Trusted Computing Systeme

Thomas Müller

Trusted Computing Systeme

Konzepte und Anforderungen

Springer

Thomas Müller
Hagenholzstrasse 92
CH-8050 Zürich
Thomas.Mueller@tmxm.de

ISBN 978-3-540-76409-0 e-ISBN 978-3-540-76410-6

DOI 10.1007/978-3-540-76410-6

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Einbandgestaltung: KünkelLopka Werbeagentur, Heidelberg
Satz und Herstellung: le-tex publishing services oHG, Leipzig

Gedruckt auf säurefreiem Papier.

9 8 7 6 5 4 3 2 1

springer.com

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1
Einleitung

1.1 Motivation

Ein wichtiger Bestandteil der heutigen IT-Landschaft ist der Bereich der IT-Sicherheit, der sich mit den Gefährdungen für und durch IT-Systeme beschäftigt. Ein Grund für das konstante Wachstum dieser Branche in den vergangenen Jahren sind die zahlreichen Sicherheitslücken aktueller IT-Systeme, die zu einer globalen Verbreitung von Schadsoftware (wie z. B. Viren, Würmer, Trojanische Pferde und Spyware) auf Computern von Privatpersonen und Unternehmen führen. Die Ausnutzung von Sicherheitslücken von IT-Systemen ist grundsätzlich keine neue Erscheinung der Computerbranche, jedoch ist in den letzten Jahren eine deutliche Zunahme im Bereich der finanziell motivierten Internetkriminalität zu erkennen. So enthielt anfangs ein Großteil der heute als Schadsoftware bezeichneten Programme keine wirkliche Schadfunktion, sondern diente vielmehr der Demonstration einer Sicherheitslücke (*Proof-of-Concept*) zur Sensibilisierung für die daraus entstehenden Gefahren. Die Hauptaufgabe heutiger Schadsoftware ist meist der Aufbau einer technischen Infrastruktur für die organisierte Internetkriminalität. Typische Schadfunktionen hierbei sind das Ausspähen von privaten Daten (z. B. Passwörter, Kreditkartennummern, Softwarelizenzen), der Aufbau von zentral gesteuerten Rechnerverbänden (Bot-Netze) zum Versand von Spam oder zum großflächigen Angriff auf die Verfügbarkeit einzelner Systeme (DDoS[1]) sowie die automatische Suche nach weiteren Systemen mit dieser Sicherheitslücke zur eigenen Replikation. Ein weiterer aktueller Trend ist im Wandel hin zu sehr gezielten Angriffen auf die Infrastruktur eines einzelnen Unternehmens mit dem primären Ziel der Wirtschaftsspionage zu erkennen. Schätzungen der Hersteller von Virenschutzsoftware zufolge sind aktuell zwischen 50% und 90% aller mit dem Internet verbundenen Computer mit einer oder mehreren Arten von Schadsoftware infiziert. Diese Infrastruktur ermöglicht der so genannten *Underground Economy*[2] die einfache und nahezu risikofreie Erschließung eines neuen Marktsegments. Nach Aussage des US-

[1] DDoS – Distributed-Denial-of-Service-Angriffe

[2] [URL01]

Finanzministerium liegt der jährliche durch die Internetkriminalität erzeugte Umsatz oberhalb von 100 Milliarden US-Dollar und beläuft sich damit auf mehr als im weltweiten Drogenhandel jährlich umgeschlagen wird[3].

Unterstützt wird die Verbreitung von Schadsoftware durch die nahezu flächendeckende Verfügbarkeit von breitbandigen Internetzugängen für private Haushalte. Der eigentliche Ursprung jedoch liegt in den Sicherheitslücken der eingesetzten Betriebssysteme und Anwenderprogramme. Speziell die Betriebssysteme konnten mit der rasanten Weiterentwicklung bei den Techniken für Schadsoftware nicht Schritt halten. So basieren zum Beispiel die in aktuellen Desktop-Betriebssystemen implementierten Zugriffskontrollsysteme immer noch auf den zu Beginn der PC-Ära entwickelten Konzepten. Zu diesem Zeitpunkt war der PC überwiegend ein Single-User-System ohne Zugriff auf ein öffentliches Netzwerk wie das Internet. Ein weiteres Manko heutiger Betriebssysteme ist die Verwendung von unsicheren Programmiersprachen (wie z. B. C und C++) für deren Entwicklung. Die dabei häufig auftretenden Fehler bei der Programmierung bilden den Angriffsvektor (z. B. durch Buffer Overflows) für das Einschleusen von Schadcode in das Betriebssystem oder in eine Applikation.

Um die Anzahl möglicher Angriffsvektoren zu reduzieren oder deren Auswirkung auf die Systemintegrität abzuschwächen, wurden in der nahen Vergangenheit unterschiedliche neue Konzepte entwickelt. Hinter dem Begriff *Trusted Computing* verbirgt sich eines dieser Konzepte zur Verbesserung der Computersicherheit. Dieses Buch beschäftigt sich ausführlich mit den Konzepten und Anforderungen dieser noch verhältnismäßig jungen Technologie.

1.2 Ziele des Buches

Das Schlagwort *Trusted Computing* ist immer häufiger das Thema von Fachartikeln, Internetbeiträgen und sogar ganzen Konferenzen und Workshops. Der Begriff ist inzwischen auch bei Personen außerhalb der IT-Sicherheitsbranche bekannt, wenn auch überwiegend auf Grund einer gewissen Nähe zum Thema *Digital Rights Management* (DRM).

Trotz der erlangten Aufmerksamkeit und der verfügbaren Publikationen ist es selbst für Fachleute auf dem Gebiet der IT-Sicherheit schwierig, sich einen klaren Überblick über diese Technologie zu verschaffen. Dies liegt darin begründet, dass sich viele der Artikel nur mit einem kleinen Teilbereich der Technologie beschäftigen, den Leser aber häufig über diesen Missstand nicht informieren. So wird das Thema *Trusted Computing* oft auf das im Zuge der Entwicklungen eingeführte *Trusted Platform Modul* (TPM) und dessen Funktionen beschränkt. Weiter ist zum Zeitpunkt der Entstehung dieses Buches keine Grundlagenliteratur zum Thema *Trusted Computing* verfügbar.

[3] [URL02]

Das Ziel dieses Buches ist es daher, dem Leser einen tiefgehenden Einblick in die Thematik zu verschaffen. Hierbei stehen neben den technischen Grundlagen vor allem die Ziele und Konzepte sowie die Grenzen des *Trusted Computing* im Vordergrund. Es werden die technischen Spezifikationen der *Trusted Computing Group* (TCG), wissenschaftliche Publikationen zum Thema sowie bereits verfügbare Lösungsansätze für *Trusted-Computing-Systeme* (TCS) betrachtet und für den Leser verständlich aufbereitet und zusammengefasst. Ein weiteres Ziel des Buches ist die Definition eines Anforderungskatalogs für zukünftige *Trusted-Computing-Systeme* und zur Bewertung heutiger Computersysteme.

1.3 Gliederung des Buches

Das Buch gliedert sich in elf Kapitel. Kapitel 2 liefert die für das Verständnis der nachfolgenden Kapitel notwendigen Grundlagen. Kapitel 3 enthält eine Einführung in das Thema *Trusted Computing* und dessen Ziele. Kapitel 4 ist eine Zusammenfassung der Spezifikationen der TCG und beschreibt unter anderem das TPM. In Kapitel 5 werden Alternativen und Erweiterungen für das Konzept der TCG vorgestellt. Kapitel 6 definiert einen Anforderungskatalog auf abstrakter Ebene. In Kapitel 7 wird die für die Umsetzung des *Trusted Computing* notwendige Infrastruktur und die damit verbundene Problematik beschrieben. In Kapitel 8 werden dann ausführlich vorhandene theoretische und praktische Lösungsansätze für den Bau von *Trusted-Computing-Systemen* vorgestellt. In welchen Projekten diese Ansätze bereits zum Einsatz kommen, erläutert Kapitel 9. In Kapitel 10 werden im Rahmen des Fazits der heutige Stand des *Trusted Computing* und der Trend der nächsten Jahre beleuchtet. Als Abschluss liefert Kapitel 11 einen Überblick über einige der neuen Sicherheitsfunktionen in Windows Vista und erläutert, welche *Trusted-Computing*-Konzepte hierbei bereits in die Entwicklung des neuen Betriebssystems eingeflossen sind.

Kapitel 2
Grundlagen

Für das bessere Verständnis der folgenden Kapitel werden zunächst einige grundlegende Aspekte beim Design von Betriebssystemen besprochen. Ebenfalls wird kurz das Konzept der Zero-Knowledge-Beweise und ein Integritätsmodell vorgestellt. Kenntnisse über kryptographische Grundlagen wie symmetrische und asymmetrische Verschlüsselung, Hashfunktionen sowie über Verfahren zur digitalen Signatur und *Public-Key-Infrastrukturen* (PKI) werden vorausgesetzt. Einen guten Einstieg in diese Thematik bietet ein Grundlagenbuch[1] zum Thema Kryptographie.

2.1 CPU-Ringe

Die Sicherheit eines Computersystems hängt nicht nur von der Sicherheit der darauf laufenden Software, sondern auch von den von der Hardware bereitgestellten Schutzfunktionen ab. Ein wichtiges Beispiel einer solchen Schutzfunktion sind die Privilegierungsstufen einer CPU. Intel 80286-kompatible[2] Prozessoren unterscheiden vier Privilegierungsstufen, auch Ringe genannt: *Ring 0, 1, 2* und *3*. Dabei stellt *Ring 0*, genannt „*supervisor mode*“ oder auch „*kernel mode*“, die höchste Privilegierungsstufe dar, die bis zur *Stufe 3* immer weiter eingeschränkt wird. Die folgende Abb. 2.1 verdeutlicht das Konzept der Ringe.
Der volle Umfang des Befehlssatzes der CPU wird nur Prozessen zur Verfügung gestellt, welche im *Ring 0* ausgeführt werden. In allen anderen Ringen steht nur ein Subset der Funktionen zur Verfügung. So wird unter anderem sichergestellt, dass unprivilegierte Prozesse nicht direkt auf die Hardware zugreifen und sich auch nicht aus ihrer Privilegierungsstufe befreien können. Der Zugriff auf den Speicherbereich anderer Prozesse wird durch Speichervirtualisierung verhindert. Somit wird gewährleistet, dass Prozesse zum Beispiel im *Ring 3* in keinem Fall Prozesse im *Ring 0* oder auch andere Prozesse im *Ring 3* beeinflussen können. Übergänge von

[1] [Schmeh2001]

[2] Intel 80286 ist Mikroprozessor der Firma Intel aus dem Jahre 1982.

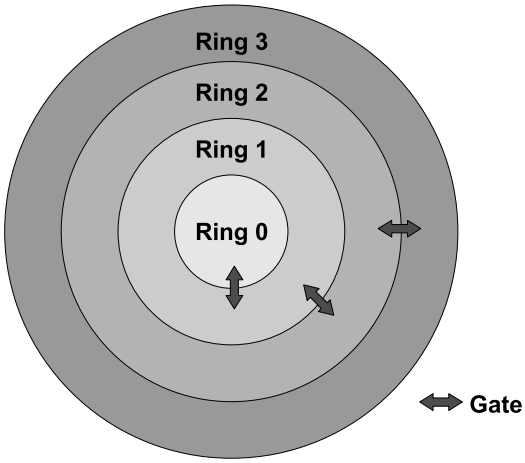

Abb. 2.1 CPU-Ringe

einer unprivilegierten Stufe in eine Eben mit mehr Umfang werden als *Gate* bezeichnet. Sie stellen die sichere Kommunikation zwischen Prozessen in unterschiedlichen Privilegierungsstufen zur Verfügung. Die dabei ausgelösten Interrupts führen in der Regel zu einem so genannten Kontextwechsel (*context switch*). Dabei wird der Kontext (im Wesentlichen die Prozessor-Register) des aktuellen Prozesses gesichert und der Kontext des neuen restauriert bzw. erstellt.

Schon beim Design der Privilegierungsstufen einer CPU wurde definiert, welche Klasse von Prozessen in welcher Stufe ausgeführt werden soll um die Integrität des Betriebssystems zu gewährleisten. Die nachfolgende Abbildung veranschaulicht diese Aufteilung:

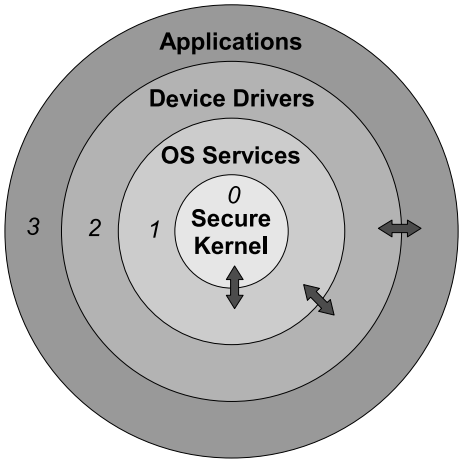

Abb. 2.2 Verwendung der CPU-Ringe

So sollten im *Ring 0* nur die Grundfunktionen eines Betriebssystems (z. B. Speicherverwaltung, Scheduling) sowie ein Zugriffskontrollsystem (*Access Control*) ausgeführt werden. Die rechtlichen Dienste des Betriebssystems sowie die Treiber für die Hardware des Systems sollen nur mit eingeschränkten Rechten in den *Ringen 1* und *2* laufen. Anwendungen, mit denen der Benutzer direkt interagiert, erhalten die wenigsten Rechte und kommen im *Ring 3* zur Ausführung. Eine Umsetzung dieser Aufteilung ist häufig ein Aspekt bei der Implementierung eines Betriebssystems in Form eines Mikrokernels[3].

2.2 Verwendung der CPU-Ringe

Verbreitete Betriebssysteme wie Linux und Windows folgen jedoch nicht vollständig dem Konzept der Privilegierungsstufen, sondern verwenden nur zwei der vier Ringe. Sie nutzen *Ring 0* für den Kernel und die Gerätetreiber. *Ring 1* und *Ring 2* werden nicht verwendet. Die Anwendungen der Benutzer laufen wie vorgesehen im *Ring 3*. Für die Kommunikation zwischen Applikationen und dem Kernel stellt das Betriebssystem Systemaufrufe (*system calls*) zur Verfügung. Die folgende Abbildung beschreibt diese Aufteilung:

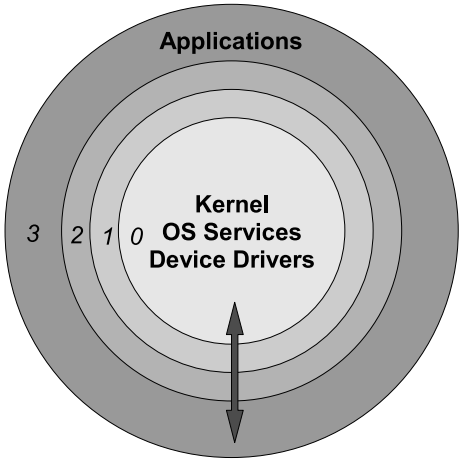

Abb. 2.3 Windows und CPU-Ringe

Somit sind zwar Applikationen und das Betriebssystem voneinander getrennt, ein Zugriff auf den Speicherbereich des Kernels mittels einer unprivilegierten Applikation ist nicht möglich, jedoch werden sämtliche Treiber im selben Kontext wie der Kernel ausgeführt und erhalten somit vollständigen Zugriff auf dessen Ressourcen. Dies wirkt sich negativ auf die Sicherstellung der Integrität des Kernels aus,

[3] Ein Beispiel für einen Mikrokernel ist beispielsweise der L4 Kernel.

da Treiber häufig aus nicht vertrauenswürdigen Quellen stammen und zusätzlich die Qualität des Codes nicht sichergestellt werden kann. So kann beispielsweise ein Druckertreiber unter Windows sämtliche Sicherheitsfunktionen des Betriebssystems untergraben.

Der Hauptgrund für die Wahl einer solchen Architektur (Monolithischer Kernel) ist die im Vergleich zu einem Mikrokernel meist bessere Performance. Diese liegt hauptsächlich im Wegfall von Kontextwechseln bei Operationen innerhalb des Betriebssystemkerns begründet. Ein weiterer Grund dürfte die zum Zeitpunkt der Entwicklung der ersten Generationen heutiger Betriebssysteme geforderte Kompatibilität zu Prozessoren mit nur zwei Privilegierungsstufen sein.

2.3 Virtualisierung und die CPU-Ringe

Vor allem im Bereich der Virtualisierung von Servern wird eine zusätzliche Schicht zwischen der Hardware und den virtualisierten Betriebssystemen eingefügt. Diese als *Virtual Machine Monitor* (VMM) oder auch *Hypervisor* bezeichnete Schicht kontrolliert die Hardware des Systems und sorgt für die notwendige Isolation der einzelnen Betriebssysteme durch Aufteilung der vorhandenen Ressourcen. Um hierfür die notwendigen Rechte zu haben muss der VMM innerhalb der höchsten Privilegierungsstufe, sprich dem *Ring 0*, ausgeführt werden. Die Betriebssysteme müssten daher in den *Ring 1* verschoben werden. Da diese jedoch aufgrund ihrer Architektur auf den vollen Funktionsumfang der CPU angewiesen sind, erweist sich diese Verlagerung als nicht unproblematisch.

Es existieren mehrere Lösungen für das Problem. Die populärsten Ansätze sind dabei die Emulation einer vollständigen CPU mittels Software und die Modifikation des zu virtualisierenden Betriebssystems. Da das Nachbilden einer CPU zu erheblichen Geschwindigkeitseinbußen führt, ist es nur in wenigen Fällen ein gangbarer Ansatz. Wird das Betriebssystem für die Ausführung oberhalb eines VMM angepasst, spricht man in der Regel von *Paravirtualisierung*. Der VMM *Xen* ist der aktuell wohl populärste Vertreter dieser Gattung der Virtualisierungstechnik. Die hierbei notwendigen Anpassungen sind jedoch nur bei Betriebssystemen ohne Weiteres möglich, deren Quellcode verfügbar ist. Die Verwendung von Windows als Gastbetriebssystem ist damit nicht möglich. Um diese Einschränkung zu umgehen verändern VMM wie beispielsweise VMware ESX die Logik der virtualisierten Betriebssysteme zur Laufzeit. Dabei werden Aufrufe des Betriebssystems auf Ring-0-Funktionen der CPU beim Start des Systems im Speicher manipuliert und auf eine entsprechende Schnittstelle des VMM umgeleitet. Der VMM führt nach Kontrolle der Zugriffsrechte der VM die aufgerufene Operation im Auftrag der VM aus. Anschließend übergibt der VMM das Ausführungsrecht wieder zurück an die VM.

Eine einfachere Möglichkeit zur Virtualisierung von unmodifizierten Betriebssystemen bieten aktuelle CPUs der Hersteller Intel und AMD. Intel nennt seine

Technik zur Virtualisierung der CPU *Intel VT-x*[4], AMD hat mit *AMD SVM*[5] eine vergleichbare Lösung im Programm. Beide Hersteller erweitern hierbei den Befehlssatz ihrer CPUs um weitere Befehle zur Ausführung eines Betriebssystems innerhalb einer isolierten Umgebung. Ebenfalls erweitert wurde hierfür das Konzept der CPU-Ringe. Die folgende Abbildung visualisiert die neue Aufteilung:

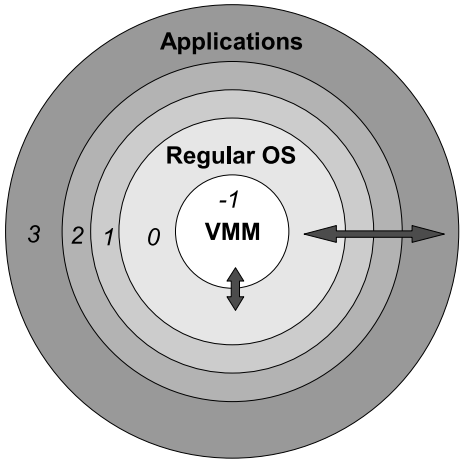

Abb. 2.4 Zusätzlicher CPU-Ring

Der VMM wird dabei innerhalb des *Ringes −1* ausgeführt. Dieser Ring verfügt über sämtliche Rechte zur Ausführung von Operationen auf der CPU. Für jede VM erstellt die CPU auf Anfrage des VMM eine virtualisierte CPU mit den bekannten *Ringen 0–3*. Aus Sicht des Gastbetriebssystems befindet es sich innerhalb des *Ringes 0* und unterliegt somit scheinbar keinerlei Einschränkungen. Greift nun eines dieser Betriebssysteme auf eine Funktion des *Ringes 0* zu, erzeugt die CPU einen Interrupt und aktiviert den im *Ring −1* ausgeführten VMM. Dieser kann nun nach Prüfung der angeforderten Operation diese, transparent für das Gastbetriebssystem, in dessen Auftrag ausführen. Eine Anpassung des Betriebssystems ist daher nicht mehr notwendig.

Aus Sicht der VMs ist die Existenz eines VMM in der Regel nicht mehr zu erkennen. Obwohl durchaus gewünscht, kann dieser Umstand auch von Schadsoftware missbraucht werden. So verwenden beispielsweise so genannte Virtualisierungs-Rootkits diese Funktion um ein Betriebssystem zur Laufzeit in eine virtuelle Maschine zu verschieben[6]. Die Rolle des VMM übernimmt hierbei das Rootkit selbst. Daher wird empfohlen, den Zugriff auf diese Befehlssatzerweiterung bei Nichtverwendung durch das System im BIOS zu deaktivieren.

---

[4] Auch als Intel „Vanderpool“ bekannt

[5] AMD *Secure Virtual Machine*, auch als „Pacifica“ bekannt

[6] Demonstriert wurde diese Möglichkeit u. a. durch „BluePill“ [URL28].

2.4 Zero-Knowledge-Beweis[7]

Das Konzept der *Zero-Knowledge-Proofs* wurde 1985 von Goldwasser, Micali und Rackoff eingeführt. Zero-Knowledge-Beweissysteme sind spezielle interaktive Beweissysteme. Bei interaktiven Beweissystemen wird ein Protokoll zwischen Beweiser A und Verifizierer B durchgeführt. A kennt ein Geheimnis, B nicht. In dem Protokoll überzeugt der Beweiser den Verifizierer davon, dass er das Geheimnis kennt. Ein Betrüger kann das nicht. (Diese Eigenschaften haben auch die Public-Key-Verfahren zur Authentifizierung.)

Hinzu kommt bei Zero-Knowledge-Beweissystemen folgende Forderung:

- Der Verifizierer erfährt nur, dass der Beweiser das Geheimnis kennt, aber nichts weiter (gleichgültig, welche Strategie er anwendet).

Wir geben ein Beispiel, das so genannte Fiat-Shamir-Verfahren (1986):

A, die Beweiserin, wählt p, q, große Primzahlen, bildet $n = p \cdot q$. Sie wählt $s \in \{1, \ldots n-1\}$ zufällig mit $\text{ggT}(s,n) = 1$. Sie berechnet $v = s^2 \bmod n$. (v,n) ist öffentlich, s geheim. (Beachte: s ist nur dann aus n zu bestimmen, wenn man die Faktorisierung von n kennt.) A beweist B, dass sie eine Quadratwurzel von v modulo n kennt.

Protokoll:

1. A wählt zufällig und gleichverteilt $r \in \{1,2, \ldots n-1\}$. Sie berechnet $x = r^2 \bmod n$.
2. A sendet x an Verifizierer B.
3. B wählt zufällig mit Wahrscheinlichkeit $1/2$ ein $e \in \{0,1\}$ und sendet e an A (Challenge).
4. Erhält A den Wert $e = 0$, so schickt sie r (die Wurzel von x) zurück.
   Erhält A den Wert $e = 1$, so schickt sie $y = rs \bmod n$ (also Wurzel von xv) an B (Response).
5. B verifiziert, dass $r^2 \equiv x \bmod n$ (im Fall $e = 0$) bzw. $y^2 \equiv xv \bmod n$ (im Fall $e = 1$).

Analyse des Verfahrens:

a) Kennt A das Geheimnis, so kann sie jede der Fragen von B beantworten.
b) Ein Betrüger kann aus (n,v) das Geheimnis nicht entschlüsseln (Ziehen von Quadratwurzeln modulo n genauso schwierig wie Faktorisierung von n).
c) Was passiert, wenn ein Betrüger trotzdem das Protokoll mit B durchführt, um sich als A auszuweisen? Er kann nicht beide möglichen Fragen beantworten, denn kennt er r und rs, so auch s. Er hat also nur zwei Möglichkeiten:

[7] Aus [Haug2002]

- Er wählt r wie vorgesehen und sendet $x = r^2 \bmod n$.
  Sendet B $e = 0$, so antwortet er r, korrekt.
  Sendet B $e = 1$, so kann er nicht antworten.
- Er wählt y vor und sendet $x = y^2 v^{-1} \bmod n$.
  Bei $e = 1$ kann er jetzt korrekt antworten.
  Bei $e = 0$ müsste er die Wurzel von v bestimmen. Das kann er nicht. Da er x angeben muss, bevor er e erhält, kann er mit einer Wahrscheinlichkeit von $1/2$ richtig antworten. Nach k Wiederholungen ist die Wahrscheinlichkeit $1/2k$, dass er immer richtig antwortet. Bei immer richtiger Antwort ist B mit einer Wahrscheinlichkeit von $1 - 1/2^k$ überzeugt, dass A tatsächlich das Geheimnis s kennt.

d) Man kann zeigen, dass B aus diesem Beweissystem nichts über s erfährt, außer dass A s kennt. (Gleichgültig, ob er sich dem Protokoll entsprechend verhält (e zufällig wählen) oder eine andere Strategie verfolgt.) (Zero-Knowledge-Eigenschaft)
   Man formalisiert dies dadurch, dass man fordert:
   Ist die Behauptung des Beweises wahr, so kann die Interaktion zwischen A und B in polynomialer Zeit (von B) simuliert werden, ohne dass eine Interaktion mit dem Beweiser stattfindet.

2.5 Clark-Wilson-Integritätsmodell[8]

Das *Clark-Wilson-Modell* ist ein Zugriffsmodell, das vor allem vor Manipulationen schützt und somit die Integrität adressiert. Innerhalb des *Clark-Wilson-Modelles* können Akteure (Benutzer, Prozesse) nie direkt auf ein Objekt (Datei, Gerät) zugreifen, sondern müssen zwingend mittels einer Applikation zugreifen. Das Konzept ist nicht analog zu dem des Referenzmonitors. Der Referenzmonitor fällt die Entscheidung, welche Applikation für den Akteur ausführbar ist. Der Zugriff über die Applikation erzeugt eine zusätzliche Schicht, in welcher applikatorische Einschränkungen implementiert werden können.

Das Modell beschreibt mittels Einhaltungs- (enforcement) und Zertifizierungsregeln (certification) die informationstechnischen Daten und Prozesse. Diese Regeln bilden die Basis zur Sicherstellung der Integrität eines Systems. Das Modell basiert immer auf einer in sich geschlossenen Transaktion.

- Eine gültige Transaktion ist eine Abfolge von Operationen, welche das System von einem Zustand in den nächsten Zustand bringt. Die Transaktion muss immer atomar sein. Dies bedeutet, dass die Zustandsänderung nur erfolgt, wenn die Transaktion keine Fehler aufweist.
- Im Clark-Wilson-Modell wird die Integrität über die Transaktionskontrolle sichergestellt.

[8] Aus [Clark1987]

- Das Prinzip der Gewaltentrennung (seperation of duty) erfordert, dass der Prozess, der eine Transaktion kontrolliert, unabhängig von dem transaktionsausführenden Prozess ist.

Dazu werden folgende Konstrukte verwendet:

- Constrained Data Item (CDI)
  Daten, die einen sicheren Zustand im System repräsentieren.
- Unconstrained Data Item (UDI)
  Daten, die noch nicht gesichert sind, z. B. Benutzer-Eingaben.
- Integrity Verification Procedure (IVP)
  Stellt sicher, dass alle CDI in einem System einen gültigen Zustand besitzen.
- Transformational Procedures (Tps)
  Die Transaktion nimmt ein CDI oder UDI entgegen und nimmt die Transaktion auf den Daten vor. Die Transaktion muss sicherstellen, dass immer wenn sie ein UDI entgegennimmt, ein sicheres CDI produziert wird; dies wird über eine Zertifizierung gelöst.

Die Regeln des Integritätsmodells sind dabei wie folgt:

- C1: Wenn ein IVP ausgeführt wird, muss es sich immer in einem gültigen Zustand befinden.
- C2: Alle Daten (CDIs) müssen von einem TP, von einem gültigen Zustand in den nächsten übergeführt werden.
- E1: Das System muss eine Liste von zertifizierten Verknüpfungen führen, welche sicherstellen, dass für ein CDI nur zertifizierte TP erlaubt werden.
- E2: Das System muss jedem TP und CDI einen Benutzer zuweisen. Der Tupel {Benutzer,TP,CDI} wird auch als allowed relations bezeichnet.
- E3: Jeder Benutzer, welcher auf ein TP zugreift, muss auf Anfragebasis (request) authentifiziert werden.
- C4: Jeder TP muss ein Transaktionsjournal schreiben, welches erlaubt, alle Operationen zu rekonstruieren.
- C5: Jeder TP, welcher ein UDI als Eingangsdaten verarbeitet, darf nur Operationen auf diesem UDI ausführen. Kann er das UDI nicht zu einem CDI konvertieren, muss er die Verarbeitung abbrechen.
- E4: Nur der Zertifizierer (Hersteller oder Besitzer) des Systems darf Einträge in den Verknüpfungen {TP,CDI} vornehmen.

Kapitel 3
Trusted Computing

Der Begriff *Trusted Computing* (TC) oder auch *Trustworthy Computing* wurde erstmals im Jahre 1983 vom Verteidigungsministerium des USA (Department of Defense) im Rahmen des Standards *Trusted Computer System Evaluation Criteria* (TCSEC) definiert [DoD1983]. Der Standard dient der Bewertung und Zertifizierung des Sicherheitsniveaus von Computersystemen. Das Dokument ist auch unter dem Namen *Orange Book* bekannt. Die TCSEC bildet die Grundlage für den Standard *Common Criteria For Information Technology Security Evaluation* (CC), dem aktuell gültigen und international anerkannten Standard für die Zertifizierung von IT-Systemen.

Für die zivile IT-Landschaft wurde der Begriff erst durch die Gründung eines Konsortiums, der *Trusted Computing Platform Alliance* (TCPA), im Jahre 1999 relevant. Dieses Konsortium wurde von den Mitgliedern Microsoft, IBM, Hewlett-Packard und Compaq ins Leben gerufen, um einen Industriestandard für vertrauenswürdige Computersysteme zu entwickeln. Popularität erlangte der Begriff *Trusted Computing* im Jahre 2002 durch die Ankündigung von *Palladium* durch Microsoft. Im Rahmen des *Palladium*-Projektes sollten die von der TCPA erarbeiteten Konzepte erstmals innerhalb eines Betriebssystems umgesetzt werden. Die Präsentation der Konzepte von *Palladium* stieß jedoch fast ausschließlich auf Ablehnung auf Seiten der Fachwelt, da man befürchtete, diese Technologie diene nur dem Schutz von Systemen zur digitalen Rechteverwaltung (*Digital Rights Management* – DRM) und entziehe dem Anwender die Kontrolle über sein System. Durch die Veröffentlichung zahlreicher negativer Artikel und kritische Äußerungen von anerkannten IT-Sicherheitsspezialisten, wie z. B. Ron Rivest, Whitfield Diffie und Bruce Schneier, entstand eine weltweite Gegenbewegung zur *Trusted-Computing*-Initiative, die bis zum heutigen Zeitpunkt aktiv ist.

Wohl auf Grund des Mangels an konkreten Ergebnissen löste sich die TCPA im Jahre 2003 auf. Ein Teil der ehemaligen Mitglieder gründete noch im selben Jahr die *Trusted Computing Group* (TCG). Die TCG gliedert sich in mehrere Arbeitsgruppen, deren übergeordnetes Ziel nach wie vor die Entwicklung von Industrie-

standards für vertrauenswürdige Computersysteme ist[1]. Ebenfalls 2003 stellte Microsoft eine überarbeitete Version der mit *Palladium* eingeführten Konzepte unter dem Namen *Next Generation Secure Computing Base* (NGSCB) vor. Die Ansätze der NGSCB sollten in die Entwicklung der nächsten Generation der Windows-Betriebssysteme mit dem damaligen Codenamen „Longhorn" einfließen.

Nachdem es in den Jahren 2003 und 2004 sehr ruhig um das Thema *Trusted Computing* geworden war, ist seit Anfang 2005 wieder deutlich mehr Bewegung auf diesem Gebiet zu erkennen. So wurde mit der *European Multilaterally Secure Computing Base* (EMSCB) ein Projekt mit dem Ziel gestartet, eine quell-offene Plattform für *Trusted-Computing*-Anwendungen zu entwickeln. Das Projekt wird von Vertretern aus der Industrie und mehreren Hochschulen vorangetrieben und unter anderem durch das Bundesministerium für Wirtschaft und Technologie finanziert. Das Europäische Pendant zur EMSCB ist das *OpenTC*-Projekt, das ebenfalls auf Basis von Open-Source-Software an einem *Trusted-Computing*-System arbeitet. Im Bereich der kommerziellen Software gibt es zu vermerken, dass auch Microsoft mit Windows Vista und Windows Server 2008 einen Teil der mit NGSCB vorgestellten Konzepte umgesetzt haben will[2].

Auch wenn dem Thema *Trusted Computing* nach wie vor ein negatives Image anhaftet, scheint das Interesse der IT-Branche an der Technologie stetig zuzunehmen. Dazu tragen auch die Bestrebungen des *Bundesamtes für Sicherheit in der Informationstechnik* (BSI) und Fachkonferenzen zu diesem Thema bei.

3.1 Definition des Begriffs „Trusted Computing"

Der Versuch, eine eindeutige Definition für den Begriff des *Trusted Computing* zu finden, zeigt die Vielseitigkeit des Themas. Der Begriff existiert seit ca. 20 Jahren und stand während dieser Zeit für unterschiedliche Ziele und Konzepte. Die folgenden chronologisch sortierten Definitionen liefern einen Überblick über die Etappen des *Trusted Computing*:

1. *„The heart of a trusted computer system is the Trusted Computing Base (TCB) which contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based."* [DoD1983]
2. *„A Trusted System or component is one whose failure can break the security"* [NSA1998]
3. *„Treacherous computing is a more appropriate name, because the plan is designed to make sure your computer will systematically disobey you. In fact, it is designed to stop your computer from functioning as a general-purpose computer."* [Stallmann2002]

[1] Siehe Kapitel 4.

[2] Siehe Kapitel 11.

4. *„Trusted Computing is the expectation that a device will behave in a particular manner for a specific purpose“* [TCG2004]
5. *„Trustworthiness is assurance that a system or a component will perform as expected“* [CIST1999]
6. *„Trusted computing is a family of open specifications whose stated goal is to make personal computers more secure through the use of dedicated hardware.“* [Wikipedia2005]

Aus dieser Menge eine einzelne gültige Definition abzuleiten, fällt auf Grund der teilweise gegenläufigen Aussagen schwer. So beschreibt etwa [TCG2004] *Trusted Computing* als die *Erwartung,* dass ein System entsprechend der definierten Arbeitsweise agiert, [CIST1999] hingegen spricht von der *Zusicherung* der korrekten Arbeitsweise. Eine Zusicherung der korrekten Arbeitsweise ist jedoch nur durch den Einsatz fehlerfreier Hard- und Software zu gewährleisten. Die dafür notwendige formelle Verifikation ist jedoch vor allem für umfangreiche Software wie z. B. ein Betriebssystem eine zum aktuellen Zeitpunkt unlösbare Anforderung. Nach Meinung des Autors ist dies ohnehin eine Anforderung, die dem Bereich des *Secure Computing* zuzuordnen ist[3].

