultat der Attestierung aus. Das Attest selbst wird in der *integrity measurement layer* (IML) erstellt und ausgewertet. Das Resultat der Messung wird an den PEP zurückgegeben, dass die Zugangskontrolle erwirkt.

2.6.3. Schnittstellen

Die TNC beschreibt keinen Algorithmus für das entfernte Attestieren. Diese Algorithmen, die in der IML-Schicht angesiedelt sind, sollen durch Dritte implementiert werden. Dabei sind proprietäre Lösungen angedacht. Allein die Schnittstelle zur IEL-Schicht ist vorgegeben. Ähnlich ergeht es der NAL-Schicht. Da die Vielfalt der Übertragungsmedien sehr groß ist, beschränkt sich die TNC lediglich darauf, Schnittstellen bereit zu stellen. Wie das Protokoll letztendlich auf ein Übertragungsmedium umgesetzt wird, wird in zusätzlichen Standards beschrieben. Für viele Anwendungen sind jedoch noch keine Standards vorhanden. Die Schnittstellen werden durch XML-Schemata beschrieben, und auch für den Informationsaustausch ist XML vorgesehen.

Aus diesen Gründen versteht sich die TNC eher als Rahmenwerk für das entfernte Attestieren.

2.6.4. Relevanz der TNC für diese Arbeit

In dieser Arbeit steht nicht die Integration eines bestehenden Algorithmus für das entfernte Attestieren mit anderen Systemen im Vordergrund, sondern es sollen Untersuchungen in Bezug auf den Datenschutz und das Messen von Eigenschaften durchgeführt werden. Das Resultat dieser Arbeit ist eine Komponente aus der IML-Schicht, die das TNC als Übertragungsmedium nutzen kann.

Das Augenmerk der TNC liegt auf der Seite des Herausforders (Rollen PEP und PDP). Die Maschine, die ihren Zustand darlegen soll, besitzt lediglich die Rolle AR, die das TPM-Gerät repräsentiert. Spätere Untersuchungen (Kapitel 3) werden jedoch verdeutlichen, dass dieser Ansatz unzureichend ist, einen effektiven Datenschutz zu gewähren, und dass eine Zweiteilung des AR angebracht ist. Die erste Rolle verwaltet das TPM-Gerät, die zweite Rolle wird von der Anwendung wahrgenommen, die gemessen werden soll. Die zweite Rolle besitzt eine ähnliche Funktion wie die PEP. Sie kann das Resultat der Attestierung einsehen und gegebenenfalls den Prozess abbrechen, wenn beispielsweise bestimmte Datenschutzkriterien nicht gewahrt wurden.

Da die TNC sich explizit aus der Gestaltung der eigentlichen Messung des Systems heraus hält (Untersuchung der Algorithmen) und weil das gewählte Modell für die Analyse in Bezug auf den Datenschutz nicht vollständig ist, ist die TNC komplementär zu dieser Arbeit. Sie spielt jedoch eine eher untergeordnete Rolle.

2.7. Verwandte Arbeiten

Neben den bisher erwähnten Arbeiten sind noch folgende Arbeiten mit meinem Vorhaben in dieser Arbeit verwandt.

B. Kauer untersuchte in seinem Großem Beleg [Kau04] bereits das Thema überprüfbares Urladen unter L4.

In [PSHW04] beschreiben Poritz *et al.* die Vorteile, die das Attestieren von Eigenschaften mit sich bringt. Auch Sadeghi und Stüble [SS04] beschreiben, wie das Attestieren von Eigenschaften mit einem TPM gekoppelt werden kann.

Weitere Projekte, die sehr nah mit dem Vorhaben dieser Arbeit verwandt sind, sind die Untersuchungen im Rahmen von EMSCB [ems], und das europäische Projekt OpenTC [ope]. Sie befassen sich mit der Aufgabe, eine sichere Software-Architektur zu entwickeln. Ihr erklärtes Ziel ist es, eine Sicherheitsschicht zwischen einem Mikrokern und den Anwendungen zu schaffen.

Die Grundlage für die Referenzimplementierung stammt von der Technischen Universität Dresden, die in den letzten Jahren sowohl eine breite Softwarebasis, als auch eine Entwicklungsumgebung für Mikrokerne entwickelt hat [tud].

Meine Arbeit nimmt die Ansätze von Kauer, Sadeghi, Stüble und Poritz auf, fügt sie zu einem System zusammen und setzt es anschließend auf einer Nizza-Architektur um. Neu sind eine Sprache für das Beschreiben von Eigenschaften, ihr Einsatz in RA und das Konzept der *sicheren Zonen*. Die *sicheren Zonen* sind ein Ansatz, das entfernte Attestieren auf die *Trusted Computing Base*[5] einer Anwendung zu beschränken.

[5] vgl. Abschnitt 2.4.

3. Enferntes Attestieren

In der Einleitung dieser Arbeit wurde das entfernte Attestieren (engl. *remote attestation* – RA) bereits kurz vorgestellt. In diesem Abschnitt sollen die Analysen vertieft werden. Als Ziel sollen die Anforderungen an das Protokoll von RA selbst und an das System an sich erarbeitet werden.

Dieser Abschnitt betrachtet das entfernte Attestieren zuerst auf einer abstrakten Ebene. Anschließend werden einige technische Details ausgearbeitet. Danach wird das entfernte Attestieren mit Hilfe des Ladeprotokolls analysiert und mit Hilfe des Models bewertet. Zuletzt folgen noch eine Betrachtung hinsichtlich des Datenschutzes der vorgestellten Techniken und ein kurzer Ausblick in weitere Formen für entferntes Attestieren.

3.1. Rollen

Bei der entfernten Attestierung sind zwei Rollen beteiligt: Der Herausforderer (engl. *challenger*) möchte einen Dienst in Anspruch nehmen, doch dafür zuerst prüfen, ob der Dienstleister vertrauenswürdig ist. Der Dienstleister händigt ein Attest aus, auf dem bescheinigt wird, dass er vertrauenswürdig ist. Der Herausforderer prüft dieses Attest und entscheidet, ob er den Dienst in Anspruch nimmt.

Um das entfernte Attestieren umsetzen zu können, muss es eine Möglichkeit geben, den Zustand der entfernten Maschine sicher zu ermitteln. Insbesondere darf der Dienstleister nicht in der Lage sein, einen falschen Systemzustand vorzutäuschen.

Die Lösung für dieses Problem ist das Einführen von Gutachtern, denen der Herausforderer vertraut und die den Dienstleister prüfen können. Auf der Seite des Anwenders wird der Gutachter fortan *Orakel* und auf der Seite des Dienstleisters das *sicheres Attestierungssystem* (SAS) genannt[1]. Der Dienstleister lässt das Attest durch das SAS erstellen bzw. beglaubigen. Der Herausforderer prüft es mit Hilfe des Orakels.

Das Attest wird immer im Zusammenhang mit einem Dienst angefordert. Dieser Dienst wird von einer Anwendung bereit gestellt (im Fallbeispiel wird die Rolle *Anwendung* durch das Programm `WahlAgent` besetzt). Diese Rolle wird auch *Proxy* genannt, da sie Herausforderer und SAS verbindet.

3.2. Sicherheit von RA

Die Sicherheit von RA gründet auf unterschiedlichen Faktoren, die in diesem Abschnitt untersucht werden sollen. Dabei wird auch das Modell sukzessiv verfeinert.

Während das Orakel als sicher vorausgesetzt wird, wird die Sicherheit des SAS durch das TPM und das Ladeprotokoll gewährleistet.

[1] Das SAS beinhaltet in einem System alle Anwendungen, die notwendig sind, um den Systemzustand sicher zu ermitteln und ein Attest sicher zu erstellen. Für gewöhnlich beinhaltet das SAS damit den Betriebssystem-Kern, die Ansteuerung des TPM-Geräts, die Protokollierung des System-Zustands, die Erstellung des Attests und die TCB der hier aufgezählten Anwendungen.

In dieser Arbeit sollen nicht die eingesetzten kryptographischen Algorithmen untersucht werden. Sicherlich ist festzuhalten, das RA höchstens solange sicher ist, solange die eingesetzten kryptographischen Algorithmen es sind. Hinzu kommt, dass die meisten dieser Algorithmen eine zuverlässige Zufallszahlen-Quelle benötigen. Auch diese wird als gegeben vorausgesetzt[2].

3.2.1. RA als Authentifizierung

Das entfernte Attestieren kommt einer Authentifizierung gleich. Die Authentifizierung erfolgt üblicherweise zu Beginn einer Sitzung und insbesondere nur einmal pro Sitzung. Das bedeutet, dass die Verbindung zwischen Herausforderer und Dienstleister dahingehend geschützt werden muss, dass nach erfolgter Authentifizierung die Sitzung nicht von einem Angreifer übernommen werden kann. Die Authentifizierung erfolgt durch das entfernte Attestieren. Solche gesicherten Sitzungen sind Aufgabe der Kryptographie und werden hier nicht weiter behandelt[3].

Die Authentifizierung und somit auch das entfernte Attestieren lassen sich wiederum in zwei Abschnitte gliedern. Zuerst wird ein Attest erstellt und anschließend, im zweiten Schritt, geprüft. Dabei muss das System sicherstellen, dass dem Herausforderer kein veraltetes, jedoch gültiges Attest übermittelt wird. Das folgende Beispiel soll Klarheit bringen.

Beispiel. Ein Angreifer fängt ein Attest ab und bewahrt es auf. Er übernimmt die Kontrolle über den Dienstleister und ersetzt die Anwendungen gegen eigene Programme. Das SAS würde diese Veränderung in einem neuen Attest berücksichtigen. Um dennoch erfolgreich zu sein, verwendet der Angreifer das SAS nicht und antwortet stattdessen mit dem abgefangenen, alten Attest. (engl. *replay*-Angriff, [Sch95]).

Um ein *frisches* Attest zu erzwingen, kann der Herausforderer fordern, dass eine selbst gewählte Zahl (engl. *nonce* oder *challenge*) im Attest mitberücksichtigt wird. Der Angreifer kann diese Zahl nicht in das abgefangene Attest einbauen, da das Attest durch eine Signatur des SAS geschützt ist und er diese Signatur nicht fälschen kann. Der Angreifer ist auch weiterhin auf das SAS angewiesen und somit wird der Angriff erkannt. Dieses Verfahren wird in der Kryptographie *challenge-response*-Verfahren genannt (engl. für Herausforderung-Antwort-Verfahren).

Die entfernte Attestierung erfolgt über eine Sitzung, die zu Beginn der Kommunikationsbeziehung erstellt wird. Darf der Herausforderer die Sitzungsnummer frei wählen, so kann diese im *challenge-response*-Verfahren eingesetzt werden.

3.2.2. Erstellen des Attests

Das Erstellen des Attests erfolgt durch das SAS. Was im SAS passiert, wird in Abbildung 3.1 verdeutlicht. Nachdem die Anwendung vom Herausforderer eine Anfrage auf ein Attest und eine Sitzung erhalten hat, wendet sie sich an das SAS mit der Bitte um ein Attest für die gegebene Sitzung.

[2] Das TPM-Gerät verfügt über einen Zufallszahlengenerator, der den kryptographischen Ansprüchen genügen sollte.

[3] TLS (engl. *transport layer security*) und seine Vorgänger SSL (engl. *secure soket layer*) sind Mechanismen für die sichere Ende-zu-Ende-Übertragung im Internet. Sie garantieren Authentizität und Vertraulichkeit. Wird eine Ende-zu-Ende-Verschlüsselung eingesetzt, so findet diese zwischen dem Herausforderer und dem SAS statt, womit der Proxy keine Einsicht in den Datenstrom besitzt. RA fordert lediglich den Schutz der Verbindung (Authentikation), setzt jedoch nicht zwangsläufig den Schutz der übermittelten Daten voraus (Konzelation).

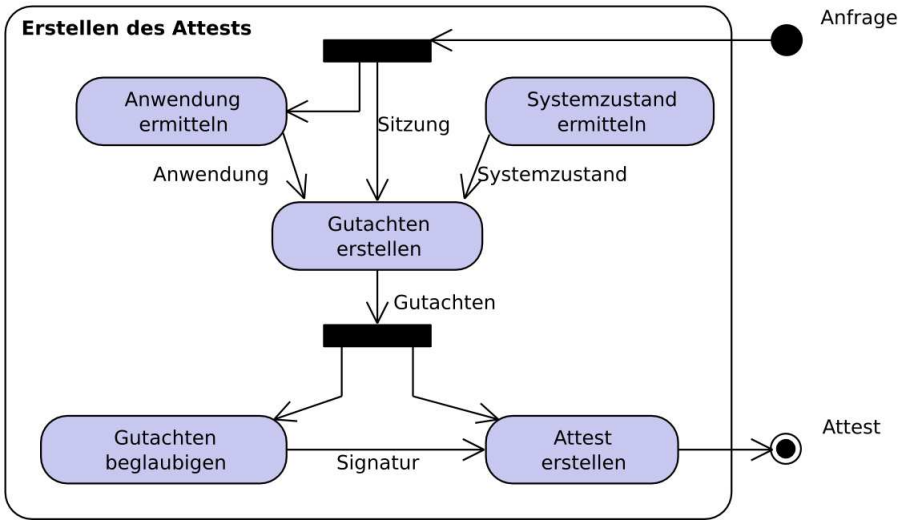

Abbildung 3.1.: Erstellen eines Attests. Das SAS kennt sowohl den Systemzustand als auch die Anwendung, die das Attest anfordert. Zusammen mit einer Sitzungsnummer werden diese Informationen zu einem Gutachten zusammengefasst, signiert und übermittelt.

Das SAS muss an erster Stelle prüfen, für wen es ein Attest ausstellt, denn diese Information könnte vor allem dann interessant sein, wenn sowohl sichere als auch unsichere Anwendungen auf einem System existieren dürfen, wie es bei Nizza-Architekturen der Fall ist. Zusätzlich kennt das SAS den Zustand des Systems[4]. Diese beiden Informationen fließen zusammen mit der Sitzung in ein Gutachten ein.

Im nächsten Schritt wird dieses Gutachten durch das SAS signiert und somit gegen Manipulationen geschützt. Die Signatur und das Gutachten ergeben zusammen das Attest:

$$\begin{aligned} \texttt{Attest} &:= (\texttt{Gutachten}, \texttt{Sign}_{SAS}(\texttt{Gutachten})) \\ \texttt{Gutachten} &:= (\text{Anwedungs-Identifikation}, \text{System-Zustand}, \text{Sitzung}) \end{aligned}$$

3.2.3. Prüfen des Attests

Das Prüfen dieses Attests ist wesentlich aufwendiger und wird in der Abbildung 3.2 dargestellt. Zuerst muss der Herausforderer prüfen, ob die Signatur im Attest zu dem Gutachten passt. Anschließend muss geprüft werden, ob die Sitzung korrekt angegeben ist. Der Herausforderer will zudem prüfen, ob das Attest auch für die erwartete Anwendung erstellt wurde.

Um zu ermitteln, ob ein System-Zustand z als sicher angesehen werden kann, muss der Anwender eine vertrauenswürdige Referenz besitzen. Sei diese Referenz durch Δ gegeben. Sei $\mathcal{I}$ eine Funktion, die einen System-Zustand interpretieren kann. Es gilt:

Definition 3.2.1: *Sicherer Systemzustand.* Ein System-Zustand z wird genau dann als sicher angesehen, wenn die Interpretation des übermittelten Zustands z in der Interpretation einer Referenz Δ enthalten ist:

$$z^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$

◇

[4]Der Zustand kann *sicher* oder *unsicher* sein. Es können aber auch Daten gemeint sein, die den realen Zustand ermitteln lassen, wie es das Ladeprotokoll erlaubt. In diesem Fall obliegt die Interpretation des Ladeprotokolls nicht dem SAS, sondern dem Herausforderer.

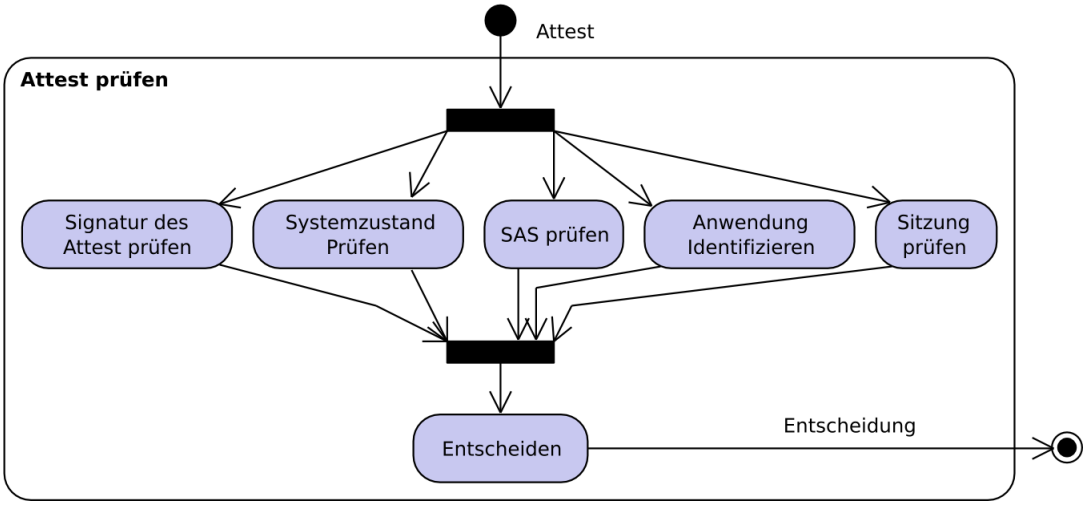

Abbildung 3.2.: Das Attest muss fünf Prüfungen bestehen; es muss eine gültige elektronische Signatur besitzen, die vom SAS ausgestellt wurde. Der Systemzustand muss als *sicher* interpretiert werden. Dies geschieht zum Beispiel durch die Auswertung des Ladeprotokolls. Dann muss das SAS gültig sein (z.B. Signierter Schlüssel EK durch MK). Verlaufen alle Prüfungen erfolgreich, so kann dem entfernten Rechner vertraut werden.

Wie diese Interpretation aussieht, ist den einzelnen Implementierungen überlassen. In diesem Kapitel wird das entfernte Attestieren mittels Ladeprotokoll vorgestellt. Das Resultat dieser Arbeit ist eine weitere Interpretierungsfunktion, die Eigenschaften auswerten kann. Die Referenz Δ wird durch das Orakel gestellt.

Das SAS muss ebenfalls geprüft werden. Wie dies geschieht, ist ebenfalls Aufgabe der konkreten Implementierung. Es müssen jedoch zwei Bedingungen gestellt werden:

Annahme 3.2.2: *Überprüfbar sicheres Attestierungssystem (SAS).* Das *sichere Attestierunssystem* (SAS) muss bedingungslos sicher oder nachweislich (überprüfbar) sicher sein. ◇

Annahme 3.2.3: *Authentischer System-Zustand.* Das SAS muss über einen authentischen System-Zustand verfügen. ◇

3.2.4. Zurückziehen der Sicherheitseigenschaft

Wird die Sicherheitsaussage einer Anwendung aufgrund einer Annahme getroffen, sollte entferntes Attestieren *revocation* erlauben (vgl. Abschnitt 2.5.6). Für RA muss die Referenz Δ immer dann angepasst werden, wenn neue Informationen dazu geführt haben, dass eine Anwendung nicht mehr vertrauenswürdig ist. Ähnliches gilt für die Menge aller SAS, denen der Herausforderer nicht mehr trauen darf.

3.3. Wer, wie, was?

3.3.1. Was soll attestiert werden?

Der klassische Ansatz von RA, nach der TCG, attestiert Prüfsummen. Die Prüfsummen stehen für einen ausgeführten Code. Im obigen Modell wird der Zustand attestiert. Die Sicherheitsaussage ensteht jedoch durch die Interpretierung (des *Wie* – Abschnitt 3.3.3).

Somit ist die Frage nach dem, *was* attestiert werden soll, von der Interpretation abhängig, obwohl eigentlich immer nur die Frage nach der Vertrauenswürdigkeit gestellt wird:

Definition 3.3.1: *Optimales entferntes Attestieren.* Das optimale entfernte Attestieren, bezüglich der Komplexität des Attests, sollte immer nur mit *sicher* oder *unsicher* antworten, nicht aber mit einem Ladeprotokoll oder sonstigen Informationen. ◇

Ziel ist, eine Aussage über die Vertrauenswürdigkeit zu ermitteln. Optimal bezüglich des Datenschutzes ist zudem die Aussage, dass nur genau soviel Informationen preisgegeben werden, wie unbedingt notwendig ist, um die Vertrauenswürdigkeit zu messen.

3.3.2. Wer soll attestiert werden?

Bisher beschränkten sich alle Beispiele darauf, nur ausgeführte Anwendungen zu messen. Konfigurationsdaten wurden nicht berücksichtigt. Dabei bilden besonders sie ein wichtiges Fundament für die Sicherheit eines Systems. Ein solches Beispiel ist die Wahlliste aus dem Fallbeispiel, oder Passwort-Datenbanken. Ein ähnliches Problem besteht mit dem Nachladen von Code[5] oder beim Einsatz von Bibliotheken.

Diese Arbeit untersucht nicht, wie verhindert werden kann, dass unerlaubter Programmtext nachgeladen und anschließend ausgeführt wird. Es sollen lediglich Methoden erarbeitet werden, die Konfigurationsdaten und dynamische Bibliotheken in den Attestierungsprozess mit einbinden können.

Darüber hinaus können die Beziehungen unter der Anwendung relevant sein. Daraus folgt, dass RA die TCB einer Anwendung und die Anwendung selbst messen soll. Wird das gesamte System gemessen, kann das insbesondere beim Einsatz von Nizza zu Widersprüchen führen, da Nizza explizit das Ausführen unsicherer Anwendungen erlaubt (dieser Punkt wird in Abschnitt 3.4.5.7 genauer betrachtet).

3.3.3. Wie soll attestiert werden?

Die Interpretierungsfunktion $\mathcal{I}$ ist bereits ein Teil der Antwort, wie RA stattfinden soll. In diesem Abschnitt sollen jedoch weitere Aspekte, inbesondere im Zusammenhang mit Nizza, betrachtet werden.

3.3.3.1. Verbindungswege

Herausforderer und Dienstleister sind über ein Verbindungsnetzwerk voneinander getrennt. Moderne Systeme stellen dem Programmierer eine Vielzahl solcher Netzwerke zur Verfügung: D-Bus, Java-RMI und viele mehr. L4 kennt im Wesentlichen nur die *Inter-Prozess-Kommunikation* (IPC) für die Kommunikation zwischen den Komponenten. Jedes weitere Protokoll muss auf IPC aufgebaut werden. Diese Vielfalt der Übertragungswege erzwingt ein Protokoll für RA, das möglichst universell einsetzbar ist. Idealerweise erlaubt ein Kommunikationskanal die Übertragung von unterschiedlichen Protokollen.

[5] Die Operation `dlopen` auf POSIX-kompatiblen Systemen erlaubt Programmierern das Nachladen von Bibliotheken. Diese Operation wird nicht durch das Betriebssystem ausgeführt, und somit könnte ein Programmierer auch Bibliotheken nachladen, deren Einsatz auf sicheren Plattformen nicht erlaubt ist. Ein weiteres Beispiel für das Nachladen von Code ist eingebetteter Schadcode durch Pufferüberläufe, der ebenfalls nicht gemessen wird.

3.3.3.2. Zusatzdienst

Ein wesentliches Merkmal von entferntem Attestieren ist, dass es sich um einen Zusatzdienst handelt.

Beispiel. Ein Dateisystem bietet den Zugriff auf Dateien an, die auf einem persistenten Speichermedium abgelegt sind. Darüber hinaus könnte es die Möglichkeit bieten, eine Attestierung zuzulassen.

Diese Tatsache hat einige Folgen für die Gestaltung der Schnittstellen der Dienste. Es bestehen drei Möglichkeiten, die Dienstleistung *entferntes Attestieren* in Anspruch zu nehmen:

Aufzählung 3.3.2:

1. Die Attestierung wird durch eine unabhängige Komponente, ohne Mitwirkung der attestierten Komponente durchgeführt.

2. Die Komponente bietet eine einzige Schnittstelle, über die sie sowohl ihre Dienstleistung als auch RA anbietet.

3. Die Komponente stellt mehrere Schnittstellen bereit. RA und die Dienstleistung sind über getrennte Schnittstellen verfügbar.

Der erste Ansatz besitzt den Vorteil, dass er sich in nahezu jedes System integrieren lässt, ohne die vorhandenen Komponenten anpassen zu müssen. Die Komponenten können jedoch keinen Einfluss auf die Attestierung ausüben, da sie nicht im Kontrollfluss mit eingebunden sind. Darüber hinaus benötigt der Herausforderer eine Methode, die Anwendung eindeutig zu identifizieren, um anschließend beim SAS ein Attest anzufordern. Vor allem, wenn der Herausforderer über ein externes Netzwerk mit dem System verbunden ist, kann sich dies als problematisch erweisen.