Die Bandbreite der möglichen Definitionen für *Trusted Computing* reicht von sehr allgemeinen und somit wenig aussagekräftigen Umschreibungen wie: „Eine Technologie um die Sicherheit von Computersystemen zu erhöhen“ über kritische Aussagen wie: „Eine Technologie um den Benutzern die Kontrolle über ihre Systeme zu entziehen“ bis hin zur sehr ehrgeizigen Zielsetzung wie: „Eine Technologie für beweisbar sichere Computersysteme“.

Auch die verfügbaren Spezifikationen der *Trusted Computing Group* tragen nur bedingt zur Klärung der eigentlichen Bedeutung bei, da sie sich nur mit Teilbereichen des *Trusted Computing* beschäftigen und für den Leser nur schwer zu verstehen sind. Der derzeitige Mangel an Fachliteratur und vollständigen Spezifikationen für diese Technologie erschwert einen einfachen Einstieg in das Thema *Trusted Computing* und führt damit auf Seiten der Benutzer zu Misstrauen und Abwehrhaltung gegenüber der entstehenden Technologie. Im kommerziellen Umfeld besteht hingegen aktuell die Gefahr der weiteren Aufweichung der Bedeutung des Begriffes.

Offensichtlich ist jedoch, dass sich die Bedeutung des Begriffs im Laufe der Jahre stark gewandelt hat. So stand *Trusted Computing* zu Beginn im Wesentlichen für die Implementierung und den Schutz eines Zugriffskontrollsystems (*Access Control System*) innerhalb eines Betriebssystems. Ein solches Zugriffssystem kann beispielsweise als Implementierung des *Bell-LaPadula*-Zugriffsmodells [Bell1976] ausgeführt sein. Der dafür verwendete *Reference Monitor* soll an jedem Zugriffsvorgang beteiligt, manipulationssicher und in seiner Funktionsweise verifizierbar sein. Für die Bewertung der Vertrauenswürdigkeit einer Implementierung sind im Rahmen der TCSEC sechs Beurteilungsklassen definiert worden. Auch wenn eine Zertifizierung innerhalb einer dieser Klassen nach wie vor ein Indiz für die Vertrauenswürdigkeit einer Software darstellt, umschreibt *Trusted Computing* in seiner aktuellen Bedeutung eine Reihe neuer Konzepte.

[3] Eine Abgrenzung zwischen *Secure Computing* und *Trusted Computing* ist ab Seite 27 zu finden.

Geprägt durch die Spezifikationen der *Trusted Computing Group* (TCG), wird *Trusted Computing* in der Regel immer dann als beschreibender Ausdruck gewählt, wenn ein *Trusted Platform Modul* (TPM) zum Einsatz kommt. Das TPM ist ein kryptographischer Prozessor, der fest mit der Hauptplatine (Mainboard) eines Computersystems verbunden ist. Es bietet unter anderem die Möglichkeit Prüfsummen über beliebige Daten zu erstellen und diese in einem geschützten Bereich zu hinterlegen. Viele der publizierten Anwendungsfälle von *Trusted Computing* basieren auf dieser Funktion des TPM. Sie verfolgen hierbei den Ansatz den Zustand eines Systems in Form von Prüfsummen über Hard- und Softwarekomponenten zu dokumentieren.

Ein Vergleich der während des Startvorgangs und zur Laufzeit des Systems erzeugten Prüfsummen mit vorhandenen Referenzwerten soll eine Aussage über die Vertrauenswürdigkeit des Systems ermöglichen. Das Vertrauen in die korrekte Arbeitsweise eines *Trusted-Computing-Systems* basiert somit nicht notwendigerweise auf der formellen Verifikation aller Softwarekomponenten, sondern auf dem Vergleich des aktuellen Systemzustands mit einem bekannten und als vertrauenswürdig eingestuften Zustand. Für den Rahmen dieses Buches ist deshalb folgende vom Autor erstellte Definition gültig:

> *„Trusted Computing bezeichnet die Ausführung von sicherheitskritischer Software auf einem System, dessen aktueller Zustand als vertrauenswürdig eingestuft wurde und das geeignete Maßnahmen zum Erhalt dieses Zustands aufweist."*

3.2 Ziele des Trusted Computing

Äquivalent zu den unterschiedlichen Definitionen des *Trusted-Computing*-Begriffs existiert eine ganze Reihe unterschiedlicher Ziele des *Trusted Computing*. Je nach Publikation variiert die Anzahl der beschriebenen Ziele und teilweise sogar deren Beschreibung. Nachfolgend werden die wichtigsten dieser Ziele benannt und näher erläutert.

Aus den Spezifikationen der *Trusted Computing Group*[4] abzuleitende Ziele:

- **Hardware-Based Root of Trust**
  Das Konzept des *Vertrauensankers* (*Root of Trust*) ist ein gängiger Ansatz in der IT-Sicherheit und beschreibt die Ursprungskomponente eines Sicherheitssystems oder einer Sicherheitsarchitektur. Der *Vertrauensanker*, ist die kritischste Komponente, da eine Kompromittierung ihrer Integrität die Sicherheit des gesamten Systems gefährdet. Als eingängiges Beispiel für einen *Vertrauensanker* eignet sich die *Root-CA* einer *Public Key Infrastructure* (PKI). Eine Kompromittierung des privaten Schlüssels der Root-CA führt zum Verlust der Vertrauenswürdigkeit der gesamten Zertifikats-Hierarchie.

[4] TCG Spezifikationen [TCG*]

Da Hardwarekomponenten im Allgemeinen ein höheres Sicherheitsniveau bieten als eine äquivalente Implementierung in Software, ist eines der Ziele der TCG die Realisierung des *Vertrauensankers* innerhalb der Hardware des Systems.

- **Chain of Trust**
  Das Ziel der *Vertrauenskette* (*Chain of Trust*) ist es, aufbauend auf der Integrität und somit der Vertrauenswürdigkeit des *Vertrauensankers* die Integrität des gesamten Systems zu gewährleisten. Hierfür ist es notwendig, dass zwischen jeder sicherheitskritischen Komponente des Systems und dem *Vertrauensanker* eine Beziehung besteht (*Vertrauenspfad*). Auf ein Computersystem übertragen bedeutet dies, dass jede auf dem System gestartete Software vor ihrer Ausführung protokolliert werden muss. Dieses Protokoll muss entweder vom *Vertrauensanker* selbst oder von einer bereits protokollierten und somit zur *Vertrauenskette* gehörigen Software geführt werden. Das Protokoll repräsentiert den Zustand des Systems und bildet dadurch die Grundlage für eine Bewertung der Vertrauenswürdigkeit.

- **Integrity Measurement, Storage and Reporting**
  Für die Erzeugung eines Protokolls über die auf dem System ausgeführte Software ist es notwendig diese eindeutig zu identifizieren. Eine Möglichkeit hierfür ist die Erzeugung von Prüfsummen über diese Komponenten durch den Einsatz einer kryptographischen Hashfunktion. Die erzeugten Prüfsummen müssen in einem geschützten Speicherbereich hinterlegt werden und gegen Manipulation gesichert werden. Das System muss eine Schnittstelle bieten, die das Auslesen der Prüfsummen erlaubt. Die Integrität der Daten während des Auslesens muss sichergestellt werden.

- **Protected Capabilities**
  Unter dem Begriff der *Protected Capabilities* sind Funktionen eines Systems zu verstehen, deren korrekte Ausführung nicht von der Integrität des Systems abhängen darf. Beispiele hierfür sind die Bereitstellung von sicherem Datenspeicher, die Erzeugung von echten Zufallszahlen[5] und die Ausführung von kryptographischen Algorithmen. Hierfür müssen diese Funktionen in einen speziellen geschützten Bereich ausgelagert werden und dürfen nur auf eigenen abgeschirmten Ressourcen (wie z. B. eigenem Arbeitsspeicher und Register) operieren. Ein Beispiel hierfür ist die beim Homebanking nach dem FinTS-Standard[6] eingesetzte Prozessor-Chipkarte. Sie bildet in Kombination mit einem geeigneten Kartenleser ein geschlossenes System für die Ein- und Ausgabe von Informationen und zur Erzeugung einer Signatur der zu übermittelnden Daten.

- **Sealed/Protected Storage**
  Eine Erweiterung des sicheren Datenspeichers ist das Konzept der versiegelten

[5] True Random Number Generator (TRNG)

[6] Financial Transaction Services (FinTS) ist der Nachfolger des HBCI-Standards.

Speicherung. Das Ziel ist es, Daten an einen bestimmten Zustand eines Systems zu binden. Hierfür fließen Prüfsummen, die den Zustand des Systems repräsentieren, in den Verschlüsselungsprozess ein. Dies stellt sicher, dass die Daten nur entschlüsselt werden können, wenn sich das System in einem identischen Zustand wie zum Zeitpunkt der Verschlüsselung befindet.

- **Platform Attestation**
  Attestation (Nachweis/Bescheinigung) bezeichnet den Vorgang der Bewertung der Vertrauenswürdigkeit eines Systems. Eine solche Bewertung lässt sich in zwei Schritte aufteilen. Im ersten Schritt muss überprüft werden, ob das System auf einer *Trusted Computing Platform* aufbaut. Dies entspricht im Wesentlichen einer Validierung des *Vertrauensankers*. Hierfür müssen Informationen über das System, wie zum Beispiel der Hersteller und die verwendeten Herstellungsprozesse, zugänglich sein. Sie liefern einen Nachweis über die Vertrauenswürdigkeit der Systemhardware (*Attestation to the platform*). Im zweiten Schritt findet eine Bewertung der Vertrauenswürdigkeit des aktuellen Zustands des Systems statt (*Attestation of the platform*). Die Grundlage hierbei liefert das im Rahmen der *Vertrauenskette* erzeugte Protokoll über die ausgeführte Software. Finden diese beiden Schritte auf einem entfernten Computersystem statt, wird dieser Vorgang als *Remote Attestation* bezeichnet.

- **Platform Authentication**
  Ein weiteres Ziel des *Trusted Computing* ist die Möglichkeit zur eindeutigen Identifizierung eines Systems. Während für die meisten Szenarien außerhalb des *Trusted-Computing*-Umfeldes eine schwache Identifizierung (zum Beispiel anhand von IP-Adresse, MAC-Adresse[7], Rechnername oder Betriebssystem-ID) ausreichend ist, muss beim *Trusted Computing* die Fälschung oder die Kopie der Identität ausgeschlossen werden. Dies kann jedoch bei einer durch Software generierten und geschützten Identität nicht garantiert werden. Daher muss die Identität in Hardware verankert werden, welche nicht aus dem System entfernt werden kann.

Ergänzend zu den von der TCG definierten Ziele sind vor allem die folgenden Aspekte in zahlreichen Publikationen zum Thema *Trusted Computing* zu finden und von zentraler Bedeutung:

- **Protected Execution**
  Die „geschützte Ausführung“ ist verwandt mit dem Konzept der *Protected Capabilities*. Hiermit wird sichergestellt, dass die Integrität eines Systemprozesses nicht von der Integrität eines anderen Prozesses abhängig ist. Der wesentliche Unterschied liegt in der Anforderung, beliebige Software in einem solchen geschützten Bereich ausführen zu können. Es muss daher sichergestellt werden, dass der Speicherbereich einer Applikation vor dem Zugriff und vor Veränderungen durch andere Applikationen geschützt ist. Die strenge Isolation einzelner

[7] Obwohl die MAC-Adresse ebenfalls in Hardware verankert ist, ist sie, aufgrund der einfachen Möglichkeiten zur Fälschung, für eine starke Authentifizierung des Systems ungeeignet.

Prozesse ist eine der Herausforderungen im Bereich der Computer-Sicherheit, da die gemeinsame Verwendung eines Speicherbereichs (shared memory) häufig für die Kommunikation zweier Prozesse verwendet wird. Alternativ zu findende Bezeichnungen für dieses Konzept sind unter anderem *Memory Curtaining*, *Protected Launch* und *Process Isolation*.

- **Protected Input/Output**
  Alle Daten, die vom Benutzer in das System eingegeben werden, und Informationen, die das System wieder verlassen, müssen im Hinblick auf Integrität und Vertraulichkeit gesichert werden. Hierfür muss der Kommunikationskanal zwischen Eingabegeräten (wie z. B. Tastatur und Maus) und dem Betriebssystem beziehungsweise den Applikationen geschützt werden. Nur so kann das Abhören der Eingaben mittels Hardware- und Software-Keyloggern unterbunden werden. Ebenso muss der Kommunikationskanal mit den am System angeschlossenen Displays verschlüsselt werden. Die Umsetzung der beiden Anforderungen setzt die Unterstützung durch die Hardware voraus. Für die verschlüsselte Ausgabe existiert mit HDCP[8] schon ein erster Standard. HDCP wird von vielen aktuellen Grafikkarten und Displays bereits unterstützt. Das Verfahren kommt momentan aber in erster Linie bei der Wiedergabe von hochauflösendem Videomaterial zum Einsatz.

- **Trusted-GUI**
  Eine spezielle Form der geschützten Ausgabe von Informationen ist der vertrauenswürdige Desktop (auch als *Trusted-GUI* bezeichnet). Er soll zusätzlich zur Verschlüsselung von ein- und ausgehenden Informationen sicherstellen, dass schadhafte Applikationen nicht in der Lage sind, sicherheitskritische Dialoge des Betriebssystems oder anderer Applikationen zu imitieren. Damit soll verhindert werden, dass Schutzfunktionen des Betriebssystems durch Fehlverhalten des Benutzers umgangen werden oder persönliche Daten des Benutzers wie z. B. Passwörter unerlaubt abgefragt werden (*Phishing*).

3.3 Das Trusted Computing System (TCS)

Für die Untersuchung möglicher Umsetzungen der vorgestellten Ziele muss zunächst der mehrfach verwendete Begriff *Trusted Computing System* (TCS) eindeutig definiert werden. Die beiden bedeutendsten Bestandteile eines *Trusted Computing System* sind eine sichere Rechnerplattform (*Trusted Computing Platform*) und das darauf aufsetzende Betriebssystem (*Trusted Operating System*). Die Abb. 3.1 beschreibt den logischen Aufbau eines *Trusted Computing System*.

Vor allem zwischen den beiden Begriffen *Trusted Computing System* (TCS) und *Trusted Computing Platform* (TCP) wird in vielen Publikationen nicht unterschie-

[8] High-bandwidth Digital Content Protection

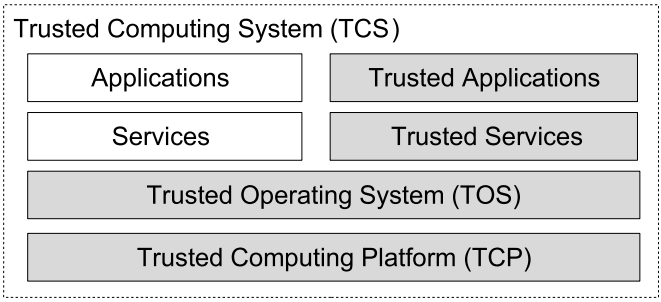

Abb. 3.1 Trusted Computing System

den, was häufig das Verständnis der vorgestellten Konzepte erschwert. Eine Unterteilung in diese beiden Bestandteile ist nach Meinung des Autors jedoch zwingend notwendig, um das Zusammenspiel von Hardware und Software für *Trusted Computing* zu verstehen.

Eine zusätzliche Abgrenzung erlaubt die Zuordnung der zuvor definierten Ziele zu den beteiligten Komponenten. Maßgeblich ist hierbei die Unterscheidung zwischen der *Trusted Computing Platform* und dem *Trusted Operating System* (TOS). Es gilt jedoch zu beachten, dass sich einige der Aufgaben nicht eindeutig zuordnen lassen, da sie entweder ein Zusammenspiel der beiden Komponenten voraussetzen oder wahlweise innerhalb der einen oder der anderen Komponente umgesetzt werden können:

Tabelle 3.1 Abgrenzung TCP und Trusted OS

Ziel/Aufgabe	**Trusted Computing Platform**	**Trusted Operating System**
Root of Trust	X	–
Chain of Trust	X	X
Integrity Measurement, Storage and Reporting	X	–
Protected Capabilities	X	–
Sealed/Protected Storage	X	–
Attestation	X	X
Platform Authentication	X	–
Protected Execution	X	X
Protected Input/Output	X	X
Trusted Desktop	–	X

3.3.1 Die Trusted Computing Platform (TCP)

In der Regel definiert die *Trusted Computing Platform* die für die Umsetzung des *Trusted-Computing*-Ansatzes notwendigen Erweiterungen an der Hardware des Systems. Diese Erweiterungen reichen vom Aufbringen neuer Bauteile auf die Hauptplatine eines Systems über das einfache Aufrüsten mit Erweiterungskarten bis zur Erweiterung der Funktionalität von bestehenden Bauteilen wie z. B. der CPU oder dem Chipsatz. Neben den hardwareseitigen Modifikationen ist häufig auch eine Erweiterung der entsprechenden Firmware notwendig, um die neuen Komponenten zu unterstützen. Eine mögliche Realisierung der *Trusted Computing Platform* in Form des TPM ist durch die Spezifikationen der TCG definiert und wird in Kapitel 4 beschrieben. Alternative und ergänzende Konzepte beschreibt Kapitel 5.

3.3.2 Das Trusted Operating System (TOS)

Die Definition des *Trusted Operating System* (*Trusted-OS*) ist im Vergleich zur TCP weitaus komplexer und umfasst zahlreiche Sicherheitsfunktionen und -konzepte. Eine eindeutige Definition des Begriffs oder sogar Spezifikationen, wie für die Umsetzungen einer *Trusted Computing Platform*, sind zum aktuellen Zeitpunkt nicht verfügbar. Ein *Trusted Operating System* lässt sich im Allgemeinen als eine Kombination aus einem sicheren Betriebssystemkern (*Trusted Kernel*), vertrauenswürdigen Diensten und Applikationen verstehen. Die Aufgaben eines *Trusted-OS* ergeben sich aus der Verteilung der Zuständigkeiten, wie in Tabelle 3.1 dargestellt. Eine detaillierte Beschreibung dieser Aufgaben und des daraus entstehenden Anforderungskatalogs erfolgt in Kapitel 6.

3.3.3 Die Trusted Computing Base (TCB)

Die *Trusted Computing Base* (TCB) bildet den Kern eines *Trusted Computing System*. Sie enthält alle Komponenten, deren Integrität für die Vertrauenswürdigkeit des Systems maßgeblich ist. Die Kompromittierung einer dieser Komponenten muss daher verhindert oder zumindest vom System erkannt werden. Die Kompromittierung einer Komponente außerhalb dieser TCB darf sich nur auf die Komponente selbst oder eine Gruppe von Komponenten, jedoch nicht auf das Betriebssystem auswirken. Die TCB kann als Querschnittsfunktion durch die bereits erwähnte TCP und das *Trusted-OS* verstanden werden. Sie enthält somit nicht zwangsläufig nur reine Softwarekomponenten. Die Abb. 3.2 veranschaulicht dies.

Die TCB eines Systems kann hardwareseitig beispielsweise aus der CPU, der *Memory Management Unit* (MMU) und einem TPM bestehen. Auf der Seite der Software ist der Kern des Betriebssystems die zentrale Komponente einer TCB. Beim Entwurf eines TCS ist einer der wichtigsten Grundsätze die Minimierung der

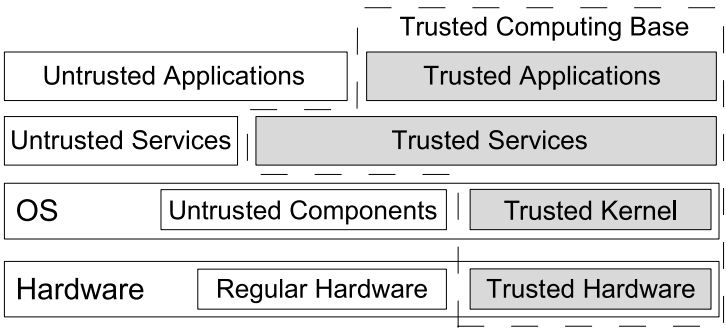

Abb. 3.2 Trusted Computing Base

Größe und somit der Komplexität der TCB. Konkret bedeutet dies vor allem eine Reduzierung der Größe des OS-Kerns, um eine Prüfung seiner korrekten Arbeitsweise zu ermöglichen. Genau hier liegt jedoch eine der Schwächen von heutigen Desktop-Betriebssystemen. Zum einen wird die durch die CPU bereitgestellte Funktion der Privilegierungsstufen nicht oder nur unzureichend verwendet und somit ein Großteil des Betriebssystems mit unnötig vielen Rechten ausgeführt. Zum anderen sind sie in der Regel in Form eines einzigen großen Kerns (Monolithischer Kernel) realisiert, wodurch sämtliche Funktionen des Betriebssystems im selben Speicherbereich ausgeführt werden und nicht voneinander isoliert sind. Noch kritischer ist jedoch der Umstand, dass der Kern durch das Laden von Treibern und Kernel-Modulen zusätzlich erweitert werden kann und auch diese Erweiterungen mit uneingeschränkten Rechten auf die Hardware des Systems zugreifen können. Dadurch gelangt nicht-vertrauenswürdiger Code in die TCB, was die Sicherstellung der korrekten Arbeitsweise unmöglich macht. So kann zum Beispiel die Integrität eines Betriebssystems durch das Installieren eines Drucker-Treibers vollständig kompromittiert werden.

3.4 Trusted Computing und Secure Computing

Wie bereits angedeutet, hat sich die Bedeutung des *Trusted-Computing*-Begriffs über die Jahre deutlich gewandelt. Stand der Begriff zunächst für die Überwachung von Zugriffsrechten durch einen Referenzmonitor und den Beweis der Korrektheit von dessen Implementierung, steht er heute stellvertretend für eine ganze Reihe von Konzepten. Als Kernkonzepte des modernen *Trusted Computing* lassen sich die Protokollierung des Systemzustands (*Integrity Measurements*) und der Nachweis der Vertrauenswürdigkeit (*Remote Attestation/Authentication*) sowie die Partitionierung des Systems in unterschiedliche Sicherheitsbereiche (*Protected Execution*) identifizieren.

Für Verwirrung sorgt oft der Begriff des *Secure Computing*. Er ist ebenfalls nicht eindeutig definiert und wird oft äquivalent zu *Trusted Computing* verwendet. Eine

exakte Abgrenzung der beiden Begriffe ist auf Grund der sich teilweise überschneidenden Ziele nur schwer möglich. Für dieses Buch ist dennoch folgende Abgrenzung gültig:

Secure Computing umschreibt alle Konzepte, die zur Erhöhung von Softwaresicherheit beitragen. Unter Sicherheit versteht der Autor in diesem Zusammenhang in erster Linie die Korrektheit und Zuverlässigkeit von Software. Somit fallen hauptsächlich Konzepte aus dem Bereich der Softwareentwicklung in diese Kategorie. Beispiele hierfür sind die Entwicklung von Betriebssystemen mit sicheren Programmiersprachen[9], die Weiterentwicklung von Compilertechniken und das automatisierte Testen von Software. Das Endziel des Secure Computing ist die Erstellung von mathematisch beweisbar sicherer Software. Da die Verifikation von Quelltext selbst für sehr kleine Programme sehr aufwendig ist, wird es in den nächsten Jahren vermutlich keine Desktop-Betriebssysteme geben, die mathematische Korrektheit vorweisen können. Die Entwicklung eines beweisbar sicheren Mikrokernels wird im Rahmen des VFiasco[10]-Projekts angestrebt.

Es ist jedoch anzumerken, dass die Verfügbarkeit von beweisbar sicherer Software einen Teil der Ansätze des *Trusted Computing* überflüssig machen würde. Es ist sogar davon auszugehen, dass viele der Konzepte im Bereich des *Trusted Computing* ihren Ursprung in ungelösten Problemen des *Secure Computing* haben. Auch wenn aktuelle Entwicklungen im Bereich des *Secure Computing* einen direkten Einfluss auf das Gebiet des *Trusted Computing* haben können, sollen im Rahmen dieses Buches in erster Linie die modernen Konzepte des *Trusted Computing* untersucht werde.

[9] Ein Beispiel hierfür ist das Forschungsbetriebssystem Singularity von Microsoft [URL03].

[10] [URL04]

Kapitel 4
Die TCP der Trusted Computing Group

Eine mögliche Umsetzung einer *Trusted Computing Platform* (TCP) ist definiert durch die Spezifikationen der *Trusted Computing Group* (TCG) [TCG*]. Sie beschreiben die Maßnahmen zur Überprüfung der Integrität von Hard- und Software durch Einsatz eines *Trusted-Platform-Moduls* (TPM). Das TPM ist eine zusätzliche Hardwarekomponente, die fest mit der Hauptplatine des Computersystems verbunden ist. Die Funktionen des TPM entsprechen in etwa denen einer *SmartCard*, mit dem Unterschied, dass das TPM an ein Computersystem gebunden ist und dieses dadurch eindeutig identifiziert werden kann. Die *SmartCard* hingegen ist nicht an ein System gebunden und wird i. d. R. zur Identifikation einer natürlichen Person verwendet. Ein wesentliches Unterscheidungsmerkmal des TPM sind die *Platform Configuration Register* (PCR); sie dienen der sicheren Speicherung von Prüfsummen, die den aktuellen Zustand des Systems repräsentieren.

Neben dem TPM benötigt eine *Trusted Computing Platform* eine weitere Komponente, die als *Core Root Of Trust For Measurement* (CRTM) bezeichnet wird. Sie ist in der Regel als Erweiterung des BIOS, und somit innerhalb der Firmware des Systems, implementiert. Die CRTM enthält die ersten Instruktionen, die beim Einschalten des Systems ausgeführt werden, und erstellt Prüfsummen über die am Startvorgang beteiligten Komponenten. Durch das Erzeugen dieser Prüfsummen initiiert die *Trusted Computing Platform* eine *Vertrauenskette* (*Chain of Trust*) vom BIOS POST[1] bis zum *Master Boot Record* (MBR) und stellt damit sicher, dass alle in dieser Betriebsstufe ausgeführten Komponenten anhand ihrer Prüfsumme eindeutig identifizierbar sind[2].

Die *Trusted Computing Platform* der TCG ist also ein Computersystem, das um ein TPM und eine CRTM erweitert wurde, und bildet dadurch den *Vertrauensanker* (*Root of Trust*) für darauf aufsetzende *Trusted-Computing*-Anwendungen. Die *Trusted Computing Platform* besteht nur aus Hardware und Firmware und arbeitet dadurch unabhängig von installierten Betriebssystemen. Implementierungen, die

[1] POST – Power-On-Self-Test

[2] Der Aufbau der Vertrauenskette wird im Abschnitt „Initialisierung der Chain of Trust“ näher erläutert.

konform zu den Spezifikationen der TCG sind, bieten die Möglichkeit, die CRTM und das TPM im BIOS zu aktivieren bzw. zu deaktivieren.

Im Rahmen dieses Kapitels werden die einzelnen Komponenten der TCP im Detail beschrieben und anschließend mit der zu Beginn der Arbeit erstellten Liste der zu erreichenden Ziele verglichen. Das Kapitel „Erweiterungen und Alternativen zur TCG“ liefert einen Überblick zu Alternativen zur TCP der TCG und zu möglichen bzw. nötigen Erweiterungen.

4.1 PC-Referenzarchitektur

Im Rahmen dieser Arbeit werden primär Ansätze zur Umsetzung einer *Trusted Computing Platform* in den Bereichen der Personal-Computer und Server-Systeme betrachtet. Umsetzungen für die Bereiche Embedded-Computing und Mobil-Computing weisen zwar durchaus Parallelen auf, werden aber in dieser Arbeit nicht näher betrachtet[3]. Aus diesem Grund wird im Folgenden eine PC-Referenzarchitektur definiert, an der die für die TCP notwendigen Erweiterungen veranschaulicht werden können.

Rechner aus den Bereichen Personal-Computing und kleinere Server-Systeme basieren in der Regel auf einer einheitlichen Architektur, die in der folgenden Abbildung vereinfacht dargestellt wird:

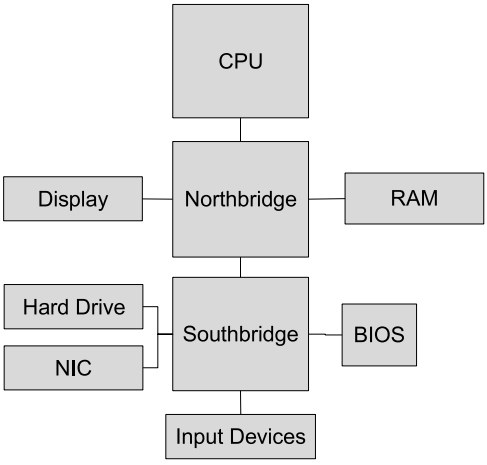

Abb. 4.1 PC-Referenzarchitektur

Southbridge und *Northbridge* sind neben der CPU ein wichtiger Bestandteil der Mainboards. Die beiden Chips werden auch als Chipsatz bezeichnet. Über den Chip

[3] Die TCG liefert auch Spezifikationen für diese Bereiche [URL05].

der *Southbridge* erfolgen der Datentransfer und die Datensteuerung zwischen peripheren Geräten (PCI-Bus, ISA-Bus, ATA, etc.) und weiteren Schnittstellen. Außerdem ist bei modernen Mainboards oft ein Teil der Peripherie schon auf der *Southbridge* integriert, z. B. der USB-Controller. *South-* und *Northbridge* kommunizieren entweder ebenfalls über PCI bzw. PCI-X oder eine proprietäre Schnittstelle. Der Chip der *Northbridge* synchronisiert den Datentransfer und die Datensteuerung zwischen CPU, Arbeitsspeicher, Cache und AGP- oder PCI-Express-Grafikkarte. In der Regel ist die Leistungsfähigkeit des Chips der *Southbridge* deutlich geringer als die der *Northbridge.*

4.2 Trusted Building Block (TBB)

Das Kernelement der TCP der TCG ist das Konzept des *Vertrauensankers* (*Root of Trust*). Ein *Vertrauensanker* ist die Basis einer *Vertrauenskette* (*Chain of Trust*) und somit der Startpunkt für die Funktionen einer *Trusted Computing Platform*. Die Basis einer Sicherheitsfunktion ist immer eine Komponente, deren Vertrauenswürdigkeit nicht durch die Plattform selbst überprüft werden kann, ihr muss deshalb bezüglich der korrekten Ausführung ihrer Aufgabe vertraut werden. Die Vertrauenswürdigkeit des *Vertrauensankers* kann dabei nur außerhalb der Trusted Platform sichergestellt werden. Dies geschieht durch digitale *Plattform-Zertifikate*[4], die durch externe Instanzen erstellt werden müssen. Die Idee des *Vertrauensankers* ist an sich kein neues Konzept der TCG, sondern Bestandteil jedes Sicherheitssystems, neu an der Umsetzung der TCG ist jedoch die Verlagerung des *Vertrauensankers* in Hardware. Dies soll zu einer Verbesserung der Vertrauenswürdigkeit beitragen, da Hardware eine weitaus geringere Angriffsfläche für Manipulationen aufweist, als eine Implementierung in Software.

Die TCG definiert drei dieser *Vertrauensanker*, den *Root of Trust for Measurement* (RTM), den *Root of Trust for Storage* (RTS) und den *Root of Trust for Reporting* (RTR). Gemeinsam bilden sie den *Trusted Building Block* (TBB). Die nachfolgende Abbildung zeigt die Zuordnung dieser *Vertrauensanker* zu den für die Realisierung zuständigen Komponenten der TCP:

Der *Trusted Building Block* ist somit die Kombination aus dem CRTM, dem TPM, der Verbindung zwischen dem CRTM und dem Mainboard sowie der Verbindung zwischen dem TPM und dem Mainboard. Daraus ergibt sich implizit auch eine Vertrauensstellung zwischen dem CRTM und dem TPM. Die Funktionen der einzelnen *Vertrauensanker* werden auf den folgenden Seiten näher erläutert.

[4] Struktur und Funktion dieser Zertifikate werden im Abschnitt „Plattform-Zertifikate (Platform Credentials)“ erläutert.

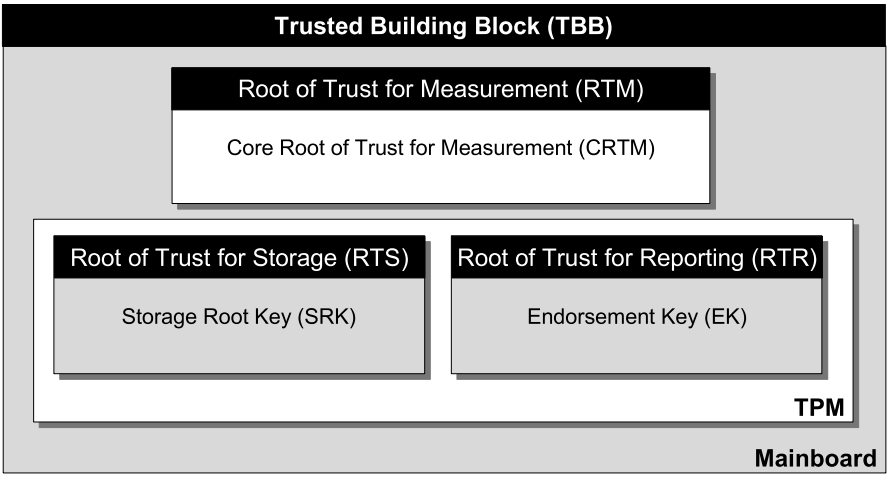

Abb. 4.2 Trusted Building Block

4.2.1 Root of Trust for Measurement (RTM)

Der erste *Vertrauensanker* ist der *Root of Trust for Measurement*, also die Basiskomponente für Integritätsmessungen (*Integrity Measurement*). Realisiert ist dieser *Vertrauensanker* durch die von den *Core Root of Trust for Measurement* erzeugten Prüfsummen. Sie stellen sicher, dass die Plattform auf Basis eines bekannten Zustands startet. Da die CRTM als erste Komponente auf dem System ausgeführt wird, ist es möglich, bereits vor der Initialisierung des BIOS Prüfsummen zu erzeugen. Diese werden benötigt um auch die Integrität der Firmware der Plattform bewerten zu können. Aufbauend auf diesem *Vertrauensanker* ist eine *Trusted Computing Platform* in der Lage, die *Vertrauenskette* zu initiieren.

4.2.2 Root of Trust for Reporting (RTR)

Neben dem RTM, der in der Regel in Form einer BIOS-Erweiterung umgesetzt ist, befinden sich die beiden weiteren *Vertrauensanker* innerhalb des *Trusted Platform Module* (TPM). Der *Root of Trust for Reporting* (RTR) ist verantwortlich für die Etablierung einer Plattform-Identität und den Schutz der übertragenen Prüfsummen während des Nachweises des Zustands der Plattform (*Remote Attestation*). Der RTR bildet eine kryptographische Identität, welche die Identifizierung und Authentifizierung des TPM und somit der Plattform ermöglicht. Hierfür ist im TPM ein eindeutiges RSA-Schlüsselpaar hinterlegt. Dieses Schlüsselpaar wird als *Endorsement Key* (EK) bezeichnet. Die *Chain of Trust* im Falle des RTR bilden die so genannten *Attestation Identity Keys* (AIK). Sie dienen als Aliase bei der Authentisierung

bzw. bei der Attestierung über ein Netzwerk. Beide Schlüsselpaare werden unter „TPM-Einheiten“ auf Seite 31 beschrieben.

4.2.3 Root of Trust for Storage (RTS)

Der *Root of Trust for Storage* (RTS) dient dem Schutz von Daten, die durch das TPM verarbeitet werden (in der Regel kryptographische Schlüssel), jedoch außerhalb des TPM gespeichert werden. Hierdurch wird die Vertraulichkeit sowie die Integrität dieser Daten sichergestellt. Der RTS ist identisch zum RTR ebenfalls ein RSA-Schlüsselpaar, welches im TPM hinterlegt ist. Dieses Schlüsselpaar wird als *Storage Root Key*[5] (SRK) bezeichnet. Im Gegensatz zum RTM und RTR ist die Integrität des RTS nicht durch die erwähnten Plattform-Zertifikate sichergestellt. Hierfür wird der SRK mit Hilfe des EK im TPM verschlüsselt und somit dessen Vertrauenswürdigkeit sichergestellt. Das Pendant zur *Chain of Trust* ist im Falle des RTS die durch die Schlüsselverwaltung des TPM aufgebaute Schlüsselhierarchie[6].

4.3 Das Trusted Platform Module (TPM)

Die Beschreibung des TPM basiert auf den umfangreichen Spezifikationen der TCG. Diese Spezifikationen liefern die Vorgabe für Hardwarehersteller für die Entwicklung konkreter TPM-Implementierungen. Da einige Abschnitte der TPM-Spezifikation keine konkrete Lösung vorschreiben, sondern eher als Empfehlung zu sehen sind, ist es möglich, dass ein TPM von dem beschriebenen Funktionsumfang und der beschriebenen Funktionsweise abweicht. Zum aktuellen Zeitpunkt sind TPMs von folgenden Herstellern verfügbar:

- Amtel
- Infineon
- Sinosun
- STMicroelectronics
- Winbond/National Semiconductors

Das TPM ist ebenso wie der CRTM eine zusätzliche Komponente, die mit dem Mainboard der Plattform verbunden wird. Im Gegensatz zur CRTM handelt es sich beim TPM jedoch um einen zusätzlichen Hardwarebaustein, der über den LPC-Bus[7] mit der *Southbridge* verbunden wird. Über diesen LPC-Bus findet die Kommunikation mit dem TPM auf Basis von Steuerbefehlen statt. Einen Überblick über die

[5] Der Storage Root Key (SRK) wird später im Buch näher beschrieben.

[6] Die Schlüsselverwaltung wird im Abschnitt „TPM-Schlüsseltypen und Schlüsselverwaltung“ beschrieben.

[7] Der Low-Pin-Count-Bus ist ein serieller Bus zur Anbindung von Hardwarebausteinen.

implementierten Befehle liefert die TCG-Spezifikation [TCG2006-5]. Im Rahmen dieser Arbeit werden die Befehle als TPM-Kommandos bezeichnet und entsprechen der folgenden Darstellung:

TPM_GetRandom

Hier am Beispiel des Kommandos zum Abfragen von durch das TPM erzeugten Zufallszahlen. Die Übergabe- und Rückgabe-Parameter der Kommandos werden zugunsten der Übersichtlichkeit nicht angegeben, sind aber in der Spezifikation zu finden. Die folgende Abbildung zeigt die Integration von CRTM und TPM in die bereits erwähnte Referenzarchitektur:

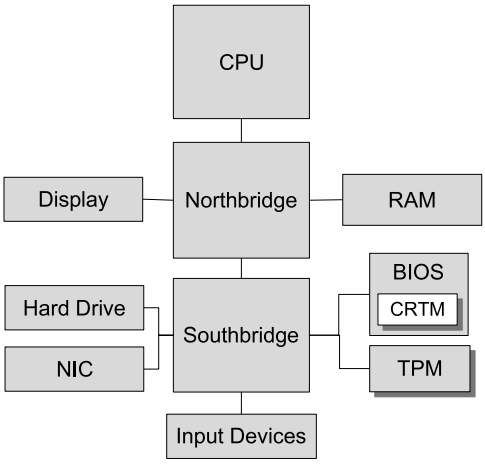

Abb. 4.3 Referenzarchitektur mit TPM

Das TPM ist in Form eines Mikrocontrollers realisiert und bildet somit ein so genanntes *Ein-Chip-Computersystem*. Mikrocontroller integrieren hierfür nahezu sämtliche Komponenten (wie z. B. den Hauptprozessor, den Programmspeicher, den Arbeitsspeicher sowie Ein-/Ausgabe-Schnittstellen) auf einem einzigen Chip. Dadurch kann sichergestellt werden, dass sicherheitskritische Operationen nur in diesem geschlossenen System durchgeführt werden und von außen nur schwer angreifbar sind. Das TPM bildet dadurch eine mögliche Umsetzung der *Protected Capabilities*. Eine ähnliche Architektur ist auch die Grundlage für die im IT-Sicherheitsumfeld häufig eingesetzten Prozessor-Chipkarten (oft auch als *Smart-Card* oder *Integrated Circuit Card* bezeichnet), weshalb das TPM oft fälschlicherweise mit solchen Chipkarten gleichgestellt wird. Die folgende Abbildung veranschaulicht die interne Struktur eines TPM:
Zusätzlich zu den Standardbausteinen eines Mikrocontrollers (grau) verfügt ein TPM über spezielle Erweiterungen (weiß) für die sichere Speicherung erzeugter Prüfsummen (PCR) und für die Bereitstellung kryptographischer Funktionen. Das folgende Kapitel beschreibt diese TPM-spezifischen Funktionen im Detail.

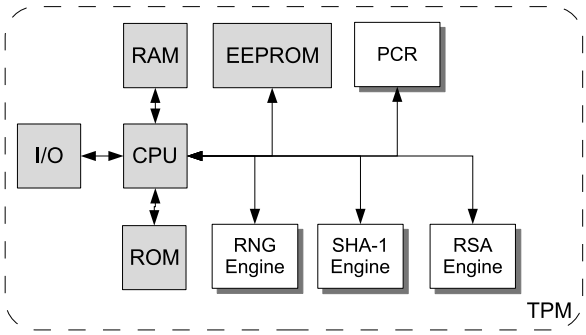

Abb. 4.4 Blockschaltbild eines TPM

4.3.1 TPM-Einheiten

Die Einheiten des *Trusted Platform Module* (TPM) können in drei logische Bereiche aufgeteilt werden. Man unterscheidet zwischen den funktionalen Einheiten (*functional units*), die die verschiedenen kryptographischen Funktionen bereitstellen, und dem Speicherbereich. Der Speicherbereich teilt sich in einen nichtflüchtigen (*non-volatile*) und einen flüchtigen (*volatile*) Speicher. Die folgende Abbildung zeigt die Aufteilung und die jeweils enthaltenen Einheiten.

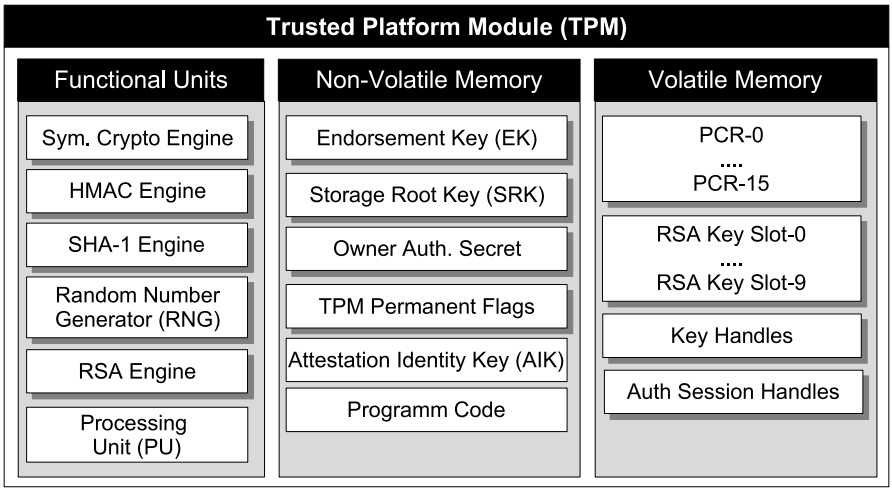

Abb. 4.5 TPM-Komponenten

4.3.1.1 Symmetrische Verschlüsselung (Sym. Crypto Engine)

Das TPM verwendet symmetrische Kryptographie zur Verschlüsselung von TPM-intern genutzten Authentisierungsdaten (z. B. Passwörter, PINs), zum Schutz der Daten auf den Kommunikationswegen zwischen TPM und System- oder Softwarekomponente und zur Verschlüsselung von Datenblöcken, die außerhalb des TPM abgelegt werden. Nach TCG-Spezifikation ist lediglich die Verwendung eines Vernam-Einwegpasswortes (*one-time-pad*) mit einer Länge von 160 Bit in Kombination mit einer XOR-Operation als kryptographisches Verfahren verpflichtend, es steht den Herstellern jedoch frei, weitere symmetrische Verfahren (z. B. AES) zu integrieren. Sämtliche symmetrische Operationen werden nur TPM-intern verwendet und nicht den externen Komponenten zur Verfügung gestellt.

4.3.1.2 HMAC-Komponente (HMAC engine)

Beim HMAC (*keyed-hash message authentication code*) handelt es sich um ein Verfahren zur Erzeugung eines *Message Authentication Code* (MAC). Ein MAC basiert auf einer kryptographischen Hash-Funktion und einem geheimen symmetrischen Schlüssel und wird hauptsächlich zur Integritätsprüfung von Nachrichten verwendet. Im Falle des TPM kommt der HMAC bei der Kommunikation mit anderen Teilen des Systems zum Einsatz und erfüllt hierbei zwei Aufgaben. Er stellt sicher, dass die Nachricht an das TPM auf dem Kommunikationskanal nicht verändert wurde, und authentisiert den Absender der Nachricht durch einen Vergleich des verwendeten symmetrischen Schlüssels mit dem beim Aufbau des Kommunikationskanals erzeugten Sitzungsschüssel[8]. Die HMAC-Funktionalität ist ebenfalls nur TPM-intern verfügbar, die Erzeugung durch HMAC geschützter Nachrichten ist Aufgabe der sendenden Einheit.

4.3.1.3 Erzeugung von Zufallszahlen (Random Number Generator)

Die TCG-Spezifikation schreibt vor, dass jedes TPM über einen Zufallszahlengenerator in Hardware verfügen muss. Um die Zufälligkeit der ausgegebenen Daten gewährleisten zu können, verwendet das TPM als Entropie die Eigenschaften der Systemhardware, wie zum Beispiel das thermische Rauschen[9] der elektronischen Bauteile. Diese Zufallszahlen werden TPM-intern für die Erzeugung von kryptographischen Schlüsseln verwendet. Diese Zahlen können jedoch auch außerhalb des TPM verwendet werden; hierzu dient das Kommando *TPM_GetRandom*, mit dem bis zu 32 Bytes an Zufallsdaten pro Aufruf ausgelesen werden kön-

[8] Der Aufbau eines gesicherten Kommunikationskanals wird im Abschnitt „TPM-Zugriffskontrolle und Kommunikationsprotokoll“ beschrieben.

[9] Thermisches Rauschen bezeichnet ein weißes Rauschen, das sich aus der thermischen Bewegung der Ladungsträger in elektrischen Schaltkreisen erklärt.

nen. Diese Werte können dann wiederum als Eingabedaten (Salt) für Software-Zufallsgeneratoren verwendet werden, welche in der Regel schneller sind als der Generator des TPM.

4.3.1.4 Kryptographische Hash-Funktion (SHA-1-Engine)

Die SHA-1-Komponente (*Secure Hash Algorithm*) implementiert eine kryptographische Hash-Funktion und erzeugt eine Ausgabe der Länge 160 Bit. Die Komponente wird hauptsächlich von der HMAC-Komponente zur Erzeugung von MACs und zur Aktualisierung der *Platform Configuration Register* (PCR) während der Bootphase und zur Laufzeit verwendet. Auch ein direkter Zugriff auf die Funktion mittels TPM-Kommando ist möglich, dies sollte jedoch nur dann in Anspruch genommen werden, wenn keine anderen Optionen verfügbar sind. Da die TCG keine Vorgaben bezüglich der Leistungsfähigkeit der Implementierung macht, ist zu erwarten, dass eine Softwareimplementierung bessere Ausführungszeiten liefert und deshalb zu bevorzugen ist. Das Kommando *TPM_SHA1Start* startet die Berechnung eines Hash-Wertes und erwartet die Übergabe der Eingangsdatenblöcke mittels *TPM_SHA1Update*. Sind alle relevanten Blöcke übertragen, schließt *TPM_SHA1Complete* den Vorgang ab und liefert den 160-Bit-Hash-Wert zurück. Während dieses Vorgangs verweigert das TPM die Ausführung anderer Kommandos. Soll der berechnete Wert in einem der PCR des TPM abgelegt werden, wird der Vorgang mittels *TPM_CompleteExtend* terminiert. Die beschriebenen Kommandos erfordern keine Authentisierung des Aufrufers und können dadurch von sämtlichen Hard- und Softwarekomponenten des Systems verwendet werden.

4.3.1.5 Kryptographische Funktionen und Schlüsselerzeugung (RSA-Engine)

Die RSA-Komponente des TPM implementiert den asymmetrischen Algorithmus RSA. Die TCG-Spezifikation schreibt die Unterstützung von 512, 768, 1024 und 2048 Bit langen Schlüsseln vor, empfiehlt jedoch, ausschließlich Schlüssel der Länge 2048 Bit zu verwenden. Die *RSA-Engine* wird zur Ver- und Entschlüsselung und für digitale Signaturen verwendet. Beide Funktionen können sowohl TPM-intern als auch außerhalb verwendet werden. Im Falle der digitalen Signatur können beliebige Daten mittels *TPM_Sign* durch das TPM signiert werden. Das Ausgabeformat des Signaturvorgangs ist durch IEEE P1363 definiert.

Die asymmetrische Ver- und Entschlüsselung dient zusammen mit dem *Storage Root Key* in erster Linie dem Aufbau einer hierarchischen Schlüsselverwaltung zur sicheren Ablage von symmetrischen und asymmetrischen Schlüsseln sowie kleineren Datenmengen. Eine weitere Aufgabe ist die Erzeugung von sicheren RSA-Schlüsselpaaren innerhalb des TPM. Hierzu greift die RSA-Komponente auf die Funktion des Zufallsgenerators zurück. Die durch das TPM erzeugten Schlüssel werden innerhalb der Schlüsselverwaltung geschützt abgelegt. Alle von der RSA-Komponente verwendeten Schlüssel müssen vor ihrer Verwendung aus der Schlüs-

selverwaltung in einen der RSA-Key-Slots geladen und entschlüsselt werden. Hierdurch wird verhindert, dass der private Teil des Schlüsselpaares das TPM im unverschlüsselten Zustand verlässt.

4.3.1.6 Endorsement Key (RTR)

Der *Endorsement Key* (EK) ist ein RSA-Schlüsselpaar mit der Länge 2048 Bit. Der *Endorsement Key* wird während des Fertigungsprozesses innerhalb des TPM erzeugt und im nichtflüchtigen Speicherbereich abgelegt. Hierzu verwendet der Hersteller das Kommando *TPM_CreateEndorsementKeyPair*. Einmal eingerichtet, kann der *Endorsement Key* nicht mehr gelöscht oder geändert werden[10]. Um sicherzustellen, dass der hinterlegte EK nur von einem autorisierten Hersteller und im Rahmen eines definierten Prozesses erzeugt wurde, verfügt er über ein EK-Zertifikat. Das EK-Zertifikat ist Teil der so genannten *Platform Credentials*, deren Funktion ab Seite 61 beschrieben wird. Es ist nicht möglich, den privaten Teil des EK auszulesen, auch nicht durch den Eigentümer. Der öffentliche Teil kann, sofern nicht deaktiviert, von jeder Instanz mittels *TPM_ReadPubek* ausgelesen werden.

Der EK ist für jedes TPM einzigartig und bietet dadurch, zusammen mit den *Platform Credentials*, die Möglichkeit, von einem TPM signierte Nachrichten zu identifizieren und die Vertrauenswürdigkeit des TPM zu überprüfen. Die Möglichkeit, die Identität eines TPM eindeutig zu bestimmen, ist jedoch unter Datenschutzaspekten als äußerst kritisch einzustufen, da hierdurch u. a. die Möglichkeit gegeben ist, ein Bewegungsprofil der Plattform zu erstellen. Deshalb wurden mit den *Attestation Identity Keys* (AIK) weitere Schlüssel zum Signieren von Nachrichten eingeführt. Der *Endorsement Key* wird daher nur zur Entschlüsselung des im TPM hinterlegten Eigentümer-Passwortes und zur Entschlüsselung und Signatur von Nachrichten während der AIK-Erzeugung verwendet.

4.3.1.7 Storage Root Key (RTS)

Der *Storage Root Key* (SRK) ist ebenfalls ein RSA-Schlüsselpaar der Länge 2048 Bit. Der SRK wird während der Einrichtung eines TPM-Eigentümers durch das TPM erzeugt und im nichtflüchtigen Speicherbereich abgelegt. Der SRK ist neben dem EK der einzige Schlüssel, der laut TCG-Spezifikation zwingend im TPM und nicht auf einem externen Speichermedium abgelegt werden muss. Analog zum EK ist es auch beim SRK nicht möglich den privaten Teil des Schlüssels auszulesen. Allerdings kann der EK durch Entfernen und die anschließende Einrichtung eines neuen Eigentümers beliebig oft geändert werden. Der SRK wird verwendet, um weitere kryptographische Schlüssel oder Daten sicher außerhalb des TPM zu speichern und bildet damit den RTS (*Root of Trust for Storage*).

[10] Nur in der TPM-Spezifikation Version 1.1b. Wie ein EK entfernt werden kann, ist auf Seite 57 beschrieben.

4.3.1.8 Eigentümer-Passwort (Owner Authorization Secret)

Das *Owner Authorization Secret* ist eine 160 Bit lange Hash-Repräsentation des Eigentümer-Passwortes, welches während der Einrichtung des Eigentümers im TPM abgelegt ist. Das Passwort wird mit dem öffentlichen Teil des *Endorsement Key* verschlüsselt und kann daher nur vom TPM selbst zur Authentisierung des Eigentümers entschlüsselt werden.

4.3.1.9 TPM-Konfigurationsspeicher (TPM-Permanent Flags)

Die *Permanent Flags* werden verwendet, um die Konfiguration des TPM dauerhaft zu speichern. Sie befinden sich im nichtflüchtigen Speicherbereich des TPM. Neben dem gewünschten Arbeitszustand[11] können auch noch erweiterte Konfigurationsoptionen[12] gesetzt werden, die sich auf den Funktionsumfang des TPM auswirken.

4.3.1.10 Attestation Identity Key (AIK)

Der *Attestation Identity Key* (AIK) ist ein RSA-Schlüsselpaar mit der Länge 2048 Bit. Wie bereits beschrieben, ist die Möglichkeit, die Vertrauenswürdigkeit eines Systems über ein Netzwerk abzufragen (*Remote Attestation*), eines der grundlegenden Konzepte des *Trusted Computing*. Werden die während dieses Vorgangs ausgetauschten Nachrichten mit dem *Endorsement Key* des TPM signiert, ist es möglich, das System bei erneuter Durchführung des Vorgangs eindeutig zu identifizieren. Um dieses Datenschutzproblem zu vermeiden, werden die Nachrichten mit einem Schlüssel (AIK) signiert, der nicht mit dem EK in Verbindung gebracht werden kann, und durch eine vertrauenswürdige dritte Partei (*Trusted Third Party*) signiert wurde. Für jede *Remote Attestation* kann ein neuer temporärer AIK verwendet werden. Ein AIK kann zur späteren Verwendung innerhalb der Schlüsselverwaltung des TPM verschlüsselt abgelegt werden. Er wird somit in der Regel nicht im nichtflüchtigen Speicherbereich des TPM, sondern auf einem beliebigen externen Speichermedium abgelegt.

4.3.1.11 Platform Configuration Register (PCR)

Die *Platform Configuration Registers* (PCR) sind 160 Bit lange Speicherbereiche im flüchtigen Speicher des TPM. Sie werden verwendet, um SHA1-Hash-Werte von beliebigen Datenblöcken (z. B. Boot-Loader-Image, Betriebssystemkern, Treiber oder Applikationen) während des Systemstarts oder zur Laufzeit zu speichern. Diese Werte repräsentieren den aktuellen Zustand des Systems und erlauben eine

[11] Beschrieben unter „Betriebszustände des TPM (Opt-In)“.

[12] Beschrieben unter „Erweiterte TPM-Konfiguration (Opt-In)“.

Bewertung der Integrität des Systems. Um eine Manipulation dieser Register zu verhindern, ist es nicht möglich, einen eingetragenen Hashwert zu entfernen. Beim Update eines Registers wird der neue Hashwert mit dem bereits vorhandenen Wert entsprechend der folgenden Formel verknüpft:

$$\mathrm{PCR}_i\,New = \mathrm{HASH}(\mathrm{PCR}_i\,Old\,\mathrm{value}\,||\,\mathrm{value\ to\ add})$$

Diese Funktion ermöglicht zusätzlich die Speicherung mehrere Prüfsummen innerhalb eines Registers. Um die einzelnen Eingangswerte und den zeitlichen Ablauf der Erzeugung dieser Hashwerte nachvollziehen zu können, wird zusätzlich bei jedem Update eines Registers ein entsprechender Eintrag im *Stored Measurement Log*[13] (SML) anlegt.

Da sich die PCR im flüchtigen Speicher des TPM befinden, werden sie beim Ausschalten des Systems gelöscht und müssen bei jedem Systemstart neu berechnet und abgelegt werden. Nach aktueller TCG-Spezifikation (V1.2) besitzt jedes TPM mindestens 23 dieser Register, es ist jedoch den Herstellern überlassen, mehr Register zur Verfügung zu stellen. Die TPM-Spezifikation der TCG sieht folgende Verwendung der PCR vor:

Tabelle 4.1 Verwendung der Platform Configuration Registers

PCR Index	PCR Verwendung
0	CRTM, BIOS und Host Platform Extensions
1	Host Platform Configuration
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL Code (usually the MBR)
5	IPL Configuration and Data (for use by the IPL Code)
6	State Transition and Wake Events
7	Host Platform Manufacturer Control
8–15	Defined for use by the Static Operating System
16	Debug
17–23	Defined for use by the Dynamic Operating System

Die Register 0–7 repräsentieren den pre-OS-Zustand des Systems und werden während des Startvorgangs durch den CRTM und das BIOS erzeugt. Register 8–15 sind für Prüfsummen über die restlichen Komponenten des Systemstarts reserviert. Um welche Komponenten es sich hierbei handeln muss, ist nicht näher spezifiziert und obliegt der Zuständigkeit des Betriebssystems. Das Register 4 enthält die Prüfsummen über den Code, der den Ausführungspfad an das Betriebssystem übergibt, in der Regel ist dies der ausführbare Teil des *Master Boot Record* (MBR). Eine detaillierte Betrachtung der durch die TCP während des Startvorgangs erzeugten Prüfsummen ist im Abschnitt „Initialisierung der Chain of Trust“ zu finden.

[13] Der Stored Measurement Log ist Teil des Spezifikation des Trusted Software Stack [TCG2006-2].

4.3.2 TPM-Zugriffskontrolle und Kommunikationsprotokoll

Jeder Zugriff auf Befehle das TPM, die hinsichtlich Sicherheit oder Datenschutz als kritisch einzustufen sind, darf nur nach vorausgehender Autorisierung ausgeführt werden. Hierfür muss jedes Subjekt (Prozess, Thread oder embedded controller), das mit den TPM kommunizieren möchte, zunächst einen sicheren Kanal aufbauen. Durch die Angabe eines Kennworts ist das TPM in der Lage, den Aufrufer zu authentisieren und gleichzeitig zu autorisieren. Das verwendete Kennwort (*Authorization Secret*) wird durch die *SHA-Engine* des TPM auf einen 160 Bit langen Wert abgebildet. Jedes der durch das TPM verwalteten Objekte (z. B. Schlüssel, Befehle oder das TPM selbst) kann mit einem Passwort verknüpft werden. Ein Beispiel hierfür ist das Eigentümer-Passwort (*Owner Authorization Secret*), welches während des *Take-Ownship*-Prozesses erzeugt und direkt im TPM hinterlegt wird. Durch Angabe des Eigentümer-Passworts erlangt ein Subjekt Zugriff auf alle Befehle des TPM. Ein weiteres Beispiel hierfür wäre ein Passwort, welches nur Zugriff auf die *SHA-Engine* und *RSA-Engine* und einen bestimmten Schlüssel unterhalb des SRK erlaubt. Im Gegensatz zum Eigentümer-Passwort sind Schlüssel-Passwörter außerhalb des TPM direkt zusammen mit dem entsprechenden Objekt hinterlegt.

Jeder der aktiven Kommunikationskanäle wird durch ein so genanntes *Authorized Session Handle* repräsentiert. Die folgende Tabelle beschreibt die einzelnen Felder dieses Handles:

Tabelle 4.2 Authorized Session Handle

Feldname	Beschreibung
Session ID	Eindeutige ID des aktiven Kanals
Nonce	Zufallszahl zur Verhinderung von Replay-Attacken und Man-In-The-Middle-Angriffen. Wird für jedes Nachricht/Antwort-Paar neu erstellt.
Message Digest	Hashwert über alle im Rahmen der Sitzung ausgetauschten Nachrichten. Wird nach den Nachrichten jeweils erneuert.
Ephemeral Secret	Einmalpasswort zur wiederholten Autorisierung und optional zur Verschlüsselung der Nachrichten.

Neben der Autorisierung des Aufrufers ist die wesentliche Aufgabe eines authentisierten Kanals die Sicherstellung der Integrität von übertragenen Nachrichten. Diese Funktion wird als *Command Validation* bezeichnet und stellt sicher, dass die Integrität der Nachricht auch im Falle einer kompromittierten Komponente im Kommunikationspfad[14] gewährleistet werden kann. Die für den Aufbau des Kommunikationskanals zuständigen Protokolle verwenden hierfür eine Implementierung eines HMAC (*keyed-hash message authentication code*) nach RFC-2104[15]:

[14] In der Regel wird für die Kommunikation zwischen Subjekten und dem TPM ein Trusted Software Stack (TSS) verwendet. Die Architektur eines TSS wird im Abschnitt „Trusted Software Stack“ auf Seite 75 erläutert.

[15] [URL06]

H(K XOR opad, H(K XOR ipad, text))

Die TCG spezifiziert zwei unterschiedliche *Command-Validation*-Protokolle, das *Object-Independent Authorization Protocol* (OIAP) und das *Object-Specific Authorization Protocol* (OSAP).

4.3.2.1 OIAP

Das *Object-Indenpendent Authorization Protocol* ermöglicht den Aufbau eines integeren Kommunikationskanals unabhängig vom verwendeten TPM-Objekt. Wurde der Kanal aufgebaut, kann die dabei erzeugte Zufallszahl (Nonce) für den autorisierten Zugriff auf alle durch das TPM verwalteten Objekte verwendet werden. Für die Übermittlung der für den Zugriff auf die Objekte notwendigen Passwörter wird somit immer derselbe Kanal verwendet. Ein mit dem OIAP aufgebauter Kanal bleibt bis zur Terminierung durch das Subjekt oder das TPM unbegrenzt gültig. Eine OIAP-Sitzung wird mittels des TPM_OIAP-Kommandos initiiert.

4.3.2.2 OSAP

Das *Object-Specific Authorization Protocol* wird für den Aufbau eines integeren Kommunikationskanals im Kontext eines einzelnen TPM-Objekts verwendet. Die erzeugte Zufallszahl autorisiert das Subjekt somit für die Ausführung mehrerer TPM-Kommandos, jedoch beschränkt auf die Verwendung des einen Objekts. Dies hat den Vorteil, dass für sämtliche Operationen auf einem Objekt nur einmalig ein Kennwort angegeben werden muss. Hierfür wird aus dem Objekt-Passwort ein für die Sitzung gültiges Einmalpasswort generiert und für alle folgenden Autorisierungen verwendet. Dieses Einmalpasswort kann optional auch zur Verschlüsselung der übertragenen Nachrichten verwendet werden um die Geheimhaltung der Informationen zu gewährleisten. Der Aufbau einer OSAP-Sitzung wird innerhalb des *TPM_OSAP-Kommandos* auf das gewünschte TPM-Objekt verwiesen.

Die meisten der TPM-Kommandos erlauben die Verwendung beider *Command-Validation*-Protokolle. In der Regel wird jedoch das OAIP bevorzugt, da es für die Verwendung mehrerer TPM-Objekte nur eine einzelne Sitzung aufbauen muss. Für Kommandos, die Passwörter für Objekte einrichten oder verändern, ist die Verwendung einer OSAP-Sitzung aufgrund der optionalen Verschlüsselung verpflichtend.

Wurde eine Sitzung erfolgreich aufgebaut, können TPM-Kommandos an das TPM übertragen werden. Alle Kommandos besitzen das gleiche Format und unterscheiden sich nur in der Anzahl der verfügbaren Ein- bzw. Ausgabeparameter. Die folgende Tabelle beschreibt die Felder eines TPM-Kommandos:

Tabelle 4.3 TPM-Kommando-Felder

Feldname	Beschreibung
Message Container	Enthält Informationen zum Nachrichtentyp, der Größe der Nachricht und der Formatierung
TPM Command	Name des Kommandos, Ein- bzw. Ausgabeparameter und der Return Code
Session State	ID der zu verwendenden Session, Control Flags und Hashwert der Nachricht

4.3.2.3 Proof of Physical Presence

Eine weitere Form der Zugriffskontrolle ist die Überprüfung des physikalischen Zugriffs auf die Hardware des Systems. Hierfür muss beim Start des Systems eine Interaktion mit der Plattform stattfinden, um ein entsprechend gesichertes Kommando ausführen zu können. Die TCG spezifiziert zwei mögliche Realisierungen dieser Funktion. Die erste und sicherere Lösung ist die Verbindung einer bestimmten Taste auf der Tastatur (z. B. bei T40-Thinkpads die FN-Taste) mit einem Steuersignal für das TPM. Hierbei muss beim Einschalten des Systems die entsprechende Taste betätigt werden. Die zweite Lösung ist die Signalisierung des physikalischen Zugriffs mittels eines TPM-Kommandos (*TSC_PhysicalPresence*) durch eine Softwarekomponente. In diesem Fall muss jedoch sichergestellt sein, dass die Softwarekomponente diese Operation nur ausführen kann, nachdem eine Interaktion mit dem Benutzer stattgefunden hat. Die TCG empfiehlt die Verwendung des CRTM für diese Aufgabe, da er Teil des *Trusted Building Block* (TBB) ist.

Diese Form der Zugriffskontrolle wird z. B. bei der Aktivierung des TPM verwendet, um sicherzustellen, dass das TPM nicht ohne Wissen bzw. Interaktion des Besitzers verwendet werden kann. Die Implementierung dieser Sicherheitsfunktion wird durch die Spezifikationen der TCG vorgeschrieben und ist Aufgabe des Plattform-Herstellers[16]. Die folgende Tabelle listet alle Kommandos, deren Ausführung einen Nachweis des physikalischen Zugriffs erfordern:

Tabelle 4.4 TPM-Kommandos mit Nachweis des physikalischen Zugriffs

TPM-Kommando	Beschreibung
TPM_PhysicalEnable	Einschalten des TPM-Chips
TPM_PhysicalDisable	Ausschalten des TPM-Chips
TPM_PhysicalSetDeactivated	Deaktivieren des TPM
TPM_SetTempDeactivated	Temporäre Deaktivierung des TPM
TPM_SetOperatorAuth	Setzen des Operator-Passworts
TPM_ForceClear	Rücksetzung des TPM in den Auslieferungszustand ohne Angabe des Eigentümer-Passworts
TPM_CreateRevocableEK	Generierung eines widerruflichen Endorsement Keys
TPM_RevokeTrust	Löschen des Endorsement Key[17]

[16] Für Aufregung sorgte die Integration von TPM-Chips in einige Modelle des MacBook und des MacBook Pro von Apple ohne die Umsetzung dieser Schutzfunktion.

[17] TPM_RevokeTrust ist ein optionales Kommando. Mehr Information zur Entfernung des Endorsement Key ist ab Seite 60 zu finden.

4.3.3 TPM-Initialisierung

Bevor das TPM verwendet werden kann, muss es zunächst in einen Zustand überführt werden, in dem es seine Funktionalität anderen Systemkomponenten zur Verfügung stellen kann. Das TPM durchläuft hierbei die Stationen *Power off*, *Initialization* und *Operational*. Das folgende Diagramm zeigt die möglichen Zustände und deren Übergänge:

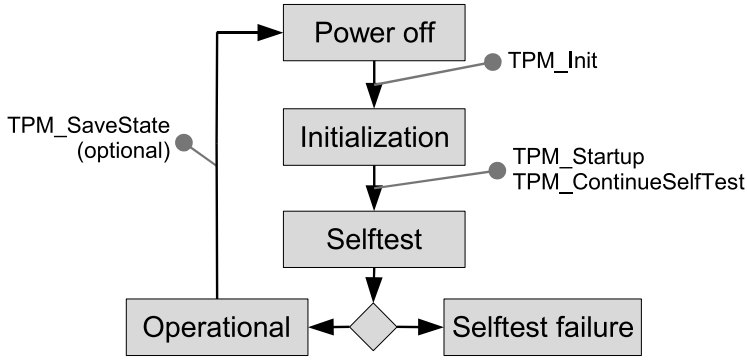

Abb. 4.6 TPM-Lebenszyklus

Das erste Kommando, das an das TPM gesendet wird, ist immer *TPM_Init*. Hierbei handelt es sich nicht um ein Kommando von einer Softwarekomponente, sondern um einen in Hardware verankerten Prozess. Dieses Kommando wird bei jedem Start des Systems oder nach einem Hardware-Reset ausgeführt. Hierdurch wechselt das TPM in den Zustand *Initialization,* in dem es auf die Ausführung weiterer Kommandos wartet.

Das zweite Kommando in diesem Prozess ist *TPM_Startup*, es muss von einer Komponente des Systems an das TPM gesendet werden, welche für die Verwendung eines TPM in seiner Funktion erweitert wurde. Dies ist in der Regel das BIOS oder die EFI[18], es ist aber auch denkbar, dass diese Aufgabe vom Prozessor oder Chipsatz übernommen wird. Diese Erweiterung entspricht dem bereits beschriebenen CRTM. *TPM_Startup* kennt mehrere Modi, in denen das TPM gestartet werden kann:

- **„Clear"-Start**
  Entspricht dem Normalfall, was bedeutet, dass alle Konfigurationsparameter des TPM auf die Werte im nichtflüchtigen Speicherbereich zurückgesetzt werden. Sämtliche Bereiche des flüchtigen Speicherbereichs sind leer.

- **„Save"-Start**
  Wird ausgeführt, wenn das System aus einem Stromsparmodus (Standbymodus oder Ruhezustand) heraus gestartet wird. Damit dieser Modus gewählt werden

[18] Extensible Firmware Interface: Potentieller Nachfolger des BIOS

kann, muss der Zustand des TPM zuvor mittels *TPM_SaveState* gesichert werden. Wird es erfolgreich ausgeführt, befindet sich das TPM im selben Zustand wie vor dem Stromsparmodus.

- **„Deactivated"-Start**
  Hierbei wird das TPM ausgeschaltet und kann bis zur nächsten Ausführung des *TPM_Init* nicht mehr verwendet werden.