Bei der zweiten Variante ist das Problem der Identifizierung der Anwendung besser gelöst, da diese in Verbindung mit den SAS tritt und dafür eine lokale Absenderadresse verwendet (z.B. eine Prozessnummer). Doch die Veränderung der Schnittstellen ist nicht angebracht, da sich dadurch das Verhalten der Komponente verändert. Zusätzlich müssen alle abhängigen Anwendungen angepasst werden. Dieser Ansatz widerspricht dem Prinzip der komponentenbasierten Softwareentwicklung, weil es unterschiedliche Rollen in einer Schnittstelle vermengt.

Nach den Prinzipien der komponentenbasierten Softwareentwicklung ist die dritte Lösung die richtige: Die Komponente bietet zwei Schnittstellen, die getrennt voneinander genutzt werden können.

In dieser Arbeit wird eine Vereinigung des ersten und des dritten Ansatzes gewählt. Das SAS stellt durch den *trusted attestation manager* (TAM) eine zentrale Instanz zur Verfügung, die alle Atteste erstellt. Doch jede Anwendung kann als Mittler bzw. als Proxy auftreten und die entfernte Attestierung auf das jeweils eingesetzte Protokoll abbilden. Der TAM wird in Abschnitt 6.1.4 genauer erläutert.

3.4. Attestieren durch das Ladeprotokoll

Das Ladeprotokoll wurde bereits in Abschnitt 2.1.6 vorgestellt. In diesem Abschnitt soll das entfernte Attestieren mit Hilfe des Ladeprotokolls (BPRA, engl. für *boot protocol remote attestation*) untersucht werden. Es wird angenommen, dass das Ladeprotokoll fortlaufend

geführt und nicht zu einem bestimmten Zustand des Systems durch andere Protokolle ersetzt wird. Diese Betrachtung erfolgt zum Ende dieses Abschnitts, wo auch die Probleme des Ladeprotokolls erläutert werden.

3.4.1. Ablauf

Das entfernte Attestieren durch das Ladeprotokoll besteht aus zwei Phasen und einer einmaligen Initialisierungsphase.

In der Initialisierungsphase wird das System gestartet und das Ladeprotokoll erstellt. Das System kann schließend das entfernte Attestieren erlauben. In der ersten von zwei Phasen wird das Attest erstellt, und anschließend, in der zweiten Phase ausgewertet.

Algorithmus 3.4.1: *RA mit Ladeprotokoll, Phase I: Erstellen des Attests.*

1. Im ersten Schritt baut der Herausforderer eine sichere Verbindung zu der Anwendung auf und startet eine Sitzung, indem er eine zufällige Zahl $z \in H$ wählt. Zu dieser Sitzung fordert er ein Attest an.

2. Die Anwendung erhält die Anforderung. Sie kann die Anfrage abweisen oder sie an das SAS weiterleiten.

3. Das SAS erhält die Anfrage und fordert beim TPM ein Quote q an, was eine signierte Kopie (Signatur s) des aktuellen PCR-Registersatzes $\mathbf{r}$ zurückliefert:
$$q = (s, z, \mathbf{r}) = \texttt{Quote}(z, \mathbf{r}, EK)$$

4. Das SAS beschafft sich das aktuelle Ladeprotokoll $\mathbf{l}$. Das Ladeprotokoll und die Quote bilden das Attest a. Dem Attest wird zusätzlich der öffentliche Schlüssel des TPM-Geräts und die Signatur über diesen Schlüssel durch den Hersteller der Plattform beigefügt.
$$a = (\mathbf{l}, s, z, \mathbf{r}, EK, \texttt{Sign}_{MK}(EK))$$

5. Die Anwendung leitet dieses Attest als Antwort auf die Anfrage an das mobile Gerät weiter.

An dieser Stelle möchte ich nochmal auf die Sicherheit der TPM-Geräte (Annahme 2.1.1) und auf die Signatur $s_{EK}^{MK} = \texttt{Sign}_{MK}(EK)$ eingehen. Diese Signatur belegt die Mitgliedschaft von EK zu der Gruppe, die durch den Schlüssel MK erzeugt wird. Die Sicherheit leitet sich aus dem Vertrauen in die Instanz hinter den MK-Schlüssel ab. Die Signatur s_{EK}^{MK} stellt damit keine Messung der Sicherheit, sondern eine identifizierung dar.

Das Quote ist eine Signatur für das Ladeprotokoll (vgl. Abschnitt 2.1.6). Der Herausforderer erhält das Attest und prüft es durch folgende Schritte:

Algorithmus 3.4.2: *RA mit Ladeprotokoll, Phase II: Prüfen des Attests.*

6. Er berechnet den Soll-Zustand der PCR-Register anhand des Ladeprotokolls mit Hilfe der Funktion Γ (vgl. Seite 11). Das Resultat muss mit übermittelten PCR-Registersatz $\mathbf{r}$ überein stimmen:
$$\Gamma(\mathbf{l}) \stackrel{?}{=} \mathbf{r}$$

7. Die Signatur von *Quote* durch EK wird geprüft[6]:

$$\texttt{Verify}_{EK}(q)$$

8. Ob der mitgelieferte EK auch tatsächlich einem vertrauenswürdigen TPM-Gerät gehört, kann durch die Signatur $\texttt{Sign}_{MK}(EK)$ ermittelt werden. Das Orakel kennt MK^{pub}, was dem Herausforderer erlaubt EK wie folgt zu prüfen:

$$\texttt{Verify}_{MK}(\texttt{Sign}_{MK}(EK))$$

9. Zuletzt wird geprüft, ob jeder Eintrag $l = (d, i)$ im Ladeprotokoll einem sicheren Programm entspricht. Das Programm wird durch seine Prüfsumme d identifiziert. Das Orakel $\mathcal{O}$ besitzt eine Tabelle, die alle Prüfsummen vertrauenswürdiger Anwendungen speichert.

$$\forall l \in \mathbf{l}, d \text{ aus } l : \mathcal{O} \text{ kennt } d$$

Sind alle Überprüfungen erfolgreich gewesen, so kann dem entfernten System vertraut werden. Im realen Einsatz eines TPM-Geräts wird statt des EK ein zuvor erstelltes AIK für das Attestieren eingesetzt wird. Zugunsten der Einfachheit verzichte ich auf dieses Detail.

3.4.2. Die Interpretierungsfunktion unter BPRA

Die Interpretierungsfunktion $\mathcal{I}$ für das entfernte Attestieren mit dem Ladeprotokoll ist im Wesentlichen in der zweiten Phase, Schritt 9 erläutert. Dieser Schritt gründet auch auf folgender Definition:

Definition 3.4.3: *Sichere Komponente für BPRA.* Eine Komponente d wird genau dann als sicher angesehen, wenn das Orakel die Prüfsumme über den Programmtext von d kennt. ◇

3.4.3. Das Orakel unter BPRA

Das Orakel wird vollständig auf der Seite des Herausforderers umgesetzt. Der Herausforderer muss eine Liste von MK-Schlüssel und und eine Liste der sicheren Anwendungen besitzen. Die Liste der Anwendungen bildet die Referenz Δ.

3.4.4. Das SAS unter BPRA

Das sichere Attestierungs-System wird durch das Betriebssystem und das TPM-Gerät bereitgestellt. Das SAS muss den zwei Anforderungen 3.2.2 und 3.2.3 genügen. Diese beiden Anforderungen werden erfüllt, denn der sichere Systemzustand (Anforderung 3.2.3) wird durch das sicher geführte Ladeprotokoll gewährleistet.

Das SAS ist selbst im Ladeprotokoll enthalten. Damit muss nur gezeigt werden, dass das TPM-Gerät sicher ist. Das geschieht in Schritt 8 und über die Annahme, dass ein TPM-Gerät immer sicher ist (Annahme 2.1.1). Damit ist auch Anforderung 3.2.2 erfüllt.

[6] Dieser Schritt beweist auch, dass das TPM-Gerät tatsächlich den geheimen Teil von EK des mitgelieferten öffentlichen Schlüssels EK besitzt, denn wäre dem nicht so, könnte das TPM-Gerät die Signatur nicht erzeugen.

3.4.5. Probleme von BPRA

3.4.5.1. Aufwendiges Orakel

Die Referenz Δ wächst linear mit der Anzahl der erlaubten Anwendungen und besitzt die Komplexität $\mathcal{O}(n)$. Eine Anpassung muss immer dann erfolgen, wenn eine neue Anwendung hinzugefügt oder entfernt wird. Jede Komponente d besitzt eine gewisse Wahrscheinlichkeit a_d, dass sie *unsicher* ist. Die Wahrscheinlichkeit w, dass das Orakel eine *sichere* Referenz besitzt, ergibt sich durch folgende Gleichung:

$$w = \prod_{d} P(1 - a_d) \tag{3.1}$$

Für unendlich viele Komponenten d und für $a > 0$ ergibt sich $w = 0$. Damit ist die Referenz immer *unsicher*.

Für eine endliche Anzahl von d und einem beschränkten a erlaubt w eine Aussage zur Häufigkeit, in der die Referenz angepasst werden muss. Ist das einmal der Fall, so muss jeder Teilnehmer eine neue Kopie der Referenz erhalten.

3.4.5.2. Nachladen von Programmtext

Das Entfernte Attestieren kann nicht verhindern, dass Anwendungen dennoch erfolgreich angegriffen und schadhafter Code eingeschleust und ausgeführt wird. Das ist ein grundsätzliches Problem, das immer dann entsteht, wenn die Messung der Sicherheit nur zu einem Zeitpunkt erfolgt.

In Abschnitt 3.6.3 wird eine Lösung für dieses Problem vorgestellt, die jedoch nicht auf diese Arbeit angewendet werden kann.

3.4.5.3. Konfigurationsdaten

Konfigurationsdaten lassen sich nicht effizient in den Prozess für entferntes Attestieren einbinden. Die Referenz kennt nur Prüfsummen. Sie kann anhand dieser nicht unterscheiden, ob es sich um eine Konfigurationsdatei oder eine Anwendung handelt. Dabei hilft es nicht, das Ladeprotokoll um ein Feld zu erweitern, das den Typ des gemessenen Eintrags festschreibt, denn das System kann nicht ermitteln, ob es sich bei den Daten nur um eine unbekannte Konfigurationsdatei handelt, oder um Schadcode, der als unbekannte Konfigurationsdatei getarnt ist. Grundsätzlich muss jede sichere Konfigurationsdatei in der Referenz mit aufgeführt werden. Dadurch würde die Referenz sehr groß werden, mit den in Abschnitt 3.4.5.1 beschriebenen Folgen.

3.4.5.4. Beziehungen unter Programmen

Das einfache Ladeprotokoll (vgl. Gleichung 2.6) gibt nur Rückschlüsse über die gestarteten Anwendungen. Es enthält keine Informationen, ob das Programm noch läuft und mit wem es kommuniziert. Daraus folgt die Annahme, dass jedes Programm mit jedem weiteren kommuniziert. Aus diesem Grund müssen auch alle Schichten im System sicher sein, auch solche Schichten, die auf die gemessene Anwendung keinen Einfluss ausüben.

3.4.5.5. Keine partielle Auswertung des Ladeprotokolls

In Abschnitt 2.1.6 wurde bereits gezeigt, dass das Ladeprotokoll nicht partiell ausgewertet werden kann. Diese Tatsache führt dazu, dass das System immer seinen gesamten Zustand

veröffentlichen muss, obwohl meistens nur die Sicherheit einer einzigen Anwendung von Interesse ist.

Zu dieser Aussage besteht eine Ausnahme. Da das TPM über mehrere PCR-Register verfügt, kann über jedes Register ein separates Ladeprotokoll geführt werden. Welches PCR-Register verwendet wurde, lässt sich aus dem PCR-Index im Ladeprotokoll ermitteln. Auch die Operation `Quote` lässt sich über eine Maske so parametrisieren, dass nur bestimmte PCR-Register in der Ausgabe berücksichtigt werden. Durch diese Technik können Einträge im Ladeprotokoll, die zu Registern gehören, die in der Maske nicht aufgeführt werden, zurückgehalten werden.

Diese Technik kann dafür eingesetzt werden das Ladeprotokoll in einen System- und einen Anwendungs-Teil zu untergliedern. Sind alle Systemressourcen gestartet worden, werden fortan keine PCR-Register mehr geändert, die dem System-Ladeprotokoll zugeordnet sind. Die Zuordnung erfolgt über die Indices (z.B.: Die PCR-Register 0-7 sind für das System- und 8-15 für das Anwender-Ladeprotokoll reserviert).

Diese Technik erlaubt jedoch nicht, für jede Anwendung ein Ladeprotokoll zu erstellen, da die Anzahl der PCR-Register begrenzt ist und diese Register nicht zurückgesetzt werden können. PCR-Register, die zurückgesetzt werden können, bilden hier eine Ausnahme. Der Einsatz dieser Register ist nicht trivial. Die Probleme werden in Abschnitt 3.4.6 erläutert. Wichtig ist noch zu erläutern, dass das System-Ladeprotokoll unter keinen Umständen mit rücksetzbaren PCR-Registern geführt werden darf. In dieser Arbeit werden solche Register nicht betrachtet.

3.4.5.6. Das falsche „Was?“

Ein Anbieter von Medieninhalten interessiert sich weniger für die Frage, welche Anwendung in welcher Version eingesetzt wird. Vielmehr möchte sich der Anbieter nur vergewissern, dass seine Daten geschützt sind. Somit liegt nahe, dass eigentlich eine Eigenschaft der Systeme der Endanwender gemessen werden soll, nicht aber die Prüfsummen der Anwendungen. Dieser Punkt wird im nächsten Kapitel genauer betrachtet.

3.4.5.7. Alles-oder-Nichts-Sicherheit

Die Referenz Δ kennt nur sichere Anwendungen und nimmt keine weiteren Einstufungen durch. Sie kennt auch keine Abhängigkeiten unter den Einträgen. Daraus folgt, dass ein System nur dann sicher ist, wenn alle Anwendungen im System sicher sind (vgl. Korollar 2.1.5).

Härtig *et al.* hingegen begründen[7] die Nizza-Architektur u.a. damit, dass Sicherheit nicht zwangsläufig eine transitive Eigenschaft ist. D.h. die Sicherheit ist nur in Bezug auf die TCB einer Anwendung zu betrachten.

Beide Ansätze stehen im Widerspruch zueinander, denn Nizza erlaubt explizit das Ausführen von ungeprüften Anwendungen, also solche die im Sinne des sicheren Urladens *unsicher* sind und die nicht in der Referenz des Orakels aufgeführt werden dürfen. Im Grundsatz sind entferntes Attestieren durch das Ladeprotokoll und Nizza nicht vereinbar. Es sind zusätzliche Annahmen notwendig, um diese Ansätze zu vereinen. Diese werden in den nachfolgenden Abschnitten erläutert.

[7] Siehe [HHF+05], Abschnitt 2

	Schicht RAL_0	Schicht RAL_1	Schicht RAL_2
Modell:	EK und AIK digitale Signaturen	sicheres Urladen überprüfbares Urladen	virtuelles TPM
Sicherheit messen:	Signiertes EK Alg. 3.4.2, Schritt 8	Ladeprotokoll Prüfsummen	virtuelles EK, virt. Ladeprotokoll
Orakel:	MK, Δ	Prüfsummen, Δ	virt. MK, Δ
Schicht-0/Rolle:	TPM	SAS	virt. TPM
Sicherheit durch:	Annahme 2.1.1	RAL_0	RAL_1

Tabelle 3.1.: Aufbau einer mehrschichtigen Architektur für das (entfernte) Attestieren.

3.4.6. Mehr-Schichten-RA

Um den Problemen der entfernten Attestierung durch das Ladeprotokoll zu entgegnen, wurden bereits mehrere Ansätze verfolgt[Kau04, SS04]. Der wichtigste ist es, das Ladeprotokoll nur für die Absicherung des SAS anzuwenden. Das SAS setzt anschließend ein anderes Protokoll für RA ein.

Dadurch entstehen unterschiedliche Schichten im Protokoll für das entfernte Attestieren (vgl. Tabelle 3.1). Die unterste Schicht, die RAL_0 (für engl. *remote attestation layer*) wird durch das TPM gestellt. Anschließend folgt die Schicht, die durch das Ladeprotokoll abgesichert wird (RAL_1), und dann die Schicht, die das SAS bereit stellt, RAL_2. Jede Schicht besitzt sein eigenes Sicherheitmodell, sein eigenes Orakel, und auch seine Schicht-0 (vgl. Lemma 2.1.7).

Als Beispiel soll das SAS ein virtuelles TPM bereitstellen, dass für jede neu gestartete Anwendung neu erstellt wird. Dieses virtuelle TPM besitzt eigene Schlüssel und führt auch ein eigenes Ladeprotokoll, in dem im Wesentlichen nur die Anwendung selbst aufgeführt wird. Tabelle 3.1 verdeutlicht den Aufbau dieses Systems.

Theorem 2.1.8 hat bereits verdeutlicht, dass jede RAL eine Schicht-0 besitzt, die sich nicht selbst absichern kann. Stattdessen werden die Schicht-0 der höheren RAL-Schichten durch die darunterliegende RAL abgesichert. Dieses Prinzip setzt sich bis zur Absicherung des TPM-Geräts fort (Annahme 2.1.1).

Der Herausforderer muss nun auf drei Ebenen prüfen. Auf der höchsten Abstraktionsebene (RAL_2) prüft er das virtuelle Ladeprotokoll, dass das virtuelle TPM und das SAS bereitstellen. Dann prüft er mit Hilfe des Ladeprotokolls aus RAL_1, dass durch das TPM-Gerät (RAL_0) abgesichert ist, ob das SAS (und damit auch das virtuelle TPM) eine sichere Schicht-0 für RAL_2 darstellt. Die RAL_0 wird durch die Signatur von EK durch MK geprüft (vgl. Algorithmus 3.4.2, Schritt 8).

In den bisher vorgestellten Abläufen wurde nicht zwischen RAL_0 und RAL_1 unterschieden, obwohl sie bereits vorhanden waren. Die Referenz der RAL_0 wurde dabei durch die Liste der MK-Schlüssel und die Referenz der RAL_1 durch die Liste der gültigen Prüfsummen gebildet.

Dieser Ansatz birgt jedoch eine große Gefahr: Wie stellt das System sicher, dass die Anwendung, die vorgibt das RAL_2 umzusetzen, auch wirklich im Ladeprotokoll aus RAL_1 aufgeführt ist? Das folgende Beispiel verdeutlicht das Problem:

Beispiel 3.4.4:

1. Ein Angreifer erlangt die Kontrolle über eine unsichere Komponente.

2. Der Angreifer startet ein Programm d, dass fortan ein virtuelles TPM bereit stellt und unter seiner uneingeschränkten Kontrolle steht. d wird nicht im Ladeprotokoll verzeichnet.

3. Die Anwendung d erstellt ein beliebiges Attest und bittet die RAL_0 das Attest zu unterschreiben.

4. Das Attest samt Ladeprotokoll wird übermittelt. Das Quote passt zu dem Ladeprotokoll aus RAL_1 und die Signatur wurde von einer gültigen RAL_0 getätigt. Obwohl kryptographisch gesehen, das Attest gültig ist, spiegelt es nicht den realen Systemzustand wieder (vgl. Abbildung 3.1 und Annahme 3.2.3).

◇

An dieser Stelle wird der Konflikt zwischen dem Prinzip des sicheren Urladens und der Nizza-Architektur sehr deutlich. Nach dem Prinzip des sicheren Urladens wäre dieser Angriff kein wirklicher Angriff, weil die Anwendung aus dem ersten Schritt offensichtlich nicht sicher ist und somit auch keine sichere Schicht darstellen kann. Nizza sagt jedoch, dass solche Anwendungen durchaus berechtigt sind, und stellt Möglichkeiten zur Verfügung, die Auswirkungen eines Angriffs einzudämmen.

Nach der Nizza-Architektur ist der Angriff genau dann erfolgreich, wenn die Anwendung aus seiner unsicheren Umgebung heraus sichere Anwendungen beeinflussen kann. Dies ist in diesem Beispiel der Fall, da der Anwender unsichere Anwendungen über ein gefälschtes Attest als sicher ausgeben kann.

Das Problem liegt in der Absicherung der RAL_2. Es gibt hierfür unterschiedliche Lösungsansätze. Im ersten Ansatz wird jeder Anwendung außerhalb der SAS der Zugriff auf RAL_0 untersagt. Damit könnte d selbst kein Attest erstellen und müsste um die Mithilfe des SAS (RAL_1) bitten, wodurch der Angriff vereitelt wird. Damit wird die Operation `Quote` zu einem schutzwürdigen Systemaufruf deklariert.

Kann im Attest festgehalten werden, wer die Operation `Quote` ausgeführt hat, ist das Problem ebenfalls gelöst, denn dann würde bei der Auswertung sofort auffallen, dass d nicht im Ladeprotokoll verzeichnet ist. Diese Information ist jedoch nicht in der Ausgabe von der Operation `Quote` vorgesehen. Doch da die Operation `Quote` einen Schlüssel als Parameter erhält, könnte darüber indirekt der Urheber ermittelt werden: die Absicherung erfolgt über den Schlüssel. Dadurch wird nicht die Operation `Quote` zu einem Systemaufruf, sondern die Anwendung ganz bestimmter Schlüssel. Nur das SAS darf über asymmetrische Schlüssel verfügen, die für RA eingesetzt werden können. Es folgt:

Theorem 3.4.5: Das Erstellen eines Attest mit einer gültigen Unterschrift ist ausschließlich dem SAS vorbehalten. ◇

Diese schutzwürdige Schlüsselmenge besteht im einfachsten Fall aus dem EK-Schlüssel. Werden AIK-Schlüssel verwendet, so sind diese ebenfalls zu schützen.

Im vorherigen Abschnitt wurden zurücksetzbare PCR-Register erwähnt. Diese Register sind vergleichbar mit dem Einsatz einer virtuellen TPM. Sie besitzen auch den gleichen Nachteil wie der Einsatz einer RAL_2, denn das Programm, das die Kontrolle über dieses Register ausübt, muss in der RAL_1 enthalten sein und somit ist es Teil der SAS. In diesem Fall kann aber davon ausgegangen werden, dass die virtuellen TPM-Geräte in nahezu unbeschränkter Menge vorhanden und zudem wesentlich effizienter sind als der Einsatz dieser PCR-Register, woraus sich aus zurücksetzbaren PCR-Register keine Vorteile ergeben.

3.5. Folgen für den Datenschutz

Bisher wurden BPRA und die TNC im Zusammenhang mit der entfernten Attestierung vorgestellt. In diesem Abschnitt werden sie in Hinblick auf den Datenschutz analysiert.

3.5.1. Verkettbarkeit durch BPRA

Das Protokoll bietet an zwei Stellen eine Verkettung der Atteste: Durch Identifizierung der Schicht-0 und durch das Ladeprotokoll.

3.5.1.1. Indentifizierung durch das Ladeprotokoll

Im klassischen Ansatz des entfernten Attestierens durch das Ladeprotokoll wird jedes gestartete Programm im Ladeprotokoll verzeichnet. Ist ein Ladeprotokoll hinreichend lang, so sinkt die Wahrscheinlichkeit, dass zwei Anwender ein identisches Ladeprotokoll besitzen, gegen Null.

Jeder Eintrag im Ladeprotokoll besitzt eine gewisse Menge an Information, die einen Anwender identifiziert. Dieser Wert soll *Entropie* genannt werden, da sich ähnliche Analysen wie bei der Kryptologie anwenden lassen. Ist genug Entropie gesammelt worden, so steigt die Wahrscheinlichkeit, einen Anwender zu identifizieren, stark an: Ist n die Anzahl der Anwender, so sind $\lceil \log_2 n \rceil$ Bit Entropie notwendig, um sie alle voneinander zu unterscheiden[SW63]. Bei einer Weltbevölkerung von ca. 6 Milliarden Menschen entspricht dies 33 Bit. Diese Zahl kann durch weitere Verkettung noch eingeschränkt werden. Beispiele dafür sind angeforderte Sprache, geographischer Standort, der Besitz eines Computers oder die IP-Adresse.