Im Falle einer erfolgreichen Ausführung von *Clear*- oder *Save*-Start veranlasst die CRTM-Erweiterung durch Ausführung von *TPM_ContinueSelfTest* einen Selbsttest des TPM. Schlägt dieser Test fehl, wechselt das TPM in den *Self-Test-Failure*-Modus, in dem es nur das Auslesen des Testergebnisses mittels *TPM_GetTestResult* erlaubt. Dieser Report ist herstellerspezifisch und enthält in der Regel Informationen zur Fehlersuche. Wird der Test erfolgreich abgeschlossen, wechselt das TPM in den Zustand *Operational* und ist damit für das System verfügbar und kann verwendet und konfiguriert werden. Welche Funktionen nun zur Verfügung stehen, hängt vom konfigurierten Betriebszustand des TPM ab.

4.3.4 Betriebszustände des TPM (Opt-In)

Ein wichtiger Punkt in der TPM-Spezifikation ist die volle Kontrolle über das TPM durch den Eigentümer der Plattform. Deshalb muss gewährleistet sein, dass nur der Eigentümer in der Lage ist, den Zustand (*operation mode*) des TPM zu ändern. Er soll die Wahl haben, ob und wie das TPM verwendet werden kann. Der gewählte Zustand muss dauerhaft im TPM hinterlegt werden und darf nicht durch einen Reset verloren gehen, weshalb die Zustände als Teil der *Permanent Flags* im nichtflüchtigen Speicherbereich des TPM angelegt sind. Kritische Zustandsänderungen dürfen nicht von Software ausgeführt werden, deshalb sind sie nur mit physikalischem Zugriff auf die Plattform durchführbar. Die Abb. 4.7 beschreibt die hierbei möglichen Zustände.

Das TPM besitzt drei getrennte Zustandspaare *Disabled/Enabled*, *Deactivated/Activated* und *Owned/Unowned*. Werden diese Paare kombiniert, ergeben sich hieraus acht mögliche Zustände (*S1–S8*). Die Zustände *S2* und *S6* haben insofern eine Sonderstellung, als dass sie sich in ihrem Funktionsumfang nicht von den Zuständen *S4* bzw. *S8* unterscheiden. Ist das TPM im Zustand *Disabled*, kann auch nicht auf den größeren Funktionsumfang des Zustands *Activated* zugegriffen werden. Da der Zustand *Activated* jedoch auch beim Wechsel in den Zustand *Disabled* erhalten bleibt, sind auch diese beiden Zustände möglich.

Im Folgenden werden die Zustandspaare *Disabled/Enabled* und *Deactivated/Activated* hinsichtlich resultierendem Funktionsumfang und korrelierenden TPM-Kommandos näher erläutert. Ebenfalls werden die möglichen Zustandstransformationen aufgezeigt. Die beiden weiteren Zustände (*Owned/Unowned*) werden im Abschnitt „TPM Eigentümer einrichten und entfernen" (S. 44) beschrieben.

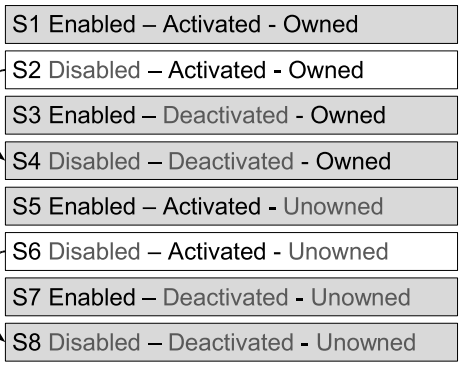

Abb. 4.7 Zustände des TPM

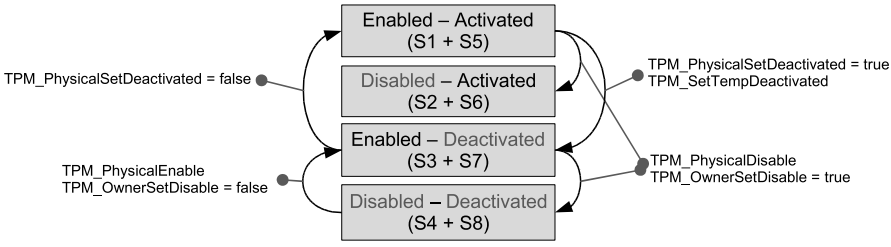

Abb. 4.8 Zustandsübergänge des TPM

4.3.4.1 Disabled/Enabled (Ausgeschaltet/Eingeschaltet)

Ist das TPM im Zustand *Disabled*, sind alle Funktionen des TPM nicht verfügbar. Eine Ausnahme hierbei sind die Kommandos, die das Abfragen von TPM-Informationen erlauben (z. B. Versionsnummer und implementierte Funktionen) und Kommandos, die das TPM in den Zustand *Enabled* überführen. Der Wechsel zwischen den beiden Zuständen hat keine Auswirkung auf die vom TPM verwalteten Objekte oder die *Permanent Flags*. Im Zustand *Enabled* sind alle Funktionen des TPM verfügbar, sofern sie nicht durch die Zustände *Deactivated* oder *Unowned* blockiert werden.

Wurde im TPM noch kein Eigentümer eingerichtet, befindet es sich also im Zustand *Unowned*, kann die Zustandstransformation nur mittels des TPM-Kommandos *TPM_PhysicalEnable* bzw. *TPM_PhysicalDisable* durchgeführt werden. Hierbei muss zusätzlich zur Ausführung durch z. B. das BIOS der Beweis erbracht werden, dass der Eigentümer physikalischen Zugriff auf das System hat. Dies wird in der Regel durch Halten einer Taste während des Systemstarts erreicht (z. B. die FN-Taste bei Notebooks). Besitzt das TPM bereits einen Eigentümer, kann dieser die Zustandstransformation auch ohne physikalischen Zugriff durchführen. Hierzu dient das Kommando *TPM_OwnerSetDisable,* welches durch das Passwort des Eigentümers (*Owner Auth Secret*) geschützt ist.

4.3.4.2 Activated/Deactivated (Aktiviert/Deaktiviert)

Im Zustand *Deactivated* unterliegt das TPM den gleichen Einschränkungen wie im Zustand *Disabled*. Es ist in diesem Zustand jedoch möglich, das Kommando *TPM_TakeOwnership* auszuführen, um so einen Eigentümer im TPM zu hinterlegen. Im Zustand *Activated* sind alle Funktionen des TPM verfügbar, sofern sie nicht durch den Zustand *Unowned* blockiert werden.

Unabhängig davon, ob bereits ein Eigentümer eingerichtet ist oder nicht, kann die Zustandstransformation nur mittels *TPM_PhysicalSetDeactivated* durchgeführt werden, wofür der Beweis des physikalischen Zugriffs erbracht werden muss. Eine durch das Eigentümer-Passwort geschützte Variante wie beim *Disabled/Enabled*-Zustandspaar gibt es für diese Transformation nicht.

Es gibt jedoch die Möglichkeit, das TPM bis zum nächsten Systemstart in den Zustand *Deactivated* zu versetzen. Hierzu dient das Kommando *TPM_SetTempDeactivated*, für welches entweder der Beweis des physikalischen Zugriffs erbracht oder das Passwort des Benutzers (*Operator Auth Secret*) übergeben werden muss. Die Rollen Eigentümer und Benutzer können unterschiedlichen Personen bzw. Personengruppen zugeordnet sein. Hierdurch kann zum Beispiel der aktive Benutzer des Systems das TPM temporär in den Zustand *Deactivated* überführen obwohl der Eigentümer *Activated* als den permanenten Zustand konfiguriert hat.

4.3.5 Erweiterte TPM-Konfiguration (Opt-In)

Neben den möglichen Zuständen bietet das TPM weitere *Permanent Flags* zur Konfiguration des Funktionsumfangs. Diese Optionen werden im Folgenden hinsichtlich Funktion, korrelierendem TPM-Kommando und benötigter Zugriffsrechte erläutert:

TPM_SetOwnerInstall = {true, false}
Aktiviert oder deaktiviert das TPM-Kommando *TPM_TakeOwnership* und somit die Möglichkeit, den Eigentümer der Plattform einzurichten. Das Kommando wird in der Regel deaktiviert, nachdem der Eigentümer eingerichtet wurde, um zu verhindern, dass jemand ohne physikalischen Zugriff auf die Plattform den Eigentümer ändern kann. Diese Option verhindert jedoch nicht das Löschen des Eigentümers, hierfür wird die Option *TPM_DisableOwnerClear* verwendet. Zur Ausführung des Kommandos ist der Beweis des physikalischen Zugriffs zu erbringen.

TPM_SetOperatorAuth
Dieses Kommando ermöglicht das Setzen eines Passwortes für den Benutzer der Plattform. Dieses Passwort dient nur der Autorisierung des Kommandos *TPM_SetTempDeactivated,* welches das temporäre Deaktivieren des TPM erlaubt. Das Einrichten des Passwortes ist nur mit physikalischem Zugriff auf das System

möglich, es kann aber jederzeit durch Überschreiben des alten Wertes ein neues Passwort vergeben werden.

TPM_DisablePubekRead
Verhindert das Auslesen des öffentlichen Teils des *Endorsement Key* ohne Kenntnis des Eigentümer-Passwortes. Da dieser Schlüssel die Plattform eindeutig identifiziert, sollte diese Option aktiviert werden. In der aktuellen Version des TPM (Version 1.2) wird dies automatisch während des *TPM_TakeOwnership*-Kommandos gesetzt und muss somit nicht mehr manuell vom Eigentümer ausgeführt werden. Der Eigentümer kann, unter Angabe des entsprechenden Passwortes, mittels *TPM_OwnerReadInternalPub* den öffentlichen Teil des EK und des SRK auslesen.

TPM_DisableOwnerClear
Deaktiviert die Ausführung des TPM-Kommandos *TPM_OwnerClear* und somit die Möglichkeit, den Eigentümer der Plattform zu löschen ohne physikalischen Zugriff auf den Rechner zu haben. Zur Ausführung des Kommandos ist eine Authentifizierung mittels des Eigentümerpassworts nötig. Eine Reaktivierung des Kommandos ist nicht vorgesehen. Erst nach der Ausführung von *TPM_ForceClear* und der damit verbundenen Entfernung des Eigentümers ist das Kommando wieder verfügbar.

TPM_DisableForceClear
Deaktiviert die Ausführung des TPM-Kommandos *TPM_ForceClear* und somit die Möglichkeit den Eigentümer der Plattform zu löschen, auch wenn der physikalische Zugriff auf den Rechner gegeben ist. Diese Einstellung ist jedoch nur bis zum nächsten Start der Plattform aktiv, da ein dauerhaftes Setzen den TPM unwiederbringlich an einen Eigentümer bindet. Falls dies gewünscht sein sollte, muss das Kommando bei jedem Systemstart erneut und direkt nach der TPM-Initialisierung ausgeführt werden (z. B. durch das BIOS). Zur Ausführung des Kommandos ist keine Authentifizierung nötig.

4.3.6 TPM-Eigentümer einrichten und entfernen

Der Eigentümer des TPM hat als einzige Instanz die volle Kontrolle über alle Funktionen des TPM. Welcher Person bzw. Personengruppe die Rolle zugeteilt wird, hängt vom jeweiligen Einsatzszenario ab. Für alle Szenarien ist das Einrichten eines Eigentümers jedoch unerlässlich, da nur darauf aufbauend komplexe Anwendungsfälle konstruiert werden können. Selbst im Zustand *Enabled/Activated* (S5) stehen nicht alle Funktionen zur Verfügung, erst das Einrichten eines Eigentümers (S1) aktiviert den vollen Funktionsumfang. Dies macht die Einrichtung des Eigentümers zu einem sicherheitskritischen Vorgang, der mehrstufig gegen Missbrauch geschützt werden muss.

Das Einrichten des Eigentümers wird durch das Kommando *TPM_TakeOwnership* ausgelöst, dieses wird jedoch nur ausgeführt, wenn es nicht durch das Kommando *TPM_SetOwnerInstall* zuvor deaktiviert wurde. Zusätzlich muss sich das TPM in einem Zustand befinden, der die Ausführung von *TPM_TakeOwnership* er-

laubt (S5–S7). Der Vorgang schlägt ebenfalls fehl, wenn bereits ein Eigentümer eingerichtet ist, dieser muss zunächst entfernt werden.

Während des Vorgangs wird das Eigentümer-Passwort (*Owner Auth Secret*) abgefragt und verschlüsselt an das TPM übergeben, wo es im nichtflüchtigen Speicherbereich ablegt wird. Die Verschlüsselung erfolgt asymmetrisch unter Verwendung des öffentlichen Teils des *Endorsement Key* (EK). Hierdurch ist gewährleistet, dass das Passwort nur dem Eigentümer und dem TPM bekannt ist, somit ist die Kenntnis des Passwortes der Beweis, der Eigentümer der Plattform zu sein. Zusätzlich werden Parameter zur Erzeugung eines RSA-Schlüssels an das TPM übertragen, das daraus den *Storage Root Key* (SRK) generiert und ebenfalls im nichtflüchtigen Speicherbereich ablegt. Der SRK bildet den *Root of Trust for Storage* (RTS) und dient damit dem Aufbau einer Schlüsselverwaltung. Um den späteren Zugriff auf den SRK zu autorisieren wird, identisch zum Passwort des Eigentümers, ein weiteres Passwort verschlüsselt an das TPM übergeben. Es gibt keine Möglichkeit, die beiden Passwörter im Falle eines Verlustes zu rekonstruieren. Beim Einrichten eines neuen Passwortes wird das alte Passwort überschrieben. Dies hat zur Folge, dass alle Informationen (z. B. Schlüssel, Daten), die mit dem alten Passwort in Beziehung stehen, unwiederbringlich verloren gehen. Beide Passwörter können aber durch Eingabe des alten Passwortes mittels *TPM_ChangeAuthOwner* geändert werden.

Abb. 4.9 TPM-Eigentümer einrichten und entfernen

Soll ein bereits eingerichteter Eigentümer entfernt werden, muss das TPM in den Auslieferungszustand zurückgesetzt werden. Bei diesem Vorgang werden folgende Schritte ausgeführt:

- Löschen des SRK – Hierdurch werden sämtliche Informationen, die unter dem SRK gespeichert sind (z. B. Schlüssel von Benutzern und Applikationen), unlesbar. Die Informationen, welche in der Regel außerhalb des TPM abgelegt sind (z. B. die Daten der Schlüsselverwaltung), werden zwar nicht gelöscht, jedoch können sie ohne den SRK nicht mehr entschlüsselt werden.
- Löschen des Eigentümer-Passwortes – Es können bis zur Einrichtung eines neuen Eigentümers keine Kommandos ausgeführt werden, die eine Autorisierung mittels *Owner Auth Secret* erfordern.
- Löschen sämtlicher Daten in den flüchtigen und nichtflüchtigen Speicherbereichen mit Ausnahme des *Endorsement Key*.
- Löschen des Benutzer-Passwortes – Da in der Regel mit einem Wechsel des Eigentümers auch der Benutzer wechselt, wird sein Passwort ebenfalls gelöscht.

Die Überführung in den Auslieferungszustand kann durch Ausführung von *TPM_OwnerClear* forciert werden. Für die Ausführung des Kommandos muss sich

der Eigentümer mit seinem Passwort authentisieren. Dies führt jedoch nur zum Erfolg, falls die Ausführung nicht durch *TPM_DisableOwnerClear* vom Eigentümer deaktiviert wurde. Ist dies der Fall, kann der Eigentümer nur noch entfernt werden, wenn der physikalische Zugriff auf das System gewährleistet ist. In diesem Zusammenhang wird *TPM_ForceClear* verwendet, welches ebenfalls das TPM in den Auslieferungszustand versetzt. Jedoch kann auch diese Möglichkeit unter Einsatz von *TPM_DisableForceClear* entfernt werden. Wie bereits beschrieben, muss dies jedoch nach jedem Systemstart wiederholt werden. Das Ergreifen von Maßnahmen zum Schutz gegen einen Reset des TPM ist notwendig, da dies andernfalls einen einfachen *Denial-of-Service*-Angriff ermöglicht.

4.3.7 TPM-Schlüsseltypen und Schlüsselverwaltung

Eine wichtige Funktion des TPM ist die Verwaltung von kryptographischen Schlüsseln durch den Aufbau einer Schlüsselhierarchie. Das TPM ist in der Lage, RSA-Schlüssel gesichert intern zu erzeugen. Hierfür werden die *RSA-Engine* und der *Randon Number Generator* (RNG) verwendet. Durch die Erzeugung von RSA-Schlüsselpaaren innerhalb des TPM kann sichergestellt werden, dass der private Teil des Schlüsselpaares (private key) zu keinem Zeitpunkt ausgelesen werden konnte und somit nur dem TPM bekannt ist. Die Schlüsselpaare können in Schlüssel für die Digitale Signatur (*signing keys*) und Schlüssel zum verschlüsselten Speichern von Daten (*storage keys*) unterteilt werden. Alle vom TPM für diese Zwecke erzeugten Schlüssel sind für die spätere Verwendung durch die *RSA-Engine* vorgesehen und haben deshalb eine der drei für asymmetrische Verfahren typischen Schlüssellängen (512, 1024, 2048 Bit). Die TCG empfiehlt die ausschließliche Verwendung der Schlüssellänge 2048 Bit, die auch standardmäßig bei der Erzeugung neuer Schlüssel verwendet wird. Eine weitere Option bei der Erzeugung von Schlüsseln ist die Wahl zwischen Schlüsseln, die an eine bestimmte Instanz des TPM gebunden sind (*non-migrateable*), und solchen, die für die Verwendung auf ein anderes TPM transferiert werden können (*migrateable*). Jeder Schlüssel erhält ein Passwort, welches bei der Verwendung durch das TPM angegeben werden muss (*key password*). Handelt es sich um einen Schlüssel, welcher auf eine andere Plattform migriert werden darf, kann hierfür ein weiteres Passwort vergeben werden (*migration password*). Die TCG definiert insgesamt vier unterschiedliche Schlüsseltypen:

- **Storage Keys**
  Dies sind Schlüssel zur Verschlüsselung von beliebigen Daten oder anderen digitalen Schlüsseln. Storage Keys dienen somit hauptsächlich dem Aufbau einer Schlüsselhierarchie für die sichere Ablage von Schlüsseln außerhalb des TPM. Deshalb werden sie auch als Wrapping Keys bezeichnet. Eine Sonderform des *Storage Keys* ist der *Storage Root Key* (SRK), der die Wurzel der Schlüsselhierarchie darstellt und somit den *Root of Trust for Storage* (RTS) bildet.

- **Signing Keys**
  Sie sind Schlüssel zum digitalen Signieren von beliebigen Daten (z. B. Anwendungsdaten und Nachrichten). *Signing Keys* können sowohl an eine Plattform gebunden sein als auch für eine mögliche Migration auf andere Plattformen konfiguriert sein.

- **Bind Keys**
  Sie bilden das Gegenstück zu den *Signing Keys* und dienen der Verschlüsselung von kleinen Datenmengen. Bei den verschlüsselten Daten handelt es sich in der Regel um einen symmetrischen Schlüssel, mit dem größere Mengen an Daten außerhalb des TPM verschlüsselt werden. Dieses Verfahren wird als hybride Verschlüsselung bezeichnet. Wie *Signing Keys* können auch *Bind Keys* auf eine andere Plattform migriert werden.

- **Legacy Keys**
  Sie bilden eine Kombination aus *Signing* und *Bind Keys* und können somit sowohl für digitale Signaturen als auch für die Verschlüsselung verwendet werden.

Die bereits beschriebenen *Attestation Identity Keys* (AIKs) dienen der Identifikation der Plattform und sind dadurch bei Definition an die Plattform gebunden und können nicht migriert werden (*non-migrateable*). Die folgende Abbildung beschreibt den Aufbau eines durch das TPM verwalteten Schlüssels:

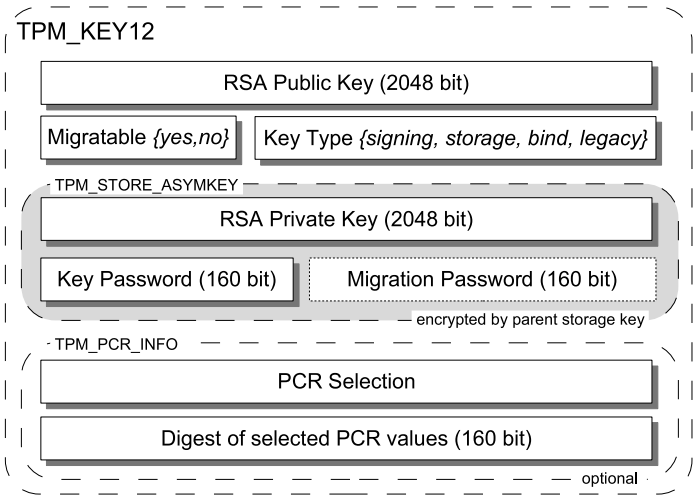

Abb. 4.10 TPM-Schlüsselobjekt

Neben dem öffentlichen und dem privaten Teil des RSA-Schlüsselpaares enthält die durch das TPM generierte Datenstruktur weitere Felder. So wird das Passwort, das zur Verwendung des Schlüssels angegeben werden muss, hier abgelegt. Falls es sich um einen Schlüssel mit Option zur Migration auf andere Plattformen handelt,

wird das hierfür benötigte Passwort ebenfalls hier hinterlegt. Des Weiteren wird der Typ des Schlüssels definiert. Optional können Schlüssel bei ihrer Erzeugung an den aktuellen Zustand des Systems gebunden werden. Ist diese Option aktiv, kann ein Schlüssel nur dann verwendet werden, falls sich das System in selben Zustand wie zum Zeitpunkt der Erstellung befindet. Hierfür wird der Hashwert der ausgewählten PCR (PCR Selection) mit dem in der Datenstruktur hinterlegten Wert (PCR Digest) verglichen.

Mit Ausnahme des *Endorsement Key* (EK) und des *Storage Root Key* (SRK) werden alle weiteren vom TPM verwalteten Schlüssel außerhalb der TPM gespeichert. Um die Geheimhaltung des privaten Teils des Schlüssels sowie der spezifischen Passwörter sicherzustellen, muss dieser Teil der Datenstruktur verschlüsselt werden. Die folgende Abbildung veranschaulicht die daraus entstehende Schlüsselhierarchie:

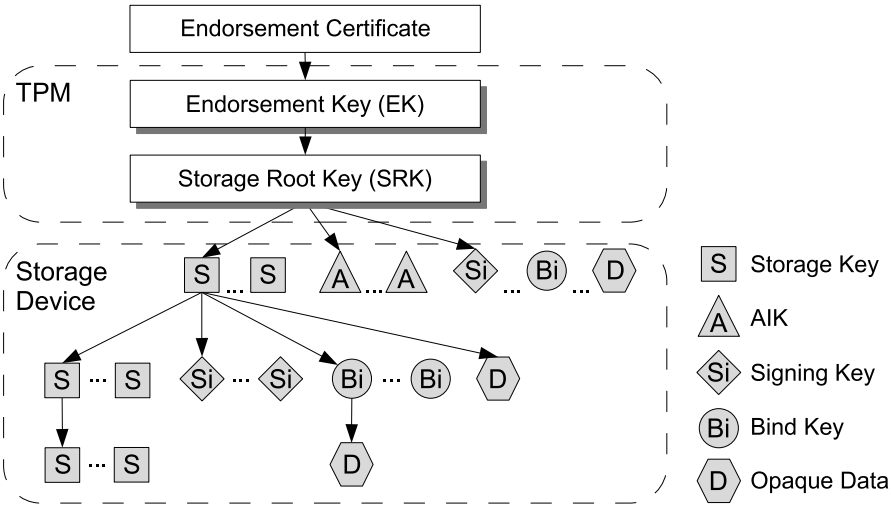

Abb. 4.11 TPM-Schlüsselhierarchie [TCG2004]

Die oberste Ebene der Schlüsselhierarchie besteht aus Schlüsseln oder Daten, die direkt durch den *Storage Root Key* (SRK) verschlüsselt wurden. Dadurch ist sichergestellt, dass sämtliche Schlüssel nur innerhalb des TPM entschlüsselt werden können und sich der private Teil eines Schlüsselpaares somit zu keinem Zeitpunkt außerhalb des TPM befindet. Der SRK selbst wird während des *Take-Ownership*-Prozesses erzeugt und ist durch den *Endorsement Key* (EK) verschlüsselt und mit dem Eigentümer-Passwort gesichert. Die Integrität des EK wird wiederum durch das *Endorsement Certificate* bescheinigt, welches dadurch streng genommen den eigentlichen *Vertrauensanker* (*Root of Trust*) stellt.

Auf dieser ersten Ebene können sämtliche Schlüsseltypen (*storage, signing, bind* und AIK) abgelegt werden. Ein *Storage Key* der ersten Ebene kann mit Ausnahme der AIKs ebenfalls sämtliche Schlüsseltypen als Kinder ablegen, hierdurch wird ei-

ne weitere Ebene in der Schlüsselhierarchie erzeugt. Ein Schlüssel der Ebene N ist somit immer mit einem Schlüssel der Ebene $N-1$ verschlüsselt. Für die Entschlüsselung eines Schlüssels müssen folglich zunächst alle *Storage Keys* auf dem Pfad zum SRK geladen und entschlüsselt werden.

Während die erste Ebene der Hierarchie in der Regel verwendet wird, um die Schlüssel des Plattform-Eigentümers zu hinterlegen, können die Schlüssel weiterer Benutzern oder Applikationen als weitere Äste hinterlegt werden. Hierbei ist es zusätzlich denkbar, nur vertrauenswürdigen Benutzern oder Applikationen Zugriff auf einen *Storage Key* zu gewähren, um zu verhindern, dass ohne das Wissen des Eigentümers zusätzliche Schlüssel durch das TPM erzeugt werden. So wäre es z. B. möglich, der E-Mail-Anwendung lediglich das Passwort für den Zugriff auf einen der *Signing Keys* zu überlassen und einer Dateiverschlüsselungssoftware für die sichere Speicherung ihres symmetrischen Schlüssels einen *Bind Key* zuzuweisen.

Um einen der gespeicherten Schlüssel zu verwenden, muss zunächst der private Teil des Schlüssels in das TPM geladen und dort entschlüsselt werden. Wurde der zu verwendende Schlüssel nicht direkt als Kind des SRK gespeichert, müssen zunächst rekursiv alle *Storage Keys* auf dem Pfad vom SRK zum eigentlichen Schlüssel geladen werden. Der Schlüssel wird dann in einem der *RSA-Key-Slots* abgelegt und der aufrufenden Instanz wird eine Referenz (*key handle*) auf diesen Schlüssel übergeben. Durch Angabe dieser Referenz kann der Schlüssel nun entsprechend des Schlüsseltyps durch die *RSA-Engine* verwendet werden. Die folgende Abbildung zeigt die bei der Verwendung und dem Ladevorgang eines Schlüssels beteiligten Komponenten:

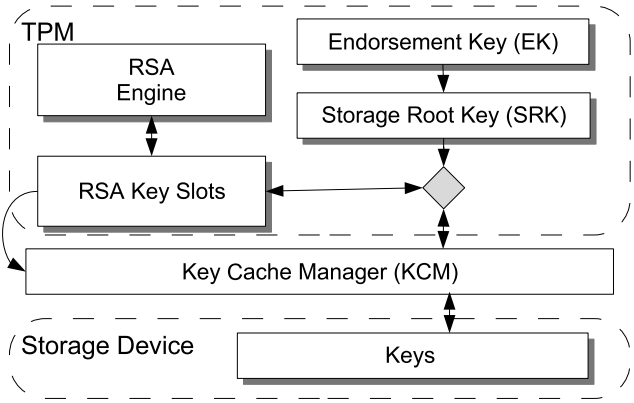

Abb. 4.12 TPM-Schlüssel-Ladevorgang [TCG2004]

Da die Anzahl der *RSA-Key-Slots* beschränkt ist, empfiehlt die TCG die Verwendung eines *Key Cache Managers* (KCM), welcher Schlüsselmaterial temporär aus dem TPM auf ein Speichermedium auslagern kann, um Platz für weitere Schlüssel zu schaffen. Die Implementierung dieser Komponente wird jedoch nicht näher spezifiziert. Es ist denkbar, dass diese Komponente Teil des *Trusted Software Stack*

(TSS) ist. Die Verwaltung der beschriebenen Schlüsselhierarchie, also die Zuordnung von Kindern zu ihren Eltern, ist ebenfalls Aufgabe des TSS.

4.3.8 TPM-Selbstschutzmaßnahmen (Tamper-Resistant)

Der TPM repräsentiert zwei der drei beschriebenen *Vertrauensanker* (Roots of Trust), den *Root of Trust for Reporting* (RTR) und den *Root of Trust for Storage* (RTS). Der RTR bildet die Grundlage für die eindeutige Identifizierung der Plattform, der RTS ist die Basis für den sicheren Datenspeicher (protected storage). Um die Integrität dieser beiden *Vertrauensanker* zu schützen, muss das TPM resistent gegen Hardware- und Software-Angriffe sein. Die TCG spezifiziert hierfür eine Reihe von zu implementierenden Schutzfunktionen.

- Es darf nicht möglich sein, sicherheitskritische Informationen (z. B. den *Endorsement Key*) über die physikalischen Kontakte des TPM auszulesen (pin probing).
- Das TPM muss fest mit der Plattform verbunden sein. Dies kann mittels einer Lötverbindung oder einer Integration in andere Komponenten (z. B. *Southbridge*) des Mainboards erreicht werden. Weiter muss das TPM einen eventuellen Entfernungsversuch erkennen und geeignete Gegenmaßnahmen ergreifen (z. B. Löschen des EK und SRK).
- Das TPM muss gegen Seitenkanal-Analysen geschützt werden. Bei der Seitenkanal-Analyse wird versucht, auf der Basis von physikalischen Messwerten (z. B. Ausführungszeit und Leistungsaufnahme) Rückschlüsse auf den verwendeten Schlüssel zu ziehen. Diese Anforderung gilt ebenso auch für *SmartCards*. Eine Übersicht über den Stand der Forschung im Bereich Seitenkanal-Analyse liefert [Lemke01].

Neben diesen physikalischen Schutzfunktionen muss das TPM gegen Wörterbuch- bzw. Brute-Force-Angriffe auf die TPM- bzw. Schlüsselpasswörter gesichert werden. Diese als *Dictionary Attack Mitigation* bezeichnete Technik soll einen effektiven Angriff auf diese Passwörter verhindern, indem die Antwortzeit auf Anfragen beim Aufbau eines Kommunikationskanals nach mehrfach fehlgeschlagenen Autorisierungsversuchen (*TPM_AUTHFAIL*) verlängert wird. Nach einer bestimmten Anzahl an fehlgeschlagenen Versuchen soll das TPM für einen nicht spezifizierten Zeitraum sämtliche Autorisierungsversuche ablehnen. Diese Sperre kann jedoch unter Angabe des Eigentümer-Passworts mit dem Kommando *TPM_ResetLockValue* wieder aufgehoben werden. Der Plattform-Eigentümer ist somit in der Lage, einen entsprechenden Angriff auf die Schlüssel unterhalb des SRK durchzuführen.

4.4 Sicherheitsfunktionen der TCP

Durch die Verwendung der unter „TPM-Einheiten“ beschriebenen Komponenten ist das TPM in der Lage, eine Reihe von Sicherheitsfunktionen anzubieten. Diese Funktionen decken einen Teil der unter „Ziele des Trusted Computing“ an eine *Trusted Computing Platform* gestellten Anforderungen ab. Auf den folgenden Seiten werden diese Funktionen im Detail beschrieben und falls möglich die korrelierenden *TPM-Kommandos* aufgezeigt.

4.4.1 Integrity Measurement, Storage and Reporting

Integrity Measurement bezeichnet die Generierung von Prüfsummen (Hashwerte) über Eigenschaften der Plattform, welche sich auf die Vertrauenswürdigkeit selbiger auswirken. Dies können z. B. Prüfsummen über das ROM-Abbild einer Hardwarekomponente, einer ausführbaren Datei oder über wichtige Bibliotheken des Betriebssystems sein. Um die Integrität dieser Prüfsummen gewährleisten zu können, werden diese in den *Platform Configuration Register* (PCR) des TPM abgelegt. Wird der Wert eines Registers überschrieben, wird die neue Prüfsumme zunächst mit der vorhandenen Prüfsumme verknüpft, um die Vortäuschung eines als vertrauenswürdig eingestuften Zustands zu verhindern. Der exakte Ablauf der Aktualisierung eines PCR ist auf Seite 35 beschrieben. Es gibt hierbei zwei unterschiedliche Möglichkeiten, eine Prüfsumme in einem PCR zu hinterlegen. Die folgende Abbildung beschreibt diese Optionen:

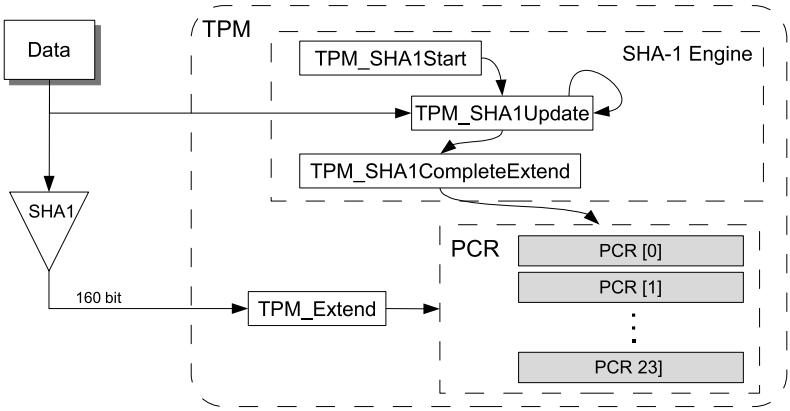

Abb. 4.13 Update der PCR

Die erste Option ist die Verwendung der *SHA-Engine* des TPM. Hierbei werden die Daten direkt an das TPM übergeben, dies erfolgt durch den Aufruf von

TPM_SHA1Start, gefolgt von *TPM_SHA1Update*. *TPM_SHA1Update* kann innerhalb einer Sitzung mehrfach aufgerufen werden, um weitere Datenblöcke hinzuzufügen. Wurden alle Daten übergeben, legt *TPM_SHA1CompleteExtend* die resultierende Prüfsumme im angegebenen Register ab. Aufgrund der beschränkten Ressourcen des TPM sollte diese Methode nur gewählt werden, falls keine andere Implementierung des SHA1-Algorithmus vorhanden ist oder deren Integrität nicht sichergestellt werden kann. Für alle anderen Fälle empfiehlt die TCG die Verwendung einer SHA-1-Implementierung außerhalb des TPM. Hierbei kann es sich um eine Implementierung in Software oder innerhalb von HSM[19], mit mehr Leistung als das TPM, handeln. Die dabei erzeugte Prüfsumme wird mittels *TPM_Extend* direkt im entsprechenden PCR abgelegt.

Die Ausführung der beschriebenen Kommandos erfordert keine Autorisierung. Das Auslesen der *Platform Configuration Register* ist ebenfalls ohne Angabe eines Passworts mittels *TPM_PCRRead* möglich.

4.4.2 Initialisierung der Chain of Trust

Eine der wichtigsten Funktionen eines *Trusted Computing System* ist der Aufbau einer lückenlosen *Vertrauenskette* (*Chain of Trust*) über alle auf dem System ausgeführten Instruktionen. Hierbei muss jede Komponente vor der Übergabe der Ausführungsrechte an die nächste Komponente diesen Schritt protokollieren. Ziel dieses Konzeptes ist es, die Vertrauenswürdigkeit eines Systems bewerten zu können. An dieser Stelle muss jedoch darauf hingewiesen werden, dass die *Chain of Trust* lediglich eine Grundlage für eine solche Bewertung darstellt. Hierbei werden nämlich lediglich Prüfsummen über die ausgeführten Komponenten erzeugt und gesichert abgelegt, eine Bewertung dieser Prüfsummen erfordert zusätzliche Maßnahmen. Für die sichere Speicherung der Prüfsummen wird die *Integrity-Measurement*-Funktion des TPM verwendet.