Die Entropie, die in einem Eintrag im Ladeprotokoll steckt, ist zudem nicht konstant und wahrscheinlich schwer zu messen. Doch es gibt Faktoren, die nahelegen, dass zum Teil sehr viel Entropie in einem einzigen Eintrag stecken kann. Theoretisch betrachtet kann jeder Eintrag im Ladeprotokoll $\lceil \log_2(|H \times \mathbb{N}|) \rceil$ Bits enthalten (vgl. Gleichung (2.6) auf Seite 10). Wird anstatt $\mathbb{N}$ eine endliche Menge, beispielsweise ein `integer`-Feld mit 32 Bit und für H die Ausgabe der SHA1-Operation gewählt (160 Bit), so enthält jeder Eintrag im Ladeprotokoll bis zu 192 Bit Entropie. Zudem enthält die Reihenfolge der Einträge, und sogar Wiederholungen, die auftreten, wenn ein Programm mehrmals gestartet wird, ebenfalls Entropie.

Dagegenzuhalten ist die Tatsache, dass viele Einträge im Ladeprotokoll eines Anwenders mit den Einträgen des Ladeprotokolls anderer Anwender identisch sind. Dies ergibt sich beispielsweise durch den Einsatz desselben Betriebssystems. Doch je länger das Ladeprotokoll ist, desto mehr Entropie kann es beinhalten. Aus diesem Grund sollte es nur für das Absichern der Schicht-0 eingesetzt werden.

Nicht nur die ausgeführten Anwendungen auf dem System beeinflussen das Ladeprotokoll. Während des Startprozesses des Rechners prüft das TPM-Gerät nicht nur sich selbst, sondern auch das BIOS. Das BIOS prüft die sogenannten *opt-ins* oder *expansion ROM*, kleine Programme, die von den Peripherie-Geräten (PCI-Karten, usw.) in den Ladeprozess eingebunden werden. Diese unterscheiden sich je nach Gerät und Version. Je nach Bestückung des Rechners könnte auch die Reihenfolge, in denen diese *expansion ROM* ausgewertet werden, unterschiedlich sein.

Die Vielfältigkeit und die Kurzlebigkeit des PC-Markts, und der schnelle Entwicklungszyklus von Software tragen dazu bei, dass die Wahrscheinlichkeit sehr gering ausfallen

dürfte, dass zwei Rechnersysteme, die an unterschiedlichen Standorten erworben werden, exakt die gleiche Konfiguration aufweisen.

Den Entropiegehalt der gemessenen Schichten genauer zu untersuchen und zu beziffern, und somit die obige Vermutung zu untermauern, ist nicht Gegenstand dieser Arbeit.

3.5.1.2. Identifizierung durch die Schicht-0

Unabhängig vom gewähltem Modell kann über die Schicht-0 immer ein System identifiziert werden. Bei dem TPM geschieht dies über den mitgelieferten öffentlichen Schlüssel EK des TPM (vgl. Schritt 4, Algorithmus 3.4.1).

Wird ein mehrschichtiges Modell eingesetzt, so lässt sich das System über jede Schicht-0 im System identifizieren, denn jede Schicht besitzt einen solchen Schlüssel oder wendet den Schlüssel einer darunterliegenden Schicht-0 an[8].

3.5.2. Anonymität durch vertraute dritte Instanz

Um eine uneingeschränkte Anonymität gegenüber dem Herausforderer zu gewährleisten, ist eine dritte Instanz notwendig, die sowohl das Ladeprotokoll als auch die Identität verschleiert.

Das Protokoll könnte folgendermaßen aussehen: Eine dritte vertrauenswürdige Instanz (engl. *trusted third party*, TTP) kann über einen gleichnamigen Schlüssel identifiziert werden. Sie bildet eine Gruppe, der jeder Dienstleister beitreten kann. Das Schema ist den Gruppensignaturen entlehnt.

Algorithmus 3.5.1: *Anonymes RA.*

1. Die TTP erstellt eine Sitzung für das entfernte Attestieren
2. Der Dienstleister erstellt eine Menge von Schlüsseln, die er für das Attestieren einsetzen möchte. Ein solcher Schlüssel wird AIK (engl. für *attestation identity key*) genannt.
3. Der Dienstleister signiert die AIK mit dem EK[9] und übermittelt der TTP die jeweiligen öffentlichen Anteile aller AIK, das Ladeprotokoll und ein aktuelles Quote.
4. Die TTP prüft RAL_0 und RAL_1 des Dienstleisters durch das übermittelte Ladeprotokoll. Anschließend signiert die TTP die AIK und übermittelt diese Signatur an den Dienstleister.
5. Der Dienstleister verwendet jede AIK für höchstens ein Attest. Er signiert ein Attest mit einem AIK und sendet diese Signatur, das Attest und die Signatur des AIK durch das TTP an den Herausforderer.
6. Der Herausforderer prüft die Signatur des AIK durch die TTP. Ist diese gültig, schreitet es analog zu Protokoll 3.4.2 fort, wobei Schritt 8 entfällt und statt EK der AIK-Schlüssel eingesetzt wird.

[8] *Beispiel.* Für das Attestieren durch das Ladeprotokoll wendet die RAL_1 durch Anwendung der Operation `Quote` den Schlüssel der RAL_0 an.

[9] Damit wird sichergestellt, dass die AIK vom SAS verwaltet werden (vgl. Theorem 3.4.5).

Der Einsatz von *direct anonymous attestation* (DAA, Abschnitt 2.5.5) bewirkt zugleich, dass eine AIK mehrmals eingesetzt werden kann, ohne dass eine Verkettung stattfindet. Damit beschränkt sich die Anzahl der notwendigen AIK auf eins[10]. Wird die Operation `Seal` (Abschnitt 2.1.3.3) eingesetzt, kann das Beitreten in die Gruppe (Schritte 1 bis einschließlich 4) zeitlich entkoppelt von der Anwendung der AIK geschehen (Schritte 5 und 6). In diesem Fall ist der AIK-Schlüssel an den PCR-Registerzustand zu binden, der auch der TTP im Schritt 3 vorlag. Damit ist keine Verbindung zur TTP zur Laufzeit notwendig.

Die Anonymität ist selbstverständlich nur solange gewährleistet, solange die anschließend übertragenen Daten und das Übertragungsmedium ebenfalls Anonymität erlauben. Werden auf RAL_2 ebenfalls Prüfsummen und das Ladeprotokoll eingesetzt, so bestehen die gleichen Probleme wie in der darunterliegenden Schicht.

3.6. Weitere Formen für RA

Entferntes Attestieren ist an das überprüfbare Urladen gekoppelt. Das überprüfbare Urladen ist wiederum auf das Prinzip der sicheren Schichten (Definition 2.1.4) des sicheren Urladens angewiesen. Arbaugh *et al.* fordern eine sichere Plattform, bzw. eine sichere Schicht-0 (Lemma 2.1.7) – sollen Alternativen zu dem entfernten Attestieren gefunden werden, muss die Untersuchung unterschiedliche Aspekte betrachten.

Eine Alternative zum sicheren Urladen (und somit zu Lemma 2.1.7) kann ich nicht erkennen. Zwar könnten sogenannte *System-on-Chips* (SOC) eingesetzt werden, doch diese unterscheiden sich nicht wesentlich von dem Einsatz eines TPM-Geräts. SOC sind hochintegrierte Schaltkreise, in denen auch die Funktionalität, die sonst in Software ausgeführt wird, in den Chip eingearbeitet wurde. In diesem Fall muss die Sicherheit durch *Anti-Tamper*-Methoden gewährleistet werden, wie es auch bei TPM-Geräten der Fall ist. Der gesamte Chip bildet eine Schicht-0, die Lemma 2.1.7 genügen muss, und die letztendlich auf die Annahme 2.1.1 zurückgeführt wird. Theoretisch betrachtet sind beide Ansätze, also der Einsatz eines SOC, oder der Einsatz eines TPM-Geräts, identisch.

Es bestehen jedoch Alternativen, wie die Interpretierungsfunktion $\mathcal{I}$ und die Referenz Δ gestaltet werden können. Neben dem Attestieren von Eigenschaften, das in den nachfolgenden Kapiteln sehr ausführlich erarbeitet wird, sind noch folgende ergänzende und gegensätzliche Ansätze möglich.

3.6.1. RA durch sicheres Urladen und Sealing

Wird sicheres Urladen und die Operation `Seal` angewendet, so dann kann ein Schlüssel an einen bestimmten Zustand des Systems gekoppelt werden. Über diesen Schlüssel wird ein kryptographisch gesicherter Kommunikationskanal zu einem entfernten System aufgebaut. Die Attestierung wird durch diesen Kommunikationsaufbau nachgewiesen.

Die Interpretierungsfunktion $\mathcal{I}$ würde lediglich auf den Kommunikationskanal hin prüfen. Die Referenz würde sich auf die Werte $\Delta := \{\text{wahr}, \text{falsch}\}$ beschränken[11].

Dieses Verfahren lässt sich nicht für große Netzwerke einsetzen, da die Menge des Schlüsselmaterials die Komplexität $\mathcal{O}(n^2)$ besitzt: Für n Teilnehmer in einem vollvermaschten Netz müssen $n * (n-1) * \frac{1}{2}$ Schlüssel für eben so viele Verbindungen

[10] Diese Eigenschaft wird durch das *zero-knowledge-proof*-Verfahren gewährleistet. Vgl. Abschnitt 2.5.5.

[11] Analog zu Definition 3.2.1 lautet das Beispiel für einen Systemzustand z, indem der Kommunikationskanal vorhanden ist: $z^{\mathcal{I}} = \{\text{wahr}\}$, $\Delta^{\mathcal{I}} = \{\text{wahr}\}$ und $z^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.

vorhanden sein. Die Frage nach der Anonymität ist hinfällig, da über den Schlüssel immer der Kommunikationspartner eindeutig bestimmt ist.

Der Vorteil dieses Ansatzes ist jedoch, dass die Attestierung ausschließlich auf der Seite des Dienstleisters erfolgt und damit der Systemzustand nicht übertragen werden muss.

Unter bestimmten Voraussetzungen bietet sich diese Architektur an. Das unter Abschnitt 3.5.2 vorgestellte Protokoll für anonymes RA ähnelt dem hier vorgestellten Ansatz in dem Punkt, das die AIK an den Systemzustand gebunden werden können, den das TTP geprüft hat. Der Einsatz der asymmetrischen Kryptographie stellt wieder eine linear wachsende Menge des Schlüsselmaterials her. Die RAL_1 wird durch das sichere Urladen der Operation `Seal` abgesichert.

3.6.2. Terra

Garfinkel *et al.* [GPC+03] schlagen eine Architektur vor, in der eine sichere Plattform durch einen *trusted virtual machine monitor* (TVMM) auf unterschiedliche virtuelle Maschinen (engl. *virtual machine* – VM) aufgeteilt werden. Im Grunde entspricht das TVMM dem Mikrokern und die VM den Legacy-Partitionen (vgl. Abschnitt 2.3). Garfinkel *et al.* unterscheiden zwischen sogenannten *offenen* und *geschlossenen* Boxen (engl. *open* und *closed-boxes*). Die offenen Boxen sollen die Vielfältigkeit moderner PC-Systeme wiederspiegeln, wobei die geschlossenen Boxen für sichere Anwendungen vorgesehen sind.

Unter bestimmten Einschränkungen kann behauptet werden, dass der Einsatz eines TPM-Geräts und der Nizza-Architektur eine Terra-Architektur bildet. Doch bei Terra erfolgt die Trennung der Partitionen auf der Ebene von unterschiedlichen virtuellen Maschinen. Nizza vollzieht diese Trennung auf der Ebene der Anwendungen und durch den Einsatz der *trusted wrappers*.

3.6.3. Symmetrische Verhaltens-Basierte Vertrauenswürdigkeit

Haldar und Franz [HF04] greifen ein grundsätzliches Problem des sicheren Urladens auf: Es prüft lediglich den Programmtext zur Startzeit. Weitere Untersuchungen zur Laufzeit werden nicht durchgeführt. Haldar und Franz setzen den Fokus, das Verhalten der Anwendung zur Laufzeit kontinuierlich zu messen, beispielsweise durch den Einsatz von Invarianten. Am Beispiel von *Terra* könnte der TVMM um die Funktionalität erweitert werden, die Invarianten in regelmäßigen Abständen zu prüfen.

Dieser Ansatz wird in dieser Arbeit nicht weiter verfolgt.

4. Eigenschaften

4.1. Motivation

Die Motivation für den Einsatz von Eigenschaften im entfernten Attestieren gründet im Wesentlichen auf zwei Aspekte. Eigenschaften erlauben die Attestierung flexibler zu gestalten. Zudem wird der schlichte Programmcode mit zusätzlichen Informationen versehen, die automatisiert ausgewertet werden.

Abschnitt 3.4.5 hat die Nachteile hinsichtlich des Ladeprotokolls verdeutlicht. Auch in Bezug auf den Datenschutz weist das Ladeprotokoll große Nachteile auf (vgl. 3.5). In beiden Fällen gründet das Problem in der Anwendung der Prüfsummen und des Ladeprotokolls für das Attestieren.

Der Einsatz von Eigenschaften ermöglicht eine Indirektionsstufe zwischen den Prüfsummen der Anwendungen und der Sicherheitsaussage einzubauen. Dafür wird eine Eigenschaft e mit einem Programm d verknüpft. Anstatt die Prüfsumme über d für das Attestieren zu verwenden, wird e genommen. Während d sich nun verändern kann, bleibt e konstant. Auch der Inhalt und die Interpretation von e können nun frei gewählt werden; e kann komplexe Sachverhalte beschreiben. Im Gegensatz dazu besitzt die Prüfsumme an sich keine weitere Bedeutung und ihre Interpretation beschränkt sich auf den Test auf Gleichheit.

Konkret könnte das bedeuten, dass eine Softwarekomponente d die Eigenschaft besitzt, als *sicher* eingestuft zu sein (Eigenschaft e). Das Programm d kann im Verlauf der Jahre weiterentwickelt werden und jede neue Version d' von d besitzt ebenfalls die Eigenschaft e. Ein System kann somit alle Varianten von d einsetzen und gilt immer noch als sicher. Würde statt e, die Prüfsumme $\mathcal{H}(d)$ von d der Prüfung unterliegen, so müsste ein älteres System das neuere Programm d' als unsicher einstufen, da es d' und folglich $\mathcal{H}(d')$ nicht kennt.

Da Eigenschaften weitaus mehr Informationen abbilden können als Prüfsummen, ist ein weiterer großer Vorteil das Berücksichtigen der Abhängigkeiten zwischen den Programmen (vgl. TCB aus Abschnitt 2.4).

Dieser Abschnitt beschränkt sich darauf, eine Syntax für das Beschreiben von Eigenschaften vorzustellen. Aufgrund dieses Modells wird im anschließendem Kapitel das entfernte Attestieren von Eigenschaften vorgestellt.

4.2. Modell der Eigenschaften

> **Eigenschaft.** Eine Eigenschaft bezeichnet ein Merkmal, eine Funktion oder eine Qualität, die einer Klasse von Objekten, Prozessen, Relationen oder Ereignissen gemeinsam ist und sie von anderen unterscheidet (nach [wik]).

Aus dieser Definition lässt sich bereits ableiten, dass der Begriff *Eigenschaft* sehr umfassend ist und deshalb im Verlauf dieser Arbeit eingeschränkt werden muss. Für den weiteren Verlauf werde ich mich zunächst auf Eigenschaften von Objekten beschränken. Objekte können ausführbare Programme sein, die auf dem System gestartet werden.

Solche Objekte vereinen oftmals mehrere Eigenschaften zu einer Menge. Diese Menge wiederum bildet eine Beschreibung des Objekts:

Definition 4.2.1: *Objektbeschreibung.* Sei $\mathcal{E}_i$ die Menge von Eigenschaften, die einem Objekt d_i zugeordnet ist. Sei e eine Eigenschaft von d_i und somit $e \in \mathcal{E}_i$. e ist innerhalb $\mathcal{E}_i$ eindeutig definiert. $\mathcal{E}_i$ heißt *Objektbeschreibung* von d_i. ◇

Beispiel. Das Programm `foobar` ist in der Version 1.51 verfügbar. Der Name des Programms bildet selbst eine Eigenschaft. Die Menge aller Eigenschaften von `foobar` könnte somit folgendermaßen lauten: $\mathcal{E}_{\texttt{foobar}} = \{\text{foobar}, \text{Version 1.51}\}$.

Die Objektbeschreibung kann dafür verwendet werden, ein Programm eindeutig zu identifizieren. Die Identität eines Programms ergibt sich aus:

Definition 4.2.2: *Identität.* Die Identität $\mathcal{J}$ eines Programms d ist eine Teilmenge $\mathcal{J} \subseteq \mathcal{E}$, anhand der das Objekt d eindeutig von allen anderen Objekten unterschieden werden kann. ◇

Die Versionsnummer lässt sich dafür jedoch nicht verwenden, da es denkbar ist, dass mehrere Programme aktuell in der Version 1.51 vorliegen. Doch unter bestimmten Voraussetzungen könnte der Name des Programms dafür verwendet werden. Somit würde gelten: $\mathcal{J}_{\texttt{foobar}} = \{\text{foobar}\}$.

Die Identitätsmenge kann gegebenenfalls sehr klein sein, beispielsweise wenn Personen-Identifikationsnummern oder kollisionsresistente Prüfsummen eingesetzt werden. Zudem ist nicht ausgeschlossen, dass ein Objekt mehrere Identitäten besitzt.

Idealerweise setzt sich $\mathcal{E}$ aus Eigenschaften zusammen, die sowohl die Identität eines Objekts ausmachen, als auch das Objekt einer Klasse von Objekten zuordnen. Besitzt ein Objekt d_i mehrere Eigenschaften, so lassen sich Objekte bezüglich einer Eigenschaft $e \in \mathcal{E}$ vergleichen. Daraus folgt unmittelbar:

Korollar 4.2.3: *Gleichheit.* Die festgestellte Gleichheit bezüglich einer Eigenschaft e zweier Objekte d_i und d_j, mit $e \in \mathcal{E}_i \cap \mathcal{E}_j$ sagt nichts über die Gleichheit anderer Eigenschaften der Objekte aus. ◇

Korollar 4.2.3 gilt analog für den Vergleich von Mengen. Deshalb sind zwei Objekte d_i und d_j noch nicht identisch, wenn beide je eine Teilmenge besitzen, die gleich sind.

Korollar 4.2.4: Seien d_i und d_j identische Objekte. Beschreibt $\mathcal{E}^A$ eine bestimmte Teilmenge von $\mathcal{E}$. Damit gilt

$$d_i = d_j \Longrightarrow \mathcal{E}_i^A = \mathcal{E}_j^A$$

◇

Die Umkehrung gilt nicht. D.h. zwei Anwendungen, die die gleichen Eigenschaften besitzen müssen nicht identisch sein.

$\cdot^A$ Beschreibt eine Bildungsvorschrift, wie die Teilmenge $\mathcal{E}^A$ aus $\mathcal{E}$ zu bilden ist. A könnte lauten *alle Eigenschaften, die die Version eines Programms ausdrücken.* Dem Beispiel zufolge wäre $\mathcal{E}^A_{\texttt{foobar}}$ mit $\mathcal{E}^A_{\texttt{foobar}} = \{\text{Version 1.51}\}$ gegeben. Die Bildungsvorschrift A wird auch Objektklasse genannt.

Definition 4.2.5: *Objektklasse.* Sei $\mathcal{E}$ eine Objektbeschreibung und $\mathcal{E}^A$ eine Teilmenge von $\mathcal{E}$. A ist eine Bildungsvorschrift für die Teilmenge $\mathcal{E}^A$. A wird eine Objektklasse genannt. ◇

4.2.1. Rollen, Abstraktionen und Klassen

Bestimmte Eigenschaften lassen sich am besten ausdrücken, wenn sie auf mehrere Eigenschaften aufgeteilt werden. So ist eine Versionsnummer meist in der Form $x.y$ gegeben, mit $x, y \in \mathbb{N}$ (z.B. Version 1.51). x und y ergeben getrennt keinen Sinn. Dennoch ist es sehr häufig vorteilhaft, x und y getrennt zu speichern.

Solche korrelierenden Eigenschaften werden zu einer Objektklasse zusammengefasst. Doch es gibt einen weiteren Punkt, Eigenschaften in einer Objektklasse zusammenzufassen: Das Ausdrücken von *Rollen*.

Rollen können mit Schnittstellen oder der Mehrfachvererbung verglichen werden. Eine Rolle A kann dabei alle notwendigen Eigenschaften vereinen, die für die Ausübung der Rolle A relevant sind. *Beispiel.* Der Namensdienst bietet eine Namesndatenbank an, der Wahlagent die Möglichkeit eine Stimme abzugeben. Beide Anwendungen sind Programme mit einem Namen und einer Prozessnummer. *Namesdienst*, *Wahlagent* und *Anwendung* sind unterschiedliche Rollen im System.

Zwei unabhängige Rollen A und B (mit $A \cap B = \emptyset$) können zu einer neuen Rolle C vereint werden, was durch die Vereinigung ausgedrückt werden soll: $C = A \cup B$.

Eine Rolle kann zudem in unterschiedlichen Abstraktionsniveaus vorhanden sein. Solche Konstrukte sind mit der einfachen Vererbung objektorientierter Programmiersprachen zu vergleichen. Bei der Abstraktion gibt es zu einem A ein B mit $B \subset A$. Daraus folgt

$$B \subset A \Longrightarrow \mathcal{E}^B \subseteq \mathcal{E}^A \tag{4.1}$$

A drückt eine *Spezialisierung* von B aus.

Beispiel. Die Dateisysteme *ext3* und *xfs* sind zwei unterschiedliche Dateisysteme für Unix-Betriebssysteme mit unterschiedlichen Eigenschaften. Beide gehören jedoch der Klasse *Dateisystem* an. Es gilt

$$\mathcal{E}^{\texttt{Dateisysteme}} \subseteq \mathcal{E}^{\texttt{ext3}} \cap \mathcal{E}^{\texttt{xfs}}$$

Setzt ein Programm ein Dateisystem voraus, drückt es dies durch die Forderung aus, dass es ein weiteres Programm benötigt, dass von der Objektklasse *Dateisystem* ist. Damit bieten sich sowohl *ext3* als auch *xfs* an.

4.2.2. Wesentliche und schwache Eigenschaften

Eigenschaften können nicht nur Objekten zugeordnet werden, sondern auch Handlungen oder den Relationen zwischen den Objekten. Bestimmte Merkmale entstehen erst durch eine Kooperation der Objekte. Andere wiederum gehen verloren. Daraus folgt:

Lemma 4.2.6: *Wechselwirkung von Eigenschaften.* Objekte können anhand von Wechselwirkungen mit anderen Objekten Eigenschaften verlieren oder hinzu bekommen. ◇

Definition 4.2.7: *Wesentliche Eigenschaft.* Eine Eigenschaft eines Objekts d, die unabhängig von Zeit oder von der Wechselwirkung dieses Objekts mit weiteren Objekten ist, wird eine *wesentliche Eigenschaft* von d genannt. ◇

Definition 4.2.8: *Schwache Eigenschaften.* Alle nicht wesentlichen Eigenschaften werden *schwache* Eigenschaften genannt. ◇

Im Gegensatz zu den *wesentlichen* Eigenschaften sind schwache Eigenschaften nur aufwendig zu realisieren, denn sie sind mit Nebenbedingungen verknüpft, die zumeist zur Laufzeit ausgewertet werden müssen. Doch ausgerechnet die Sicherheit eines Systems ist eine schwache Eigenschaft, da sie in Wechselwirkung zu allen gestarteten Anwendungen

steht[1]. Die Versionsnummer einer Anwendung hingegen ist ein Beispiel für eine wesentliche Eigenschaft.

4.3. Technik

4.3.1. Eigenschaft

Eine Eigenschaft ist folgendermaßen definiert:

Definition 4.3.1: *Eigenschaft.* Eine Eigenschaft ist ein n-Tupel mit folgenden Werten:

1. Einem Namen, der innerhalb der Objektbeschreibung eindeutig ist,
2. einem Wertebereich oder Typ, der den Wert einschränkt, den die Eigenschaft speichern kann,
3. einer Kardinalität, und
4. zusätzlichen Modifikatoren (konstant, vorausgesetzt usw.).

◇

Der Name dient der einfachen Referenzierung einer Eigenschaft innerhalb einer Objektbeschreibung. Die Modifikatoren, die Kardinalität und der Wertebereich der Eigenschaft sind nützliche Hilfskonstruktionen, die das Bilden von Standards vereinfachen. Mit Hilfe dieser Informationen lässt sich das gesammelte Wissen, also die Menge aller Objektbeschreibungen, automatisiert auf Konsistenz prüfen.

Die Syntax der Eigenschaften ist in Anhang A gegeben.

4.3.2. Referenz

Eine Referenz ist eine besondere Art der Eigenschaft, deren Wertebereich die Menge aller Objektbeschreibungen im System ist. Sie kann somit auf eine andere Objektbeschreibung verweisen. Seien d_i und d_j zwei Objekte im System, und $\mathcal{E}_i$ und $\mathcal{E}_j$ die zugehörigen Objektbeschreibungen. Die Referenz $r = \vec{\mathcal{E}}_j$ ist eine Eigenschaft aus $\mathcal{E}_i$, die auf die Menge $\mathcal{E}_j$ verweist. $\vec{}$ soll darauf deuten, dass eine Referenz auf die Menge $\mathcal{E}_j$ gemeint ist.

Die Objekte, auf die die Referenz verweisen kann, können selbst durch den Typ der Referenz eingeschränkt werden. Die Notation $r\langle A\rangle$ deutet darauf hin, dass die Referenz r nur Verweise beinhalten darf, die selbst von der Objektklasse A sind: $r\langle A\rangle = \vec{\mathcal{E}}_j$ und $\exists \mathcal{E}_j^A$ mit $\mathcal{E}_j^A \subseteq \mathcal{E}_j$. Auch diese Einschränkung dient, wie auch die Kardinalität, zur rechnergestützten Prüfung des Modells.

Folglich ist eine Referenz folgendermaßen definiert:

Definition 4.3.2: *Referenz.* Eine Referenz ist eine Eigenschaft mit:

1. einem Namen, der innerhalb der Objektbeschreibung eindeutig ist,
2. einer Objektklasse, die den Wertebereich definiert,
3. einer Kardinalität,
4. zusätzlichen Modifikatoren.

◇

[1] Die Tatsache, dass die Sicherheit eine schwache Eigenschaft ist, wird in Theorem 5.2.1 genauer begründet.

4.3.3. Zonen

Durch den Einsatz von Referenzen können die Objekte miteinander verknüpft werden. Diese Verknüpfungen bilden eine transitive Hülle, die *Zone* genannt wird.

Definition 4.3.3: *Zone.* Die Zone $\mathcal{Z}_i$ eines Objekts d_i ist eine Vereinigung der

1. Objektbeschreibungen $\mathcal{E}_i$
2. und der Zonen $\mathcal{Z}_j$ der Objekte d_j, die mit d_i in Verbindung stehen.

Ein Objekt d_j steht mit d_i in Verbindung, wenn es eine Referenz in $\mathcal{E}_i$ gibt, die auf $\mathcal{E}_j$ verweist: $\exists r \in \mathcal{E}_i$ mit $r = \vec{\mathcal{E}_j}$. ◇

4.3.4. Farbige Zonen und Referenzen

Eine Spezialisierung der Referenzen und folglich auch der Zonen, sind die farbigen Referenzen und Zonen. Sei F eine Menge von Farben.

Definition 4.3.4: *Farbige Referenz.* Eine farbige Referenz ist eine Referenz, die zusätzlich um eine Farbe ergänzt wird. ◇

Die Farben spielen insbesondere bei der Bildung der Zonen eine wesentliche Rolle.

Definition 4.3.5: *Farbige Zone.* Die farbige Zone $\mathcal{Z}_i^f$ der Farbe $f \in F$ eines Objekts d_i ist die Menge aller Objekte d_j, die mit d_i in Verbindung stehen. Ein Objekt d_j steht mit d_i in Verbindung, wenn es eine Referenz in $\mathcal{E}_i$ gibt, die auf $\mathcal{E}_j$ verweist und diese Referenz besitzt die Farbe f. ◇

4.3.5. Objektklasse

Die Syntax der Objektklasse ist in Appendix A gegeben. Ein Beispiel für eine Objektklasse lautet:

```
objectdef WahlAgent : L4Task {
    required integer versionMajor;
    required integer versionMinor;
    required reference<WahlListe> wahlListe;
};
```

4.3.6. Objektbeschreibung

Bei der Objektbeschreibung (Definition 4.2.1) handelt es sich um eine Menge von Eigenschaften. Zusätzlich besitzt die Objektbeschreibung einen Typ, der die Rollen und Abstraktionen der Objektbeschreibung festhält.

Zudem besitzt jede Objektbeschreibung einen eindeutigen strukturierten Namen. Ein Beispiel für einen solchen Namen ist `list001.wahl2006.wahllisten.de`. Ein solcher Namensraum erlaubt die Objekte zusätzlich logisch anzuordnen. Suchen können sich somit auf Teilbäume beschränken.

Ein Beispiel für eine Objektbeschreibung lautet:

```
object WahlAgent x01_5.wahlurne.de {
    versionMajor = 1;
    versionMinor = 5;
    payloadHashSHA1 = "2f6ae490b2317f7..." ;
};
```

Es ist zu beachten, dass Objektbeschreibungen zur Laufzeit vervollständigt werden können. Dies ist auch notwendig, denn bestimmte Informationen sind nur zur Laufzeit bekannt. Dies gilt zum Beispiel für Prozessnummern. Das Vervollständigen dieser Informationen ist Aufgabe der Anwendung und des SAS. Welche Informationen von wem vervollständigt werden können, wird in Abschnitt 5.3.2 erläutert.

4.3.7. Verhalten der Eigenschaften bei der Vereinigung von Rollen

Die Abstraktion und die Rollenbildung (Abschnitt 4.2.1) bringt ein Problem mit sich. Angenommen eine Eigenschaft e ist sowohl in $\mathcal{E}^A$ als auch in $\mathcal{E}^B$ enthalten. Sei $C = A \cup B$. Durch die Vereinigung der Rollen A und B entsteht das Problem, das Eigenschaft e zweimal vorhanden ist. Dies ist jedoch nicht erlaubt, da Eigenschaften innerhalb von $\mathcal{E}$ eindeutig definiert sein müssen.

Ein ähnliches Problem tritt auf, wenn A eine Spezialisierung von B sein soll.

Eine zufriedenstellende Lösung für dieses Problem zu finden, ist aufwendig. Die Programmiersprache C++, die ebenfalls dieses Problem besitzt, löst es durch den Einsatz von Namensräumen. Solche Namensräume würden die vorgestellte Sprache sehr aufblähen und deshalb wird darauf verzichtet.

In dieser Arbeit wird gefordert, dass eine Eigenschaft nur dann wieder genannt werden darf, wenn auch der Typ und die Kardinalität identisch sind. Dies ist bei Spezialisierungen der Fall, wenn beispielsweise die abstrakte Klasse eine Prüfsumme fordert, die in der Spezialisierung als Konstante mit angegeben wird.

4.3.8. Bilden von Prüfsummen

Die Datenbasis soll später im Zusammenhang mit kryptographischen Algorithmen eingesetzt werden. Inbesondere das Signieren der Objektbeschreibung und der Objektklassen muss gewährleistet werden.

Der klassische Ansatz für eine elektronische Signatur beruht darauf, dass Daten, die im Nachhinein nicht mehr verändert werden, bereits in einer serialisierten Form, zum Beispiel in Form einer Datei, zur Verfügung stehen, über die dann die Signatur errechnet wird.

Im Gegensatz zur klassischen Signatur sollen die Objekte und Objektklassen im Speicher signiert werden. Im Speicher besitzen die Objekte naturgemäß eine andere Darstellung als in der Datei. Wird die Signatur im Speicher durchgeführt, so muss sichergestellt werden, dass die Prüfsummenbildung jedesmal dasselbe Ergebnis für identische Objekte berechnet, unabhängig davon, wann sie auf welchem System in den Speicher geladen wurden.

Zwei Gründe führen mich zu diesem Ansatz. Zum einem besitzen Herausforderer und Dienstleister kein gemeinsames Dateisystem, in dem alle signierten Dateien abgelegt werden können. So ein zentrales Dateisystem zu erstellen, ist ebenfalls aufwendig. Wesentlich interessanter ist der Ansatz, dass jeder Dienstleister seine Zertifikate aufbewahrt und auf Anfrage übermitteln kann. Zudem soll durch die Signatur nicht das Übertragungsmedium bestimmt werden. Wird die Signatur über die serialisierte Form, also der Datei, ausgeführt, wäre dies aber der Fall.

Der klassische Ansatz, Dateien zu signieren, geht vom Prinzip aus, dass nach der Prüfung die Signatur unwichtig ist. Doch im entfernten Attestieren wird die Signatur im weiteren Verlauf noch benötigt. Sie wird beispielsweise bei einem Attest mit übertragen. Ebenso ist die Kenntnis der Signaturen notwendig, um einige von ihnen gezielt zurückziehen zu können, ohne das gesamte Datenmodell verwerfen und neu laden zu müssen. Dies setzt Voraus, das auch die Signaturen in der Datenbasis enthalten sein müssen.

Diese Arbeit ordnet die Eigenschaften in lexikographische Reihenfolge an und bildet darüber eine Prüfsumme, die anschließend signiert wird. Die Signatur ist ebenfalls ein Objekt in der Datenbasis. Durch diesen Ansatz muss das bestehende Modell nicht weiter angepasst werden.

Wird die Prüfsumme über eine Objektbeschreibung gebildet, so muss diese nicht nur die Werte, die es speichert, sondern auch ihren Typ (die Objektklasse) und den Typ der einzelnen Eigenschaften berücksichtigen. Auch die Referenzen müssen samt ihres Typs und Inhalts mit in die Prüfsumme einfließen.

Der Nachteil ist, dass insbesondere die Repräsentation der ganzen Zahlen auf unterschiedlichen Plattformen unterschiedlich sein können (*little* und *big endian*) wodurch Problem in der Portierung entstehen. Das Bilden der Prüfsumme muss diese Tatsache berücksichtigen. In dieser Arbeit wird stets das höchswertigste Byte vorne (links) angenommen (*big endian*).

4.4. Interpretation der Eigenschaften

4.4.1. Die Interpretierungsfunktion

Bisher wurde nur eine Sprache für das Beschreiben von Eigenschaften vorgestellt. Mit Hilfe dieser Beschreibungssprache lässt sich eine Datenbasis beschreiben, die fortan mit Δ bezeichnet wird. Zusätzlich muss die Interpretierungsfunktion $\mathcal{I}$ bestimmt werden.

Die einfachste Form anhand dieser Datenbasis Schlüsse zu ziehen, ist das Prüfen auf eine bestimmte, wesentliche Eigenschaft. Angenommen jedes sichere Programm ist mit einem Prüfzertifikat ausgestattet. Soll geprüft werden, ob ein bestimmtes Programm d sicher ist, so muss seine Objektbeschreibung $\mathcal{E}_d$ dieses Zertifikat aufweisen. Es gilt:

$$\mathcal{E}_d^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ gdw. für } d \text{ ein Zertifikat vorliegt.}$$

Die Existenz dieses Zertifikats bildet eine wesentliche Eigenschaft von d (siehe auch Definition 4.2.7). Erheblich schwieriger ist zu prüfen, ob das gesamte System sicher ist, da die Sicherheit des Systems eine *schwache* Eigenschaft ist[2].

4.4.2. Beschreibungssprachen

Um eine schwache Eigenschaft ermitteln zu können, muss die Interpretierungsfunktion $\mathcal{I}$ erweitert werden. Diese Funktionalität kann auf zwei Wegen umgesetzt werden:

statisches Regelwerk – In diesem Fall werden die notwendigen Funktionen in die Interpretierungsfunktion aufgenommen und algorithmisch umgesetzt. Die Interpretierungsfunktion wird somit speziell für das entfernte Attestieren entwickelt.

dynamisches Regelwerk – Die Interpretierungsfunktion wird möglichst generisch gehalten. Sie kennt im Wesentlichen nur Mengenoperationen. Im Gegenzug wird die Datenbasis um Funktionen bzw. Operatoren erweitert. Die Interpretierungsfunktion kann anhand der Funktionen nun komplexe Sachverhalte interpretieren.

In Bezug auf das dynamische Regelwerke sind besonders die DL-Sprachen (engl. *description logics*) [BCM+03, Bra06, DLNS96] sehr interessant, denn sie betrachten auch

[2] Die Erläuterung dafür wird mit Theorem 5.2.1 erbracht.

das automatisierte Schließen (engl. *resoning*), und können sehr komplexe Sachverhalte ausdrücken. DL-Sprachen sollen einen großen Beitrag zum Finden, Speichern und Verknüpfen von Wissen in großen Netzwerken leisten. OWL[BvHH+04] ist eine DL-Sprache, die in diesem Zusammenhang eingesetzt wird.

Diese Flexibilität muss sich der Anwender mit Ressourcen erkaufen. Zudem ist bei einigen Dialekten der DL-Sprachen (z.B. OWL-Full [BvHH+04]) die Entscheidbarkeit nicht gegeben[3]. Einige Algorithmen können somit entweder falsche bzw. unvollständige Ergebnisse liefern oder sich in einer Endlosschleife verfangen.

4.4.3. Strukturierte Abfragesprache

Unabhängig davon, wie das Regelwerk in der Interpretierungsfunktion $\mathcal{I}$ eingebunden wird, kann die Frage nach der Sicherheit auf unterschiedliche Weisen gestellt werden. Der Vergleich der Prüfsummen kann durch eine strukturierte Abfragesprachen ersetzt werden. Diese Sprache, ich nenne sie *remote attestation query language* (RAQL) in Anlehnung an die SQL-Sprache (engl. *structured query langauge*), erlauben weit mehr als nur die Frage: *Sind alle Prüfsummen bekannt?*

Beispiel. Die Anfrage

```
?type=WahlAgent AND forall(zone(), hasCertificate(X))
```

würde bedeuten: Attestiere die Anwendung vom Typ `WahlAgent`, wobei alle Objektbeschreibungen der Zone eine Signatur besitzen, die durch die Instanz `X` ausgestellt wurde.

4.4.4. Abschließendes zu der Interpretierungsfunktion

Dieser Abschnitt lässt das Attestieren durch das Ladeprotokoll (BPRA) unter einem anderen Licht erscheinen. So handelt es sich bei BPRA auch um das Attestieren von Eigenschaften, wenngleich die Sprache wesentlich einfacher gestaltet ist, da sie nur die Prüfsumme als Eigenschaft zulässt. Die Interpretierungsfunktion besitzt ein statisches Regelwerk und, die Sicherheit wird als schwache Eigenschaft betrachtet.

4.5. Weitere Bewertungen des Systems

Dieser Abschnitt soll die vorgestellte Datenbasis in Hinblick auf ihren Einsatz unter Nizza und für das entfernte Attestieren beurteilen. In Teilen ist dies bereits im vorhergehenden Text geschehen. In diesem Abschnitt werden die Beweggründe nochmals zusammengefasst und ergänzt.

4.5.1. Wahl der Beschreibung der Eigenschaften

Die Datenbasis ist sehr flexibel gehalten und eignet sich nicht nur für den Einsatz im entfernten Attestieren, sondern für weit mehr Einsatzgebiete. Dies ist auch in Bezug auf das entfernte Attestieren notwendig, das sich den Anforderungen der Einsatzgebiete anpassen soll – die Anforderungen an elektronischen Wahlurnen unterscheiden sich von denen kommerzieller Anbieter von Musik-Stücken. Darüber hinaus beschränkt sich die Sicherheit nicht allein auf das entfernte Attestieren. Ist das Musik-Stück einmal verkauft, soll es in Zukunft an bestimmte Bedingungen geknüpft sein, wie zum Beispiel, dass es nur

[3] Ein Problem ist *entscheidbar*, wenn es einen Algorithmus gibt, der in endlicher Zeit und für jede Eingabe ein Ergebnis liefert (vgl. Berechenbarkeitstheorie von Alan Turing).

fünfmal abgespielt werden darf. Umso mehr ist die Wahl einer Beschreibungssprache für Eigenschaften sinnvoll, die über das entfernte Attestieren hinaus eingesetzt werden kann.

4.5.2. Begründung für die Objektklassen

Objektklassen dienen dazu, den Objektbeschreibungen eine bindende Struktur zu verleihen, also die Namen der Eigenschaften und ihren Typ vorzugeben und darüber hinaus eine Menge von Eigenschaften zusammenzufassen. Diese Struktur erlaubt das Bilden eines Standards, über den Informationen effizient ausgetauscht werden können.

Die bindende Struktur ermöglicht den Rechnern die Prüfung auf vollständige Angaben, was wiederum den Programmierer dazu zwingt, die Angaben vollständig zu tätigen. In diesem Sinne sind auch die Modifikatoren für die Deklaration der Eigenschaften vorgesehen.

Das Unterstützen der Rollen und der Spezialisierungen verleiht der Objektklassen-Hierarchie die notwendige Flexibilität. Ihr Einsatz erlaubt das entfernte Attestieren auf Abstraktionsebenen: Sei die Objektklasse `WahlAgentV1_05` eine Spezialisierung der Objektklasse `WahlAgent`. Letztere bezeichnet eine Produktfamilie, ersteres ein konkretes Produkt in einer bestimmten Version. Der Herausforderer kann eine Anwendung des Typs *WahlAgent* fordern, ohne sich auf eine konkrete Version festlegen zu müssen. Ein detaillierteres Beispiel wird in Abschnitt 5.5.1 gegeben.

4.5.3. Referenzen und Zonen

Im vorherigen Kapitel ist erläutert worden, dass das entfernte Attestieren mit Hilfe des Ladeprotokolls einer *Alles-oder-nichts-Philosophie* entspricht (Abschnitt 3.4.5). Vor allem *Nizza*-Systeme erlauben das nebenläufige Ausführen von sicheren und unsicheren Anwendungen. Das Konzept der Zonen kommt für Nizza wie gerufen, denn es erlaubt die Abhängigkeiten zwischen den Anwendungen abzubilden und bestimmte Bereiche gesondert zu betrachten, während andere Anwendungen, die nicht in der Zone aufgeführt werden, unbeachtet bleiben. Mit Hilfe der Referenzen kann die TCB einer Anwendung als Zone beschrieben werden. In dieser Arbeit ist eine Anwendung a von einer Anwendung b genau dann abhängig, wenn, analog zur Definition 2.4.1 für das TCB, b Auswirkungen auf die Sicherheit von a haben kann.

Härtig *et al.* begründen den Einsatz der *trusted wrappers* unter anderen mit der Aussage, dass die Sicherheit nicht zwangsläufig eine transitive Eigenschaft darstellt[HHF+05]. Das bedeutet, dass sichere Komponenten mit unsicheren kommunizieren dürfen, ohne dass die Sicherheit dabei verloren geht. Farbige Referenzen erlauben es, die Abhängigkeiten zwischen den Anwendungen detailliert zu beschreiben und der Referenz eine Wichtung zu geben, wie stark sie sich auf die Sicherheit auswirkt.

4.5.4. Bilden von Standards

Das Messen von Eigenschaften ergibt insbesondere in offenen Netzwerken einen Sinn, also Netzwerke, in denen jeder Anwender die Wahl seiner Anwendungen frei bestimmen kann. Bei der Vielzahl von Systemen ist ein Standard wichtig, dem alle Systeme folgen. Zu diesem Standard gehören unter anderen die Syntax, die Bezeichnung und Vererbungen der Objektklassen, der Typ und Bezeichnung der Eigenschaften.

Diese Arbeit beschäftigt sich vordergründig nicht mit dem Bilden dieses Standards, doch die Referenzimplementierung bietet einen Ansatz für spätere Weiterentwicklungen.

5. Entferntes Attestieren von Eigenschaften

In diesem Kapitel werden die Ergebnisse der bisherigen Analysen zu einem Modell für das entfernte Attestieren von Eigenschaften auf Mikrokern-Architekturen zusammen geführt (PBRA, engl. für *property based remote attestation*).

Dieses Kapitel beschreibt sowohl das Rahmenwerk für das entfernte Attestieren, als auch die Architektur für das Attestieren von Eigenschaften.

5.1. Zielstellung

Anhand der bisher geführten Diskussionen ist ein System zu entwerfen, dass wie die TNC unterschiedliche Formen für RA auf einer Nizza-Architektur zulässt. Darauf aufbauend wird das entfernte Attestieren von Eigenschaften entwickelt. Folgende Anforderungen sind zu beachten:

Aufzählung 5.1.1: *Anforderungen an die Referenzimplementierung.*

/Req 1/ RA anhand des Ladeprotokolls erlauben.

/Req 2/ RA anhand von Eigenschaften erlauben.

/Req 3/ Zurückziehen der Sicherheitsaussage erlauben.

/Req 4/ Datenschutz wahren und Anonymität erlauben.

/Req 5/ Unverkettbarkeit der Atteste gewährleisten.

/Req 6/ Ausgrenzung verhindern.

/Req 7/ Modular gestaltet sein und geringe Komplexität besitzen.

/Req 8/ Sicher sein.

/Req 9/ Mit der Nizza-Architektur vereinbar sein. Das beinhaltet insbesondere das Messen der Zonen.

/Req 10/ Messen einer TCB unterstützen.

5.2. Modell

5.2.1. Die Sicherheit

5.2.1.1. Sicherheit des Systems als schwache Eigenschaft

Theorem 5.2.1: *Sicherheit des Systems als schwache Eigenschaft.* Die Sicherheit eines Systems ist eine schwache Eigenschaft. ◇

Beweis. Laut Korollar 2.1.5 ist ein System genau dann sicher, wenn alle Schichten im System sicher sind. Damit steht die Eigenschaft *sicheres System* in Wechselwirkung zu anderen Eigenschaften. Nach Definition 4.2.8 ist die Sicherheit somit eine *schwache Eigenschaft*[1].

5.2.1.2. Messen der TCB

Gilt Theorem 5.2.1, so folgt, dass alle Anwendungen im System sicher sein müssen. Die Frage nach der Sicherheit von Nizza-Systemen müsste konsequenterweise stets mit *unsicher* beantwortet werden. Doch moderne Betriebssysteme unterscheiden zwischen *bevorzugten* und *normalen* Anwendungen (engl. *kernel-* und *user-space*).

Jede Anwendung besitzt einen eigenen Adressraum. Normale, dritte Anwendungen können diesen Adressraum nicht manipulieren. Diese Isolation wird durch das Betriebssystem gewährleistet. Werden zwei normale Anwendungen a und b in dieser Reihenfolge gestartet und wird das Prinzip des sicheren Urladens angewendet, so bedeutet dies, dass die Sicherheit von b von a abhängt, obwohl beide Anwendungen sich nicht beeinflussen können.

Das Schichtenmodell hat seine Berechtigung für den Startprozess eines Rechners. Doch ab dem Zeitpunkt, in dem das Betriebssystem die Isolation durch Adressraumgrenzen durchsetzen kann, ist die Menge der Anwendungen, die sicher sein müssen damit eine Anwendung b sicher ist genau die TCB von b (vgl. Definition 2.4.1). Mit dieser Änderung lässt sich RA effizient auf Nizza-Systemen umsetzen (siehe auch [HHF+05]).

5.2.1.3. Die sicherheitsrelevanten Rollen

Die Rollen sind bereits aus den vorherigen Kapiteln bekannt. Die TTP wird um die Rolle der CA (engl. *certificat authority*, vgl. Seite 14) ergänzt. Damit bilden die öffentlichen Schlüssel der TTP die Wurzelknoten des PKI-Baums, der in dieser Referenz-Implementierung zum Einsatz kommt.

Auch das SAS besitzt ein asymmetrisches Schlüsselpaar, ein AIK-Schlüssel, und bildet eine CA. Diese CA ist im PKI-Baum enthalten, ist aber kein Wurzelknoten diese Baums.

5.2.1.4. Sichere Zonen

Das System wird in Zonen unterteilt. Jede Zone besteht aus der Anwendung selbst und den Unter-Zonen, die durch die Referenzen gebildet werden. Die Vereinigung aller Unter-Zonen bildet das TCB der Anwendung.

Definition 5.2.2: *Sichere Zone.* Eine Zone $\mathcal{Z}_i$ ist genau dann sicher, wenn

1. Jede Unter-Zone $\mathcal{Z}_j \subset \mathcal{Z}_i$ sicher ist.
2. Alle Elemente d aus $\mathcal{Z}_i$, die nicht in den Unter-Zonen erfasst sind, ebenfalls sicher sind[2].

◇

Abbildung 5.1 zeigt exemplarisch den Aufbau einer Zone C mit seinen Unter-Zonen.

[1] Das entfernte Attestieren von Eigenschaften verwendet nicht das Konzept der sicheren Schichten (Definition 2.1.4), sondern das der sicheren Zonen, das in Kürze in Definition 5.2.2 eingeführt wird. Für diesen Fall ist der Beweis analog zu führen.

[2] Für gewöhnlich handelt es sich immer nur um ein Element, die Anwendung selbst.

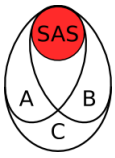

Abbildung 5.1.: Das System ist in Zonen unterteilt. Jedes Programm besitzt eine Zone. Dem Programm c gehört die Zone C. c ist von den Zonen A und B abhängig, A und B umfassen die TCB für Anwendung c. Diese Zonen, wiederum basieren auf der Zone SAS (bzw. der TCB von A und B). Ist das Programm c sicher und sind die Zonen A und B sicher, dann ist auch die Zone C sicher (Definition 5.2.2).