Die Basis für die iterative Erzeugung von Prüfsummen muss bereits eine vertrauenswürdige Komponente sein, da es nicht möglich ist, die Basis selbst zu prüfen. Um dieses Vertrauen zu rechtfertigen ist der *Vertrauensanker* (*Root of Trust*) in Hardware realisiert. Der *Vertrauensanker* für die Erzeugung der Prüfsummen ist der bereits beschriebene *Core Root of Trust for Measurement* (CRTM). Eine stark vereinfachte Darstellung der *Chain of Trust* liefert die folgende Abbildung:

Hierbei werden beginnend mit dem CRTM sämtliche am Startvorgang des Betriebssystems beteiligten Komponenten vor ihrer Ausführung mit Hilfe einer Prüfsumme protokolliert. Im optimalen Fall sollte die Prüfsumme über eine Komponente immer direkt von ihrem Vorgänger erzeugt werden, um zu verhindern, dass zwischen dem Zeitpunkt der Erstellung der Prüfsumme und der Ausführung der Instruktionen unbemerkt Änderungen vorgenommen werden können. Über den Startvorgang hinaus sollten auch sämtliche zur Laufzeit des Betriebssystems geladenen

[19] Hardware Security Module

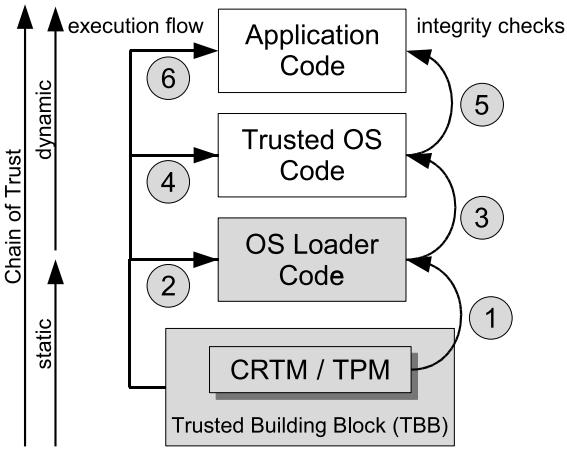

Abb. 4.14 Chain of Trust [TCG2004]

und ausgeführten Instruktionen protokolliert werden. Bei der beschriebenen Abbildung handelt es sich um eine Erweiterung der Grafik aus den Spezifikationen der *Trusted Computing Group*. Leider steht die Grafik dort in einem falschen Kontext, da die Spezifikationen eine mögliche *Trusted Computing Platform* (TCP) beschreiben, welche aber per Definition nur für einen kleinen Teil dieser *Vertrauenskette* zuständig ist. Korrekt wäre, diese *Vertrauenskette* in zwei Teile zu zerlegen. Der erste Teil wird durch die TCP aufgebaut, die zu einem bestimmten Zeitpunkt im Startvorgang die Ausführungsrechte an das Betriebssystem übergibt; das Betriebssystem ist nun verantwortlich für die Weiterführung der *Vertrauenskette*. Diese beiden Teile der *Vertrauenskette* seien für die Arbeit wie folgt definiert:

- **Static Chain of Trust**
  Bezeichnet den durch die TCP erzeugten Teil der *Vertrauenskette*. Die Umsetzung dieses Teils ist durch die Spezifikation der TCG exakt festgestellt, erfolgt auf jeder TCG-konformen Plattform identisch und arbeitet unabhängig vom Betriebssystem.

- **Dynamic Chain of Trust**
  Bezeichnet den durch das *Trusted-OS* oder eine vergleichbare Instanz erzeugten Teil der *Vertrauenskette*. Der *Vertrauensanker* hierfür ist die letzte von der TCP protokollierte Komponente, die zugleich für den Übergang zwischen TCP und Betriebssystem zuständig ist; in der Regel ist dies der *Master Boot Record* (MBR). Für die *Dynamic Chain of Trust* existieren momentan keine Spezifikationen und somit keine exakten Vorgaben, welche Komponenten hierbei protokolliert werden müssen. Mögliche Konzepte und konkrete Lösungsansätze für diesen Teil der *Vertrauenskette* werden ab Seite 91 vorgestellt.

Der Rest dieses Kapitels widmet sich dem Aufbau und den Teilen der *Static Chain of Trust* sowie den daran beteiligten Komponenten. Die folgende Abbildung skizziert den Ablauf beim Start des Systems:

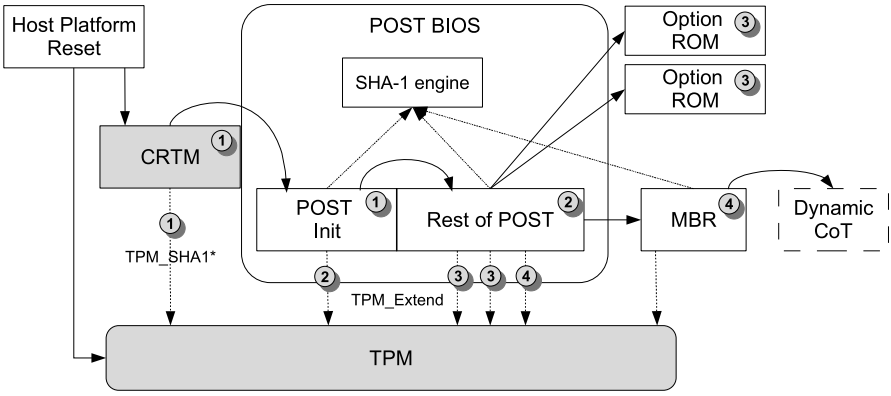

Abb. 4.15 Static Chain of Trust [TCG2005]

Beim Start (Kaltstart, Reset oder Rückkehr aus dem Ruhezustand) des Systems befindet sich das TPM im Ausgangszustand, d. h. alle *Platform Configuration Register* sind zu diesem Zeitpunkt leer. Um das TPM verwenden zu können, durchläuft es zunächst die auf Seite 40 beschriebene Initialisierungssequenz. Die ersten Instruktionen, die auf der CPU des Systems ausgeführt werden, sind Teil des CRTM und werden aus einem ROM[20] gelesen. Der CRTM erzeugt eine Prüfsumme über sich selbst und über den statischen Teil des BIOS. Der statische Teil des BIOS enthält den Programmcode und wird aus einem EEPROM[21] gelesen. Da der CRTM selbst über keine Implementierung des SHA-1-Algorithmus verfügt, verwendet er hierfür die beschriebene Funktionalität des TPM (*TPM_SHA1**). Nach Ablage der Prüfsummen im TPM wird der Ausführungspfad an den BIOS POST[22] übergeben. Neben der eigentlichen Aufgabe des POST, der Überprüfung der grundlegenden Komponenten des Systems, erzeugt der erste Teil des POST eine Prüfsumme über die aktuelle Konfiguration des BIOS. Diese Einstellungen werden in einem CMOS-Speicher[23] gespeichert, der über die Mainboard-Batterie mit Strom versorgt wird. Während der weiteren Ausführung des POST werden Prüfsummen über die auf Erweiterungskarten vorhandenen Option-ROMs erzeugt. Beispiele für gängige Option-ROMs sind das BIOS der Grafikkarte oder die Firmware eines SCSI-Controllers. Erst nachdem diese Prüfsummen erzeugt wurden, wird der Ausführungspfad an diese Komponenten übergeben. Wurden alle Erweiterungskarten initialisiert, erzeugt der BIOS POST als letzten Schritt die Prüfsum-

[20] Read-Only Memory

[21] Electrically Erasable Programmable Read-Only Memory. Auch als E^2PROM bezeichnet.

[22] Power-On Self-Test

[23] Complementary Metal Oxide Semiconductor-RAM

me über den *Master Boot Record* (MBR) des ersten bootfähigen Speichergeräts (z. B. Festplatte, Boot-CD, USB-Stick). Die TCG empfiehlt die Verwendung einer SHA-1-Implementierung außerhalb des TPM während des POST-Vorgangs. Der MBR selbst ist nicht mehr Teil der Plattform, er unterscheidet sich je nach installiertem Betriebssystem bzw. dem installierten Bootmanager. Somit ist der MBR bereits Teil der *Dynamic Chain of Trust*.

4.4.3 Remote Attestation

Attestation (Nachweis, Bescheinigung) bezeichnet den Vorgang der Beweiserbringung der Vertrauenswürdigkeit eines Systems. Die Vertrauenswürdigkeit eines Systems hängt im Wesentlichen vom Zustand, in dem es sich zum aktuellen Zeitpunkt befindet, ab. Der Zustand eines Systems ist definiert durch sämtliche auf dem System ausgeführten Instruktionen. Dies umfasst sowohl die Prozesse, die sich gegenwärtig im Speicher des Betriebssystems befinden, als auch sämtliche Prozesse, die seit dem Einschalten des Systems ausgeführt wurden. Dieser Zustand wird repräsentiert durch den Inhalt der *Platform Configuration Register*. Wie bereits im vorhergehenden Abschnitt erläutert, ist die *Trusted Computing Platform* nur für die Erzeugung einer Untermenge dieser Prüfsummen verantwortlich; ohne die Unterstützung durch ein vertrauenswürdiges Betriebssystem beschränkt sich der im Folgenden beschriebene Prozess auf die *Static Chain of Trust*.

Unter „Integrity Measurement, Storage and Reporting“ wurde die Speicherung sowie die Abfrage der PCR beschrieben. Für die Abfrage der Register kommt das TPM-Kommando *TPM_PCRRead* zum Einsatz. Eine solche Abfrage kann ohne Angabe eines Passwortes durchgeführt werden und erfordert somit keinen sicheren Kommunikationskanal zwischen der aufrufenden Instanz und dem TPM. Somit kann die Integrität der erhaltenen Prüfsummen nicht gewährleistet werden. Findet die Bewertung der Vertrauenswürdigkeit direkt auf dem zu bewertenden System statt, kann die Integrität durch den Aufbau eines sicheren Kommunikationskanals (*Channel Level Security*) sichergestellt werden. In der Regel findet eine solche Abfrage jedoch aus der Ferne über ein privates oder sogar öffentliches Netzwerk statt. Es muss also sichergestellt werden, dass die Integrität der übertragenen Prüfsummen nicht von der Integrität des Netzwerks und den an dieser Kommunikation beteiligten Softwarekomponenten beeinflusst wird. Dies ist ein gängiges Problem im Bereich der Informationssicherheit und wird durch den Einsatz von Sicherheit auf der Nachrichtenebene (*Message Level Security*) erreicht.

Hierfür werden die PCR vor dem Versenden innerhalb der TPM digital signiert. Die Abb. 4.16 beschreibt den Ablauf dieses Vorgangs.

Die abfragende Instanz (*Remote Service*) sendet eine *Challenge* (i. d. R. eine Zufallszahl) mittels *TPM_Quote* an das TPM[24]. Zusätzlich werden die Indizes der zu signierenden PCR übergeben. Die Datenstruktur bestehend aus der *Challenge* und

[24] Hierfür muss das Trusted-OS einen geeigneten Netzwerkdienst zur Verfügung stellen (siehe S. 74)

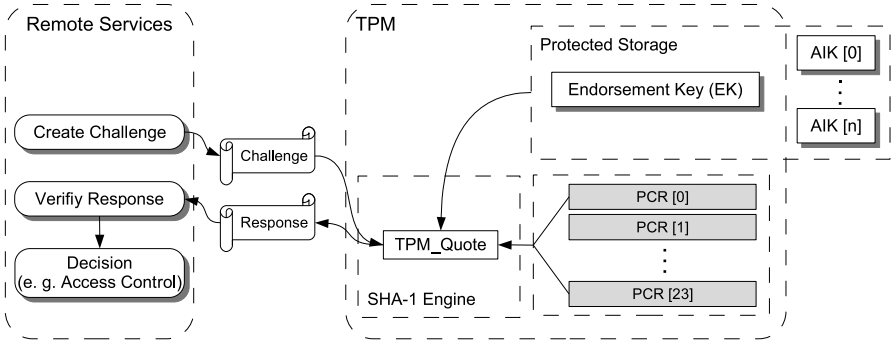

Abb. 4.16 Remote Attestation

den ausgewählten PCRs bildet die *Response* und wird innerhalb des TPM mit einem *Attestation Identity Key* signiert. Der *Remote Service* kann nun unter Verwendung des zum verwendeten Schlüssel passenden Zertifikats[25] (*AIK Credential*) die Identität des Systems sowie die Integrität der übermittelten Prüfsummen verifizieren.

Die Identität und der durch die Prüfsummen repräsentierte Zustand des Systems bilden die Grundlage für die Bewertung des Systems. Werden die PCR nicht verwendet, findet also nur eine Signatur der *Challenge* statt, wird dieser Vorgang als *Platform Authentication* bezeichnet und dient somit der Authentisierung des Systems.

Hinweis: Dieser Vorgang erfordert neben einer TCP auch die Unterstützung durch ein Trusted-OS.

4.4.4 Kryptographische Operationen

Durch die integrierten Crypto-Engines (*RSA-Engine*, *SHA1-Engine* und RNG) bietet das TPM eine Reihe von kryptographischen Funktionen vergleichbar mit denen einer *SmartCard*. Dies umfasst die folgenden Operationen:

- Asymmetrische Ver- und Entschlüsselung von beliebigen Daten nach dem RSA-Verfahren. Die TCG bezeichnet diesen Vorgang auch als *Binding*. Wird für die Verschlüsselung ein Schlüssel verwendet, der nicht auf eine andere Plattform übertragen werden kann (non-migratable), sind die Daten fest an ein bestimmtes TPM gebunden.
- Erzeugen und Verifizieren von digitalen Signaturen über beliebige Daten nach dem RSA-Signatur-Verfahren.

[25] Die Erzeugung des zum AIK gehörigen Zertifikats wird im Kapitel „Public Key Infrastructure (PKI)“ beschrieben.

- Erzeugung von Prüfsummen über beliebige Daten nach dem SHA-1 (SHA-180)-Standard.
- Erzeugung von echten Zufallszahlen (*True Random Number Generator*).

Über diese Funktionen hinaus bietet das TPM durch den Einsatz der *Platform Configuration Register* weitere Funktionen:

- **Sealing**
  Sealing ist die erweiterte Version von Binding. Die Idee hierbei ist, Daten auf der lokalen Plattform oder Nachrichten an andere TCP an einen bestimmten Zustand des Systems zu binden. Eine mittels Sealing geschützte Nachricht kann vom Empfänger nur entschlüsselt werden, wenn er sich aus Sicht des Senders in einem vertrauenswürdigen Zustand befindet. Für die Erzeugung einer solchen Nachricht wird eine Untermenge von PCR-Werten zusammen mit einem symmetrischen Schlüssel asymmetrisch mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Die Nachricht selbst ist mit diesem symmetrischen Schlüssel geschützt. Befindet sich der Empfänger im gewünschten Zustand, kann dieser mittels TPM den symmetrischen Schlüssel extrahieren und somit die Nachricht entschlüsseln.

- **Sealed-Signing**
  Dies ist die erweiterte Version des RSA-Signatur-Verfahrens. Hierbei fließen in den Prozess der Signatur zusätzlich zu den eigentlichen Daten beliebige PCR-Werte ein. Hierdurch kann z. B. der Sender einer signierten Nachricht zusätzlich zu seiner Identität auch Informationen über den aktuellen Zustand des Systems übertragen. Das *Sealed-Signing*-Verfahren kommt u. a. bei der bereits beschriebenen *Remote Attestation* zum Einsatz.

4.5 Identität der TCP und entstehende Datenschutzprobleme

Einer der Hauptkritikpunkte des *Trusted-Computing*-Ansatzes ist die mögliche Verletzung der Privatsphäre der Benutzer und die damit verbundenen datenschutzrechtlichen Probleme. Das durch den Einsatz eines TPM entstehende Problem ist in der Einmaligkeit des *Endorsement Key* begründet. Kommt dieser Schlüssel bei der Kommunikation über ein Netzwerk zum Einsatz, ist es möglich dieses System bei allen zukünftigen Unterhaltungen eindeutig zu erkennen. Diese Information kann zum Beispiel verwendet werden, um ein Bewegungsprofil des Systems zu erstellen. Dies entspricht der Funktionsweise von HTTP-Cookies, ohne dass jedoch auf dem System zusätzlich zum EK etwas hinterlegt werden muss. Erschwerend kommt im Falle des EK noch hinzu, dass dieser im Gegensatz zu HTTP-Cookies in der Regel nicht gelöscht[26] werden kann. Weitere Datenschutzprobleme und einige Lösungsansätze beschreibt [Kursawe2003].

[26] Ab der Version 1.2 der TPM-Spezifikation ist eine optionale Funktion zum Löschen des EK beschrieben.

Um den Missbrauch des EK zu verhindern, muss sichergestellt werden, dass der öffentliche Teil des EK nur unter bestimmten Umständen ausgelesen werden kann. Hierfür definiert die TCG seit Version 1.1 der TPM-Spezifikation zwei Gegenmaßnahmen:

- Das Auslesen des öffentlichen Teils des EK kann deaktiviert werden (*TPM_DisablePubekRead*). Ist diese Option aktiv, kann dieser Schlüssel nur noch unter Angabe des Eigentümer-Passworts ausgelesen werden. Ab Version 1.2 der Spezifikation ist dies die Standardeinstellung.
- Für die Authentifizierung der Plattform während des *Remote-Attestation*-Vorgangs können an Stelle des EK die *Attestation-Identity-Keys* (AIK) verwendet werden.

Ab der Version 1.2 der Spezifikation bietet das TPM zusätzlich zum AIK-Verfahren das *Direct-Anonymous-Attestation*-Verfahren an. Beide Konzepte werden in den folgenden Abschnitten näher erläutert.

4.5.1 Erzeugung eines Attestation-Identity-Zertifikats

Das Ziel beim Einsatz der AIK ist es, während der *Remote Attestation* nachzuweisen, dass der für die Signatur der Nachrichten verwendete Schlüssel nur dem TPM selbst bekannt ist und es sich um eine konforme Implementierung des TPM handelt – jedoch ohne die bei der Verwendung des EK entstehenden Datenschutzprobleme. Es muss der Nachweis erbracht werden, dass das TPM die Nachrichten signiert hat, ohne jedoch den EK zu verwenden. Der AIK dient somit als Pseudonym (Alias) für den EK. Hierfür wird für jede Instanz, gegenüber der sich das TPM authentisiert, oder sogar für jeden neuen Authentisierungsvorgang ein neuer AIK erzeugt. Für die Ausstellung eines solchen Aliases kommt eine *Privacy Certificate Authority* (PCA) zum Einsatz. Hierbei handelt es sich um eine CA, der sowohl der Eigentümer des TCS als auch der Anbieter des Services, gegenüber dem sich die Plattform ausweisen will, Vertrauen schenkt (*Trusted Third Party* – TTP)[27]. Die Abb. 4.17 veranschaulicht den Prozess der Erzeugung und Signatur eines AIK.

Der neue AIK wird innerhalb des TPM durch die *RSA-Engine* erzeugt. Der private Teil des Schlüsselpaares wird mit dem SRK verschlüsselt und außerhalb des TPM abgelegt. Da es sich bei der Erzeugung von AIKs um eine sicherheitskritische Operation handelt, muss bei der Ausführung von *TPM_MakeIdentity* das SRK-Passwort angegeben werden. Die TCG empfiehlt zusätzlich die Angabe eines AIK-Passwortes, welches bei jeder *Remote Attestation* angegeben werden muss. Wie alle anderen Schlüssel, die innerhalb der Schlüsselverwaltung des TCS gespeichert werden, können auch die AIKs an einen bestimmten Zustand der *Platform Configuration Register* (PCR) gebunden werden.

[27] Eine für Testzwecke zugängliche Privacy-CA wird im Rahmen des OpenTC-Projektes zur Verfügung gestellt [URL07].

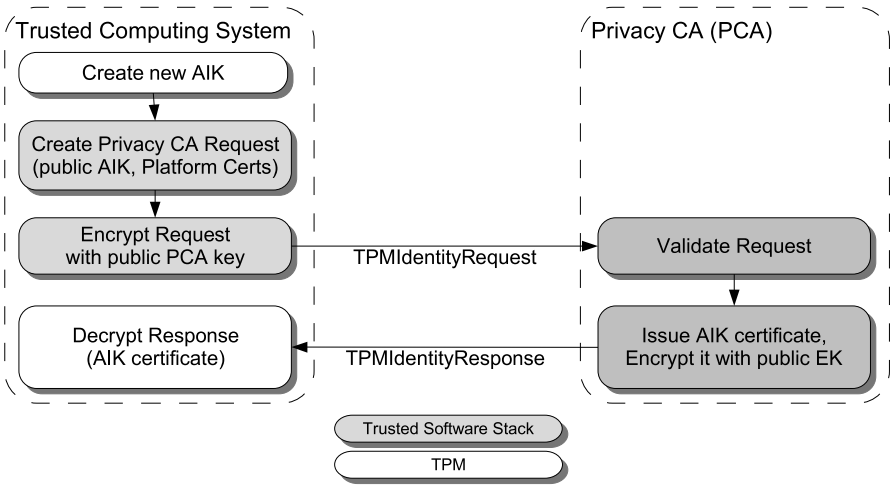

Abb. 4.17 AIK-Erzeugung

Aus dem öffentlichen Teil des AIK und den Plattform-Zertifikaten (*EK Credential*, *TCPA Conformity Certificate*, *Platform Credential*) wird die Datenstruktur für die Anfrage erzeugt. Da diese Anfrage für die spätere Antwort auch den öffentlichen Teil des EK enthält und sichergestellt werden muss, dass dieser nur der *Privacy-CA* bekannt ist, wird sie mit dem öffentlichen Schlüssel der *Privacy-CA* verschlüsselt. Die Aufgabe der *Privacy-CA* ist es, zu überprüfen, ob die Anfrage von einer vertrauenswürdigen TPM-Implementierung stammt. Hierfür werden die übertragenen Plattform-Zertifikate ausgewertet[28], sie weisen unter anderem nach, dass der verwendete EK in einem TPM hinterlegt ist. Wurde die Verifizierung erfolgreich abgeschlossen, erzeugt die *Privacy-CA* ein AIK-Zertifikat (*Attestation Identity Credential*). Dieses Zertifikat enthält neben dem öffentlichen Teil des neuen AIK auch einen frei wählbaren Bezeichner für diesen AIK (Identity Label). Dieses Zertifikat wird anschließend verschlüsselt mit dem öffentlichen Teil des EK zurück an das TCS gesendet. Die Verschlüsselung der Antwort dient jedoch nicht der Geheimhaltung des Zertifikates, sondern stellt sicher, dass das TPM im Besitz des privaten Teils des EK und somit berechtigt ist, diesen AIK im Rahmen der *Remote Attestation* zu verwenden. Die notwendige Entschlüsselung wird innerhalb des TPM, durch den Aufruf von *TPM_ActivateIdentity*, durchgeführt. Nach dem Aufruf dieses Kommandos steht der neue AIK zur Verfügung.

Anzumerken ist jedoch, dass auch durch den Einsatz einer *Privacy-CA* keine vollständige Anonymität erreicht werden kann. Die PCA kennt nämlich den öffentlichen Teil des EK und ist somit ihrerseits in der Lage, ein System wiederzuerkennen und somit sämtliche von ihr signierten AIKs mit der Plattform zu verknüpfen. Ein weiteres Problem besteht darin, eine geeignete Instanz für die *Privacy-CA* zu finden,

[28] Die hierfür notwendige Public Key Infrastructure (PKI) in im Abschnitt „Verwendung der AIK-Zertifikate“ erläutert.

da sowohl der Benutzer als auch der Serviceanbieter dieser Einrichtung vertrauen müssen.

Hinweis: Dieser Vorgang erfordert neben einer TCP auch die Unterstützung durch ein Trusted-OS.

4.5.2 Direct Anonymous Attestation (DAA)

Die ursprünglich vorgeschlagene Lösung (TPM-Spezifikation Version 1.1) des Datenschutzproblems benötigt eine vertrauenswürdige Dritte Partei. Diese *Privacy-CA* signiert alle neu erzeugten AIKs, sofern die Plattform bestimmte festgelegte Richtlinien erfüllt, diese werden nachgewiesen durch die Plattform-Zertifikate. Die Nachteile liegen in der notwendigen Hochverfügbarkeit und dem zentralen Angriffspunkt hinsichtlich der Privatsphäre der Benutzer.

Deshalb wurde mit der TPM-Spezifikation Version 1.2 eine als *Direct Anonymous Attestation* (DAA) bezeichnete Technik eingeführt. Durch ein kryptographisches Verfahren (Gruppensignaturschema) kann man die vertrauenswürdige Dritte Partei einsparen und die Beglaubigung direkt zwischen den beteiligten Instanzen durchführen. Einen wichtigen Baustein dieser Technik bilden so genannte *Zero-Knowledge-Protokolle*[29]. Sie zeigen einem Verifizierer (Serviceanbieter) die Gültigkeit eines erzeugten AIK, ohne dass dabei Wissen über den korrespondierenden EK preisgegeben wird. So kann man einem Dritten versichern, den Lösungsweg zu kennen, ohne diesen Weg erläutern zu müssen.

Allerdings existieren auch bei DAA Einschränkungen bzgl. der gewährten Anonymität. Beispielsweise gibt es einen bestimmten Betriebsmodus (Named-Base Pseudonym, Rogue Tagging), der auf Wunsch des Verifizierers das Erkennen einer wiederholten bzw. missbräuchlichen Nutzung erlaubt. Damit ist eine Verkettung der durchgeführten Dienstanforderungen möglich, was natürlich die Anonymität einschränkt. Ferner sieht der Standard eine optionale *Anonymity Revocation Authority* vor, um den gesetzlichen Vorschriften einiger Staaten zu entsprechen.

Der Einsatz des DAA-Verfahrens hat auf Grund der erwähnten Probleme ein niedrigeres Integritätsniveau als der Einsatz einer *Privacy-CA*. [Camenisch001] beschreibt eine von IBM Research entwickelte Modifikation des DAA-Verfahrens. Nach Aussage des Autors liefert diese Implementierung ein zum *Privacy-CA*-Verfahren äquivalentes Integritätsniveau, jedoch ohne die aufgezeigte Problematik dieses Verfahrens.

4.5.3 Löschen des Endorsement Key

Ebenfalls ab der Version 1.2 der TPM-Spezifikationen definiert die TCG eine optionale Funktion zum Entfernen des EK und somit der Plattform-Identität. Wird diese

[29] [URL08]

Funktion von einem TPM-Hersteller implementiert, kann mittels *TPM_RevokeTrust* der im TPM hinterlegte private Teil des EK unwiederbringlich entfernt werden. Ob ein TPM diese Funktion unterstützt, ist in der erweiterten Konfiguration des TPM (*Permanent Flags*) hinterlegt. Um diesen Befehl absetzen zu können, muss der physikalische Zugriff auf das System (*Proof of Physical Presence*) nachgewiesen werden. Zusätzlich ist dieser Befehl durch ein Passwort gesichert, welches ungleich dem Eigentümer-Passwort ist und vom Hersteller des TPM definiert wird.

Um den vollen Funktionsumfang eines TPM nutzen zu können, muss sich dieser im Zustand *Owned* befinden. Um das für diesen Zustandsübergang zuständige Kommando *TPM_TakeOwnership* auszuführen, muss das TPM jedoch über einen EK verfügen. Hierfür kann mittels *TPM_CreateRevocableEK* ein neues EK-Schlüsselpaar im TPM erzeugt werden.

Viele der vom TPM angebotenen Funktionen unterliegen durch die Verwendung des selbst erzeugten EK keinerlei Einschränkungen. Jedoch verlieren die zur Plattform gehörigen Zertifikate aufgrund ihrer Beziehung zum *Endorsement Key* ihre Gültigkeit. Somit kann gegenüber einer entfernten Instanz nicht mehr nachgewiesen werden, dass es sich um eine konforme TPM-Implementierung handelt. Dies kann dazu führen, dass eine *Privacy-CA* die Anfrage der Plattform ablehnt und es somit unmöglich wird, einen AIK zu erhalten. Technisch ist es zwar durchaus möglich, auch für einen selbst erstellten EK solche Zertifikate auszustellen, auf Grund der Beziehungen zwischen den einzelnen Zertifikaten und den an der Ausstellung beteiligten Parteien würde dies jedoch einen sehr komplexen Prozess erfordern.

4.5.4 Deaktivieren des TPM

Neben der Möglichkeit das TPM permanent in den Zustand *Deactivated* zu versetzen, besteht zusätzlich die Option, das TPM bis zum nächsten Systemstart zu deaktivieren. Dies kann z. B. dann sinnvoll sein, wenn der Benutzer den sicheren Schlüsselspeicher des TPM verwendet, sich aber sicher sein will, dass während eines bestimmten Vorgangs (z. B. Zugriff auf das Internet) kein Zugriff auf die Funktionen des TPM möglich ist. Diese temporäre Deaktivierung wird durch den Aufruf von *TPM_SetTempDeactivated* eingeleitet. Für diesen Aufruf wird ein Passwort benötigt, welches zuvor vom Eigentümer der Plattform mittels *TPM_SetOperatorAuth* eingerichtet wurde.

4.6 Plattform-Zertifikate (Platform Credentials)

Wie bereits mehrfach angesprochen, basiert das Vertrauen in das TPM bzw. in den im TPM hinterlegten *Endorsement Key* auf den Plattform-Zertifikaten. Durch die Verifizierung dieser Zertifikate kann sich eine entfernte Instanz davon überzeugen, dass es sich um ein zu den TCG-Spezifikationen konformes TPM handelt, und so-

mit davon, dass der private Teil des EK nur dem TPM bekannt ist. Die folgende Abbildung beschreibt die Felder der einzelnen Zertifikate und deren Beziehungen untereinander:

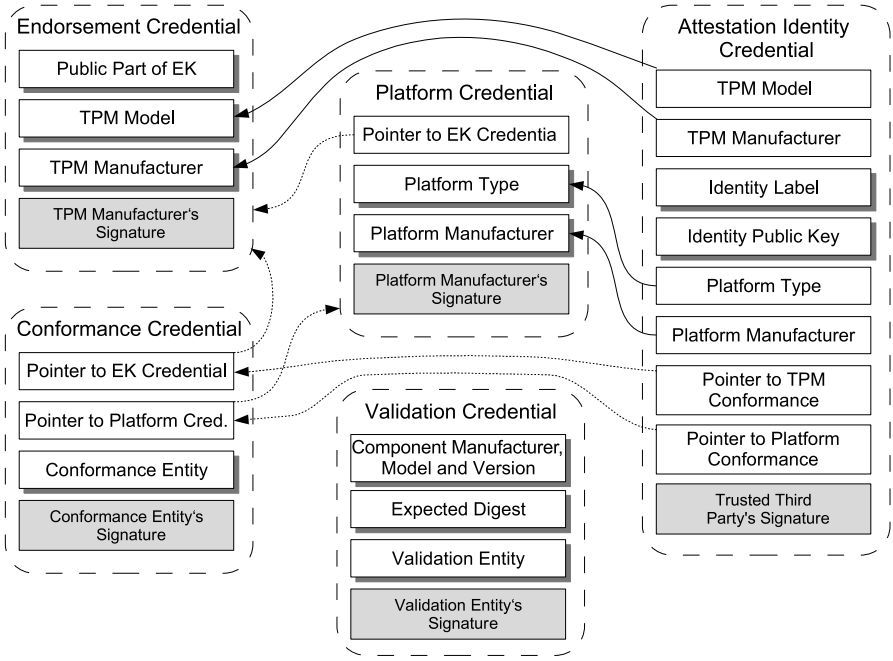

Abb. 4.18 Plattform-Zertifikate und deren Beziehungen [TCG2004]

Diese Zertifikate werden u. a. von einer*Privacy-CA* vor der Ausstellung eines *Attestation Identity Credential* ausgewertet. Jedes dieser Zertifikate bescheinigt die Vertrauenswürdigkeit einer bestimmten Komponente:

- **Endorsement-Zertifikat (Endorsement Credential)**
  Bestätigt die Echtheit des TPM. Genau genommen wird sichergestellt, dass das TPM von einem autorisierten Hersteller bereitgestellt wurde. Das TPM wird in dem Zertifikat durch ein 2048 Bit langes Schlüsselpaar, den EK, eindeutig repräsentiert. Dieses Schlüsselpaar wird entweder bei der Herstellung des TPM-Chips vom Hersteller erzeugt oder erst zu einem späteren Zeitpunkt im Chip gebildet. In beiden Fällen verlässt der *Endorsement Key* das TPM niemals. Der *Endorsement Key* wird verwendet, um sog. *Attestation Identity Keys* (AIK) herzustellen. Ein AIK repräsentiert einen Alias im Sinne einer anonymen Identität. Dienstleister können durch Prüfung eines für seinen Service erzeugten AIK sicher sein, eine vertrauenswürdige Plattform zu bedienen.

- **Plattform-Zertifikat (Platform Credential)**
  Wird vom Hersteller der Plattform, also etwa eines PCs, Laptops oder Mobil-

telefons ausgestellt. Es bestätigt, dass alle Plattform-Komponenten der TCG-Spezifikation genügen und dass die Plattform ein gültiges TPM enthält. Es wird also bescheinigt, dass das aktuelle System eine vertrauenswürdige Plattform darstellt.

- **Conformance-Zertifikat (Conformance Credential)**
  Bestätigt, dass das TPM-Design in der Plattform der TCG-Spezifikation genügt und das TPM korrekt implementiert ist.

- **Validation-Zertifikat (Validation Credential)**
  Stellt für Komponenten oder Komponentengruppen (wie Grafikkarten oder Eingabegeräte) die Übereinstimmung und Korrektheit der Implementierung gegenüber der TCG-Spezifikation sicher.

Mehr Informationen zu den einzelnen Zertifikaten und deren Beziehungen liefert die TCG-Spezifikation [TCG2006-4]. Da diese Zertifikate u. a. für die Erzeugung der AIK benötigt werden, muss der Plattform-Eigentümer bzw. eine vom Eigentümer verwendete Applikation Zugriff auf diese Zertifikate erhalten. Die TCG definiert für die Verteilung dieser Zertifikate eine Reihe möglicher Optionen:

- Speicherung auf einer der Plattform beigelegten CD
- Speicherung auf der Festplatte der Plattform
- Abruf von den Internetseiten des Plattform-Herstellers
- Speicherung im nicht-flüchtigen Speicherbereich des TPM[30]

4.7 Probleme und Einschränkungen der TCP

Die TCP der TCG unterliegt zum aktuellen Zeitpunkt einer Reihe von Einschränkungen. Diese Einschränkungen lassen sich in technische Schwächen sowie konzeptuelle bzw. infrastrukturelle Probleme aufteilen.

Als technische Kritikpunkte werden häufig der SHA-1-Algorithmus und die Anbindung des TPM an das Mainboard des Systems angeführt. So gelten die SHA-1-Hashfunktionen der Länge 160 Bit zumindest im akademischen Umfeld als „gebrochen“, nachdem es einem Team von chinesischen Forschern im Februar 2005 gelungen war, die Zeitkomplexität bei der Suche nach Kollisionen zu verringern [Wang2005]. Auch wenn der beschriebene Angriff nach Meinung vieler Experten für die Praxis bisher nur bedingt relevant ist, ist davon auszugehen, dass die nächste Generation der TPM eine Alternative zum SHA-1 enthalten wird. Weiter wird kritisiert, dass die Kommunikation zwischen dem TPM und den restlichen

[30] Die im Rahmen des Trusted-Java-Projekts entstehenden jTpm Tools bieten die Möglichkeit das EK-Zertifikat aus TPMs des Herstellers Infineon auszulesen [URL09].