Mit Hilfe der Definition für sichere Zonen kann die Interpretierungsfunktion für das Prüfen der Zone definiert werden. Sie lautet:

Korollar 5.2.3: Sei d_i die Objektbeschreibung eines zu prüfenden Programms und $\mathcal{Z}_i$ die dazugehörige Zone. Sei Δ die Referenz. Die Interpretierungsfunktion $\mathcal{I}$ lautet:

$$\mathcal{Z}_i^{\mathcal{I}} \in \Delta^{\mathcal{I}} \text{ gdw. } \mathcal{Z}_i \text{ sicher ist.}$$

◇

Diese Definition besagt, dass System genau dann sicher ist, wenn die Zone $\mathcal{Z}_i$ der Anwendung d_i sicher ist (in der Interpretation der Referenz enthalten ist). Die Definition der sicheren Zone ist rekursiv und benötigt eine Abbruchbedingung für die Rekursion, die durch die Definition der *sicheren Zone-0* gegeben wird:

Definition 5.2.4: *Sichere Zone-0.* Jedes System verfügt über eine eindeutige bestimmte und kleinste Zone, die Zone-0, in der nur die Anwendungen enthalten sind, die für das entfernte Attestieren notwendig sind. Diese Zone ist sicher. ◇

Die Sicherheit der sicheren Zone-0 wird in Abschnitt 5.3.1 behandelt. Die Zone-0 wird durch das SAS gestellt.

Nun muss noch ermittelt werden, wann ein Element einer Zone sicher ist (Definition 5.2.2, Punkt 2).

5.2.1.5. Sichere Anwendung

Die Interpretierungsfunktion $\mathcal{I}$ soll an dieser Stelle etwas genauer betrachtet werden. Dazu muss festgelegt werden, wann eine Anwendung als sicher angenommen werden kann. Dies geschieht in zwei Schritten. Zuerst wird nachgewiesen, dass der Objektbeschreibung getraut werden kann. In einem zweiten Schritt muss zusätzlich gezeigt werden, dass auch die korrekte Objektbeschreibung für die Anwendung verwendet wurde. Für die Referenzimplementierung gilt:

Definition 5.2.5: *Sichere Objektbeschreibung.* Sei $\mathcal{E}$ die Objektbeschreibung der Anwendung d. Die Objektbeschreibung wird genau dann als sicher angesehen, wenn der Typ der Objektbeschreibung einer Objektklasse gehört, die von einer vertrauten Instanz (der TTP) signiert wurde. ◇

Beispiel. Das Programm `WahlAgent` besitzt eine Objektbeschreibung $\mathcal{E}$. Diese Objektbeschreibung ist vom Typ A. Dieser Typ bestimmt die Objektklasse. Existiert für diese Objektklasse eine Signatur der TTP, so ist das Programm `WahlAgent` eine sichere Anwendung.

Korollar 5.2.6: Aus Definition 5.2.5 folgt:

$$\mathcal{E}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ für sichere Objektbeschreibungen } \mathcal{E}$$

◇

Es muss noch gewährleistet werden, dass einer Anwendung im System auch die korrekte Objektbeschreibung zugeordnet wird. Das erledigt die Funktion $\mathcal{X}$:

Definition 5.2.7: *Sichere Objekt-Objektbeschreibung-Zuordnung.* Sei d eine Anwendung und D die Menge aller Anwendungen im System. Sei $\mathcal{E}$ die Objektbeschreibung von d. Die Funktion $\mathcal{X}$ ermittelt zu einer Anwendung d die korrekte Objektbeschreibung $\mathcal{E}$:

$$\mathcal{E} = \mathcal{X}(d)$$

◇

Die Existenz dieser Funktion ist eine notwendige Voraussetzung für PBRA. Dass es sie gibt und wie sie umgesetzt wird, wird in Abschnitt 5.3.2 gezeigt.

Nun muss noch festgestellt werden, wer die Anwendung in den Speicher geladen und ausgeführt hat, denn, wie es bereits zu Beispiel 3.4.4 erläutert wurde, kann eine unsichere Komponente keine sichere Komponente erstellen.

Die Lösung liegt darin, den Ladevorgang einer Anwendung zu prüfen, indem der Vaterprozess einer Anwendung überprüft wird. Sei V die *Vater-Funktion*, die den Vater-Prozess zu einer gegebenen Anwendung d ermittelt. Damit die Anwendung sicher ist, muss auch $V(d)$ sicher sein:

Korollar 5.2.8: *Sichere Anwendung.* Sei d eine Anwendung im System. Die Anwendung ist genau dann sicher, wenn

1. $(\mathcal{X}(d))^{\mathcal{I}} \subset \Delta^{\mathcal{I}}$ und
2. $(V(d))^{\mathcal{I}} \subset \Delta^{\mathcal{I}}$

gelten. ◇

Für Anwendungen, die in der SAS enthalten sind, kann Korollar 5.2.8 nicht angewendet werden, da es mitunter keinen Vaterprozess geben muss. Es ist, wegen Theorem 2.1.8, ohnehin nicht sinnvoll die Sicherheit des SAS alleine auf Definition 5.2.7 zu gründen. Diese Anwendungen müssen durch die RAL_1 abgesichert werden. Somit greift Anforderung 3.2.2, dass das SAS sicher sein muss. Für Korollar 5.2.8 folgt daraus zusätzlich:

Korollar 5.2.9: *Sicherer Vaterprozess.* Der Vaterprozess $V(d)$ der Anwendung d ist genau dann sicher, wenn

1. der Vaterprozess in der SAS enthalten ist, $V(d) \in \text{SAS}$, oder
2. $V(d)$ nach Korollar 5.2.8 ebenfalls eine sichere Anwendung ist.

◇

5.2.2. Drei-Schichten-System

Das SAS aus Kapitel 3 (vgl. Tabelle 3.1, Schicht RAL_1) ist nicht in der Lage, eine TCB zu ermitteln und Zonen und Eigenschaften zu berücksichtigen. Aus diesem Grund wird eine neue RAL_2 eingeführt, die sich auch von der RAL_2 aus Kapitel 3 unterscheidet. Die

	Schicht RAL_0	Schicht RAL_1	Schicht RAL_2
Modell:	EK und AIK Signaturen	überprüfbares und sicheres Urladen	sichere Zonen sichere Anwendungen
Sicherheit messen:	Signiertes EK	Ladeprotokoll Prüfsummen	Zertifikate, Zonen, Objektbeschreibung
Orakel:	MK, Δ	Prüfsummen, Δ	PKI, CA
Schicht-0/Rolle:	TPM	SAS	SAS/PBRA
Sicherheit durch:	Annahme 2.1.1	RAL_0	RAL_1

Tabelle 5.1.: Aufbau der RAL-Schichten für das entfernte Attestieren von Eigenschaften.

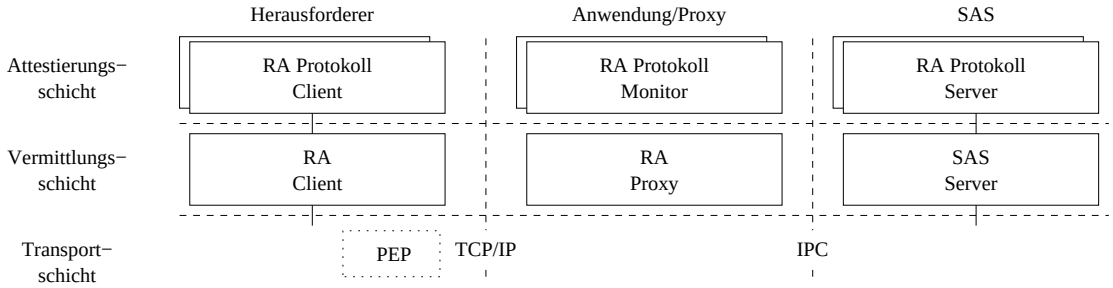

Abbildung 5.2.: *Das Rahmenwerk.* Die Architektur für das entfernte Attestieren orientiert sich an der TNC (vgl. Abbildung 2.2), legt den Fokus jedoch auf den Kommunikationspartner, der seinen Zustand offenlegen muss. Der wichtigste Unterschied ist, dass die Anwendung mit den Monitoren den Informationsfluss überwachen kann. Der PEP wird in die Rolle des Herausforderers integriert.

Tabelle 5.1 verdeutlicht diesen Unterschied. Anstatt ein virtuelles TPM für RAL_2 einzusetzen, wird ein vollständig neues Modell erarbeitet, das in der Lage ist, Anwendungen mit Eigenschaften zu versehen, Beziehungen zwischen den Anwendungen abzubilden, und diese auszuwerten.

5.2.3. Das Rahmenwerk

Um die neue Attestierungsschicht RAL_2 effizient einzusetzen, muss das System auch die RAL_1 zur Verfügung stellen. Das SAS sollte somit grundsätzlich mehrere solcher Attestierungsverfahren unterstützen. Die Referenzimplementierung lässt dies zu.

Die eingesetzte Architektur (Abbildung 5.2) gleicht auf dem ersten Blick dem Modell der TNC (vgl. Abbildung 2.2). Auch die Bezeichnungen der Schichten ist der TNC entlehnt. Die unterste Schicht, der Transportschicht, wird in dieser Arbeit nicht betrachtet.

Die Vermittlungsschicht stellt die abstrakte Schnittstelle für das entfernte Attestieren bereit. Über diese Schicht können Sitzungen verwaltet werden, und sie ermittelt zudem das konkrete Attestierungsprotokoll (RAL_1 oder RAL_2), dass eingesetzt werden soll. Jedes Protokoll wird durch ein unabhängiges Modul in der Attestierungschicht umgesetzt.

Im Unterschied zur TNC sind die Rollen und die Aufgaben etwas anders verteilt. Der große Unterschied liegt in der Tatsache, dass die Anwendung nun eine aktive Rolle im Prozess erhält. Sie nimmt die Anfrage des Herausforderers entgegen und leitet sie an das SAS weiter. Die Anwendung arbeitet als *Proxy* und kann sich der *Monitore* bedienen, um die ausgetauschten Informationen einzusehen und gegebenenfalls das Protokoll abzubrechen. Monitore kennen das eingesetzte Protokoll und können somit den gesamten Prozess des entfernten Attestierens überwachen. Die Anwendung dieser Monitore ist optional, um die

Name	Beschreibung
Schicht 3 Schema	Beschreibt einen Standard, wie Eigenschaften strukturiert und benannt werden sollen. Hier sind die Objektklassen angesiedelt. (vgl. Abbildung 5.3).
Schicht 2 Instanz-Schicht	Menge aller Komponenten im System, durch Objektbeschreibungen dargestellt.
Schicht 1 System-Schicht	Das laufende System mit all seinen aktiven Anwendungen und seinem aktuellen Zustand (vgl. Abbildung 1.1).

Tabelle 5.2.: Die Beschreibung der Eigenschaften erfolgt in mehreren Abstraktionsschichten. In der untersten Schicht befinden sich die Anwendungen, die beschrieben werden sollen. Schicht 2 beinhaltet diese Beschreibungen. Diese Beschreibung folgt einer Vorlage, die Schicht 3 bildet.

Anwendungen nicht unnötigerweise komplex zu machen. Das PEP aus der TNC wird in die Rolle des Herausforders integriert.

5.2.4. Eigenschaften unter Nizza

Das Messen der Eigenschaften erfolgt in einem Modul (fortan RAL_2 genannt), das in der Attestierungsschicht aus Abbildung 5.2 angesiedelt ist. Die Datenbasis dieses Moduls lässt sich ebenfalls in Ebenen abbilden. Diese sind in Tabelle 5.2 dargestellt.

Die Architektur setzt sich aus drei Schichten zusammen. Die unterste Schicht (Systemschicht) wird durch das reale System gebildet. Es beinhaltet die Programme und seine Beziehungen zu anderen Anwendungen. Darüber folgt die Schicht, in der die Objektbeschreibungen angesiedelt sind. Diese Beschreibungen folgen einer Vorlage, den Objektklassen. Diese bilden die dritte Schicht. Zudem wird in dieser Schicht ein Standard für die Beschreibung der Eigenschaften angesiedelt, das in Abschnitt 5.3.4 genauer erläutert wird. Die Instanz-Schicht wird durch das SAS verwaltet, das Schema ist vorgegeben.

5.3. Technik

5.3.1. Absicherung des SAS

Die sichere Zone-0 ist identisch mit dem *sicheren Attestierungs-System* (SAS, vgl. Abschnitt 3.1). Das SAS selbst ist das TCB der Anwendung, die die Atteste ausstellt. Diese Anwendung wird TAM für *trusted attestation manager* genannt.

Das SAS bildet die Schicht-0 für das entfernte Attestieren von Eigenschaften (RAL_2). Diese Schicht muss durch das Ladeprotokoll RAL_1 abgesichert werden[3]. Für diese Absicherung stehen im Wesentlichen zwei Varianten zur Verfügung:

5.3.1.1. Attestierung mit einer TTP

Analog zu dem Anonymen RA aus Algorithmus 3.5.1, prüft eine TTP zuerst die RAL_1 und stellt dann eine Gruppenmitgliedschaft aus, indem ein AIK erstellt, beziehungsweise unterzeichnet wird.

[3] vgl. Theorem 2.1.8 und Lemma 2.1.7.

5.3.1.2. Attestierung ohne TTP

Im zweiten Ansatz werden die Rollen des Herausforderers und der TTP gewissermaßen vereint, da der Herausforderer das Ladeprotokoll erhält und damit auf herkömmlicher Art das SAS prüft, also durch Einsatz des Ladeprotokolls (vgl. Abschnitt 3.4).

5.3.2. Funktion $\mathcal{X}$

Im ersten Teil dieses Kapitels wurde bereits die Funktion $\mathcal{X}$ vorgestellt, die als Verbindungsglied zwischen dem System und der Beschreibung der Eigenschaften dienen soll. Soll diese Funktion in Tabelle 5.2 eingezeichnet werden, so würde sie zwischen die Schichten Objektbeschreibung und System gehören.

Es stellt sich grundsätzlich die Frage, wie diese Objektbeschreibung an den Programmcode der Anwendung gekoppelt werden kann. Es wird schnell deutlich, dass es nicht reicht, die Objektbeschreibung in den Programmcode einzubetten. Die Objektbeschreibung muss so gestaltet werden, dass sie sich nicht ohne weiteres ersetzen oder ändern lässt.

Für die Lösung dieses Problems bieten sich digitale Signatursysteme an. Gemäß Definition 5.2.5 werden Zertifikate eingesetzt, die die Anwendung durch eine Prüfsumme über ihren Programmcode identifizieren.

Wendet sich eine Anwendung an das SAS, muss es die Identität der Anwendung ermitteln können. In den meisten Systemen ist lediglich die Absenderadresse (z.B. eine Prozessnummer) der Anwendung bekannt. Doch diese Absenderadresse genügt meistens nicht zur eindeutigen Identifizierung. Solche Adressen sind vom aktuellen Zustand des Systems abhängig. Sie können im Verlauf der Zeit mehrfach für unterschiedliche Anwendungen eingesetzt werden. Auch die Prüfsumme über eine Anwendung reicht in diesem Fall nicht für die Identifizierung der Anwendung aus, da es durchaus vorkommen kann, dass ein Programm mehrfach ausgeführt wird. Die Kombination von Prozessnummer und Prüfsumme funktioniert erst dann, wenn die Prozessnummern nicht wiederverwendet werden.

Als Identität kann ein Index in das Ladeprotokoll, eine fortlaufende Nummer oder eine zufällige Zahl genommen werden. Diese Identität wird in der Objektbeschreibung vermerkt.

Für die Referenzimplementierung wird angenommen, dass es eine Instanz gibt, die folgender Annahme genügt.

Annahme 5.3.1: Es existiert eine zentrale Instanz $\mathcal{T}$, die über alle laufenden Anwendungen informiert ist und die für jede Anwendung die

1. die Prozessnummer,
2. den Zustand (Aktiv oder Beendet),
3. den Typ (Ausführbare Datei oder Daten),
4. die Prüfsumme über ihren Programmcode,
5. und die Identität des Vaterprozess kennt.

Diese Instanz ist Teil des SAS. ◇

$\mathcal{T}$ erlaubt jeder Anwendung neue Einträge zu tätigen vermerkt die Anwendung jedoch als Vaterprozess des neuen Eintrags. Die Identität einer Anwendung wird durch $\mathcal{T}$ bestimmt.

$\mathcal{X}$ kann anhand der Prozessnummer und mit $\mathcal{T}$ eine Identität der Anwendung ermitteln, die auch in der Objektbeschreibung hinterlegt ist, und somit jedem Prozess eine Objektbeschreibung eindeutig zuordnen.

5.3.3. Erstellen und Ergänzen der Objektbeschreibungen

Das SAS benötigt auch die Zusammenarbeit mit den Anwendungen, denn diese müssen dem SAS die Informationen zu den Referenzen mitteilen. In dieser Hinsicht bietet das SAS jeder Anwendung eine Kontroll-Schnittstelle (`TamCtrlInterface`), über die die eigene Objektbeschreibung erzeugt und ergänzt werden kann. Um sicherzustellen, dass eine Anwendung nicht die Referenzen Dritter setzen kann, darf jede Anwendung nur seine eigene Objektbeschreibung verändern.

Das Vervollständigen der Referenzen ist Aufgabe der Anwendung in Zusammenarbeit mit dem SAS. Die Integrität der Referenzen wird auf das Prinzip des sicheren Urladens zurückgeführt: Ist die Anwendung sicher, so darf ihr auch getraut werden, dass sie die Referenzen korrekt angibt. Die referenzierten Objekte sind nach Korollar 5.2.8 zu prüfen und müssen somit dem Herausforderer ebenfalls übermittelt werden.

Wird ein neues Programm gestartet, so muss sichergestellt werden, das die Objektbeschreibung den korrekten Typ besitzt. Ist die Instanz $\mathcal{T}$ aus Annahme 5.3.1 vorhanden, so kann diese Auskunft über die Identität der Anwendung erteilen indem nach der entsprechenden Prozessnummer nachgefragt wird. Da die Anwendung nicht in der Datenbasis des SAS enthalten ist, muss sie angelegt werden. Die Objektklasse der Objektbeschreibung wird dabei von der Prüfsumme bestimmt, die ebenfalls $\mathcal{T}$ bekannt ist. Die Objektbeschreibung wird um die Identität ergänzt, die $\mathcal{T}$ der Anwendung gegeben hat.

Die Objektklassen besitzen ihre Gültigkeit aufgrund der Signaturen, mit denen sie versehen sind. Folglich müssen diese Daten nicht über zusätzliche Schutzmaßnahmen geschützt werden. Es ist irrelevant, wer die Objektklassen dem SAS zur Verfügung stellt. Diese Tatsache erlaubt es, jeder Anwendung selbst zu überlassen diese Zertifikate dem SAS zur Verfügung zu stellen.

Die Zertifikate könnten somit auf zentralen Dateisystemen bereit stehen, als Argumente oder als Eingabe übermittelt werden. Aber auch das Einbetten der Zertifikate in den Programmcode ist möglich, sofern die Zertifikate die Prüfsummen über den Programmcode und die Signatur besitzen[4]. Durch die Vielzahl der Wege, wie diese Information bereitgestellt werden kann, kann jedes Programm und jede Datei mit solchen Zertifikaten versehen werden.

5.3.4. Der Standard für die Referenzimplementierung

Jede Objektbeschreibung ist eine Instanz einer Objektklasse. Die Objektklassen (Bezeichnung und Typ der Eigenschaften, Name) und ihre Beziehungen (Vererbung, Rollen etc.) werden in einem Schema verbindlich vorgeschrieben. Abbildung 5.2 zeigt Auszüge des Schemas der Referenzimplementierung.

Die Objektklasse `RaData` besitzt das Feld `payloadHash` und das Feld `bpindex`, dass die Prüfsumme des Programmtextes und die Identifikation aufnimmt (in diesem Fall den Index im Ladeprotokoll). Die Prozessnummer wird in der Eigenschaft `taskID` aus der Klasse `L4Task` vermerkt. Diese Felder sind notwendig, damit die Funktion $\mathcal{X}$ jeder Anwendung im System eine Objektbeschreibung zuordnen kann.

[4] [vDBA] beschreiben, wie solche Zertifikate in ELF-Programme [elf95, You95b, You95a] eingebettet werden können.

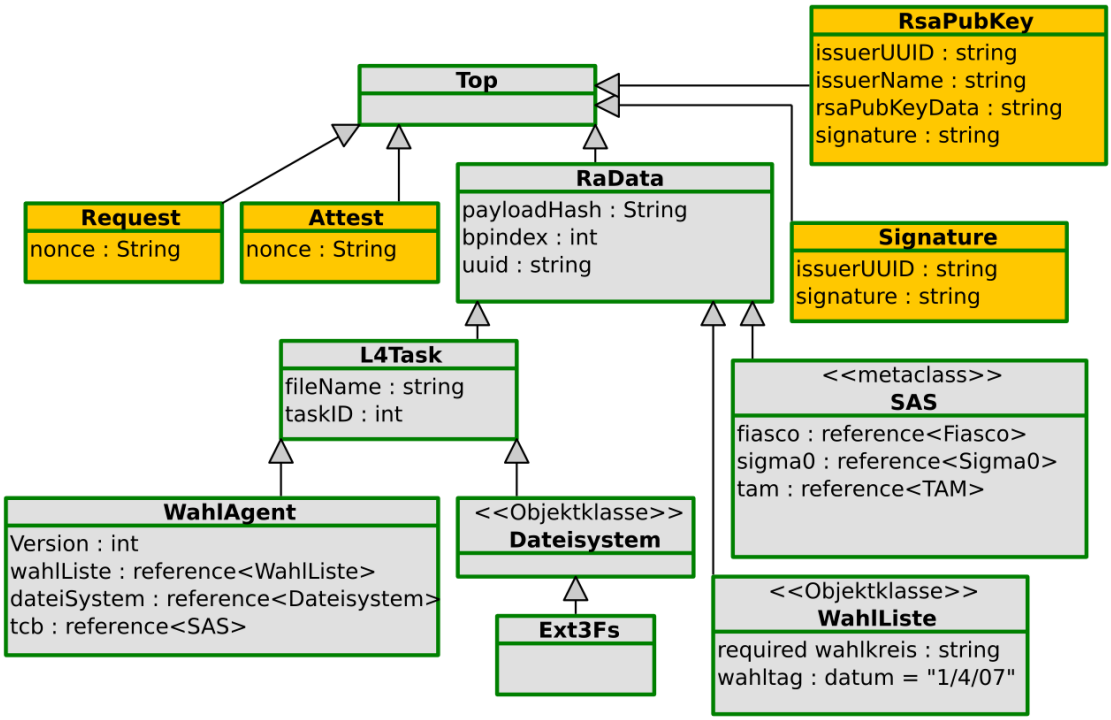

Abbildung 5.3.: Beispielhafte Umsetzung des Standards für das Beschreiben von Eigenschaften auf Nizza-Architekturen. Das Schema schreibt die Bezeichnung der Objekte und der Eigenschaften vor. Zudem gliedert es die Objekte anhand ihres Typs in unterschiedliche Klassen (`WahlAgent`, `L4Task` etc.)

5.3.5. Das Orakel

PBRA erlaubt eine interessante Umsetzung des Orakels. Vorausgesetzt das SAS wird wie in Abschnitt 5.3.1.1 beschrieben von einer TTP geprüft, so muss das Orakel für das Prüfen des Ladeprotokolls (RAL_1) nur bei der TTP implementiert sein. Der Herausforderer benötigt diese Information nicht, da er die Sicherheit des SAS nicht mehr selbst messen muss.

Werden die Objektbeschreibungen und die Objektklassen aus dem Schema bei der Attestierung mit übertragen, so kann sich das Orakel auf der Seite des Herausforderers auf die öffentlichen Schlüssel der eingesetzten TTP beschränken, die die Wurzel einer PKI bilden (vgl. Abschnitt 2.5.2); Die Liste der Prüfsummen und die Liste der MK-Schlüssel entfallen.

Die Anzahl der TTP-Schlüssel ist meist sehr begrenzt. Die Wahrscheinlichkeit w, dass die Referenz unsicher ist und angepasst werden muss, bleibt trotz wachsender Anzahl der ausgestellten Zertifikate unverändert. Zwar gilt noch immer die Gleichung (3.1) (Seite 29), doch da die Referenz nur aus den TTP-Schlüssel besteht, fällt diese Wahrscheinlichkeit gering aus.

Dieser Ansatz hat über das Problem Anwendungen als unsicher zu erklären vollständig hinweg gesehen. Dies wird in nächsten Abschnitt nachgeholt.

5.3.6. Zurückziehen der Eigenschaften

Jedes Orakel aus RAL_2 müsste eine CRL-Liste über alle Signaturen besitzen, die nicht mehr gültig sind. Wird eine Signatur für eine Objektklasse zurückgezogen, so wird die Anwendung dieser Objektklasse nicht mehr als sicher angesehen (vgl. Korollar 5.2.5).

Die Signaturen werden bei der entfernten Attestierung mit übertragen und können auf der Seite des Herausforders überprüft werden. Doch in diesem Fall fällt die Speicherkomplexität des Orakels wieder höher aus und auch die Häufigkeit der Aktualisierungen des Orakels nimmt wieder zu (vgl. Gleichung 3.1), da die CRL-Listen bei den Herausforderern vorhanden sein müssen.

Kann dem SAS getraut werden, was definitionsgemäß der Fall ist, so kann es auch selbst die CRL-Listen prüfen und nur gültige Signaturen übertragen (oder ungültige entsprechend kennzeichnen). Jeder Dienstleister wäre fortan selbst in der Pflicht, die CRL-Listen aktualisiert zu halten. Jeder Dienstleister muss zudem nur den Teil der CRL-Listen speichern, der die Anwendungen betrifft die vom Dienstleister eingesetzt werden.

Werden die CRL-Listen vom SAS eingesetzt, so stellt sich dem Herausforderer die Frage, ob die eingesetzten Listen aktuell sind. Aus diesem Grund kann das SAS die eingesetzten Listen benennen. Ob die Listen mit einem Verfallsdatum (Zeitstempel) versehen werden, oder der Herausforderer die eingesetzten Listen mit der TTP abgleicht, oder ob ganz andere Verfahren eingesetzt werden, um sicherzustellen, dass das SAS aktuelle Listen einsetzt, wird in dieser Arbeit nicht weiter untersucht. Doch diese Information stellt selbst eine Eigenschaft, in diesem Fall des SAS dar, und kann als Teil des Attest übermittelt werden.

5.3.7. Reduzible Zonen

Die Definition der sicheren Zonen kann mit einem Baum verglichen werden, der Schritt für Schritt reduziert wird. Solche Bäume effizient zu reduzieren, setzt eine strenge Hierarchie ihrer Struktur voraus. Diese Struktur kann in den Beziehungen zwischen den Anwendungen jedoch nicht vorausgesetzt werden.

Vielmehr handelt es sich bei realen Systemen um ein Geflecht, in dem Zyklen nicht ausgeschlossen werden können. Eine zyklische Abhängigkeit hat zur Folge, dass Zone und Unter-Zone gleich sind; Wird Definition 5.2.2 angewendet, verfängt sich der Algorithmus in einer Endlosschleife.

Es existieren unterschiedliche Ansätze, wie dieses Problem gelöst werden kann. Zum einen gibt es Algorithmen, die Knoten in diesem Geflecht so vereinen können, so dass es sich reduzieren lässt. Diese Ansätze werden in der Graphentheorie behandelt.

Ein zweiter Ansatz besteht darin, alle Zonen zu vereinen und alle Elemente dieser neuen Menge zu prüfen. Dieser Ansatz ist dann möglich, wenn alle Referenzen bzw. Zonen die selbe Farbe besitzen. Die Referenzimplementierung setzt diesen Ansatz ein.

5.4. Das Attest

5.4.1. Anfordern eines Attests

Das Attest wird über eine besondere Struktur angefordert, die selbst anhand einer Objektklasse festgeschrieben ist. Für die Referenzimplementierung wird folgende Objektklassen verwendet:

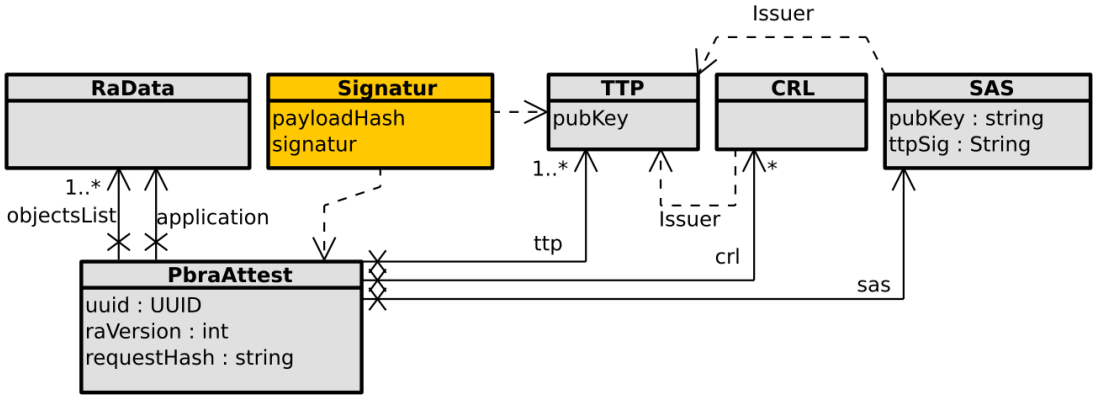

Abbildung 5.4.: Aufbau des Attests

```
objectdef RaRequest : Top {
required string nonce;
};

objectdef PbraRequest : RaRequest {
reference<RaData> application;
};
```

Die Eigenschaft `nonce` ist ein zufälliger Wert, der dem *challenge-response*-Verfahren genügen soll. Da die Anfrage jedoch mehr Informationen als lediglich das `nonce` besitzt, wird als Sitzungsnummer nicht das `nonce`, sondern die Prüfsumme über die gesamte Anfrage, also über der gesamten Objektbeschreibung verwendet[5]. Diese Information wird im Attest unter der Eigenschaft `requestHash` gespeichert. Damit ist gewährleistet, dass auch komplexe Anfragen unterstützt werden können und ihre Argumente während der Übertragung nicht verändert werden.

Das Feld `application` ist optional und bestimmt die Anwendung, die gemessen werden soll. Ist dieses Feld nicht angegeben, so wird die Anwendung gemessen, die die Anfrage an das TAM gestellt hat.

5.4.2. Der Aufbau des Attests

Das Attest besitzt den Aufbau, der in Abbildung 5.4 gezeigt wird. Jedes Attest wird mit der Version und dem Typ des Protokolls (z.B. RAL_1 oder RAL_2) vermerkt. Ein Attest besitzt zusätzlich folgende Objekte:

application – Dieses Objekt vom Typ `RaData` repräsentiert die Anwendung (Objektbeschreibung), die gemessen werden soll.

[5] Sei d die Anfrage (Objektbeschreibung). Sei $E(a)$ die Entropie von einem Datum a. Zu zeigen ist, dass $E(\texttt{nonce}) \leq E(\mathcal{H}(d))$, wobei d eine Konkatenation von `nonce` und weiteren Daten m_i darstellt. Daraus folgt: $E(\texttt{nonce}) \leq E(\mathcal{H}(\texttt{nonce} \| m_1 \| \cdots))$. Dieser Beweis ist allgemein sehr schwer zu führen, da es sehr fortgeschrittene Kenntnisse über die Operation $\mathcal{H}$ erfordert. Doch wird angenommen, dass die Prüfsummenbildung $\mathcal{H}$ die Entropie weitgehend erhält, so ist die Entropie $\mathcal{H}(d)$ ungefähr $min(log_2(|H|), E(\texttt{nonce}))$. Sind die Ausgabe von $\mathcal{H}$ und das `nonce` lang genug, so kann $E(\mathcal{H}(d))$ als ausreichend zufällig angenommen werden (vgl. [Kru04]).

objectList – Die Vereinigung aller Zonen und Unterzonen bildet eine Menge von Objektbeschreibungen. Diese Menge wird in `objectList` aufbewahrt.

ttp – Eine Liste der TTPs, die verwendet werden. Diese Liste ist insbesondere dann interessant, wenn Kindsknoten der PKI in der Lage sein sollen Objektklassen zu unterzeichnen. Sind die TTP allgemein bekannt (und somit beim Herausforderer hinterlegt), darf diese Liste leer sein.

SAS – Sowohl den öffentlichen Schlüssel, den das SAS einsetzt, als auch seine Akkreditierung. Diese Akkreditierung kann durch das Ladeprotokoll oder durch den Einsatz einer TTP erfolgen (vgl. Abschnitt 5.3.1). Dieses Objekt dient zur Absicherung der Schicht-0 von PBRA. Über dieses Objekt erfolgt die Absicherung der RAL_2 durch die RAL_1.

CRL – Eine Liste der CRL-Listen, die das SAS verwendet.

Die Signatur des Attest bildet ein ganz besonderes Objekt. Sie kann nicht im Attest referenziert werden, denn die Signatur wird über die Prüfsumme gebildet, die über die Zone berechnet wird, die das Attests aufgespannt. Die Signatur kann deshalb nicht in der Zone des Attests enthalten sein. Sie wird gesondert übermittelt.

Das Schema kann, wie in Abschnitt 5.3.3 erläutert, gesondert übertragen werden. In der Referezimplementierung werden die Objektklassen dem Attest samt ihren Signaturen angehängt.

Abbildung 5.5 zeigt das Zusammenspiel der Signaturen, Objektbeschreibungen und Objektklassen.

5.4.3. Auswerten eines Attests

Im Prinzip müssen dieselben Schritte ausgeführt werden, wie sie in Abbildung 3.2.3 auf Seite 23 gezeigt sind. Die Signatur des Attests muss als erstes geprüft werden. Die Sitzung wird über die Prüfsumme der Anfrage geprüft und muss ebenfalls im Attest hinterlegt sein. Die attestierte Anwendung wird durch das Objekt `application` bestimmt und muss den Erwartungen des Herausforderers entsprechen.

Das SAS kann, wie in Abschnitt 5.3.1 beschrieben auf, zwei Varianten geprüft werden. Wird eine TTP eingesetzt, so beinhaltet das SAS lediglich einen AIK-Schlüssel und die Signatur der TTP über den AIK. Soll das Attestieren über das Ladeprotokoll erfolgen, so ist an dieser Stelle das Ladeprotokoll und das Quote notwendig.

Zuletzt fehlt noch das Prüfen des Systems. Die Vorgehensweise wurde bereits zu Beginn dieses Kapitels erläutert. Das Attest beinhaltet alle Informationen, um diese Überprüfung durchzuführen.

5.5. Chancen von PBRA

5.5.1. Messen von Generalisierungen

Einen interessanten Ansatz bietet das Messen über Abstraktionsstufen (Generalisierungen) und Rollen, was anhand der Abbildung 5.5 verdeutlicht werden soll. Das Programm `WahlAgent` besitzt eine Objektbeschreibung d (Objekt `#9.02`) vom Typ `WahlAgentV1_05`. In dieser Objektklasse ist die Prüfsumme der Anwendung enthalten. Soll diese Information zurückgehalten werden, so kann folgender Ansatz gewählt werden:

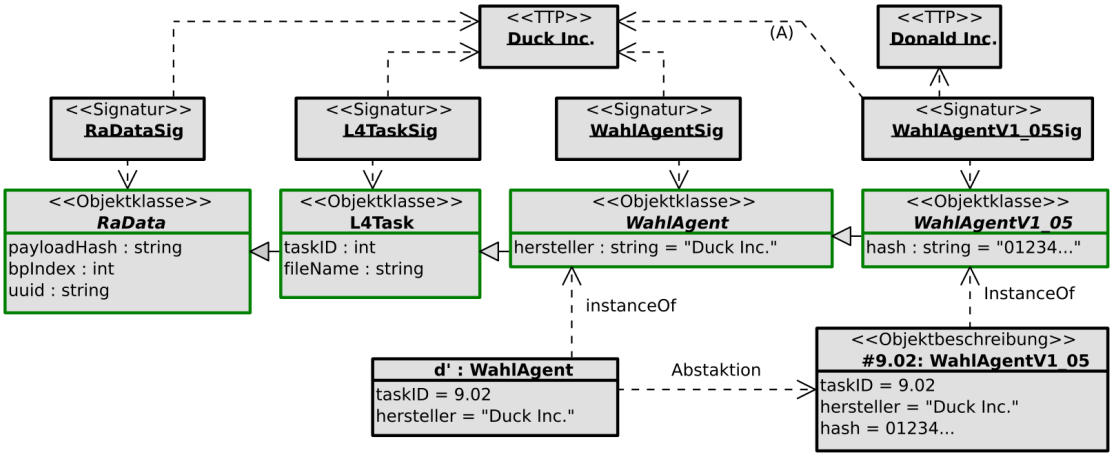

Abbildung 5.5.: Schema und Objektbeschreibung für die Anwendung `WahlAgentV1_05`.

Das TAM erstellt eine temporäre Objektbeschreibung d' vom gewünschten Typ, beispielsweise `WahlAgent` und kopiert die Werte für d' aus d. Es gilt: `WahlAgent` $\sqsubset$ `WahlAgentV1_05`. Wegen Gleichung (4.1) auf Seite 39 gilt auch $d' \subseteq d$.

Anstatt d wird beim entfernten Attestieren d' gesendet. Beim Prüfen des Attests, im Schritt *Anwendung ermitteln* aus Abbildung 3.2.3, wird nun nicht mehr die Anwendung selbst, sondern eine Abstraktion der Anwendung erwähnt. Der Herausforderer muss nun entscheiden, ob dieses Verfahren zulässig ist.

Analog kann dieses Verfahren für das Attestieren von Rollen eingesetzt werden. Soll überhaupt keine Information über das System nach außen gelangen, so wird bis zur obersten Ebene, `Top`, abstrahiert. Damit wird das Attestieren durch Eigenschaften durch das Attestieren durch Zertifikate ersetzt.

Dieses Verfahren besitzt ein Problem. Das Abstrahieren entfernt Information aus dem Attest. Wird bis zur Objektklasse `Top` abstrahiert, sind auch die Zonen nicht mehr enthalten.

In Definition 5.2.5 wird die Sicherheit eines Objekts durch die Signatur einer TTP festgesetzt. Dabei ist es durchaus erwünscht, dass unterschiedliche TTP im System vorhanden sind[6]. Angenommen die Objektklasse `WahlAgentV1_05` besäße keine Signatur durch die TTP *Duck Inc.* (vgl. Pfeil (*A*) aus Abbildung 5.5), sondern lediglich eine Signatur durch die TTP *Donald Inc.*. In diesem Fall würde die Abstraktion zu der Objektklasse `WahlAgent` diese Information vernichten. Der Herausforderer würde noch immer denken, er *redet* mit einer Wahlurne *aus dem Haus Duck Inc.*.

Die Referenzimplementierung setzt nur eine TTP ein und ist von diesem Problem nicht betroffen.

5.5.2. Der Weg zu einem DRM-System

Eine Anwendung wird dann als sicher angesehen, wenn sie bestimmte Kriterien erfüllt (vgl. das Beispiel für die RAQL-Sprache auf Seite 44). Werden diese Kriterien in der Beschreibung von Daten erwähnt, so kann damit ein DRM-System, wie folgt, aufgebaut werden:

Beispiel 5.5.1:

[6] Im Fall der elektronischen Wahlurne könnte jede Partei eine TTP stellen.

1. Ein Dateisystem erhält eine Anfrage eines Programms in der es aufgefordert wird eine bestimmte Datei zu übermitteln.
2. Das Dateisystem wertet das Zertifikat dieser Datei aus, liest die Kriterien ein und führt eine entfernte Attestierung der Anwendung aus, unter den gegebenen Kriterien.
3. Erfüllt die Anwendung alle Kriterien, dann wird die Attestierung erfolgreich durchgeführt. In diesem Fall kann die Datei übermittelt werden. Sonst wird der Zugriff verweigert.

◇

6. Umsetzung und Beispiele

Die Referenzimplementierung ist vollständig in C++ entstanden. Um den objektorientierten (OO) Ansatz durchgängig einhalten zu können, ist dadurch eine API entstanden, die das L4Env auf ein OO-Modell abbildet. Diese Arbeiten sind in dem Paket `l4cxx` zusammengefasst.

Die Implementierung folgt dem Prinzip des *test driven programming*. Bei diesem Ansatz werden die Tests zuerst geschrieben, dann erst folgt die Implementierung der Klasse bzw. der Methode. Die Tests wurden durch *unit tests* realisiert. Daraus entstand die Notwendigkeit, eine *unit-test*-Umgebung für L4 zu besitzen. Diese wurde durch die Portierung von `cppunit` [cpp] erzeugt. Zudem wurde die Entwicklungsumgebung um die Rolle[1] für Units-Test erweitert. Alle *unit-tests* sind im Verzeichnis `utests` der Applikationen abgelegt. Diese lassen sich mit dem Befehl `make runux` ausführen.

Die vorhandenen Möglichkeiten *regression-tests* auszuführen, habe ich nicht verwendet. Solche Tests sind eher für *black-box-testing* vorgesehen, wobei bei *unit-tests* eher das *white-box-testing* im Vordergrund steht, obwohl dieses auch *black-box- testing* erlauben.

Meine Anwendungen sind als unabhängige Applikationen entstanden, die nicht im L4-Baum unter dem Verzeichnis `pkg` eingebunden sind. Meine Umgebung, die bereits zu früheren Arbeiten entstanden ist, kann ganze CDs samt zusätzlicher Dateisysteme (`init-rd`, `rootfs`) erstellen. Deshalb habe ich ihr den Vorzug gegeben.

6.1. Bibiliotheken und Anwendungen

6.1.1. L4 C++ API

Die L4 C++ API (`l4cxx`) ist eine Menge von Adapterklassen und Algorithmen, die L4Env mit einer C++-Abstraktion versieht. Während der Implementierung habe ich sehr hohen Wert auf Portabilität gesetzt. Die `l4cxx`-Bibliothek kann bereits jetzt ansatzweise für Linux/POSIX als auch für L4Env umgesetzt werden. Die Speicherverwaltung in der API wird ausschließlich durch Allokatoren umgesetzt, womit sich diese Bibliothek auch für den Einsatz in eingebetteten Systemen eignen sollte, die über eine nicht POSIX-konforme Speicherverwaltung verfügen.

Die L4 C++ API besitzt momentan folgende Module:

uclibc Backend für die Anwendung unter L4Env

glibc Backend für die Anwendung unter Linux/POSIX

sys System-Modul, mit Mutex, Semaphoren und Prozessmanagement

util Namensdienste, einfacher Zufallszahlengenerator, Hex- und Base64-Codierung.

[1] Die L4Env-Entwicklungsumgebung definiert unterschiedliche Rollen, wie `prog.mk` oder `lib.mk`, je nachdem ob eine Anwendung oder eine Bibliothek gebaut werden soll. Die vollständige Liste aller Rollen ist in der Anleitung *BID Specification* unter [tud] gegeben.

crypto Kryptographische Algorithmen (RSA, SHA, MD5, Hmac), starker Zufallszahlengenerator[Kru04]

6.1.2. BP-Monitor

BP-Monitor steht für *boot protocol monitor* (engl. für Ladeprotokoll-Monitor). Er besitzt die Aufgaben, das Ladeprotokoll zu führen, die Anwendungen eindeutig zu benennen und den Zustand des Systems zu protokollieren. Dies bedeutet insbesondere, dass dieser Monitor weiss, welche Anwendungen gegenwärtig auf einem System ausgeführt werden bzw. beendet wurden. Diese Anwendung bildet die Instanz $\mathcal{T}$ aus Annahme 5.3.1.

Das wichtigste Element in diesem Paket ist die Definition eines Ladeprotokoll-Eintrags, das wie folgt definiert wurde:

```
typedef struct {
      char * name;            // Optionaler Name diese Eintrags,
      unsigned int nameSize;  // und seine Länge, ohne '\0'.
      char * hash;            // Prüfsumme über den Programmtext;
      unsigned int  hashSize; // und ihre Länge, ohne '\0'.
      l4_threadid_t threadid; // Prozessnummer der neuen Anwendung
      int state;              // Zustand: Aktiv, Beendet
      int type;               // Typ: Ausführbar, Daten, ...
      int pcrIndex;           // Angewendeter PCR-Index
      int bpIndex;            // eindeutige Indentifizierung
      int parentBpIndex;      // Identifizierung des Vater-Prozesses
} BpEntry_t;
```

Bis auf das Feld `state`, bringen alle weiteren Felder eine wesentliche Eigenschaft zum Ausdruck. Die Prüfsumme über alle diese wesentlichen Eigenschaften dient als Eingabe für die Operation `Extend`. Für weitere Informationen sei insbesondere auf die Abschnitte 5.2.1.5, 5.3.2 und 5.3.3 verwiesen.

Es muss noch erwähnt werden, dass es nicht Aufgabe dieser Arbeit war, sicheres oder überprüfbares Urladen umzusetzen. Doch eben sicheres bzw. überprüfbares Urladen und die damit notwendige Anbindung an das TPM-Gerät ist eine notwendige Voraussetzung für RA, die parallel zu dieser Arbeit, durch Mitwirken Dritter enstehen sollte. Dies ist leider nicht im ausreichenden Umfang geschehen. Die Auswirkungen auf diese Arbeit halten sich jedoch in Grenzen, da die fehlende Funktionalität durch intelligente Mocks ersetzt wurde. Sobald überprüfbares Urladen in L4Env zur Verfügung steht, lässt sich diese Funktionalität sehr einfach in die Referenzimplementierung übernehmen.

6.1.3. L4Semantics

L4Semantics ist für das Einlesen, Verwalten, Schreiben und Rechnen von und mit den Objektklassen und Objektbeschreibungen verantwortlich. Der Parser ist von der eigentlichen Bibliothek entkoppelt, was den Einsatz dieser Bibliothek mit einer anderen Beschreibung (z.B. XML) erlaubt. Momentan wird ein selbstgeschriebener Parser eingesetzt, der mit Hilfe von *yacc* geschrieben wurde.

Die Bibliothek bietet selbst keine Interpretierungsfunktionen an, doch sie verfügt über einfache Algorithmen, die das Rechnen mit Eigenschaften erleichtern (z.B. das Bilden von Zonen, Setzen von Referenzen, Suchen in der Datenbasis).

Einige Aspekte müssen in dieser Komponente noch zufriedenstellender umgesetzt werden: Zum einem soll ein Sicherheitsmechanismus eingeführt werden, der Manipulationen an der Datenbasis, die insbesondere aus dem Einlesen einer Eingabe durch den Parser entstehen, überwacht. In dieser Hinsicht ist bereits die Klasse `L4SGuard` entstanden. Dieses Konzept ist jedoch nicht durchgängig umgesetzt.

Zudem soll die Datenbasis um *snapshots* ergänzt werden, denn jede entfernte Attestierung sollte nicht mit der globalen Datenbasis, sondern mit einer eigenen Kopie arbeiten, um eine bessere Isolation und Konsistenz zu erreichen. Das TAM arbeitet bereits unter der Annahme, dass solche *snapshots* vorhanden sind, doch *L4Semantics* gibt momentan jedes Mal eine Referenz auf die globale Datenbasis zurück. Der Datenbasis fehlt zudem noch das Konzept der Transaktionen.

6.1.4. TAM

Der *Trusted Attestation Monitor* (TAM) bildet die Hauptkomponente für RA. Es sind zwei Attestierungsmodule umgesetzt. Das erste erlaubt einfaches Attestieren durch das Ladeprotokoll. Dieses muss insbesondere an das überprüfbare Urladen angebunden werden. Das zweite führt das Attestieren von Eigenschaften durch. Die Wahl, welches Protokoll eingesetzt wird, erfolgt durch den Programmierer. Werden unterschiedliche Protokolle angeboten, kann eine zusätzliche Auswahl zur Laufzeit erfolgen. Die Vermittlungsschicht prüft, ob das gewählte Protokoll angeboten wird und stellt die Verbindung her.

Das TAM stellt sowohl dem Herausforderer als auch der Anwendung Bibliotheken bereit, die RA vereinfachen sollen. In beiden Fällen hat der Programmierer die Möglichkeit, das entfernte Attestieren als schwarze Box zu betrachten, oder auch in das Protokoll, einzugreifen[2] und genauere Kenntnis über die übertragene Informationen zu erhalten.

Zu Beginn startet das TAM alle angebotenen RA-Protokolle als eigenständige Schnittstellen. Die Schnittstelle für das Attestieren von Eigenschaften führt folgende Initialisierung durch: Zuerst wird ein asymmetrisches Schlüsselpaar und das SAS-Objekt erstellt. Anschließend wird das Ladeprotokoll ausgewertet und alle bekannten Objekte angelegt. Damit werden der Kern und alle Anwendungen, die vor dem TAM gestartet wurden erfasst. Anschließend steht das TAM bereit um Atteste zu erstellen, oder neue Objektbeschreibungen aufzunehmen.

Insbesonders der TAM besitzt eine komplexe Software-Architektur, die mit UML-Diagrammen modelliert wurden. Diese Diagramme sind in den Entwickler-Dokumentationen hinterlegt.