Komponenten des Systems nicht gesichert ist. [Kursawe2005] beschreibt mögliche passive und aktive Angriffe auf den physikalischen Kommunikationskanal zum TPM. Denkbare Gegenmaßnahmen wären eine Verschlüsselung der Kommunikation auf der physikalischen Schicht oder eine Integration des TPM in den Chipsatz oder besser direkt in die CPU.

Eine andere technische Einschränkung des TPM ist die Beschränkung des TPM auf asymmetrische Verschlüsselungsverfahren. Das TPM bietet lediglich die Möglichkeit der sicheren Ablage von symmetrischem Schlüsselmaterial innerhalb seiner Schlüsselverwaltung und somit lediglich einen Schutz gegen Offline-Angriffe. Das TPM ist daher nicht wie häufig angenommen als Ersatz für *Hardware-Security-Module* (HSM) zur Beschleunigung des DES- oder AES-Verfahrens geeignet. Selbst mit Unterstützung für symmetrische Verfahren wäre der für die Anbindung des TPM verwendete LPC-Bus mit seinem maximalen Durchsatz von ca. 4 Mbyte/s für viele Anwendungen zu langsam. Für die Erreichung der beiden Ziele *Protected Execution* und *Protected I/O* bietet die TCP der TCG ebenfalls keine Lösung.

Nach Meinung des Autors jedoch deutlich schwerwiegender sind die Einschränkungen konzeptueller Natur. So sind die beiden Kernfunktionen der TCP, namentlich die *Vertrauenskette* (Chain of Trust) und die *Remote Attestation*, ohne die Unterstützung durch ein *Trusted-OS* und eine *Trusted-Computing-Infrastruktur (TCI)* nahezu wirkungslos. Dies wird in den meisten Publikationen und sämtlichen Spezifikationen der TCG nicht explizit erwähnt. Die durch die TCP aufgebaute *Vertrauenskette* reicht nämlich nur bis zum MBR des Betriebssystems und bildet somit nur einen kleinen Teil der am Startvorgang beteiligten Komponenten ab. Eine Bewertung der Integrität eines Systems auf Basis der durch die TCP erzeugten Prüfsummen ist unzulänglich, da hierbei lediglich eine Betrachtung eines in der Vergangenheit liegenden Zustands möglich ist. Die Komplexität der notwendigen Fortsetzung der *Vertrauenskette* durch das Betriebssystem wird ab Seite 71 beschrieben. Die Überprüfung der Integrität des Systems auf einem entfernten System (*Remote Attestation*) sowie die dafür notwendige Erzeugung der *Attestation Identity Keys* (AIK) erfordern Zugriff auf die Plattform-Zertifikate (*Platform Credentials*) und den Aufbau einer komplexen *Public Key Infrastruktur* (PKI). Beides ist zum aktuellen Zeitpunkt nicht gegeben. Gerade die PKI ist jedoch für nahezu alle Anwendungsfälle des *Trusted Computing* von elementarer Bedeutung. Auch dieser Aspekt wird in den meisten Publikationen vernachlässigt.

Ein weiteres Problem ist die Überprüfung der korrekten Implementierung eines TPMs. Dies sollte eigentlich durch die Spezifikationen der TCG und die Plattform-Zertifikate sichergestellt sein, jedoch zeigt eine Untersuchung [Sadeghi2006] der Ruhr-Universität Bochum, dass einige Implementierungen von TPMs erheblich von den Spezifikationen abweichen.

Kapitel 5
Erweiterungen und Alternativen zur TCG

Wie im vorhergehenden Kapitel erwähnt unterliegt die TCP der TCG einigen Einschränkung im Bezug auf die eingangs definierten Ziele des Trusted Computing. Am schwersten wiegt hierbei das Fehlen einer Lösung für den *Protected-Execution*-Ansatz. Diesen Mangel haben auch die CPU- und Chipsatz-Hersteller erkannt und integrieren ihrerseits eine Reihe neuer Sicherheitsfunktionen in die aktuelle Generation ihrer Produkte.

5.1 Intel Trusted Execution Technology (TXT)

Intel beschreibt TXT, vormals bekannt unter dem Namen *LaGrande*, als ein Bündel von Hardware-Erweiterungen an Prozessor, Chipsatz, Speichercontroller und Ein-/Ausgabe-Systemen, die einen gewöhnlichen PC in eine *Trusted Computing Platform* verwandeln sollen. Ein Großteil der Erweiterungen sind in den Business-Bürorechner auf Basis der aktuellen vPro-Plattform (vPro 2) bereits enthalten. Bei Notebooks nennt Intel die entsprechende Plattform Centrino vPro. Intel TXT adressiert einige der identifizierten Schwachstellen der TCP der TCG. So bietet sie u. a. eine mögliche Lösung für den *Protected-Execution*- und den *Protected-I/O*-Ansatz.

Hierfür kommen außer einem TXT-tauglichen Prozessor, einem erweiterten Chipsatz und einem *Trusted Platform Module* (TPM 1.2) auch noch spezielle signierte Software-Module namens *Authenticated-Code* (AC)-Modules zum Einsatz. Eines lagert als *BIOS-AC* im Flash-Speicherchip des BIOS, ein weiteres, chipsatzspezifisches *SINIT-AC* liefert Intel; es dient der Zertifizierung eines *Virtual Machine Manager* (VMM) oder *Hypervisors*. Die CPU lädt die beiden AC-Module in einen speziellen Cache namens *Authenticated Code RAM* (ACRAM). Die Abb. 5.1 zeigt die für Intel TXT notwendigen Erweiterungen der PC-Referenzarchitektur.

Durch das Zusammenspiel dieser Komponenten soll sichergestellt werden, dass kein fremder Code die Vertrauenswürdigkeit der Plattform beeinträchtigt (*Measurement of Trust*). Im Wesentlichen dienen die neuen Funktionen dem protokollierten Start eines VMM (*Measured Virtual Machine Monitor* – MVMM), der mit den

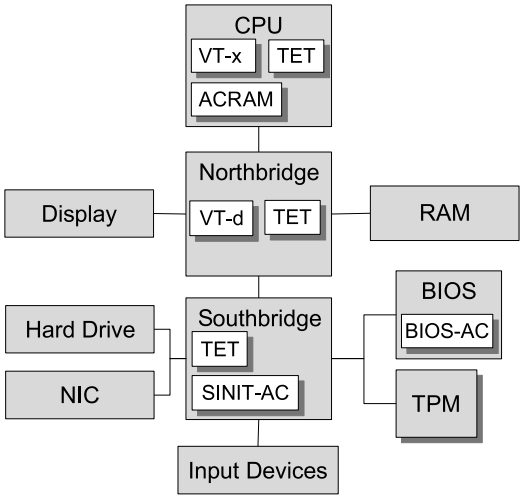

Abb. 5.1 Intel-Trusted-Execution-Technologie

maximalen Systemrechten in einer abgeschirmten Ausführungsumgebung operiert. Dieser bildet dadurch eine durch die Hardware geschützte Implementierung eines *Secure Kernels* und ist somit Teil der *Trusted Computing Base* (TCB). Die für den Start eines MVMM notwendigen CPU-Erweiterungen fasst Intel unter dem Begriff *Safer Machine Extensions* (SMX) zusammen. Sie stellen eine Erweiterung der bereits in vielen CPUs von Intel enthaltenen Funktionen zur Virtualisierung der CPU (*Virtual Machine Extensions* – VMX) bzw. VT-x dar. Über die *Safer Machine Extensions* (SMX) kann ein gestarteter MVMM zusätzlich vertrauenswürdige virtuelle Maschinen starten (*Trusted Virtual Machine*, TVM), die nicht aus ihrem Speicherbereich ausbrechen können sollen. Hierfür wurde auch der Chipsatz um einige Schutzfunktionen erweitert. Das wichtigste der neuen Features ist die *Memory Protection Table* (MPT). Die MPT, unter Kontrolle des MVMM, erlaubt es, bestimmte Bereiche des physikalischen Arbeitsspeichers vor unberechtigten Zugriffen durch andere Hardwaregeräte zu schützen. Damit ist es möglich den Speicherbereich des VMM und den der virtuellen Maschinen von einem Zugriff über DMA[1] auszuschließen.

SMX definiert zwei neue Ausführungsmodi, den *Authenticated Code Execution Mode* und den *Protected Execution Mode*. Der *Authenticated Code Execution Mode* erlaubt das Laden von digital signierten Modulen (*Authenticated Code Modules*). Diese Module werden in den CPU-internen RAM der *Authenticated Code Execution Area* (ACRAM) geladen und dort ausgeführt. Die CPU erzeugt hierbei eine Prüfsumme über das AC-Modul und vergleicht diesen Wert mit den in der Signatur hinterlegten Daten. Der Code des Moduls wird nur im Falle einer positiven Überprüfung ausgeführt. Nur Code, der innerhalb des ACRAM liegt, erhält die notwendigen Rechte zur Steuerung der erweiterten Chipsatz-Funktionen und somit die Möglichkeit zum Wechsel in den *Protected Execution Mode*. Code innerhalb des *Protected*

[1] DMA – Direct Memory Access

Execution Mode unterliegt dem Schutz durch die MPT. Die nachfolgende Abbildung beschreibt den Ablauf bei der Erzeugung einer *Protected-Execution*-Umgebung:

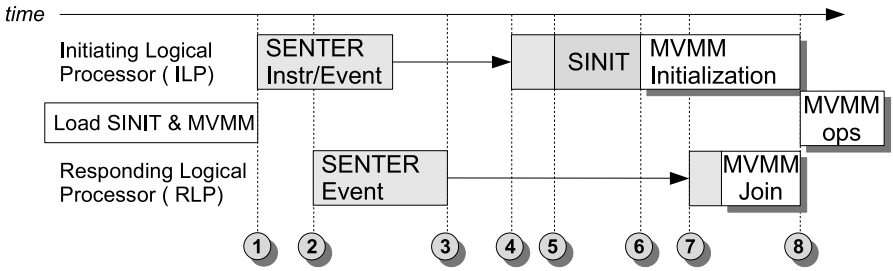

Abb. 5.2 Intel TXT – Secure Launch

1. Das SINIT-AC-Modul und der MVMM werden in den Arbeitsspeicher geladen und *SENTER* wird aufgerufen.
2. Durch Aufruf des CPU-Befehls *SENTER* werden der Chipsatz und alle weiteren Co-Prozessoren des Systems über den Aufbau einer geschützten Ausführungsumgebung informiert. Der Chipsatz sorgt hierbei für die Isolation des angeforderten Arbeitsspeichers.
3. Alle Co-Prozessoren des Systems melden ihre Bereitschaft.
4. Die CPU setzt die Ausführung fort.
5. Die CPU lädt das SINIT-AC-Modul in die *Authenticated Code Execution Area* (ACRAM) und überprüft dessen digitale Signatur. Die zur Überprüfung durch die CPU erzeugte Prüfsumme wird anschließend im *Platform Configuration Register* (PCR-17) hinterlegt. Das AC-Modul überprüft den aktuellen Zustand der CPU und des Chipsatzes und stellt sicher, dass eine legitime Konfiguration vorliegt.
6. Das AC-Modul erzeugt eine Prüfsumme über den Code des MVMM und startet diesen. Die Prüfsumme wird ebenfalls im TPM hinterlegt (PCR-18).
7. Der MVMM informiert die Co-Prozessoren über die erfolgreiche Ausführung des SINIT-Moduls.
8. Der MVMM benutzt nun die Befehle der *Virtual Machine Extensions* zur weiteren Konfiguration des Systems (z. B. Umleitung bestimmter Interrupts) und startet das erste Gast-Betriebssystem innerhalb des *Protected Execution Mode.*

Die ersten Desktop-PCs und Notebooks mit Unterstützung für Intel TXT sind bereits verfügbar. Allerdings sind diese Erweiterungen nur bei entsprechender Unterstützung durch Betriebssysteme und Virtualisierungslösungen wirkungsvoll. Im Rahmen des XenSE-Projektes (Security Enhanced Xen) sind bereits erste Patches zur Verwendung von TXT durch Xen veröffentlicht worden. Die Kombination von Intel TXT oder vergleichbaren Technologien mit den ab Seite 111 vorgestellten Lösungen definiert, nach Meinung des Autors, die nächste Generation der *Trusted-Computing-Systeme*.

5.2 AMD Presidio Technology

Auch AMD arbeitet an einer mit der Intel TXT vergleichbaren Erweiterung der Funktionalität von CPU und Chipsatz. Die unter dem Namen *Presidio* oder auch *Secure Execution Mode* (SEM) bekannte Erweiterung stellt hierbei ebenfalls die nächste Evolutionsstufe der mit *Pacifica* oder auch *Secure Virtual Machine* (SVM) eingeführten Technologie zur Virtualisierung der CPU sowie des Speicher-Controllers dar. Zur SEM wurden bisher von AMD keine Spezifikationen veröffentlicht, das Folgende ist daher eine Zusammenfassung bereits vorhandener Informationen.

Bereits mit der Einführung der Befehlssatzerweiterung SVM unterstützen die CPUs von AMD die für die Isolation von Prozessen notwendige Virtualisierung des Arbeitsspeichers. Mit dem *Device Exclusion Vector* (DEV) besitzen aktuelle CPUs auch bereits eine Funktion zum Schutz vor DMA-Zugriffen auf den Speicherbereich der virtuellen Maschinen. Eine Erweiterung der Chipsatz-Funktion wie bei Intel war nicht notwendig, da sich der Speicher-Controller in der Architektur von AMD direkt in der CPU befindet. Die folgende Abbildung veranschaulicht diesen Aufbau:

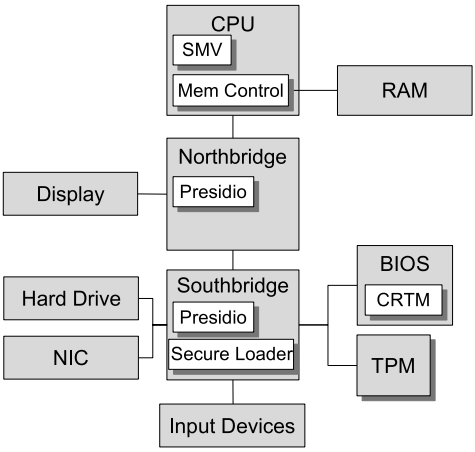

Abb. 5.3 AMD Presidio

Mit SEM neu eingeführt wurde der CPU-Befehl *SKINIT*, der das Pendant zum *SENTER*-Befehl von Intel darstellt. Auch bei AMD wird dieser Befehl dazu verwendet, einen *Virtual Machine Monitor* (VMM) in einer isolierten Ausführungsumgebung zu starten. Der *Pacifica*-Befehl SKINIT startet hierzu das *Secure Loader Image* (SL). Dieses geschützte Programm dient analog zum SINIT-AC-Modul von Intel der sicheren Initialisierung des VMM-Codes. Der Secure Loader besteht aus einem 64 kByte großen Secure Loader Image. Dieses Image liegt im so genannten Secure Loader Block, einer beliebigen 64-kByte-gebundenen Adresse unterhalb von 4 GByte. Im Gegensatz zu Intels ACRAM wird dieses Programm somit innerhalb des regulären Arbeitsspeichers ausgeführt.

Während *Pacifica* bereits in den meisten aktuellen AMD CPUs enthalten ist, ist dem Autor nicht bekannt, wann die ersten Rechner mit *Secure Execution Mode*-Support verfügbar sein werden. Auch im Falle von SEM hat das XenSE-Projekt eine kommende Unterstützung der Technologie angekündigt. Die folgende Tabelle liefert abschließend eine Übersicht über die bereits verfügbaren und noch kommenden (*) Sicherheitsfunktionen von Intel und AMD:

Tabelle 5.1 Vergleich Intel TXT und AMD Presidio

Funktion	**Intel TXT (LaGrande)**	**AMD Presidio**
CPU-Virtualisierung	VT-x, VT-x2 (Vanderpol)	SVM (Pacifica)
Speicher-Virtualisierung	SMX (TXT)	SVM (Pacifica)
I/O-Virtualisierung (IOMMU)	VT-d	SEM (Presidio)*
CPU/Chipsatz-Befehlssatz-erweiterungen	Virtual Machine Extensions (VMX), Safer Machine Extensions (SMX)	Secure Virtual Machine (SVM), Secure Execution Mode (SEM)*

5.3 IBM SecureBlue

Auch IBM hat unter dem Namen *SecureBlue* eine Erweiterung für Prozessoren angekündigt. Hierbei soll der Prozessor um Schaltkreise erweitert werden, die eine Ver- und Entschlüsselung von Daten direkt auf der CPU erlauben. Dieses Konzept wurde speziell für die PowerPC-Architektur entwickelt, soll laut IBM jedoch auch in Desktop- und Mobil-Prozessoren von Intel und AMD integriert werden können. Außer den in der Pressemitteilung[2] enthaltenen Informationen sind keine näheren Details verfügbar. So ist bisher unklar, welche Verschlüsselungsverfahren von *SecureBlue* unterstützt werden, und ob die Technologie als eine Erweiterung oder als Alternative zur *Trusted Computing Platform* der TCG zu sehen ist. Eine Integration von symmetrischen Verschlüsselungsverfahren (z. B. DES, AES) in die CPU hätte den Vorteil, dass das verwendete Schlüsselmaterial sich nicht im Hauptspeicher des Systems befindet und somit nicht extrahiert werden kann. Eine sichere Verbindung zwischen CPU und TPM vorausgesetzt, könnte dann vom TPM verwaltetes symmetrisches Schlüsselmaterial direkt in die CPU geladen werden.

[2] [URL10]

Kapitel 6
Anforderungen an vertrauenswürdige Betriebssysteme

Ein *Trusted Computing System* besteht aus der beschriebenen *Trusted Computing Platform* und einem darauf aufbauenden vertrauenswürdigen Betriebssystem (*Trusted-OS*). Für die Umsetzung eines *Trusted-OS* gibt es mindestens zwei unterschiedliche Ansätze, zum einem in Form einer um entsprechende Funktionen erweiterten klassischen Betriebssystem-Architektur. Zum anderen als Umsetzung in Form einer vertrauenswürdigen Virtualisierungsschicht (*Trusted Virtual Machine Monitor*). Der Begriff *Trusted-OS* steht daher im Rahmen dieser Arbeit stellvertretend für beide Ansätze. Unabhängig von der gewählten Form der Implementierung des *Trusted-OS* muss es eine Reihe von Anforderungen erfüllen, um die zu Beginn der Arbeit identifizierten Ziele zu erfüllen. Im Rahmen dieses Kapitels werden diese dafür notwendigen Funktionen in Form eines Anforderungskatalogs beschrieben. Die folgenden Kapitel widmen sich dann sowohl theoretischen als auch aktuell verfügbaren praktischen Lösungsansätzen. Ebenso dient dieser Anforderungskatalog als Grundlage für die Untersuchung einiger neuer Sicherheitsfunktionen in Windows Vista im letzten Kapitel des Buches.

6.1 Dynamic Chain of Trust (Integrity Measurement)

Wie beschrieben, erzeugt die *Trusted Computing Platform* Prüfsummen aller am Systemstart beteiligten Komponenten bis zum MBR. Dies wird als *Static Chain of Trust* bezeichnet. Der ausführbare Teil des MBR ist bereits Teil des Betriebssystems und enthält Instruktionen zum Start der nächsten Komponente. Dies ist in der Regel ein Bootmanager, der eine Auswahl der zu startenden Betriebssysteme anbietet. Der Bootmanager ruft entsprechend der getätigten Auswahl das Startprogramm des jeweiligen Betriebssystems (*OS-Loader*) auf, welches die nötigen Boot-Treiber und schließlich den Kern des Betriebssystems (*OS-Kernel*) startet. Der OS-Kernel lädt dann die zur Laufzeit verwendeten Treiber und startet weitere Dienste (Daemons oder Services). Um die *Chain of Trust* im Sinne des *Trusted Computing* fortsetzen zu können, muss ein *Trusted-OS* jeden der beschriebenen Schritte um die Erzeugung

entsprechender Prüfsummen ergänzen. Konkret bedeutet dies, bereits der MBR benötigt zusätzliche Routinen, die eine Messung des Bootmanagers ermöglichen, bevor dessen Instruktionen ausgeführt werden. Der Bootmanager wiederum muss eine Möglichkeit vorsehen, eine Prüfsumme über den *OS-Loader* zu erzeugen. Dieses Konzept muss bis zum vollständigen Start des Betriebssystems fortgesetzt werden und bildet somit den ersten Teil der *Dynamic Chain of Trust*. Eine Umsetzung dieses Konzeptes wird in der Fachliteratur als „*Trusted Boot*“ oder „*Authenticated Boot*“ bezeichnet [Smith2005].

6.2 Dynamic Chain of Trust (Integrity Protection)

Nach der erfolgreichen Umsetzung des ersten Teils der *Dynamic Chain of Trust* befindet sich das System in einem durch die erzeugten Prüfsummen repräsentierten und somit bekannten Zustand. Dies bedeutet, dass sämtliche am Startvorgang beteiligten und alle im Arbeitsspeicherbereich des Betriebssystems befindlichen Komponenten vor ihrer Ausführung protokolliert wurden. Eine weitere Aufgabe des *Trusted-OS* ist es nun, diese Protokollierung über die Laufzeit des Betriebssystems fortzuführen, um auch zu einem späteren Zeitpunkt eine korrekte Aussage über den Zustand des Systems geben zu können. Hierfür müssen alle Softwarekomponenten (z. B. Treiber, Applikationen, Libraries, etc.), die vom Betriebssystemkern nachgeladen werden, die *Vertrauenskette* durch die Erzeugung weiterer Prüfsummen erweitern. Diese Maßnahme ist jedoch wenig effektiv, wenn die Instruktionen nach dem Laden in den Speicherbereich des Betriebssystems verändert werden können. Es muss also sichergestellt sein, dass ein Zugriff auf diese Speicherbereiche nur von vertrauenswürdigen Komponenten durchgeführt werden kann. Zusätzlich sollten die kritischen Datenstrukturen im Speicher regelmäßig mit den vor dem Laden erzeugten Prüfsummen verglichen und somit überwacht werden. Als weiterer Schutz sollten alle zum Betriebssystem gehörigen Dateien bereits auf dem Datenträger vor unberechtigter Manipulation geschützt werden.

Somit ist *Integrity Protection* eine Ausweitung des *Integrity-Measurement*-Ansatzes auf die komplette Laufzeit des Systems. Obwohl die Implementierung dieses Konzeptes maßgeblich für die Integrität des Systems verantwortlich ist, wird dieser Aspekt bei der Beschreibung von Einsatzszenarien für *Trusted Computing* oft vernachlässigt. Hierbei wird oft verschwiegen, dass eine Auswertung der während des Bootvorgangs erzeugten Prüfsummen lediglich die Bewertung eines in der Vergangenheit liegenden Zustandes des Systems erlaubt. Eine etwaige Kompromittierung zur Laufzeit des Systems durch Schadsoftware (z. B. Kernel-Mode-Rootkits) kann dadurch nicht erkannt werden.

Allerdings muss auch erwähnt werden, dass die Fortsetzung der *Vertrauenskette* zur Laufzeit um ein Vielfaches komplexer ist als die vorausgehenden Messungen. So kann das Konzept der Protokollierung einer Komponente vor ihrer Ausführung durch ihren direkten Vorgänger nicht auf die Laufzeit des Betriebssystems übertragen werden, da moderne Betriebssysteme mehrere Applikationen gleichzei-

tig ausführen können (Multi-Tasking). So findet spätestens nach dem vollständigen Start des Betriebssystems eine Überführung von der sequentiellen Ausführung hin zur parallelen Ausführung von Prozessen statt. Hierbei übernimmt in der Regel ein Prozess[1] des Betriebssystems das Laden und Ausführen von Applikationen. Applikationen selbst haben ebenfalls die Möglichkeit weitere Applikationen zu starten, wodurch eine Eltern-Kind-Beziehung zwischen Prozessen entsteht. Die Vertrauenskette wandelt sich dadurch zum *Vertrauensbaum* (*Tree of Trust*[2]), was die Erstellung geeigneter Prüfsummen und deren Auswertung zusätzlich erschwert.

Die größte Herausforderung der Abbildung des Systemzustands zur Laufzeit ist jedoch die nahezu unbegrenzte Vielfalt an Applikationen, die auf einem System zur Ausführung kommen können. Zusätzlich zu den Applikationen selbst sind häufig auch die Konfigurationsparameter sowie die durch die Applikation verarbeiteten Daten für die Integrität des Systems relevant. Selbst wenn nur vertrauenswürdige (z. B. digital signierte) Applikationen auf dem System ausgeführt werden, kann nicht gewährleistet werden, dass diese nicht durch eine falsche Konfiguration oder durch die Manipulation der zu verarbeitenden Daten die Integrität des Systems gefährden. Auch die Erzeugung von Prüfsummen über die Eingabedaten der Applikationen ist nur zu einem bestimmten Grad praktikabel, da das hierbei entstehende Protokoll sehr schnell eine zu prüfende Größe überschreitet und die Erstellung der notwendigen Referenzwerte nicht möglich ist. Besonders problematisch sind Applikationen, deren Funktionsumfang nicht beschrieben werden kann, da er beliebig erweiterbar ist. Hierzu zählen Applikationen mit Plugin-Funktion, Interpreter von Skriptsprachen (z. B. Bash, Perl, Python, Ruby, Javascript) sowie Applikationen zur Bereitstellung von virtuellen Maschinen (z. B. Java VM, .NET CLR, VMWare). Eine Implementierung des *Integrity Protection* muss deshalb zwischen der möglichst vollständigen Erfassung des Systemzustands und einer noch verwertbaren Größe des Applikationsprotokolls abwägen.

6.3 Bewertung der Systemintegrität (Integrity Validation)

Entgegen der selbst in Fachartikeln zu findenden Aussagen, die *Trusted Computing Platform* stelle die Integrität eines Computersystems sicher, bietet sie lediglich die Möglichkeit zur Protokollierung der ersten Instanzen des Startvorgangs und somit zur Identifikation der beteiligten Komponenten. Es ist daher essentiell, zwischen der reinen Erzeugung von Prüfsummen und der Bewertung der selbigen zu unterscheiden. Eine Bewertung der Integrität dieser Komponenten basiert zwar auf den erzeugten Prüfsummen, ist jedoch Aufgabe einer weiteren Komponente und erfordert die Implementierung entsprechender Metriken. Diese Komponente kann entweder Teil des *Trusted-OS* sein oder sich auf einem externen System befinden. Auf jeden Fall muss hierfür zumindest ein Teil der durch die *Trusted Computing Platform* im TPM

[1] In UNIX-artigen Betriebssystemen ist dies der init-Prozess. Unter Windows ist dies für Dienste des Betriebssystems der wininit.exe-Prozess und für Applikation die explorer.exe.

[2] Der Begriff Tree of Trust wurde vom Autor für dieses Buch definiert.

hinterlegten und sämtliche während des restlichen Bootvorgangs (Dynamic *Chain of Trust*) oder zur Laufzeit erzeugten Prüfsummen mit entsprechenden Referenzwerten verglichen werden. Zu welchen Zeitpunkten diese Überprüfung stattfindet, ist vom konkreten Anwendungsfall abhängig. Findet die Bewertung auf dem System selbst statt, muss hierbei darauf geachtet werden, dass die prüfende Komponente ein Teil der *Vertrauenskette* ist und somit selbst über eine Prüfsumme verfügt. Der Mechanismus, mit dem sichergestellt wird, dass ein Systemstart nur im Falle einer positiven Integritätsprüfung stattfindet, heißt „*Secure Boot*“ [Smith2005].

Es sei zusätzlich erwähnt, dass eine Bewertung der Integrität durch das System selbst ein nicht zu unterschätzendes Problem darstellt. Hierbei befinden sich nämlich die zu überprüfenden Teile auf dem selben System wie die prüfende Komponente. Wurde die Integrität des Systems kompromittiert, ist auch nicht auszuschließen, dass die für die Bewertung zuständige Komponente modifiziert wurde, um auch im Falle abweichender Prüfsummen einen integeren Zustand zu attestieren. Eine zuverlässige Bewertung kann also nur auf einem separaten System erfolgen. Ein solches Verfahren wird häufig als *Remote Attestation* bezeichnet und findet im nächsten Abschnitt nähere Erläuterung.

6.4 Remote Attestation (Remote Integrity Validation)

Bei der *Remote Attestation* werden die durch die *Trusted Computing Platform* und das *Trusted-OS* erzeugten Prüfsummen zur Bewertung an ein externes System übertragen. Die aus diesen Werten gewonnenen Informationen über das System können als Grundlage für weitere Entscheidungen dienen. Beispielhaft hierfür seien Zugangskontrollsysteme genannt, die auf Basis der ermittelten Integrität des Systems den Zugang zu einem Netzwerk gestatten oder ablehnen. Ausgangspunkt für die Übermittlung der Prüfsummen muss das TPM des Systems sein, da nur so die Integrität der Daten durch das entfernte System verifiziert werden kann. Die in diesem Prozess involvierten Komponenten des TPM zeigt die Abbildung „Remote Attestation“ auf Seite 75.

Die Abb. 6.1 zeigt die an dem Gesamtprozess der *Remote Attestation* beteiligten Einheiten auf einer abstrakten Ebene und beschreibt den Ablauf des Prozesses.

Wie bereits im Kapitel zur *Trusted Computing Platform* beschrieben, erzeugen der CRTM und das BIOS den *Static Chain of Trust* (a) durch Ablegen entsprechender Prüfsummen im TPM. Der erste Teil des *Dynamic Chain of Trust* wird dann durch einen hierfür erweiterten *OS-Loader* bzw. *Bootmanager* erzeugt (b). Die Erzeugung von weiteren Prüfsummen zur Laufzeit des Systems erfolgt dann durch *Measurement Agents* (c), die als Dienst des Betriebssystems zu verstehen sind.

Um nun die Vertrauenswürdigkeit des Systems zu bewerten, fordert die entfernte Instanz, hier als *Attestation Server* bezeichnet, die erzeugten Prüfsummen über das Netzwerk an (1). Hierfür muss auf dem zu bewertenden System eine Netzwerkapplikation diese Funktionalität anbieten. Dieser als *Attestation Client* bezeichnete Dienst leitet diese Anfrage mittels TSS an das TPM weiter (2). Das TPM gibt

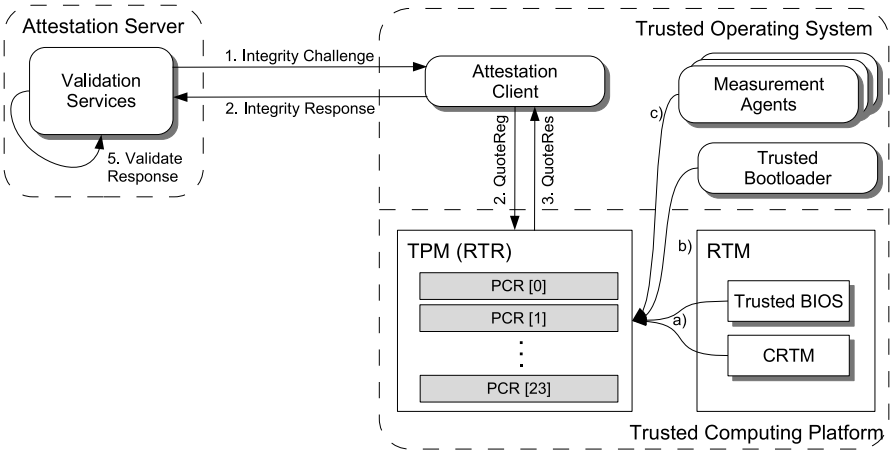

Abb. 6.1 Remote Attestation

daraufhin den Inhalt der PCR und eine digital signierte Prüfsumme dieser Werte an den *Attestation Client* zurück (3). Für die Erstellung der Signatur wird in der Regel einer der *Attestation Identity Keys* (AIK) verwendet. Diese Datenstruktur fließt zurück an die *Validation Services* auf dem *Attestation Server* (4). Nachdem die Integrität der übermittelten Daten überprüft[3] wurde, können die Prüfsummen mit Referenzwerten verglichen werden (5). Welche der Prüfsummen hierbei berücksichtigt werden oder welche Kriterien ausschlaggebend für das Ergebnis sind, hängt stark vom Einsatzszenarium dieses Konzeptes ab. Neben der Bewertung der Systemintegrität kann die übermittelte Datenstruktur auch zur Identifizierung bzw. Authentifizierung des Systems verwendet werden. Hierzu dient ebenfalls der verwendete *Attestation Identity Key* (AIK) bzw. das dazugehörige Zertifikat.

6.5 Trusted Software Stack

Um mit dem TPM zu kommunizieren und um den Applikationen eine Schnittstelle zu den Funktionen zu bieten, muss ein *Trusted-OS* einen Software Stack zur Verfügung stellen. In der Regel wird dies durch eine Implementierung der TCG-Spezifikation [TCG2006-2] für einen *Trusted Software Stack* (TSS) erreicht. Der TSS besteht aus mehreren Schichten, die unterschiedliche Dienste anbieten und den Zugriff auf das TPM für Applikationen abstrahieren. Eine vollständige Implementierung dieser Spezifikationen ist für die Umsetzung der im Zusammenhang mit *Trusted Computing* genannten Anwendungsfälle erforderlich. Die nachfolgende Abb. 6.2 beschreibt die einzelnen Schichten dieses Software Stacks und definiert deren Schnittstellen:

[3] Siehe Kapitel „Public Key Infrastructure (PKI)“

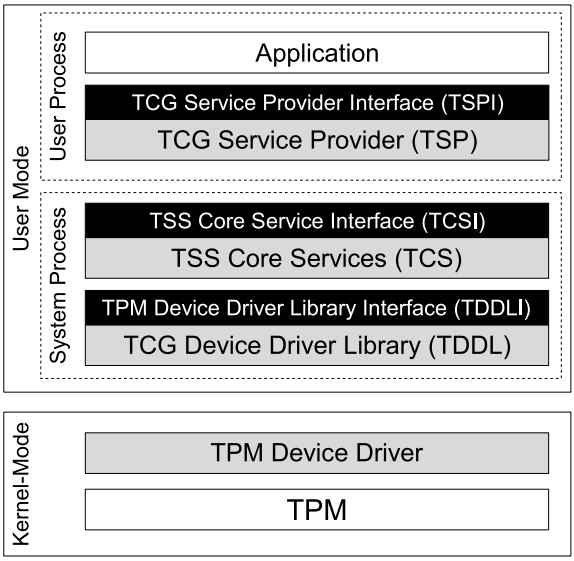

Abb. 6.2 Trusted Software Stack [TCG2006-2]

6.5.1 TPM Device Driver

TPM-Treiber (TPM Device Driver, TDD) gehören nicht direkt zum TSS. Sie werden von den TPM-Herstellern zur direkten Steuerung des TPMs zur Verfügung gestellt und müssen über die TPM Device Driver Library (TDDL) und die entsprechende Schnittstelle, das TPM Device Driver Library Interface (TDDLI), von den TSS-Komponenten ansprechbar sein. Für TPMs bis Version 1.1 musste für jeden TPM-Hersteller ein spezieller Treiber verwendet werden, ab Version 1.2 wurde auch die Schnittstelle zum TPM im Rahmen der TPM Interface Specification (TIS)[4] standardisiert. In aktuellen Betriebssystemen (Linux Kernel ab 2.6.18 und Windows Vista) gehören Treiber für diese Schnittstelle zum Lieferumfang.

6.5.2 TCG Device Driver Library (TDDL/TDDLI)

Die Device Driver Library befindet sich bereits im User Mode des Betriebssystems und stellt die Device-Driver-Library-Schnittstelle für die nächste Schicht zur Verfügung. Die Implementierung der Schnittstelle im User Mode hat einige Vorteile gegenüber einer Implementierung im Kern des Betriebssystems.

[4] [TCG2005-1]

- Sie bietet eine betriebssystemsneutrale Schnittstelle
- Sie stellt sicher, dass unterschiedliche Implementierungen des TSS mit jedem TPM kommunizieren können ohne den ganzen Stack zu modifizieren
- Sie ermöglicht die einfache Implementierung eines TPM-Simulators

Die TDDL regelt den Übergang zwischen Kernel- und User Mode, jedoch nicht den simultanen Zugriff auf das TPM, dies ist Aufgabe der höheren Schichten. Da das TPM nicht für die gleichzeitige Kommunikation mit mehreren Gegenstellen vorgesehen ist, existiert auf jeder Plattform nur eine Instanz der TDDL. Obwohl die TDDL in einigen Veröffentlichungen auch als TSS Device Driver Library bezeichnet wird, wird auch diese meist vom TPM-Hersteller zur Verfügung gestellt und ist somit nur bedingt Teil des TSS.

6.5.3 TSS Core Services (TCS/TCSI)

Die Core Services bieten eine definierte Schnittstelle für die darauf aufsetzenden Service Provider. Die TCS implementieren die fünf Basisdienste des TSS:

- Kontext Verwaltung – Regelt den simultanen Zugriff auf das TPM
- Zertifikats- und Schlüsselverwaltung – Implementiert die auf Seite 46 beschriebene Schlüsselverwaltung und somit die sichere Speicherung von Schlüsselmaterial außerhalb des TPM
- Measurement Event Management – Verwaltet die Einträge im Event Log und den Zugriff auf die zugehörigen PCR
- Generierung des Binärprotokolls – Zuständig für die Serialisierung, Synchronisierung und Übermittlung von Kommandos an das TPM

Abb. 6.3 TSS Core Services [TCG2006-2]

Die TCS arbeiten als System-Dienst im User Mode und sind in der Regel Teil des Betriebssystems.

6.5.4 TCG Service Provider (TSP/TSPI)

Eine weitere Ebene bilden die TSS Service Provider (TSP). Auch diese sind als Dienst oder Daemon implementiert, allerdings werden hier mehrere Exemplare pro TPM zur Verfügung gestellt. Nach einem objektorientierten Ansatz wird hierbei für jede Anwendung eine eigene Instanz geladen. Die Koordination des Zugriffs durch die Module erfolgt in den TCS.

In den TSPs werden unterschiedliche Funktionen zur Verfügung gestellt. Teilweise gehen diese über die TPM-Funktionalität hinaus. So werden die vom TPM gelieferten Daten hier in ein für Anwendungsprogramme besser lesbares Format umgewandelt. Dem TSS-Entwickler stehen dabei gewisse Freiheiten zu, wobei jeder TSS auf jeden Fall die Komponente TSP Cryptographic Functions (TSPCF) mit vorgegebenen kryptographischen Funktionen enthalten muss. Diese Komponente beinhaltet u. a. bekannte Krypto-Schnittstellen, wie MSCAPI[5] und PKCS#11[6], so dass existierende Anwendungen nur geringfügig angepasst werden müssen.

Damit Anwendungen die zur Verfügung stehenden Funktionen des TSP nutzen können, hat jeder TSP eine eigene Schnittstelle (ein TSP Interface), über die er angesteuert werden kann.

6.5.5 Einsatzszenarien des TSS

Für die Kommunikation mit den beschriebenen TSS Core Services gibt es eine Reihe unterschiedlicher Optionen. So kann die Verbindung zum TPM direkt über einen der TCG Service Provider hergestellt werden, hierfür muss die Applikation jedoch das TCG Service Provider Interface unterstützen. Sollen bereits vorhandene Applikationen (z. B. Email-Programme) anstelle von Software-Kryptographie-Modulen oder *SmartCards* das TPM verwenden, ist eine zusätzliche Umsetzung auf die bereits unterstützten Schnittstellen notwendig. Die beiden bekanntesten Schnittstellen dieser Art sind die Microsoft Crypto-API und der PKCS#11 Standard. Eine Umsetzung erfolgt hierbei in Form einer zusätzlichen Schicht, die als Proxy zwischen dem TCG Service Provider Interface und den Applikationen fungiert. Die Abb. 6.4 beschreibt hierbei die unterschiedlichen Zugriffsarten.

Neben den beiden lokalen zugriffsarten definiert die TCG auch einen entfernten Zugriff auf das TPM mittels *Remote Procedures Calls* (RPC). Hierfür läuft ein entsprechender RPC-Server oberhalb der TSS Core Services und erlaubt somit den Zugriff über einen RPC-Client von einer entfernten Maschine. Das zu verwendende RPC-Protokoll ist von der TCG nicht vorgegeben, jedoch ist dem Autor auch keine Implementierung dieses Zugriffsverfahrens bekannt. Wenn überhaupt, wäre eine solche Schnittstelle auch nur im Firmenumfeld als Teil der Fernwartung interessant, im privaten Umfeld entstünden dadurch zu viele Probleme hinsichtlich Sicherheit und Datenschutz.

[5] [URL11]

[6] [URL12]

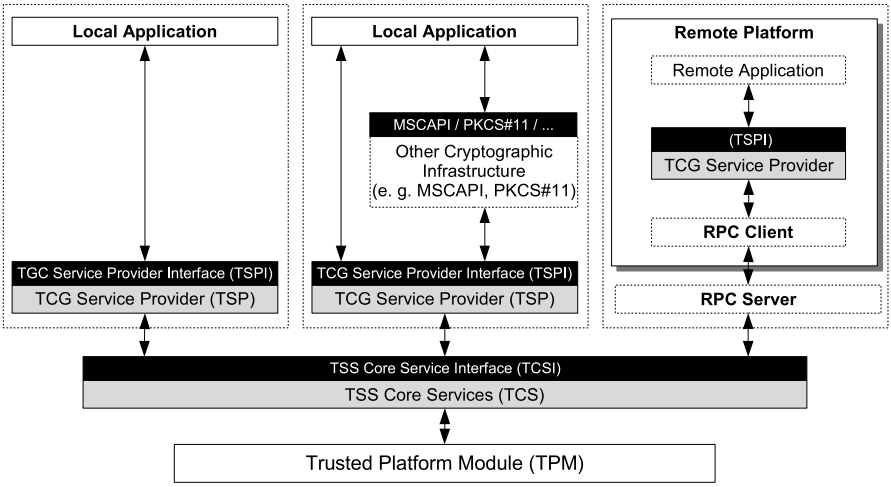

Abb. 6.4 TSS-Einsatzszenarien [TCG2006-2]

6.6 Protected Execution

Aufgabe der *Protected Execution* ist es, *die Trusted Computing Base* (TCB) eines *Trusted-Computing-Systems* und andere sicherheitskritische Komponenten vor Zugriffen durch nicht vertrauenswürdige Prozesse zu schützen. Eine Kompromittierung einer Komponente außerhalb der TCB darf sich nicht auf die Integrität des Systems auswirken. Die Architektur gängiger Desktop-Betriebssysteme sorgt hierbei für einige Schwierigkeiten. Bei der Architektur von Betriebssystemen wird grob zwischen der Umsetzung als Microkernel oder als monolithischer Kernel unterschieden. Während beim Mikrokernel nur die Grundfunktionen des Betriebssystems (wie z. B. Speicherverwaltung, Scheduling) direkt im Kern implementiert sind, integrieren die monolithischen Kernel einen Großteil der Funktionalität direkt innerhalb des Kerns. Da die Auslagerung von Funktionen aus dem privilegierten Bereich des Kernels zu häufigen Kontextwechseln führt, galten die Mikrokernel als langsam und kommen u. a. deshalb bis heute in Desktop-Betriebssystemen nicht zum Einsatz. Der Nachteil der monolithischen Kernel ist jedoch der gemeinsame Speicherbereich aller im Kernel aktiven Prozesse und die dadurch mögliche Manipulation von sicherheitskritischen Datenstrukturen des Betriebssystemkerns. Als besonders problematisch haben sich in der Vergangenheit die Gerätetreiber erwiesen. Sie stammen oft aus nicht bekannten Quellen und führen damit zur Ausführung von nicht vertrauenswürdigem Code mit den Rechten des Betriebssystems und somit dem Zugriff auf den gesamten Speicher des Systems. Exakt diese Schwäche nutzt immer mehr Schadsoftware (u. a. Kernel-Mode-Rootkit) aus, um das Betriebssystem zu kompromittieren und um sich selbst vor der Erkennung durch Virenscanner zu schützen.

Microsoft, deren Betriebssystem Windows am stärksten von dieser Art Schadsoftware betroffen ist, reagierte auf diesen Angriffsvektor mit der Überprüfung der digitalen Signatur von Gerätetreibern. Treiber ohne bzw. mit ungültiger Signatur können in der 64-Bit-Version von Windows Vista nicht mehr geladen werden. Dieser Ansatz hat jedoch gleich einige Schwächen, so eignet sich die digitale Signatur lediglich zur Identifizierung des Autors. Eine Sicherstellung der Qualität des Codes ist dadurch nicht gegeben. Darüber hinaus sind bereits Möglichkeiten zur Umgehung des Schutzes bekannt[7].

Für die Umsetzung wirkungsvoller Konzepte gibt es unterschiedliche Ansätze. Sie reichen von neuen Konzepten in der Architektur von Betriebssystemen (Microsoft Singularity[8]) über den Einsatz von Virtualisierungstechnologie in Kombination mit den bereits beschriebenen Erweiterungen in aktuellen CPUs und Chipsätzen. Mit den Lösungen auf Basis von Mikrokernen und Virtualisierungstechnologie beschäftigt sich das Kapitel „Protected Execution“ ab Seite 79. Allen Ansätzen gemein ist das Konzept der Partitionierung des Systems in Bereiche mit unterschiedlichen Schutzanforderungen. Die dabei entstehenden Bereiche werden i. d. R. als *Security Domains* oder *Security Compartments* bezeichnet und erlauben dadurch u. a. die Abgrenzung der TCB von den restlichen Teilen des Betriebssystems.

6.7 Trusted-GUI und Trusted Input/Output

Ein mit dem Ansatz der *Protected Execution* verwandtes Konzept ist das des *Trusted Graphical User Interface* (Trusted-GUI). Ziel ist es, einen vertrauenswürdigen Kommunikationspfad zwischen dem Benutzer und dem Betriebssystem aufzubauen. Die dieser Idee zugrunde liegende Problematik ist der Umstand, dass es für den Benutzer nicht möglich ist, zwischen GUI-Elementen (z. B. Buttons, Fenster, Dialoge) des Betriebssystems und denen von Applikationen zu unterscheiden. Weiter verschärft wird die Problematik durch die Funktionalität der Grafikschnittstellen der Betriebssysteme, beliebige Darstellungen auf das Display zu zeichnen, was die einfache Nachahmung von bekannten Betriebssystem-Dialogen ermöglicht. Diese Schwächen können u. a. für Phishing-Angriffe verwendet werden, um den Benutzer zur Eingabe von Anmeldeinformationen zu verleiten. Weiter erlauben die meisten Betriebssysteme die Simulation von Benutzerinteraktionen für die Automatisierung von wiederkehrenden Tätigkeiten (z. B. bei der Fernwartung oder der zentralen Softwareverteilung). Wird eine solche Schnittstelle von einer schadhaften Applikation verwendet, können u. a. sicherheitsrelevante Dialoge vor dem Benutzer verborgen werden, indem sie automatisch bestätigt werden.

Unter den Aspekten des *Trusted Computing* erzeugt dies ein weiteres Problem. Da der Benutzer den ihm angezeigten Informationen nicht vertrauen kann, ist es

[7] Siehe Kapitel „Trusted Computing mit Windows Vista“

[8] Ein Forschungsbetriebssystem, das nahezu komplett in einer typsicheren Sprache entwickelt wurde. Anstelle von gemeinsam genutztem Speicher (shared memory) tauschen Prozesse streng typisierte Nachrichten aus. [URL13]

auch nicht möglich, den Benutzer über den aktuellen Zustand des Betriebssystems zu informieren. Selbst im Falle der Bewertung der Systemintegrität auf einem externen System (*Remote Attestation*) gibt es keine Möglichkeit, ein negatives Ergebnis verlässlich auf dem bewerteten System anzuzeigen. Im Falle einer Kompromittierung der Systemintegrität muss nämlich auch mit der Kompromittierung der für die Benachrichtigung zuständigen Komponente gerechnet werden[9]. Diese Tatsache wird oft gar nicht als Problem erkannt und findet deshalb in den meisten Publikationen zu *Trusted Computing* auch keine Erwähnung.

Neben der Problematik der vertrauenswürdigen Anzeige, sprich der Ausgabe von Informationen, besteht zusätzlich noch das Problem der sicheren Eingabe von Informationen in ein Computersystem. Auch hierfür muss ein vertrauenswürdiger Pfad zwischen den Eingabegeräten (z. B. Maus und Tastatur) und der aktuell aktiven Applikation hergestellt werden.

Wie bereits eingangs erwähnt, ist die Problematik des *Trusted I/O* verwandt mit der *Protected Execution*, da auch hierfür ein vom restlichen System isolierter Bereich notwendig ist. Somit liefert erst eine Umsetzung der *Protected Execution* die notwendige Basis für das *Trusted-GUI*.

[9] Bei der Übertragung der Prüfsummen zum externen System ist eine Kompromittierung des Attestation Client jedoch aufgrund der im TPM erzeugten digitalen Signatur unkritisch, da eine Manipulation der Daten auf dem externen System erkannt werden könnte.

Kapitel 7
Trusted-Computing-Infrastruktur

Für die meisten im Rahmen des *Trusted Computing* verfolgten Ziele ist eine Interaktion zwischen unterschiedlichen Systemen an unterschiedlichen Standorten von großer Bedeutung. So kann z. B. das Konzept der *Remote Attestation*, also der vertrauenswürdigen Übermittlung des aktuellen Zustands eines Systems, nur umgesetzt werden, wenn die Interoperabilität zwischen den Systemen sichergestellt werden kann. Für diesen Bereich existiert innerhalb der *Trusted Computing Group* (TCG) eine eigene Arbeitsgruppe (WG-Infrastructure), die sich mit der Erstellung von entsprechenden Spezifikationen beschäftigt. Im Wesentlichen beziehen sich diese Spezifikationen auf die an der *Remote Attestation* beteiligten Instanzen, die verwendeten Protokolle sowie die für den Datenaustausch eingesetzten Datenstrukturen. Relevant hierbei sind die beiden Hauptdokumente:

- Reference Architecture for Interoperability (Part I) [TCG2005-2]
- Reference Architecture for Interoperability (Part II) – Integrity Management [TCG2006-6]

Daneben existiert eine ganze Reihe weiterführender Dokumente, die sich zum Großteil an Entwickler von kompatiblen Systemen richten:

- Core Integrity Schema Specification
- Credential Profiles
- Infrastructure Subject Key Attestation Evidence Extension
- Integrity Report Schema Specification

Es sei an dieser Stelle erwähnt, dass diese von der TCG erstellten Spezifikationen teilweise noch keinen stabilen Zustand erreicht haben und noch unvollständig sind. Einige der Spezifikationen wurden erst während der Entstehungsphase dieses Buches veröffentlicht. Die folgenden Abschnitte stellen eine Zusammenfassung der gelisteten Spezifikationen dar, erheben aus den genannten Gründen jedoch keinen Anspruch auf Vollständigkeit.

7.1 Public Key Infrastructure (PKI)

Im Rahmen des *Remote-Attestation*-Verfahrens kommen eine ganze Reihe unterschiedlich digitaler Zertifikate zum Einsatz. Die Verifizierung dieser Zertifikate ist das Kernelement für den Aufbau einer Vertrauenskette[1] von der Herstellung des TPM bis hin zur eigentlichen *Remote Attestation* oder *Remote Authentication* gegenüber einem Dienstanbieter. Hieraus entsteht die Notwendigkeit einer umfangreichen *Public-Key-Infrastruktur* (PKI). Die Erzeugung dieser Vertrauenskette lässt sich in drei Phasen aufteilen, die im Folgenden näher beschrieben werden:

1. Ausstellung der Plattform-Zertifikate
2. Ausstellung der AIK-Zertifikate
3. Verwendung der AIK-Zertifikate

7.1.1 Ausstellung der Plattform-Zertifikate

Während der Herstellung der Plattform müssen die auf Seite 61 beschriebenen Plattform-Zertifikate erstellt werden. Voraussetzung hierfür ist, dass sowohl der TPM-Hersteller (z. B. Infineon) als auch der Plattform-Hersteller (z. B. Lenovo) über gültige Schlüssel zur digitalen Signatur verfügen. Die zu den Schlüsseln gehörigen Zertifikate sollten von einer allgemein bekannten und als vertrauenswürdig eingestuften *Root-CA* (z. B. Verisign) ausgestellt werden. Nur so kann sichergestellt werden, dass der für eine spätere Verifizierung (z. B. bei der Ausstellung eines AIK-Zertifikats) notwendige Zertifizierungspfad verfügbar ist. Auch die für die Ausstellung des *Conformance Certificate* zuständige Instanz sollte über ein allgemein anerkanntes Zertifikat verfügen. Die Abb. 7.1 beschreibt den Vorgang der Ausstellung der Plattform-Zertifikate. Die durchgehenden Pfeile verknüpfen den Ursprung und das Ziel eines Zertifikats, die gestrichelten Pfeile kennzeichnen eine Beziehung zwischen den Zertifikaten.

Der Hersteller des TPM erstellt das EK-Zertifikat und übergibt dieses zusammen mit der TPM selbst an den Hersteller der Plattform. Dieser erzeugt das Plattform-Zertifikat und verweist hierbei auf den Inhalt des EK-Zertifikates. Die *Conformance Authority*[2] erzeugt dann auf Basis der beiden anderen Zertifikate das Conformance-Zertifikat und bescheinigt dadurch die Konformität zu den Spezifikationen der TCG. Die einzelnen Felder der Zertifikate sowie deren Beziehungen wurden bereits unter „Plattform-Zertifikate (Platform Credentials)“ auf Seite 61 beschrieben.

Die erstellten Zertifikate müssen dann an einer für den TPM-Eigentümer zugänglichen Stelle abgelegt werden. Die TCG-Spezifikationen sehen hierfür die Auslie-

[1] Diese Vertrauenskette basiert auf der Ausstellung und Verifizierung von digitalen Zertifikaten und ist nicht zu verwechseln mit der auf den erzeugten Prüfsummen basierenden Vertrauenskette (Chain of Trust).

[2] Welche Instanz die Aufgabe der Conformance Authority übernimmt, ist von der TCG nicht spezifiziert.

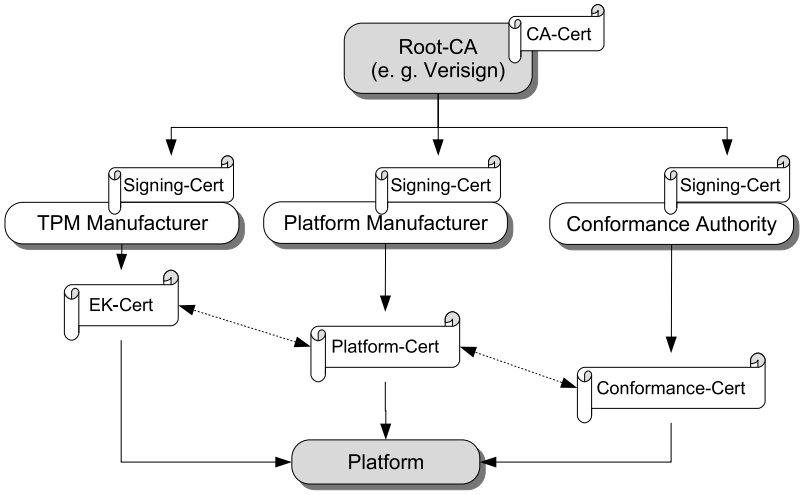

Abb. 7.1 Ausstellung der Plattform-Zertifikate

ferung auf CD, das Hinterlegen auf der Festplatte des Systems, den Download über die Seiten des Herstellers oder das direkte Hinterlegen im TPM vor. Dem Autor ist jedoch aktuell keine Plattform bekannt, für die alle drei Zertifikate erhältlich sind bzw. mit dem Rechner ausgeliefert werden.

7.1.2 Ausstellung der AIK-Zertifikate

Die nächste Phase ist die Ausstellung der zu den *Attestation Identity Keys* (AIK) gehörigen AIK-Zertifikate. Der technische Teil dieses Prozesses ist unter „Erzeugung eines Attestation-Identity-Zertifikats“ auf Seite 58 ausführlich beschrieben. Dieser Abschnitt widmet sich der dafür notwendigen PKI. Die Ausstellung eines AIK-Zertifikates basiert auf der Verifizierung der zugehörigen Plattform-Zertifikate. Die Abb. 7.2 beschreibt diesen Ablauf.

Die während der Fertigung der Plattform erstellten Zertifikate werden zusammen mit dem öffentlichen Teil des neuen AIK an eine *Privacy-CA* übertragen. Die *Privacy-CA* überprüft die Signatur der Plattform-Zertifikate, hierfür benötigt sie eine Vertrauensbeziehung zu der *Root-CA*, welche für die Ausstellung der verwendeten Signatur-Zertifikate verantwortlich zeichnet. Das Zertifikat der *Root-CA* (CA-Cert) muss also der *Privacy-CA* als vertrauenswürdige Zertifizierungsstelle bekannt sein. Zusätzlich müssen auch die Signatur-Zertifikate[3] vorliegen, um den vollständigen Zertifizierungspfad zu prüfen. Diese müssen entweder von der Plattform mitgesendet werden oder bei einem Zertifikatsserver erfragt werden. Die Adresse des Zertifikatsservers muss hierfür entweder bekannt sein oder von der Plattform über-

[3] Siehe Abb. 7.1: Ausstellung der Plattform-Zertifikate

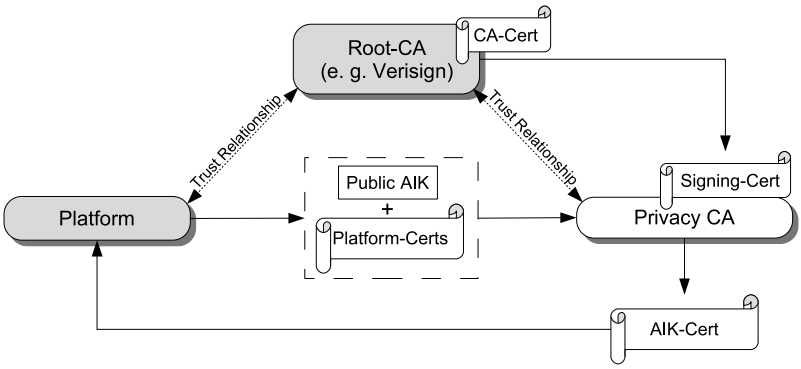

Abb. 7.2 Signatur eines AIK

mittelt werden. War die Überprüfung des Zertifizierungspfads erfolgreich, erstellt die *Privacy-CA* das AIK-Zertifikat zum erhaltenen AIK. Um auch dieses Zertifikat später verifizieren zu können, sollte das hierfür verwendete Signatur-Zertifikat ebenfalls von einer allgemein anerkannten *Root-CA* stammen.

7.1.3 Verwendung der AIK-Zertifikate

Die Plattform kann dieses AIK-Zertifikat nun verwenden, um die erzeugten Prüfsummen an eine prüfende Instanz zu übertragen, ohne dabei die eigene Identität bekannt geben zu müssen. Der technische Ablauf dieses als *Remote Attestation* bezeichneten Verfahrens wurde unter „Remote Attestation“ und „Remote Attestation (Remote Integrity Validation)“ bereits beschrieben. Die nachfolgende Abbildung zeigt die für die *Remote Attestation* notwendige PKI:

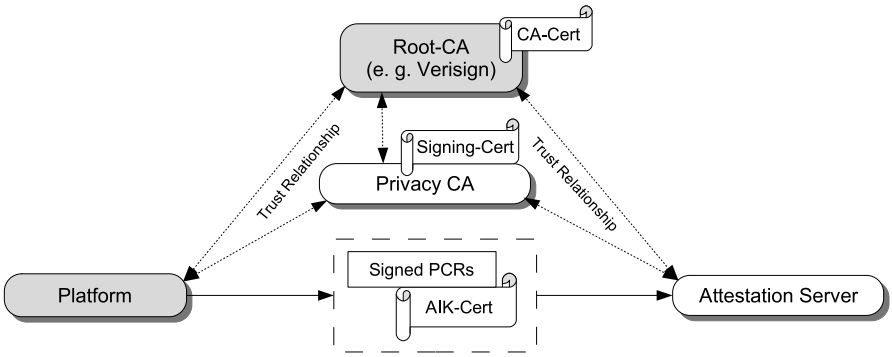

Abb. 7.3 PKI Remote Attestation

Das TPM der Plattform signiert alle oder eine Untermenge der in den PCR enthaltenen Werte und sendet sie zusammen mit dem von der *Privacy-CA* erhaltenen AIK-Zertifikat an den *Attestation Server*. Hierbei kann es sich sowohl um ein Zugangskontrollsystem im Intranet eines Unternehmens als auch um einen beliebigen Dienstanbieter im Internet handeln. Auf dem Server wird nun die Signatur mittels Auswertung des Zertifizierungspfads überprüft. Hierfür benötigt der *Attestation Server* eine Vertrauensbeziehung (*Trust Relationship*) zur *Root-CA,* die für die Ausstellung des Signatur-Zertifikats der *Privacy-CA* verantwortlich ist. Dieses Signatur-Zertifikat muss, analog zur Ausstellung des AIK-Zertifikats, von der Plattform mitgesendet oder von einem Zertifikatsserver abgefragt werden.

Eine weitere Möglichkeit für den Einsatz des AIK bzw. des AIK-Zertifikats ist die Anforderung weiterer Zertifikate bei einer regulären CA (*Classic-CA*). Ein Beispiel hierfür ist die Anforderung eines Zertifikats für die Mail-Signatur oder für den Aufbau einer SSL/TLS-Verbindung. Das AIK-Zertifikat dient hierbei der Signatur des *Certificate Request*[4]. Die *Classic-CA* wiederum kann den im AIK-Zertifikat enthaltenen öffentlichen Teil des AIK dazu verwenden, das neu erstellte Zertifikat verschlüsselt an die Plattform zurückzuübermitteln. Hierdurch kann sichergestellt werden, dass das Zertifikat nur auf der Plattform mit dem entsprechenden TPM entschlüsselt werden kann. Die nachfolgende Abbildung beschreibt diesen Vorgang:

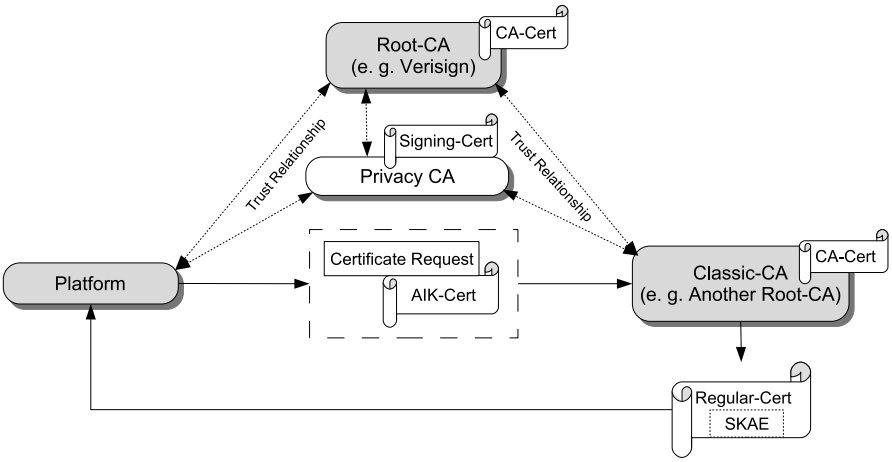

Abb. 7.4 PKI-Subject Key Attestation Evidence Extension

Die hierbei als *Classic-CA* bezeichnete *Certificate Authority* kann zusätzlich zu dem normalen Validierungsverfahren auch das AIK-Zertifikat auswerten. Hierfür benötigt diese CA ebenfalls eine Vertrauensbeziehung zur *Root-CA* der *Privacy-CA*. Auch das Signing-Zertifikat der *Privacy-CA* muss hierfür vorliegen. Um diese zusätzliche Überprüfung im neu erstellten Zertifikat abzubilden, spezifiziert die TCG die *Subject Key Attestation Evidence Extension* (SKAE), deren Format

[4] Das Format eines X.509 Certificate Request beschreibt [URL14].

in [TCG2005-3] definiert ist. Diese Zertifikatserweiterung kann später zusätzlich zu den üblichen Feldern eines Zertifikats ausgewertet werden und als Grundlage für weitere Entscheidungen dienen.

7.1.4 Zusammenfassung

Durch die Zusammenlegung der in den einzelnen Phasen verwendeten PKIs entsteht ein komplexes Netz von Vertrauensbeziehungen. Die Komplexität wird zusätzlich noch dadurch erhöht, dass es möglich ist, mehrere unterschiedliche *Root-CAs* und auch mehrere *Privacy-CAs* zu verwenden. Aufgrund der Machtposition einer einzelnen Instanz ist dies sogar das wahrscheinlichste Szenario. Die folgende Abbildung ist ein Versuch, die Komplexität einer solchen PKI zu veranschaulichen:

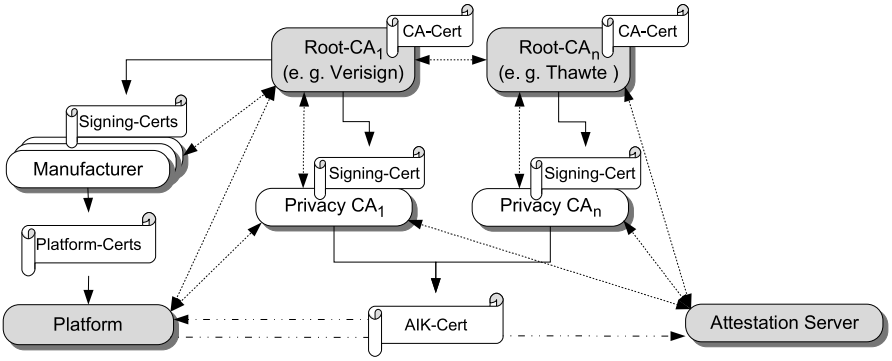

Abb. 7.5 Trusted Computing PKI

Aufgrund dieser Komplexität und dem dadurch entstehenden Abstimmungs- und Verwaltungsaufwand bleibt es abzuwarten, ob und wann eine entsprechende Infrastruktur zur Verfügung stehen wird. Aktuell wird die einzige *Privacy-CA* für AIK-Zertifikate im Rahmen des OpenTC-Projektes von der TU Graz betrieben. Sie dient jedoch in erster Linie Testzwecken und akzeptiert auch selbstsignierte Plattform-Zertifikate. Dieser Umstand liegt jedoch vor allem in der Nichtverfügbarkeit von Zertifikaten der Hersteller begründet. Eine Ausnahme hierbei ist der TPM-Hersteller Infineon, der das EK-Zertifikat bei aktuellen V1.2 TPMs direkt im TPM hinterlegt und den notwendigen Zertifizierungspfad auf den Internetseiten zum Download anbietet.

7.2 Certificate-Management-Protokoll

Weiter definieren die TCG-Spezifikationen die bei der Ausstellung der Zertifikate verwendeten Protokolle. Hierbei sind zwei mögliche Verfahren spezifiziert, das *Certificate Management Messages over CMS* (CMC) und die *XML Key Management Services* (XKMS). Bei CMC handelt es sich um eine Erweiterung des *PKCS#10 Certificate Request Syntax*[5] und des *PKCS#7 Cryptographic Message Syntax*[6] um die Funktionalitäten Verschlüsselung und Signatur von ausgetauschten Nachrichten. Bei den XKMS[7] handelt es sich um ein Management-Protokoll auf Basis von XML. XKMS verwendet den SOAP-Standard als Übertragungsprotokoll und bietet die Funktionen Ausstellung, Registrierung und Sperrung von Zertifikaten. Der Favorit der TCG ist, aufgrund des größeren Funktionsumfangs und der einfachen Erweiterung, das XKMS.

7.3 Remote-Attestation-Protokoll

Bei der Ausstellung von AIK-Zertifikaten durch die *Privacy-CA* sowie bei der darauf folgenden *Remote Attestation* müssen Informationen über den Zustand des *Trusted-Computing-Systems* über ein Netz übertragen werden. Auch für diese Kommunikation, die zwischen der Plattform und der *Privacy-CA* bzw. zwischen der Plattform und dem *Attestation Server* stattfindet, definiert die TCG eine spezielle Protokollerweiterung namens *TLS-Attestation*. *TLS-Attestation* erweitert den *Transport-Layer-Security* (TLS)-Standard um die Übermittlung von Integritäts-Reports. Die Erweiterung ist im Dokument „*TLS Extensions for Attestation*“ der TCG spezifiziert, welches jedoch bisher nicht veröffentlicht wurde deshalb nicht näher beschrieben werden kann. Unter „WS-Attestation“ wird auf Seite 105 eine Alternative zu diesem Protokoll auf Basis von Web Services beschrieben. Die bei *TLS-Attestation* ausgetauschte Datenstruktur (*Integrity-Report-Schema*) ist bereits bekannt und in [TCG2006-7] spezifiziert.

---

5 [URL15]

6 [URL16]

7 [URL17]

Kapitel 8
Theoretische und praktische Lösungsansätze

Dieses Kapitel beschreibt konkrete Ansätze zum Erreichen der vorgestellten Ziele des *Trusted Computing*. Hierbei werden sowohl Lösungsansätze theoretischer Natur als auch solche mit bereits verfügbaren Prototypen berücksichtigt. Wo möglich, werden die einzelnen Lösungsansätze zu einem Ziel in chronologischer Reihenfolge erläutert. Dies ermöglicht einen Einblick in die Entstehung der Ansätze und liefert die technischen Grundlagen für die in Kapitel 9 vorgestellten Lösungen.

8.1 Integrity Measurement und Integrity Protection

8.1.1 AEGIS

Bereits vor der breiten Verfügbarkeit von *Trusted-Platform-Modulen* und der Definition des *Integrity-Measurement*-Ansatzes und der daraus resultierenden *Vertrauenskette* gab es Bestrebungen, die Integrität des Startvorgangs abzusichern. Der erste dem Autor bekannte Ansatz ist die AEGIS-Architektur [Arbaugh1997, Arbaugh1997-2] aus dem Jahre 1997. Das Konzept basiert auf der Erstellung von Prüfsummen über alle am Startvorgang beteiligten Komponenten und deren Bewertung anhand von bekannten Referenzwerten.

Bereits 1997 wurde erkannt, dass der *Vertrauensanker* einer solchen Architektur als Teil einer Hardware- bzw. Firmware-Komponente implementiert sein sollte. Das Konzept sieht daher eine Erweiterung der Funktionen des PC-BIOS vor. Diese Erweiterung ist eng verwandt mit der CRTM der TCG und diente vermutlich als dessen Designvorlage. Mit Hilfe dieser Erweiterung werden die Prüfsummen über das BIOS sowie für evtl. vorhandene Option-ROMs erzeugt. Danach werden, bis zum vollständigen Start des Betriebssystems, weitere Prüfsummen über die Komponente erzeugt und mit Referenzwerten verglichen.