6.2. Anwendungsbeispiel

In diesem Abschnitt wird das gesamte Protokoll, vom sicheren Urladen bis zum Auswerten des Attests exemplarisch vorgestellt. Die nachfolgenden Graphiken sind aus der Referenzimplementierung heraus entstanden. Abbildung 6.1 zeigt die Notation der nachfolgenden Abbildungen. Objektklassen werden in Rechtecke, Signaturen in Parallelogramme und Objektbeschreibungen in Ellipsen dargestellt.

[2] Für den Schutz der Privatsphäre könnte die Anwendung ein Attest einsehen und es zurückhalten und somit das Attest verweigern.

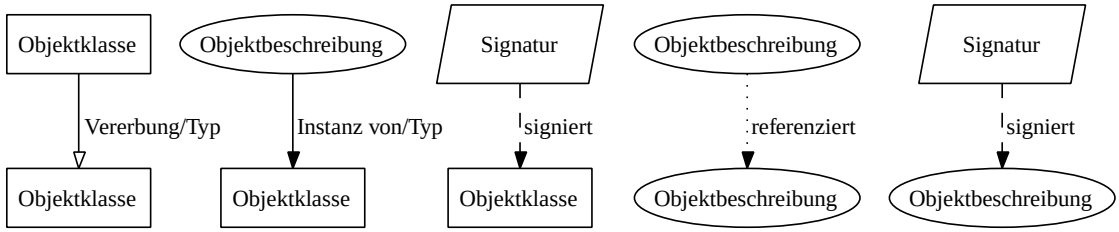

Abbildung 6.1.: Notation.

Index	Vater Index	Prüfsumme	Programm	Prozessnummer
6		E6AF23...	roottask	4
7	6	A484FE...	TAM	5
8	6	047E12...	wahlagent	9
9	8	67EFAF...	WahlListeDresden	

Tabelle 6.1.: Auszug aus dem Ladeprotokoll.

6.2.1. Ladereihenfolge und Ladeprotokoll

Die Ladereihenfolge besteht aus `roottask`, `TAM` und `WahlAgent`. Hierbei entsteht das Ladeprotokoll, dass in Tabelle 6.1 abgebildet ist. Das Ladeprotokoll wird von System sicher geführt. Aus diesem Ladeprotokoll geht hervor, welche Anwendung den Ladeprotokoll-Eintrag getätigt hat (Spalte *Vater Index*) und welche Prüfsumme jeder Eintrag besitzt. Wird eine Anwendungen gestartet, wird auch ihre Prozessnummern festgehalten. Der Index ist sowohl eine eindeutige Identifizierung eines Eintrags (und einer Anwendung), drückt aber auch einen Zeitpunkt aus.

6.2.2. Neue Anwendungen starten und registrieren

Angenommen das System befindet sich in Zeitpunkt sieben. Die `roottask` startet das Programm `wahlagent` nach dem Prinzip des sicheren Urladens:

Ablauf 6.2.1: *Sichere Urladen von Anwendung* `wahlagent`.

1. Die `roottask` bildet die Prüfsumme (047E12...) über den Programmtext der Anwendung `wahlagent`. Die `roottask` ermittelt die Prozessnummer (9) für die neue Anwendung.

2. Die `roottask` vermerkt diese Informationen in das Ladeprotokoll. Eintrag 8 des Ladeprotokolls entsteht.

3. Die `roottask` führt die Anwendung `wahlagent` aus.

Der TAM besitzt noch keine Informationen über die neue Anwendung. Zu diesem Zeitpunkt besitzt er lediglich eine initiale Datenbasis, die nur aus dem Schema besteht. Abbildung 6.2 zeigt diese Datenbasis. Diese Datenbasis muss im nächsten Schritt vervollständigt werden. Dieser Vorgang wird anhand der Anwendung `wahlagent` gezeigt und in Abbildung 6.3 verdeutlicht. Er erfolgt in vier Schritten, die in den nächsten vier Abläufen erläutert werden.

Ablauf 6.2.2: *Schritt 1: Laden der Objektklassen und Signaturen.*

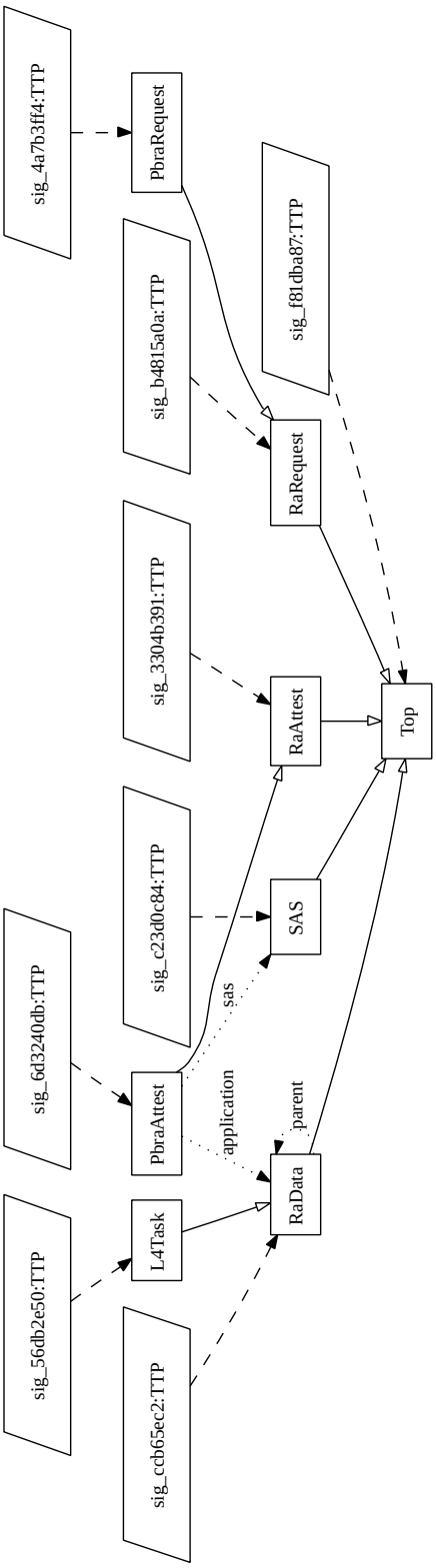

Abbildung 6.2.: Initialer Zustand. Dieser ist der initiale Kontext für die Datenbasis. Es wird vom Schema bestimmt. Herausforderer und Dienstleister besitzen den selben initialen Zustand. Die Abbildung zeigt die grundlegenden Objektklassen (Top, L4Task etc.), ihre Klassenhierarchie, die Signaturen dieser Objektklassen (in Parallelogrammen) und die Referenzen zwischen den Klassen (gepunktete, benannte Pfeile). Dieser initiale Kontext enthält noch keine Objektbeschreibungen.

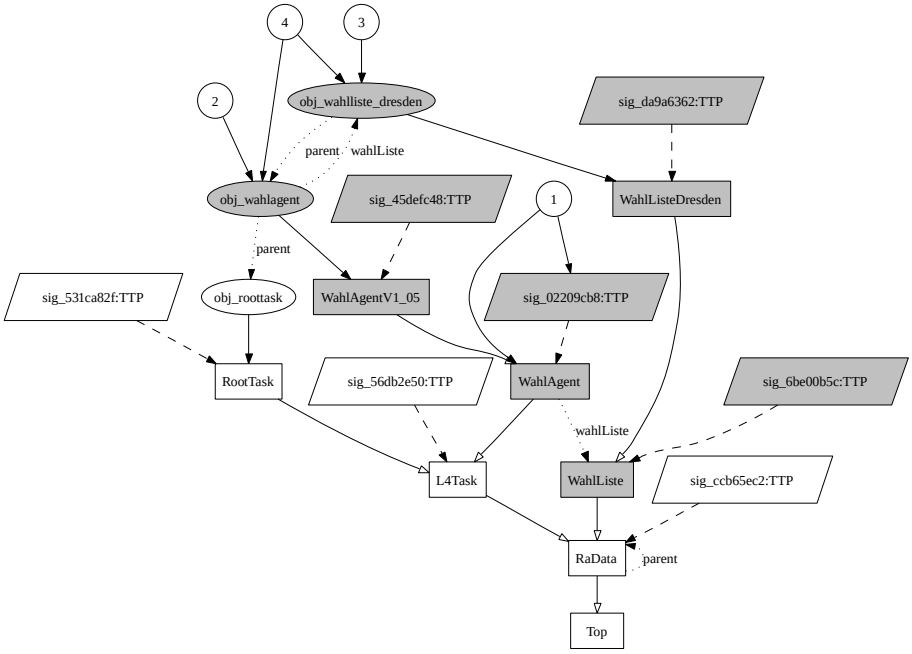

Abbildung 6.3.: Anwendungen registrieren und Referenzen setzen. Auszug aus dem Kontext des TAM. Die grauen Objekte wurden auf Anweisung der Anwendung `wahlagent` vom TAM angelegt.

1. Der Wahlagent führt die Operation `IpbRa1_1::loadObjectClass` für die Objektklassen `WahlAgent`, `WahlAgentV1_05`, `WahlListe` und `WahlListeDresden` aus.

2. Der Wahlagent fügt alle ihm bekannten Signaturen dieser Klassen dem TAM hinzu.

Die Signaturen sind von einer TTP (durch den Anhang „:TTP“ zu erkennen) erstellt worden. Diese Klassen stellen eine Erweiterung des Schemas da. Dass diese Erweiterung vertrauenswürdig ist, wird durch die Signaturen belegt. Der TAM muss an dieser Stellen nicht zwingend prüfen, ob die Signaturen gültig oder vertrauenswürdig sind, da dies beim Herausforderer geschieht. Im nächsten Schritt (Schritt zwei) registriert sich die Anwendung `wahlagent` beim TAM:

Ablauf 6.2.3: *Schritt 2: Registrieren der Anwendung* `wahlagent`.

1. Programm `wahlagent` registriert sich beim TAM, indem es die Operation `IpbRa1_1::registerObject` ausführt.

2. Daraufhin ermittelt der TAM den Index und die Prüfsumme anhand der Prozessnummer von `wahlagent` aus dem Ladeprotokoll.

3. Der TAM prüft anhand des Index, ob nicht bereits eine Instanz für diese Anwendung vorhanden ist.

4. Der TAM ermittelt anhand der Prüfsumme den Typ `WahlAgentV1_05` für die neue Objektbeschreibung. Die Prüfsumme ist in der Objektklasse `WahlAgentV1_05`, in Eigenschaft `payloadhash` hinterlegt.

5. Der TAM erstellt eine neue Instanz mit dem Namen `obj_wahlagent` vom Typ `WahlAgentV1_05`.Der Vaterprozess (`roottask`) wird durch das Setzen der Referenz

`parent` vermerkt, sowie die Identität (der Index aus dem Ladeprotokoll) der Anwendung.

Im dritten Schritt muss der Wahlagent eine Wahlliste beim TAM vermerken. Dazu muss diese Liste in der Datenbasis registriert werden:

Ablauf 6.2.4: *Schritt 3: Wahlliste registrieren.*

1. Das Programm `wahlagent` ermittelt die Prüfsumme der eingesetzten Liste.
2. Der Wahlagent erstellt einen Eintrag für die eingesetzte Wahlliste im Ladeprotokoll. Es ensteht der Eintrag 9. Dort wird auch das Programm `wahlagent` als Vater des neuen Eintrags festgehalten. Der neue Index (Index 9) ist die Identität der Wahlliste.
3. Das Programm `wahlagent` registriert eine neue Instanz der Wahlliste mit dem TAM. Dies erfolgt analog zu Ablauf 6.2.3. Da der Eintrag für die Wahlliste keine Prozessnummer besitzt, muss ihre Identität (Index 9) übermittelt werden.
4. Der TAM legt das Objekt `obj_wahlliste_dresden` vom Typ `WahlListeDresden` an. Als Vater (Eigenschaft `parent`) wird die Wahlurne vermerkt.

Im vierten Schritt teilt `wahlagent` dem TAM mit, dass er die Wahlliste einsetzt:

Ablauf 6.2.5: *Schritt 4: Wahlliste referenzieren.*

1. Der `wahlagent` führt die Operation `IpbRa1_1::assignReference` aus, übermittelt die Identität (Index 9) auf die er sich bezieht und die Referenz die er setzen möchte (`wahlListe`).
2. Der TAM prüft, ab das Programm `wahlagent` berechtigt ist, das Objekt `obj_wahlagent` zu manipulieren.
3. Der TAM ermittelt die Objektbeschreibung `obj_wahlliste_dresden` und setzt die Referenz `wahlListe` auf dieses Objekt.

Zu diesem Zeitpunkt ist die Initialisierung beendet. Der TAM kann nun ein Attest für den Wahlagenten ausstellen.

6.2.3. Das Attest erstellen und auswerten

Der Herausforderer führt folgenden Befehle aus:

```
 1 /* Verbindung herstellen und Kontext anlegen  */
 2 PbraClient *client = PbraClient::create(...);
 3 PBRA::AutoCtx reqCtx(PBRA::createPbraCtx());
 4 PbraRequest request(*reqCtx); // Anfrage erstellen

 6 /* Attest anfordern */
 7 Attest *attest = client->requestAttest(request);

 9 /* Attest Prüfen    */
10 PbraVerifier verifier;
11 verifier.getRootCA().addCA(ttpRsaCA); // TTP setzen
```

```
13 if (verifier.verifyAttest(*attest, request)) {
14  // Erfolg!
15 } else {
16  // Fehler!
17 }
```

In den ersten Zeilen wird eine Verbindung zum Wahlagenten hergestellt und der Kontext initialisiert. Der Kontext besteht zu diesem Zeitpunkt nur aus dem Schema (vgl. Abbildung 6.2). In Zeile 4 wird eine Anfrage erstellt. Zeile 7 fordert das Attest an. In Zeile 11 bestimmt der Herausforderer, welchen TTP er traut. Zeile 13 wertet die Antwort aus. Diese Antwort wird in Abbildung 6.4 gezeigt.

Anhand von Abbildung 6.4 soll das Prüfen des Attest durchgeführt werden. Im wesentlichen sind die fünf Tests auszuführen, die auch in Abbildung 3.2 (Seite 24) gezeigt werden.

Sitzung prüfen. Das Objekt `attest` besitzt eine Prüfsumme über die Anfrage (Objekt `request`). Die Prüfsumme über die Anfrage und die vermerkte Prüfsumme werden auf Gleichheit geprüft.

SAS prüfen. Die Objektbeschreibung `sas` besitzt das Ladeprotokoll, das Quote und die öffentlichen Schlüssel. Diese Schlüssel sind durch die TTP signiert. Bevor die TTP (die TTP kann ein MK-Schlüssel sein) einen solchen Schlüssel signiert, stellt sie sicher, dass dieser Schlüssel einem vertrauenswürdigem SAS zugeordnet ist. Das SAS selbst wird durch das Ladeprotokoll geprüft.

Signatur des Attests prüfen. Das Attest besitzt die Signatur *sig_attest:SAS*, die durch das SAS ausgestellt wurde. Die Prüfsumme die dieser Signatur zugrunde liegt, wird über das Attest (Objekt `attest`), seinem Typ und all seinen Referenzen gebildet. Somit sind alle Pfade vom Objekt `attest` bis zur Objektklasse `Top` abgesichert (vgl. Abschnitt 4.3.8). Der Schlüssel, der diese Signatur geleistet hat, muss einem vertrauenswürdigem SAS gehören. Dies wurde im vorherigen Schritt geprüft.

Anwendung Identifizieren In diesem Schritt geht es darum festzustellen, dass der Herausforderer mit einer Wahlurne kommuniziert. Dafür untersucht der Herausforderer den Typ von der Objektbeschreibung `obj_wahlagent`. Es muss gelten:

$$\text{Typ}(\texttt{obj\_wahlagent}) \subseteq \texttt{WahlAgent}$$

Dies ist der Fall. Im nächsten Schritt prüft der Herausforderer ob die Wahlurne auch die korrekte Wahlliste einsetzt. Da das Objekt `obj_wahlliste_dresden` vom Typ `WahlListeDresden` ist, ist auch diese Forderung erfüllt.

Der Herausforderer kann an dieser Stelle seine Prüfung beliebig ausweiten.

Systemzustand prüfen. In diesem Schritt soll ermittelt werden, ob das System in einem sicheren Zustand ist. Dies wurde in Ansätzen schon bei der Prüfung des SAS durchgeführt. In diesem Schritt müssen noch die Anwendungen geprüft werden, die nicht im System-Ladeprotokoll geführt sind. Für dieses Beispiel ist das nur die Anwendung `wahlagent`.

Es ist die Zone von `wahlagent` zu messen. Sie besteht aus den Objekten `obj_wahlagent`, `obj_roottask` und `obj_wahlliste_dresden`.

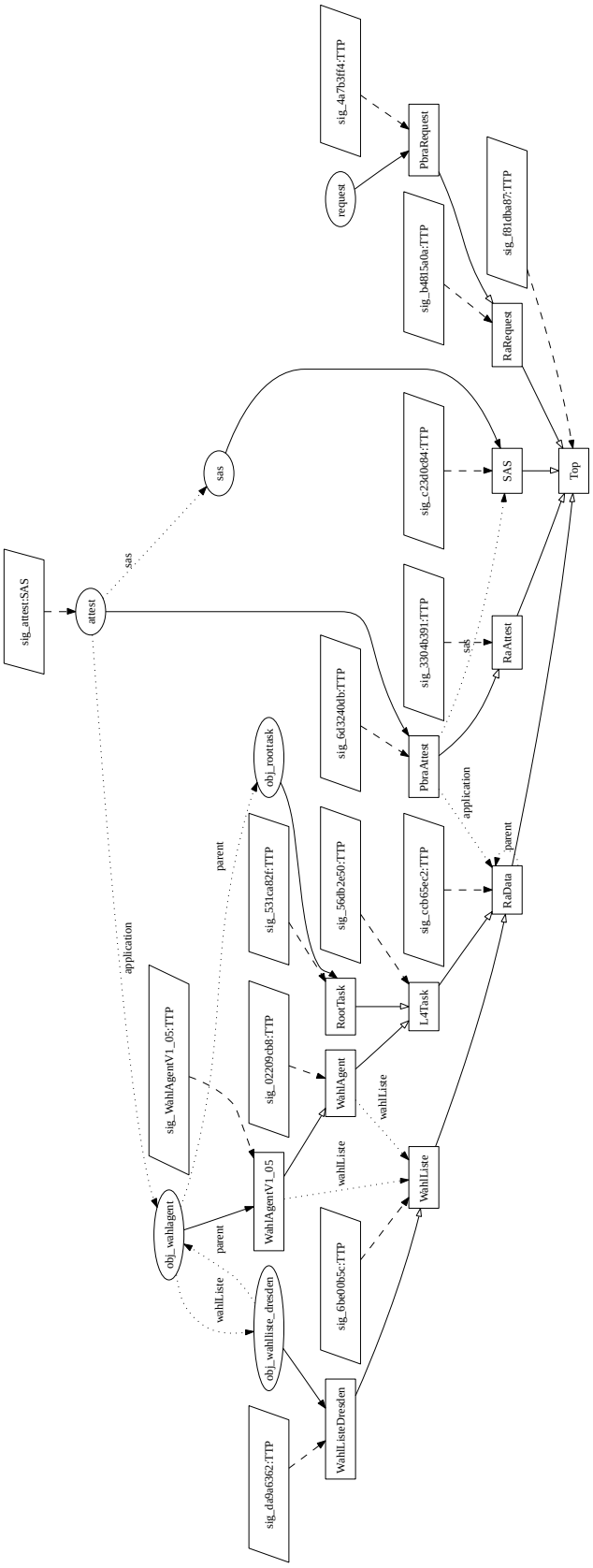

Abbildung 6.4.: Das Attest.

`obj_wahlagent`: Eine Anwendung ist genau dann sicher (Korollar 5.2.8), wenn (a) ihr Vaterprozess sicher ist, wenn (b) die Objektbeschreibung der Objektklasse folgt und wenn (c) die Objektklasse sicher ist (Definition 5.2.5).

Der Vaterprozess wird in der Referenz `parent` festgehalten. Somit sind alle Vaterprozesse in der Zone enthalten:

$$\text{Menge der Vaterprozesse einer Zone } \mathcal{Z}_i \subset \mathcal{Z}_i$$

Sind alle Prozesse der Zone sicher, so sind auch alle Vaterprozesse sicher, womit die Vaterprozesse nicht gesondert behandelt werden müssen. Punkt (b) ist technische bedingt. Die Modifikatoren (vgl. Definition 4.3.1) dienen dieser Prüfung.

Zuletzt muss noch die Objektklasse `WahlAgentV1_05` geprüft werden (Punkt (c)). Diese Objektklasse besitzt eine Signatur `sig_WahlAgentV1_05:TTP`, die durch die TTP ausgestellt wurde. Somit ist diese Objektklasse vertrauenswürdig. Die Basis-typen von `WahlAgentV1_05` (also die Klassen `Top` bis `WahlAgentV1_05`) müssen nicht gesondert geprüft werden, da die Prüfsumme über `WahlAgentV1_05` diese Klassen mit einbezieht. Somit sichert die Signatur `sig_WahlAgentV1_05:TTP` den gesamten Pfad, von `WahlAgentV1_05` bis zu `Top` ab.

`obj_wahllist_dresden`: Dieses Objekt ist analog zu Objekt `obj_wahlagent` zu prüfen.

`obj_roottask`: Auch dieses Objekt ist wie `obj_wahlagent` zu prüfen, mit einer Ausnahme: für dieses Objekt ist kein Vaterprozess gesetzt. Somit muss die `obj_roottask` in der SAS enthalten sein (vgl. Korollar 5.2.9). Somit ist zu prüfen, ob die Prüfsumme von der Anwendung `roottask` im Ladeprotokoll in der Objektbeschreibung `sas` hinterlegt ist.

Die Sicherheit der Referenzen wird durch das Prinzip des sicheren Urladens begründet: Wird die Anwendung einer bestimmten Prüfsumme als *sicher* angesehen, so kann man ihr trauen, die Referenzen korrekt gesetzt zu haben.

Damit ist das entfernte Attestieren von Eigenschaften gezeigt.

7. Auswertung und Ausblick

7.1. Kurzer Überblick

Die folgende Auflistung fasst die Forderungen aus der Liste 5.1.1 zusammen und erläutert inwiefern diese Anforderungen erfüllt werden konnten:

/Req 1/ RA durch das Ladeprotokoll. Teilweise erfüllt. Siehe Abschnitt 6.1.2

/Req 2/ RA von Eigenschaften. Erfüllt.

/Req 3/ Zurückziehen der Sicherheitsaussage. Diese Anforderung wird durch den Einsatz von CRL-Listen erlaubt. Obwohl die Referenzimplemetierung keine CRL-Listen anwendet, wurde in Abschnitt 5.3.6 gezeigt, wie sich diese Listen effizient in das Attestieren von Eigenschaften einbinden lassen.

/Req 8/ Sicherheit. Die Sicherheitsbetrachtungen wurden im gesamten Verlauf dieser Arbeit detailliert und sorgfältig durchgeführt. Die Sicherheit gründet letztendlich noch immer auf die Annahme, dass ein TPM-Gerät sicher ist. Inwiefern der Code sicher ist, muss sein Einsatz beweisen.

/Req 9/ Mit Nizza vereinbar sein. Erfüllt. Insbesondere Korollar 5.2.8 und das Messen der TCB durch den Zonen (Korollar 5.2.3) sind auf die Anforderungen von Nizza gut angepasst.

/Req 10/ Messen der TCB. Erfüllt. Siehe Abschnitt 5.2.1.2.

Alle weiteren Anforderungen werden in den nächsten Abschnitten behandelt.

7.2. Zu Datenschutz, Anonymität und Verkettbarkeit

Anforderungen: /Req 4/ und /Req 5/.

Die Arbeit hat gezeigt, dass sich entferntes Attestieren unterschiedlich durchführen lässt. Während eine elektronische Wahlurne möglichst transparent gestaltet werden soll, sollten die Kunden einer Online-Musikplattform ihre personenbezogenen Daten möglichst schützen lassen.

In Bezug auf den Datenschutz wurden folgende Ansätze diskutiert, die in Teilen ergänzend wirken.

Messen von Generalisierungen. Dieses Verfahren wurde in Abschnitt 5.5 vorgestellt und nur theoretisch betrachtet. Es erlaubt gezielte Rollen, die eine Anwendung einnimmt, zu messen. Es besitzt einige Probleme insbesondere, wenn mehrere TTP eingesetzt werden. Doch es bietet eine Möglichkeit, den Informationsfluss einzuschränken.

Einsatz einer RAQL-Sprache. Die RAQL-Sprache wurde ebenfalls nur theoretisch betrachtet. Sie bietet jedoch einen sehr guten Ansatz für den Schutz der Daten, denn die Antwort dieser Anfrage ist boolesch. Dennoch besteht weiterhin eine Verkettbarkeit durch die Anfrage an sich: Ist diese Anfrage detailliert genug formuliert, können auch weiterhin Informationen übermittelt werden, die nicht für das Attestieren notwendig sind. Bis auf diese Verkettbarkeit entspricht der Einsatz der RAQL dem Prinzip des optimalen RA (Definition 3.3.1). Siehe Abschnitt 4.4.3.

Der Einsatz der Monitore. Durch die Monitore können die Anwender den Informationsfluss effektiv überwachen und gegebenenfalls verhindern. Siehe Abschnitt 5.2.3.

Der Einsatz einer TTP. Die TTP ist die einzige effektive Methode, das Ladeprotokoll zu verbergen. Der Einsatz einer TTP ist ein sehr mächtiges Werkzeug. So mächtig, dass er Anonymität auch ohne den Einsatz von PBRA erlaubt (vgl. Algorithmus 3.5.2).

Gruppensignaturen. Die Gruppensignaturen (speziell DAA) erlauben den Einsatz von digitalen Signaturen und bewahren dabei die Anonymität der Anwender. Siehe Abschnitt 2.5.4 *ff.*

Damit stehen fünf effektive Methoden zur Verfügung, den Informationsfluss zu kontrollieren. Weitere Betrachtungen sind von den Anforderungen an den Datenschutz und an das System abhängig.

7.3. Ausgrenzung verhindern

Anforderungen: /Req 6/.

Das Thema Ausgrenzung wurde in dieser Arbeit noch nicht genauer angesprochen. Für diese Diskussion muss das Fallbeispiel geändert werden. Sei der Anwender ein Konsument, der bei einem Dienstleister ein Musikstück erwerben möchte. Der Dienstleister kann den Anwender anhand seiner Identität oder seiner Programmwahl ausgrenzen. In diesen beiden Fällen können die Methoden aus Abschnitt 7.2 eingesetzt werden, wobei der Anbieter immer noch selbst entscheiden kann, ob diese Methoden eingesetzt werden dürfen.

Die zweite Variante bezieht sich auf die Kontrolle über die TTP: Der Dienstleister bestimmt die TTP. Wendet er keine TTP an, die bereit ist, den Anwender als sicher auszuweisen, oder weigert sich der Anwender eine bestimmte TTP zu verwenden, so ist dadurch eine Ausgrenzung entstanden.

Die TTP ist aus RA nicht wegzudenken. Das TPM-Gerät selbst wird nur dann als sicher erkannt, wenn es einen EK-Schlüssel besitzt, der durch einen MK-Schlüssel signiert wurde. Die MK-Schlüssel bilden eine TTP (vgl. Abschnitt 3.4) und somit sehe ich keinen Ansatz, eine mögliche Ausgrenzung grundsätzlich zu verhindern.

7.4. Modular gestaltet, geringe Komplexität

Anforderungen: /Req 7/

Wie klein ist klein genug? Mikrokern-Systeme zeichnen sich unter anderem dadurch aus, dass sie einen modularen Aufbau besitzen. Aufgrund dieser Modularität lässt sich eine kleine TCB erreichen, die mitunter in der Anzahl der Code-Zeilen gemessen werden

kann. Doch alleine eine Zahl, z.B. 15.000 Zeilen Code, enthält an sich keine Aussage über die Sicherheit einer Anwendung.

Auch der Vergleich zwischen monolithischen Systemen und Mikrokern-Systemen ist meistens nicht gerecht, denn sobald das Mikrokern-System denselben Funktionsumfang anbietet, werden beide Systeme eine ähnliche Komplexität aufweisen.

Es ist die modulare Gestaltung eines Mikrokern-Systems, die es erlaubt, auf effiziente Art, überflüssige Funktionalität wegzulassen und die übrig gebliebenen Anwendungen voneinander isoliert in eigene Adressräume zu betreiben. Doch auch moderne Compiler, Programmiersprachen, Test- und Entwicklungsmethoden haben in den letzten Jahren einen großen Beitrag dafür geleistet, Software sicherer zu machen, und ihr Einsatz ist nicht selbstverständlich. Eben diese Anstrengungen geraten meistens in den Hintergrund, wenn die Frage nach der Anzahl der Codezeilen aufkommt.

Das Auswerten von Eigenschaften ist eine komplexe Anwendung (Paket `l4semantics`). Sie umfasst ca. 3.000 Zeilen Code (einschließlich des Parsers). Hinzu kommen ca. 1.000 Zeilen reiner Test-Code. Die `l4cxx`-API umfasst ca 4.000 Zeilen Code, wobei sowohl POSIX- als auch L4v2-Code mit eingerechnet sind. Dazu kommen ca. 1.700 Zeilen Test-Code in über 100 automatisierte Tests. TAM besitzt weitere 5.000 bzw. 1.600 Zeilen Code. Davon sind über 2.000 Zeilen alleine für PBRA zuständig und weitere 1.000 für die asymmetrische Kryptographie. Alle Angaben wurden mit Hilfe der Anwendung `sloccount` [Whe] ermittelt.

Die Stärken des Codes liegen in der Modularität und in den automatisierten Tests: PBRA lässt sich an einer einzigen Stelle im Code vollständig aus dem TAM entfernen. Nahezu überall werden Schnittstellen (in C++, abstrakte Klassen) eingesetzt. Jedes Entwicklungstadium (Meilenstein) wurde mit einem Test (Akzeptanz-Test) versehen, der auch zu späteren Zeitpunkten der Entwicklung noch durchlaufen wird. Jede Klasse und fast alle Methoden werden getestet.

7.5. Nachteile

Das entwickelte System hat den Nachteil, dass es wesentlich komplexer ist, als bisher bekannte Verfahren für RA, insbesondere als das Attestieren mit Hilfe des Ladeprotokolls. Neben den umfangreichen Anwendungen, die für das entfernte Attestieren notwendig sind, ist auch das System sehr komplex. Solche Systeme sicher umzusetzen, ist eine Herausforderung.

Das System erfordert zudem eine strikte Disziplin des Programmierers, denn die Beschreibung der Eigenschaften muss immer mit dem Entwicklungsstand der Anwendung übereinstimmen. Zudem muss jede Anwendung kooperieren. Auch besitzt jede Anwendung Routinen für das entfernte Attestieren, was die Komplexität der Anwendung wieder erhöht.

7.6. Ausblick

Diese Arbeit weist viele Stellen auf, in denen weitere Forschungen angebracht sind, die hier nicht alle wiederholt werden sollen. Doch die Beschreibung von Eigenschaften ist ein Ansatz, der auch in anderen Problemen Anwendung finden könnte. L4Semantics könnte zu einer vollständigen objektorientierten Datenbasis ausgearbeitet und beispielsweise für Verzeichnisdienste eingesetzt werden.

Das L4Env und der TAM sollten in jedem Fall das Konzept für überprüfbares Urladen vollständig umsetzen. Auch die L4 C++ API bietet eine Möglichkeit, die Entwicklung der Mikrokern-Plattform voranzutreiben.

7.7. Nachwort

Während der Ausarbeitung meiner Diplomarbeit habe ich das Konzept der komponentenbasierten Systeme und *Test-Driven-Programming* vermisst. Das Programm `dice`, wenngleich sehr nützlich, erfüllt nicht die Kriterien eines klassischen Komponentensystems (nach [Aß03]). Weitergehende Konzepte wie *grey-box-composition* oder Programmieren mit Aspekten (engl. *aspect-programming*) kommen überhaupt nicht zum Einsatz, obwohl Mikrokern-Systeme die idealen Voraussetzungen dafür bieten.

Der Erfolg moderner Rechnersysteme gründet maßgeblich auf ihren universellen Einsatzmöglichkeiten. Das sichere Urladen beschränkt diesen Grundsatz und steht zu ihm im direkten Widerspruch. Dementsprechend wird die Technologie immer weiter entwickelt, neue Funktionen werden hinzugefügt und es wird viel getan um die Universalität der Systeme beizubehalten. Die Technologie wird zunehmend komplexer, und immer mehr Annahmen und Ausnahmen müssen berücksichtigt werden. In meinen Augen vermisse ich bei dem Einsatz der TPM-Geräte ein kompaktes, einfaches Modell. Sicherheit sollte auf einfache und konsistente, vorzugsweise formal spezifizierte Modelle beruhen. Die Spezifikation der TPM-Geräte hingegen umfasst mittlerweile mehrere hunderte Seiten und duzende Funktionen. Dies macht ihren Einsatz sehr kompliziert.

Doch im Grundsatz hat diese Arbeit gezeigt, dass entferntes Attestieren von Eigenschaften auf mikrokern-basierten Systemen auch unter Berücksichtigung des Datenschutzes möglich ist.

A. Syntax und Notation der Beschreibungssprache

A.1. Notation

Für die Notation wird die *Wirth Syntax Notation angewendet*, mit der Änderung, dass Kommentare in der C-Syntax hinzugefügt wurden. Literale werden in einfachen oder doppelten Anführungsstrichen geschrieben.

Alternative	$\|$	$a\|b$ bedeutet: Entweder a oder b
Wiederholung	$\{\cdots\}$	Wiederholungen (0 bis n Wiederholungen) werden in geschweiften Klammern dargestellt. $\{a\}$ steht für $\epsilon\|a\|aa\|aaa\|\cdots$.
Optional	$[\cdots]$	Die eckigen Klammern stehen für 0 oder 1 wiederholung: $[a] = \epsilon\|a$
Grupppierung	$(\cdots)$	Zum Beispiel: $(a\|b)c = ac\|bc$
$< text >$		Steht für eine beliebige Textfolge
ϵ		Steht für die leere Zeichenfolge

A.2. Syntax der Beschreibungssprache

```
START = { OBJECTCLASS } { OBJECT }
```

Objektklasse

```
OBJECTCLASS = "objectdef" KLASSE [ ":" BASE ] "{" { DECLARATION } "}" ";"
KLASSE      = name // Name der Objektklasse
BASE = NAME { NAME } // Liste der Rollen dieser Klasse

/* Die Deklaration definiert eine Eigenschaft. */
DECLARATION = MODIFIKATOR TYP ATTRNAME [ KARDINALITÄT ] [ "=" WERT ] ";"
MODIFIKATOR = [ "const" | "required" ]
TYP         = "integer" | "string" | REFERENZ
REFERENZ    = "reference" "<" KLASSE [ "," FARBE ] ">"
FARBE       = KLASSE     // Nicht implementiert!
KARDINALITÄT =   "[]"    // Beliebig viele Werte (0..*)
               | "[" integer (":" | "," | "-" ) integer "]"
ATTRNAME    = name       // Name der Eigenschaft
```

Objekt

```
OBJECT    = "object" KLASSE OBJNAME [ ":" VORLAGE ]
                "{" { ZUWEISUNG } "}" ";"
OBJNAME   = name           // Name dieses Objekts
```

```
VORLAGE   = OBJNAME        // Nicht implementiert!

ZUWEISUNG = ATTRNAME [ INDEX ] "=" WERT ";"
INDEX     = "[" integer "]"
WERT      = name | integer | string | base64
```

Terminale

```
digits     = "0" | "1" | ... | "9"
lower-case = "a" | "b" | ... | "z"
upper-case = "A" | "B" | ... | "Z"
integer    = digits { digits }
white      = " " | "\t" | "\n" | "\r"
alpha      = lower-case | upper-case
alphanum   = alpha | digits
name       = alpha { alphanum | "-" | "." | "_" | ":" }
base64     = { alpha | digits | "+" | "/" } { "=" }
string     = "'" <text> "'" | '"' <text> '"'
```

Literaturverzeichnis

[AFS97] Arbaugh, W. A., D. J. Farber und J. M. Smith: *A secure and reliable bootstrap architecture.* In: *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Seite 65, Washington, DC, USA, 1997. IEEE Computer Society. 8

[ant] *Palladium/TCPA and DRM – Big Brother Everywhere?* `http://www-crypto.htw-saarland.de/weber/topics/pd/index_e.html`. 1

[Aß03] Assmann, U.: *Invasive Software Composition.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. 74

[BCC04] Brickell, E., J. Camenisch und L. Chen: *Direct Anonymous Attestation*, 2004. 16

[BCM+03] Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi und Peter F. Patel-Schneider (Herausgeber): *The description logic handbook: theory, implementation, and applications.* Cambridge University Press, New York, NY, USA, 2003. 43

[Bra06] Brandt, Sebastian-Philipp: *Standard and non-standard reasoning in description logics.* Doktorarbeit, Technische Universität Dresden, 2006. 43

[BvHH+04] Bechhofer, Sean, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider und Lynn Andrea Stein: *OWL Web Ontology Language Reference*, 2004. W3C Recommendation 10 February 2004, `http://www.w3.org/TR/owl-ref/`. 44

[CH02] Camenisch, Jan und Els Van Herreweghen: *Design and implementation of the idemix anonymous credential system.* In: *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, Seiten 21–30, New York, NY, USA, 2002. ACM Press. 16

[Cha03] Chaum, David: *Zero-Knowledge Undeniable Signatures.* In: Biham, Eli (Herausgeber): *Advances in cryptology - EUROCRYPT 2003, proceedings of the internarional conference on the theory and application of cryptographic techniques*, Band 2656 der Reihe *Lecture Notes in Computer Science*, Seiten 614–629, Warsaw, Poland, Mai 2003. Springer-Verlag. 16

[cpp] *CppUnit.* `http://cppunit.sourceforge.net/`. 61

[DEJ01] D. Eastlake, 3rd und P. Jones: *US Secure Hash Algorithm 1 (SHA1)*, 2001. RFC3174. 5, 7

[DLNS96] Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi und Andrea Schaerf: *Reasoning in Description Logics.* In: Brewka, Gerhard (Herausgeber): *Foundation of Knowledge Representation*, Seiten 191–236. CSLI-Publications, 1996. 43

[elf95] *Tools Interface Standard (TIS) Executable Linking Format (ELF) Specification*, 1995. 54

[ems] *European Multilaterally Secure Computing Base (EMSCB).* `http://www.emscb.org/`. 19

[FFS87] Fiege, U., A. Fiat und A. Shamir: *Zero knowledge proofs of identity.* In: *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, Seiten 210–217, New York, NY, USA, 1987. ACM Press. 16

[GPC+03] Garfinkel, Tal, Ben Pfaff, Jim Chow, Mendel Rosenblum und Dan Boneh: *Terra: a virtual machine-based platform for trusted computing.* In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, Seiten 193–206, New York, NY, USA, 2003. ACM Press. 36

[Gro91] Gross, Michael: *Vertrauenswürdiges Booten als Grundlage authentischer Basissysteme.* In: *VIS '91: Verläßliche Informationssysteme, GI-Fachtagung*, Seiten 190–207, London, UK, 1991. Springer-Verlag. 8

[HF04] Haldar, Vivek und Michael Franz: *Symmetric behavior-based trust: a new paradigm for internet computing.* In: *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, Seiten 79–84, New York, NY, USA, 2004. ACM Press. 36

[HHF+05] Härtig, Hermann, Michael Hohmuth, Norman Feske, Christian Helmuth, Adam Lackorzynski, Frank Mehnert und Michael Peter: *The Nizza Secure-System Architecture.* In: *Proceedings of CollaborateCom*, 2005. `http://os.inf.tu-dresden.de/papers_ps/nizza.pdf`. 12, 13, 30, 45, 48

[HPFS02] Housley, R., W. Polk, W. Ford und D. Solo: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 2002. RFC3280. 14, 16

[JK03] Jonsson, J. und B. Kaliski: *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, 2003. RFC3447. 5, 14

[Kau04] Kauer, Bernhard: *Authenticated booting for L4*, 2004. `http://os.inf.tu-dresden.de/papers_ps/kauer-beleg.pdf`. 19, 31

[KBC97] Krawczyk, H., M. Bellare und R. Canetti: *HMAC: Keyed-Hashing for Message Authentication*, 1997. RFC2104. 5

[Kru04] Krueger, Hans Marcus: *Zufallszahlen unter L4/DROPS*, 2004. 57, 62

[Lie95] Liedtke, J.: *On micro-kernel construction.* In: *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, Seiten 237–250, New York, NY, USA, 1995. ACM Press. 12

[MVO96] Menezes, Alfred J., Scott A. Vanstone und Paul C. Van Oorschot: *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996. 13, 14

[ope] *Open Trusted Computing (OpenTC).* `http://www.opentc.net/`. 19

[Pfi00] Pfitzmann, Andreas: *Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme*, 2000. Skript zu den Vorlesungen Datensicherheit und Kryptographie. 14, 15

[PSHW04] Poritz, Jonathan, Matthias Schunter, Els Van Herreweghen und Michael Waidner: *Property Attestation — Scalable and Privacy-friendly Security Assessment of Peer Computers*, 2004. 19

[Sch95] Schneier, Bruce: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995. 13, 14, 22

[SS04] Sadeghi, Ahmad-Reza und Christian Stüble: *Property-based attestation for computing platforms: caring about properties, not mechanisms*. In: *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, Seiten 67–77, New York, NY, USA, 2004. ACM Press. 19, 31

[SW63] Shannon, Claude E. und Warren Weaver: *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1963. 33

[tcg] *Trusted Computing Group (TCG)*. `https://www.trustedcomputinggroup.org/`. 5, 17

[tis06] *TCG PC Client Specific TPM Interface Specisifaction (TIS)*, 2006. Version 1.2. 5

[tnc06] *Trusted Network Connect (TNC) Specifications*, 2006. Version 1.1, Revision 2. 17

[tpm06] *Trusted Platform Module (TPM) Specifications*, 2006. Version 1.2, Revision 94. 5, 6, 7

[tud] *Operating Systems Group at Technical university of Dresden*. `http://www.tudos.org/`. 12, 19, 61

[vDBA] Doorn, Leendert van, Gerco Ballintijn und William A. Arbaugh: *Signed Executables for Linux*. 54

[Whe] Wheeler, David. A.: *sloccount*. `www.dwheeler.com/sloccount`. 73

[wik] *Wikipedia*. `http://www.wikipedia.org/`. 37, 81, 82

[You95a] Youngdale, Eric: *Kernel Korner: The ELF Object File Format by Dissection*. Linux J., 1995(13es):15, 1995. 54

[You95b] Youngdale, Eric: *Kernel Korner: The ELF Object File Format: Introduction*. Linux J., 1995(12es):7, 1995. 54

Glossar

Anti-Tamper Das Wort Anti-Tamper leitet sich aus dem englischen Verb *to tamper*, das soviel wie *unerlaubtes manipulieren* bedeutet. Es bezeichnet Verfahren, die sowohl in Software als auch in Hardware umgesetzt werden, um eine unerlaubte Manipulation eines Systems zu verhindern.

API *Application Programming Interface*

Authentikationssystem Ein Authentikationssystem kann anhand eines *message authentication code* (MAC) ermitteln, ob eine gegebene Nachricht n authentisch ist, bzw. während der Übertragung nicht verändert wurde.

BIOS *Basic Input/Output System* Das *Basic Input/Output System* ist das erste Programm, das IBM-kompatible System ausführen. Seine Aufgabe ist die Plattform soweit vorzubereiten, dass das Betriebssystem gestartet werden kann.

black-box-testing Siehe white-box-testing

Bootloader Siehe Bootsektor

Bootsektor Der Bootsektor ist ein besonderer Bereich einer Festplatte, Diskette oder CD-ROM, auf dem die Startsequenz eines Betriebssystems abgelegt wird. Das BIOS sucht diese Boot-Sektoren, führt die Startsequenz aus und übergibt damit die Kontrolle über den Rechner an das Betriebssystem. Sind auf einem Rechner unterschiedliche Betriebssystem installiert, so wird gewöhnlicherweise ein Boot-Loader in den Bootsektor geschrieben. Dieser Boot-Loader ermittelt das Betriebssystem, dass gestartet werden soll und startet es anschließend. Meistens präsentiert der Boot-Loader dem Anwender eine Liste, aus der der Anwender das Betriebssystem auswählen kann.

challenge-response-Authentifizierung Die *challenge-response-Authentifizierung* ist ein Verfahren in der Kryptographie in der ein Herausforderer eine Frage stellt, die von einer entfernten Instanz korrekt beantwortet werden muss. Ein sehr einfaches Beispiel ist die Abfrage nach einem Passwort. Sicherere Verfahren hingegen übermitteln nicht das Passwort, sondern nur das Resultat der Berechnung über die Frage (für gewöhnlich eine zufällige große Zahl) und das Passwort (nach [wik]).

IPC *Inter-Prozess-Kommunikation* Die IPC (engl. *inter-process communication*) ist der Systemaufruf eines Betriebssystems, dass den Austausch von Nachrichten zwischen Anwendungen erlaubt.

Konzelationssystem Ein Konzelationssystem ist ein System mit dem Nachrichten ver- und entschlüsselt werden können. Bei symmetrischen Konzelationssystemen benötigen Absender und Empfänger einen geheimen Schlüssel. Bei asymmetrischen Systemen ist der öffentliche Schlüssel allen bekannt und kann für das Verschlüsseln verwendet werden. Das Entschlüsseln erfolgt durch Anwendung des geheimen Schlüssels beim Empfänger eienr verschlüsselten Nachricht.

Kryptographisch sichere Hash-Funktionen Eine Hashfunktion ist eine besondere deterministische Einwegfunktion, die zu einer beliebigen Eingabe eine Ausgabe fester Länge errechnet (vgl. Seite 6). Einwegfunktionen sind Funktionen, die keine inverse Funktion besitzen, d.h. aus dem Ergebnis der Funktion lässt sich nicht die Eingabe errechnen. Kryptographisch sichere Einwegfunktionen müssen unter anderen kollisionsresistent sein. Das bedeutet, dass es zu einer Eingabe a und der Ausgabe h, mit $h = \mathcal{H}(a)$ unmöglich ist, mit vertretbaren Aufwand eine Eingabe b zu finden, so dass $\mathcal{H}(a) = \mathcal{H}(b)$ und $a \neq b$ gelten.

L4Env L4Env ist eine Menge von Anwendungen und Bibliotheken für die Entwicklung unter L4.

Legacy Systeme In der Informationstechnik bezeichnet *legacy* den Anteil eines Systems, dass bei einer Weiterentwicklung übernommen werden soll.

Mock Mock-Objekte werden in der testgetriebenen Softwareentwicklung Objekte genannt, die als Platzhalter für echte Objekte innerhalb von Unit-Tests verwendet werden (nach [wik]).

PCR *Platform Control Register* Das PCR ist ein besonderes Register in einem TPM-Gerät, das zur Bildung der Prüfsummenkette eingesetzt wird. Mit Hilfe dieser Prüfsummenkette kann das *überprüfbare Urladen* umgesetzt werden (vgl. Abschnitt 2.1.5). Siehe Abschnitte 2.1.2 und 2.1.3.1

POSIX *Portable Operating System Interface* POSIX ist ein gemeinsam von der IEEE und der Open Group für Unix entwickeltes standardisiertes Applikationsebeneninterface, das die Schnittstelle zwischen Applikation und dem Betriebssystem darstellt. Der (inter-)nationale Standard trägt die Bezeichnung DIN/EN/ISO/IEC 9945. Aus [wik].

Proxy Ein Proxy (aus dem lat. *proximus*, der Nächste) stellt eine Instanz in Rechnernetzen dar, die den Datenverkehr vermittelt. Der Proxy kann durch Zwischenspeichern den Netzwerkverkehr effizienter gestalten, aber auch in die Vermittlung eingreifen, und den Einsatz von Zuganskontrollmechanismen erwirken.

regression-tests Regression-Tests sind Tests, die die Ausgabe bzw. Resultate einer Methode, Klasse oder Komponente über den Entwicklungszyklen hinweg mit früheren Ausgaben vergleicht.

SQL *Structured Query Language* Die SQL-Sprache ist eine Computersprache, um Informationen aus relationalen Datenbanken zu extrahieren, zu löschen oder zu manipulieren.

test-drive-programming Bei diesem Ansatz werden die Testprogramme (Siehe unit tests) vor den Programmen geschrieben. Das Programm gilt erst dann als fertig, wenn alle Tests erfolgreich durchlaufen werden.

unit tests : Das *unit tests framework* ist ein Programmierwerkzeug, dass automatisierte Tests erlaubt. Die Tests werden in Form kleiner Programme geschrieben und in das Testverfahren eingebunden. Dieses Werkzeug verwaltet den Aufruf und die Auswertung der Tests.

white-box-testing Bei dieser Testmethode werden hauptsächlich die inneren Zustände und die Zustandsübergänge einer Methode, Klasse oder Komponente getestet. Das *black-box-testing* hingegen testet Die Schnittstellen einer Komponente uns ihr Verhalten im Zusammenspiel mit anderen Komponenten.

yacc *Yet Another Compiler-Compiler*

Index