Das System setzt eine Speicher-Erweiterungskarte (PROM-Board) ein, um den beschränkten Speicher des BIOS-ROM zu vergrößern. Hier werden neben den In-

struktionen zur Bewertung der Prüfsummen auch die zugehörigen Referenzwerte hinterlegt. Eine interessante Zusatzfunktion ist in Form eines Recovery-Kernels ebenfalls auf dieser Erweiterungskarte hinterlegt. Im Falle einer negativen Überprüfung startet nicht das installierte Betriebssystem, sondern ein minimales Rettungssystem auf Basis des Recovery-Kernels. Ein in diesem Rettungssystem enthaltener Netzwerkdienst soll die als nicht integer identifizierten Komponenten durch die ursprünglichen Versionen ersetzen und das System somit wieder in einen integeren Zustand überführen. Die folgende Abbildung zeigt die vereinfachte Architektur des Systems:

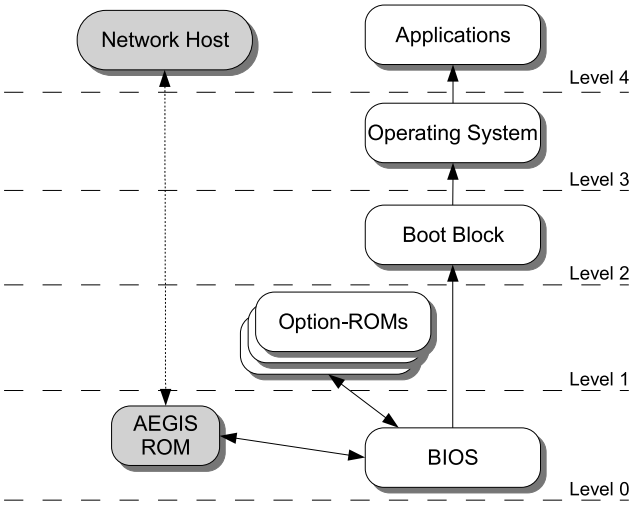

Abb. 8.1 AEGIS-Architektur

Die Idee wurde an der University of Pennsylvania entwickelt und baut auf den beiden Arbeiten „A Smartcard Protected Operating System“ [Clark1994] und „Using Secure Coprocessors“ [Yee1994] aus dem Jahre 1994 auf. [Yee1994] definierte erstmals das Konzept eines sicheren Startvorgangs (*Secure Boot* bzw. *Trusted Boot*).

8.1.2 SEBOS

Der *Security Enhanced Bootloader for Operating Systems*[1] (SEBOS) ist die direkte Weiterentwicklung der AEGIS-Architektur und wird an der University of Maryland entwickelt. Ziel dieser Arbeit ist eine Integration in das Open-Source-Projekt LinuxBIOS. Nach Informationen auf den Internetseiten des Projekts unterstützt diese

[1] [URL18]

Entwicklung auch die Verwendung von *SmartCards* und dem IBM 4758 Coprozessor. Eine Verwendung des TPM ist jedoch nicht vorgesehen.

8.1.3 Copilot

Das Projekt Copilot „*A Coprocessor-based Kernel Runtime Integrity Monitor*“ [Petroni2004] ist ebenfalls mit der AEGIS-Architektur verwandt und hat seinen Ursprung an der University of Maryland. Ziel hierbei ist es, die Integrität des Kernel bzw. dessen Datenstrukturen im Arbeitsspeicher zu überwachen. Die Überwachung von Datenstrukturen des Betriebssystemkerns ist ein häufig verwendeter Ansatz zum Schutz der Integrität eines Betriebssystems. Der wohl populärste Vertreter dieses Schutzkonzeptes ist Microsofts *PatchGuard*[2]. Allen Systemen gemein ist jedoch der Nachteil, selbst Teil des Kernels zu sein und somit ebenfalls von dessen Integrität abhängig. Dies ist ein weiteres Beispiel für die bereits beschriebene Schwäche gängiger Systeme, ihre eigene Integrität nicht selbst bewerten zu können. Ist der Kernel kompromittiert, ist es in der Regel auch recht einfach, diesen Kernelschutz zu deaktivieren, welcher genau diesen Angriff hätte erkennen sollen. [Skape2006] beschreibt beispielsweise die Umgehung von *PatchGuard* auf Basis dieser Limitierung.

Um diesen Angriffsvektor zu vermeiden, setzt *Copilot* auf das *System-in-System*-Konzept. Hierbei befindet sich ein zweites vollkommen autonomes System innerhalb der Plattform. Das zweite System läuft dazu idealerweise auf eigener Hardware. Das *Copilot*-Projekt verwendet hierfür einen so genannten *Single-Board-Computer* (Intel StrongARM EBSA-285) in Form einer PCI-Erweiterungskarte. Unter Verwendung des *Direct Memory Access* (DMA) des PCI-Bus ist es so möglich, auf den physikalischen Arbeitsspeicher des Systems zuzugreifen. Um das *System-im-System* selbst zu schützen, kann es so konfiguriert werden, dass es sämtliche Lese- und Schreibzugriffe durch den Prozessor des zu schützenden Systems verweigert und somit über diesen Kanal nicht angegriffen werden kann.

Durch den Zugriff auf den physikalischen Arbeitsspeicher des Systems ist es möglich, die sicherheitskritischen Datenstrukturen des Betriebssystemkerns zu überwachen. Ein Beispiel einer solchen Datenstruktur ist die Adresstabelle der Systemfunktionen (z. B. CreateFile). Eine Modifikation dieser Tabelle ist häufig ein Indiz für die Präsenz eines Kernel-Rootkit. Für eine erfolgreiche Überwachung muss die Semantik der überwachten Daten bekannt sein, *Copilot* muss deshalb an das jeweilige Betriebssystem bzw. dessen Kernel angepasst werden. Gleiches gilt für alle Applikationen, die durch Copilot überwacht werden sollen.

Die Steuerung der Erweiterungskarte und somit des Überwachungssystems erfolgt auf einem zweiten Rechner, der direkt mit der im ersten Rechner verbauten Erweiterungskarte verbunden ist. Die nachfolgende Abbildung beschreibt diesen Aufbau:

[2] Microsofts PatchGuard wird im Kapitel „Trusted Computing mit Windows Vista“ beschrieben.

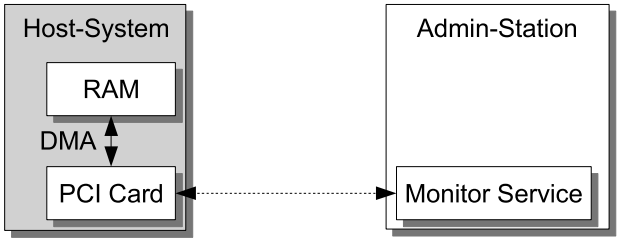

Abb. 8.2 Copilot-Architektur

Aufgrund des hohen Aufwands für die Anpassungen sowie der Notwendigkeit eines weiteren Rechners ist ein solcher Ansatz für die Praxis in den meisten Fällen jedoch unbrauchbar.

8.1.4 Trusted Grub

Die bekannteste und zugleich am meisten fortgeschrittene Umsetzung des *Integrity-Measurement*-Konzeptes, also der Fortsetzung der *Vertrauenskette* von der TCP bis zum vollständigen Start des Betriebssystems, ist eine Weiterentwicklung des *Grand Unified Bootloader* (Grub). Der als *Trusted-Grub*[3] bezeichnete Ansatz ergänzt den Grub Bootmanager um die Schnittstelle zum TPM und erlaubt dadurch den Zugriff auf dessen Funktionalität.

Der Startvorgang mittels Grub ist unterteilt in mehrere Abschnitte: Stage 1, Stage 1.5 und Stage 2. Stage 1 entspricht hierbei dem Code im *Master Boot Record* (MBR) und enthält somit die letzten von der TCP protokollierten Instruktionen. Stage 1.5 enthält die Implementierung des Dateisystems und bildet dadurch die Grundlage für das Laden der Betriebssystemkomponenten. Stage 2 schließlich enthält die rechtlichen Instruktionen des Bootmanagers sowie die zugehörige Konfiguration (grub.conf).

Für die Weiterführung der *Vertrauenskette* müssen zunächst diese Abschnitte des Bootmanagers durchlaufen werden, bevor eine Messung des Betriebssystems stattfinden kann. Hierfür müssen die Instruktionen aus Stage 1 die nächste Instanz, Stage 1.5, protokollieren und im TPM hinterlegen. Danach kommt Stage 1.5 zur Ausführung und erstellt nach dem gleichen Schema eine Prüfsumme über die Instruktionen von Stage 2. Die Abb. 8.3 beschreibt diesen Vorgang.

Stage 2 führt eine ganze Reihe von Messungen durch. Zunächst wird eine Prüfsumme über die Konfigurationsdatei (grub.conf) erzeugt, diese Datei enthält einen Verweis auf das zu ladende OS-Image. Über dieses Image wird, inklusive seiner Abhängigkeiten, direkt im Anschluss eine Prüfsumme erzeugt. *Trusted-Grub* definiert zusätzlich ein neues Schlüsselwort in der Konfigurationsdatei (measure) mit Hilfe dessen weitere Dateien zur *Vertrauenskette* hinzugefügt werden können.

[3] [URL19]

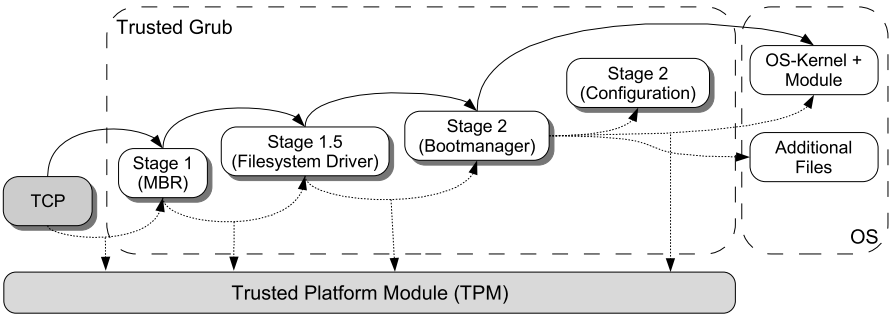

Abb. 8.3 Trusted Grub

Diese Dateien erhalten ebenfalls eine Prüfsumme und bilden den Abschluss der durch den *Trusted-Grub*-Bootmanager erstellten *Vertrauenskette*. Die standardmäßige Verwendung der PCR des TPM beschreibt die folgende Tabelle:

Tabelle 8.1 Trusted Grub – PCR-Verwendung

PCR	Verwendung
PCR[4]	Stage 1 (Master Boot Record)
PCR[8]	Stage 1.5
PCR[9]	Stage 2
PCR[12]	Kernel-Konfigurationsparameter aus menu.1st und vom Bootprompt
PCR[13]	Summe der mittels „measure“ hinzugefügten Dateien
PCR[14]	Die Dateien des OS (z. B. Linux Kernel, initrd, Module...)

Trusted-Grub schließt somit die Lücke zwischen der *Trusted Computing Platform* und einem *Trusted-OS*. Durch das *measure*-Schlüsselwort ist der Trusted-Grub-Bootmanager auch in der Lage, beliebige zusätzliche Komponenten zu protokollieren. Dies ersetzt jedoch nicht die Erweiterung der *Vertrauenskette* zur Laufzeit durch ein *Trusted-OS*. Eine vollständige Beschreibung des Bootmanagers liefert [Maruyama2003].

8.1.5 IBM Integrity Measurement Architecture (IMA)

Die *Integrity Measurement Architecture*[4] ist eine von IBM Research entwickelte Erweiterung für den Linux Kernel. Ziel hierbei ist es, den *Integrity-Measurement*-Prozess auf die Laufzeit des Betriebssystems auszuweiten. Die IMA verwendet den *Trusted-Grub*-Bootmanager, um dem Linux Kernel sowie dessen Abhängigkeiten zu laden. Zusätzlich erzeugt der *Trusted-Grub* auch eine Prüfsumme über die zur

[4] [URL20]

IMA gehörigen Komponenten. Dadurch ist die IMA selbst ein Glied in der *Vertrauenskette* und somit in der Lage, diese zur Laufzeit zu erweitern.

Das in [Sailer2004] beschriebene Ziel der IMA ist es, ein Clark-Wilson-Integritätsmodel für Linux zu implementieren. Hierfür müssen die folgenden Daten auf dem System zur Laufzeit und vor ihrer Ausführung protokolliert werden:

- Ausführbare Instruktionen: Applikationen, Bibliotheken, Kernel-Module und sonstiger Code, der ausgeführt wird, muss protokolliert werden, unabhängig davon, ob er vom Betriebssystem selbst oder von bereits laufenden Applikationen geladen wird.
- Strukturierte Daten: Daten, die zur Steuerung und Konfiguration einer Applikationen dienen werden, haben ebenfalls Einfluss auf die Integrität der Applikation und müssen deshalb ebenfalls protokolliert werden.
- Unstrukturierte Daten: Sämtliche von den Applikationen verarbeiteten Daten, die aus nicht vertrauenswürdigen Quellen stammen, müssen protokolliert werden.

Aufgrund der unter „Anforderungen an vertrauenswürdige Betriebssysteme“ bereits beschriebenen Komplexität der Abbildung des Systemzustands zur Laufzeit beschränkt sich die IMA auf die Erzeugung von Prüfsummen über die ausführbaren Instruktionen und strukturierten Daten. Hierfür wurde der Linux Kernel in Form eines *Linux Security Modules* (LSM) um einen Systemaufruf zur Erzeugung von Prüfsummen erweitert. Diese Funktion wird immer dann aufgerufen, wenn ausführbare Instruktionen vom Kernel oder von Applikation geladen werden. Die folgenden Messpunkte (*Measurement Agents*) sind hierbei bereits implementiert:

- Beim Laden von Kernel-Modulen durch eine Anpassung der insmod- bzw. modprobe-Applikation zur Protokollierung entsprechender Events.
- Beim Start von User-level-Anwendungen durch eine Anpassung des execve-Systemaufrufs. Im Falle einer statisch gelinkten Applikation wird hierbei lediglich die Datei der Applikation protokolliert.
- Beim Start einer dynamisch gelinkten Applikation werden zusätzlich die notwendigen Bibliotheken protokolliert. Hierfür wurden die entsprechenden Funktionen der Linux-Laufzeitumgebung erweitert.

Die erzeugten Prüfsummen werden nicht direkt im TPM hinterlegt, sondern in einem Messprotokoll gespeichert. Somit ist es möglich, jede Komponente mit einer eigenen Prüfsumme zu verknüpfen und die Beschränkung auf die 23 *Platform Configuration Register* des TPM zu umgehen. Ebenso ist es dadurch möglich, zusätzliche Informationen zur Prüfsumme wie den Namen der zugehörigen Datei oder einen Zeitstempel zu hinterlegen. Um die Integrität des Messprotokolls bei der Übermittlung an ein externes System zu gewährleisten, wird eine Prüfsumme über das Protokoll selbst einem der PCR des TPM hinterlegt. Eine Kombination der Prüfsummen aus dem Messprotokoll muss somit identisch zu dem im PCR enthaltenen Wert sein, andernfalls wurde das Messprotokoll modifiziert und das System befindet sich in einem nicht integeren Zustand. Die folgende Abbildung beschreibt den Aufbau der IMA:

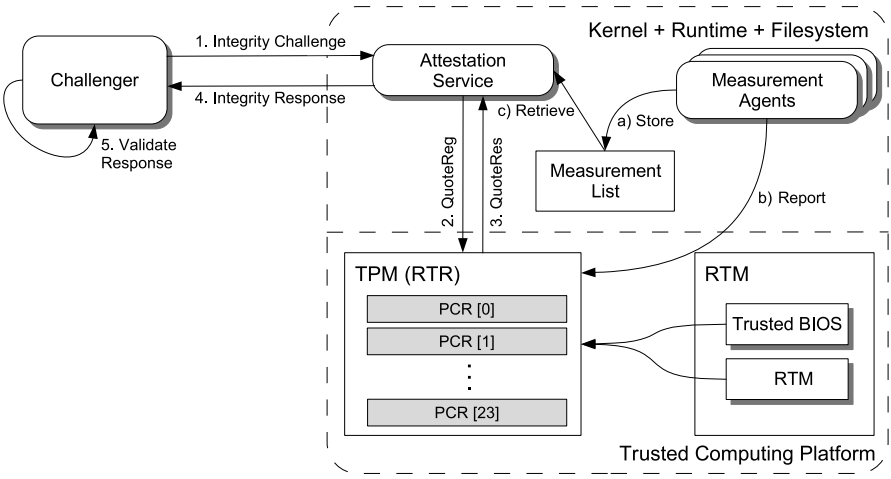

Abb. 8.4 Integrity Measurement Architecture [Sailer2004]

Das Messprotokoll (*Measurement List*) ist in Form einer verknüpften Liste im Arbeitsspeicherbereich des Kernel implementiert. Zusätzlich zur Prüfsumme enthält jeder Eintrag ein zusätzliches Flag (DIRTY/CLEAN). CLEAN bedeutet hierbei, die Datei wurde seit der Protokollierung nicht mehr mit Schreibrechten geöffnet, DIRTY bedeutet, ein schreibender Zugriff lag vor und die Datei wurde evtl. modifiziert. Um einen möglichen Zugriff mit Schreibrechten zu erkennen, erweiterte die IMA zusätzlich den Systemaufruf *open* und *unmount*. Das DIRTY-Flag wird auch gesetzt, wenn sich die Datei auf einem Netzwerklaufwerk befindet oder das zugehörige Dateisystem ausgehängt (*unmount*) wurde. Wurde eine Prüfsumme durch den beschriebenen Systemaufruf erzeugt, wird das Messprotokoll zunächst nach einem bereits vorhandenen Eintrag zur Datei durchsucht. Ist ein solcher bereits vorhanden und das DIRTY-Flag steht auf CLEAN, wird keine weitere Messung durchgeführt. Steht das Flag auf DIRTY, wird erneut ein SHA1-Hash der Datei erzeugt und mit dem vorhandenen Eintrag verglichen. Ist der Hash-Wert identisch, wird das Flag auf CLEAN gesetzt, andernfalls wird ein neuer Eintrag in der Liste erzeugt. Ist noch kein Eintrag vorhanden, wird ebenfalls ein SHA1-Hash erzeugt und an das Ende der Liste angefügt. Wurde ein neuer Eintrag erstellt, wird gleichzeitig das entsprechende PCR mit der erzeugten Prüfsumme erneuert. Ein kommentiertes Beispiel eines solchen Messprotokolls ist auf den Projektseiten[5] zu finden.

Zusätzlich zur Erweiterung der *Vertrauenskette* implementiert die IMA eine Schnittstelle zur Abfrage des erzeugten Messprotokolls zur Umsetzung des *Remote-Attestation*-Konzeptes. Die als *Attestation Service* bezeichnete Komponente überträgt hierbei das durch die *Measurement Agents* erstellte Messprotokoll zusammen mit den durch das TPM signierten Werten aus den PCR an ein externes System. Eine vorausgehende Authentisierung des externen Systems ist hierzu notwendig. Das

[5] [URL20]

bei der Kommunikation verwendete Datenformat ist XML. Beispiele für die ausgetauschten Nachrichten sowie das XML-Schema sind ebenfalls auf den Projektseiten zu finden. Der für die IMA notwendige Kernel-Patch kann auf Sourceforge[6] heruntergeladen werden. Weitere Informationen zur Implementierung liefert [Munetoh2004]. Ein interessantes Konzept zur Weiterentwicklung der IMA durch die Umsetzung eines neuen Integritätsmodels (*CW-Lite-Integrity*) beschreibt [Sailer2006].

8.1.6 BIND – Binding Instructions and Data

Ein bisher rein theoretischer Ansatz zur Erzeugung von Prüfsummen zur Laufzeit ist BIND, welches in einer Kooperation von IBM Research und der Carnegie Mellon University entstanden ist. BIND [Shi2005] adressiert die bereits mehrfach erwähnte Problematik der Skalierbarkeit von zur Laufzeit erzeugten Prüfsummen. Im Gegensatz zu Ansätzen wie z. B. der IMA setzt BIND auf die Erstellung von feinkörnigen (fine-grained) Prüfsummen. Hierbei werden nicht wie bei der grobkörnigen (coarse-grained) Erzeugung alle auf dem System ausgeführten Applikationen betrachtet, sondern nur die Komponenten, deren Integrität für eine externe Instanz relevant ist. Ein weiterer Kritikpunkt der Entwickler von BIND an Ansätzen wie der IMA ist die zeitliche Differenz zwischen der Erzeugung der Prüfsummen und deren Auswertung. So sind die beim Start einer Applikation erzeugten Prüfsummen über die gesamte Laufzeit der Applikation gültig, eine Manipulation der Applikationslogik im Arbeitsspeicher kann daher nicht erkannt werden. Deshalb sieht BIND eine Messung direkt vor und bei jeder Ausführung der Instruktionen.

BIND verwendet für die Umsetzung des Konzeptes eine Erweiterung der für die Entwicklung der betreffenden Applikationen eingesetzten Programmiersprachen. Hierdurch ist es möglich, die für die Verarbeitung kritischen Codeblöcke einer Applikation im Quelltext zu markieren (Annotations). Die markierten Codeblöcke werden dann bei jedem Durchlauf von einem *Attestation Service* protokolliert und die hierbei erzeugte Prüfsumme wird mit den durch den Code erzeugten Daten verknüpft. Somit kann nachgewiesen werden, welcher Code für die Erzeugung der Daten verwendet wurde, und ob dieser sich in einem integeren Zustand befindet. Zum Schutz des Codeblocks vor anderen Prozessen auf dem System wird er nach dem Erreichen der Startmarkierung (ATTESTATIOM_INIT) in eine Sandbox[7] verschoben und dort ausgeführt. Wurde die Endmarkierung (ATTESTATION_COMPLETE) erreicht, werden die nachfolgenden Instruktionen wieder innerhalb der allgemeinen Ausführungsumgebung ausgeführt. Dadurch ist es laut [Shi2005] möglich, vertrauenswürdigen Code selbst auf einem nicht vertrauenswürdigen System auszuführen.

Diese Aussage ist jedoch nur unter bestimmten Umständen gültig und die von BIND an das hierfür notwendige *Trusted Computing System* gestellten Anforderungen sind äußerst komplex und zum aktuellen Zeitpunkt nur auf sehr wenigen

[6] [URL21]

[7] Der Begriff Sandbox bezeichnet in diesem Kontext die Ausführung eines Prozesses in einem vom restlichen System abgeschirmten Bereich.

Systemen vollständig verfügbar. Elementar wichtig für die Umsetzung dieses Konzepts ist die Integrität der für die Erzeugung der Prüfsummen verantwortlichen Komponente, des *Attestation Service*, sowie die Wirksamkeit des angesprochenen Sandbox-Mechanismus. Hierfür nimmt BIND die folgenden Anforderungen als gegeben an:

- Hardwareseitige Unterstützung (i. d. R. CPU und Chipsatz) für das Bereitstellen einer gesicherten Ausführungsumgebung (Sandbox). Aktuelle Prozessoren von Intel und AMD erreichen dies u. a. durch die Abschirmung von Arbeitsspeicherbereichen. Mehr Informationen zu Intels TXT und AMD Presidio sind ab Seite 65 zu finden.
- Einen *Secure Kernel* (SK). Unter einem *Secure Kernel* ist im Kontext der Technologien von Intel und AMD eine Softwarekomponente zu verstehen, die als einzige Instanz Zugriff auf die erweiterten Funktionen der CPU erhält und selbst in einem abgeschirmten Bereich auf dem System ausgeführt wird. Dadurch ist der SK unabhängig vom Betriebssystem und dessen Integrität. Eine mögliche Form des SK ist etwa eine Virtualisierungsschicht (Hypervisor) unterhalb des eigentlichen Betriebssystems.
- Ein performantes TPM zur Erzeugung der Prüfsummen und deren digitale Signatur.

Da der *Attestation Service* und somit die Erzeugung der Prüfsummen innerhalb des *Secure Kernel* ausgeführt werden muss, ist BIND auf die Integrität dieser Komponente angewiesen. Hierfür verweist auch [Shi2005] auf die Erstellung von Prüfsummen während des Startvorgangs des Systems (*Secure Boot*) und beim Laden des *Secure Kernel*. BIND ist somit als Aufsatz auf die IMA bzw. auf *Trusted-Grub* zu verstehen und nicht als vollständiger Ersatz dieser Konzepte. Die Abb. 8.5 verdeutlicht den konzeptuellen Ablauf der Prüfsummenerzeugung entsprechend den folgenden Schritten:

1. Beim Erreichen der Anfangsmarkierung (ATTESTATION_INIT) löst der zu prüfende Code einen so genannten *Secure Kernel Intercept* (SKI) aus, welcher von der CPU interpretiert wird.
2. Die CPU übergibt daraufhin das Ausführungsrecht an den zuvor registrierten *Secure Kernel*.
3. Hier erzeugt der *Attestation Service* mit Hilfe des TPM eine Prüfsumme über den markierten Codeblock und dessen Eingabeparameter. Diese Prüfsumme wird in einem der PCR des TPM hinterlegt.
4. Der *Secure Kernel* erzeugt mit Hilfe der Sicherheitsfunktionen der CPU eine abgeschirmte Ausführungsumgebung (*Protected Execution*) und übergibt das Ausführungsrecht zurück an den Applikationscode.
5. Nach Erreichen der Endmarkierung (ATTESTATION_COMPLETE) wird das Ausführungsrecht wieder zurück an den *Secure Kernel* übergeben.
6. Der *Attestation Service* erstellt mit Hilfe des TPM eine digitale Signatur über die zuvor erstellte Prüfsumme und über die durch den Code generierten Daten. Diese Signatur wird den erzeugten Daten angehängt und kann für die *Remote Attestation* verwendet werden.

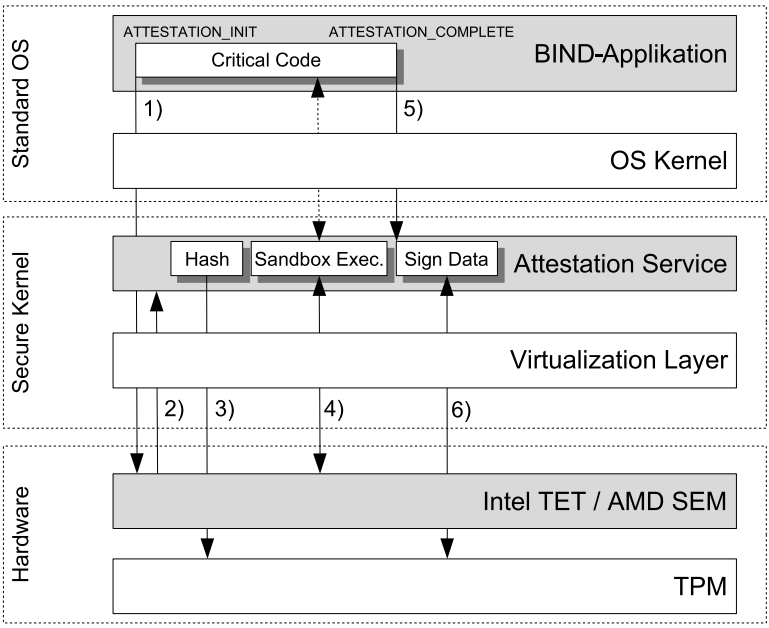

Abb. 8.5 Prüfsummenerzeugung mit BIND

Eine Implementierung von BIND war zum Zeitpunkt der Veröffentlichung von [Shi2005] aufgrund der Nichtverfügbarkeit entsprechender CPU-Erweiterungen nicht möglich. Mit der breiten Verfügbarkeit solcher Technologien ist in Laufe des Jahres 2008 zu rechnen. Es bleibt somit abzuwarten, ob das mit BIND beschriebene Konzept in naher Zukunft umsetzbar ist.

8.2 Remote Attestation

8.2.1 Trusted Network Connect (TNC)

Eine mögliche Umsetzung der *Remote Attestation* wurde unter dem Namen *Trusted Network Connect* (TNC) innerhalb einer weiteren Arbeitsgruppe der TCG (WG-TNC) spezifiziert [TCG2006]. Erklärtes Ziel hierbei ist die Schaffung eines offenen Standards für die Authentisierung von Netzwerkendpunkten sowie die Erzeugung und Übermittlung von Integritätsmesswerten. Die TCG beschreibt hierbei nur die Architektur und definiert die notwendigen Schnittstellen zwischen den Komponenten. Die Abb. 8.6 liefert eine vereinfachte Darstellung der definierten Architektur.

TNC ist zunächst eine Erweiterung der für IEEE 802.1x bereits spezifizierten Architektur zur Authentisierung und Autorisierung von Netzwerkgeräten. Die TCG sieht hierbei größtenteils die Verwendung bereits bekannter und weitläufig unter-

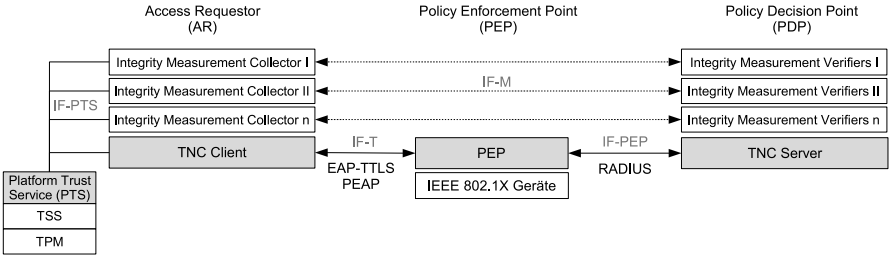

Abb. 8.6 Trusted Network Connect [TCG2006]

stützter Protokolle (z. B. EAP, RADIUS) vor. Neu hierbei ist, dass neben einem Authentisierungstoken (z. B. X.509-Zertifikat) auch noch Informationen über den Zustand eines Systems übertragen werden. Eine Zugriffsentscheidung kann somit auf Basis der Identität und des Zustands des Systems getroffen werden. Die Architektur teilt sich hierbei in drei Hauptaufgaben:

- **Access Requestor (AR)**
  Das System, das Zugriff auf ein Netzwerk erhalten möchte, wird um eine Softwarekomponente, den *TNC Client,* erweitert. Aufgabe des TNC Client ist es, den aktuellen Zustand des Systems zu ermitteln und an den TNC Server zu übertragen. Er selbst führt jedoch keine Messungen durch, sondern aggregiert die von den *Integrity Measurement Collectors* (IMC) ermittelten Informationen. Diese IMCs können von unterschiedlichen Herstellern stammen und unterschiedliche Teile des Systems beurteilen (z. B. Zustand der Hostfirewall, Version des Virenscanners). Optional sieht die TCG auch die Verwendung des TPM vor. Die Spezifikation für den Zugriff auf das TPM (IF-PTS) war jedoch bis zum Abschluss der Recherchen nicht verfügbar, daher ist unklar, welche Rolle das TPM hierbei genau spielt.

- **Policy Enforcement Point (PEP)**
  Der PEP ist in der Regel eine aktive Netzkomponente (z. B. WLAN-Access-Point, Switch, Router) mit 802.1x-Unterstützung. Er ist zuständig für die Umsetzung der vom PDP getroffenen Entscheidung und für die Übertragung der Nachrichten zwischen AR und PDP. Eine solche Komponente bietet im Falle einer negativen Überprüfung i. d. R. zwei Optionen, das Abweisen des AR und das Verschieben des AR in ein vordefiniertes Netzsegment zur anschließenden Wiederherstellung des gewünschten Zustands.

- **Policy Decision Point (PDP)**
  Der PDP ist zuständig für die Bewertung der durch die IMCs ermittelten und durch den *TNC Client* übertragenen Information. Hierfür wird dieses System um die Softwarekomponente *TNC Server* erweitert, die die Daten vom *TNC Client* entgegennimmt. Die eigentliche Bewertung erfolgt durch die *Integrity Measure-*

ment Verifiers (IMV). Hierbei existiert für jeden der IMV auf dem *TNC Server* ein IMC auf dem *TNC Client*.

Der TNC Ansatz erscheint zunächst als sehr vielversprechend, da er die Problematik der Erzeugung von Prüfsummen zur Laufzeit eines Systems durch den Einsatz von Softwareagenten (IMCs) umgeht. Problematisch hierbei ist jedoch, dass die Agenten auf einem zu diesem Zeitpunkt nicht vertrauenswürdigen System ausgeführt werden und dadurch zunächst selbst nicht als vertrauenswürdig anerkannt werden können. Dies entspricht dem bereits angesprochenen Problem eines Systems, seine eigene Integrität nicht bewerten zu können. Zwar findet die Bewertung auf dem *TNC Server* statt, jedoch ausschließlich auf Basis der vom *TNC Client* übermittelten Informationen. Eine genaue Aussage über die Sicherheit des Systems kann erst dann gemacht werden, wenn die Rolle des *Platform Trust Service* (PTS) bzw. des TPM bekannt ist.

Das TNC-Konzept ist nach Meinung des Autors nur dann als sicher zu betrachten, wenn der TNC Client auf einem *Trusted-OS* zur Ausführung kommt, und der TNC Client selbst Teil der erzeugten *Vertrauenskette* ist. Dadurch wäre der TNC Client selbst in der Lage, die *Vertrauenskette* um die einzelnen IMCs zu erweitern. Der TNC Server kann dann zunächst durch Auswertung der vom TPM signierten *Vertrauenskette* die Integrität des TNC Client und der IMCs beurteilen, um sich dann mit den Informationen der Agenten zu befassen. Auf die Notwendigkeit eines *Trusted-OS* als Grundlage für die TNC-Architektur wird innerhalb der Spezifikationen nicht hingewiesen.

8.2.2 Microsoft Network Access Protection (NAP)

Microsoft stellt mit der Einführung der nächsten Generation seiner Desktop (Windows Vista)- und Server-Betriebssysteme (Windows Server 2008) eine mit der TNC-Architektur nahezu identische Lösung vor. Unter dem Namen *Network Access Protection* (NAP) definiert auch Microsoft eine durch Drittanbieter erweiterbare Architektur zur Feststellung des Systemzustands [Micro12]. Die folgende Abbildung beschreibt dieses System:

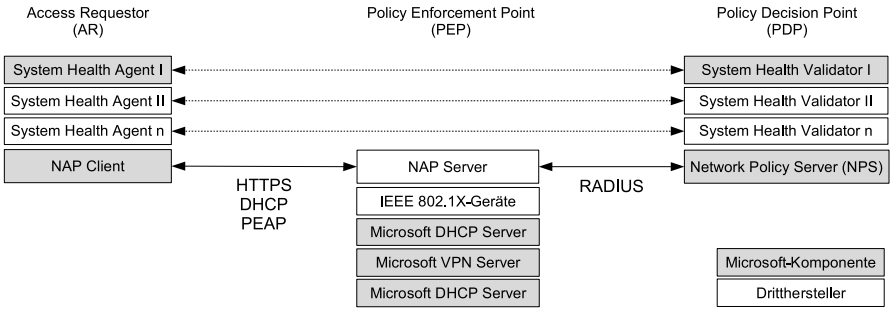

Abb. 8.7 Network Access Protection

Ein Großteil der Komponenten ist von ihrer Funktionsweise identisch zu denen der TNC-Architektur, trägt jedoch andere Namen. Der *NAP Client* entspricht hierbei dem *TNC Client,* der *TNC Server* dem *Network Policy Server* (NPS). Anstelle der *Integrity Measurement Collectors* kommen die *System Health Agents* (SHA) zum Einsatz. Die *Integrity Measurement Verifiers* sind mit den *System Health Validators* (SHV) vergleichbar. Trotz der engen Verwandtschaft sind die einzelnen Komponenten bzw. deren Schnittellen nicht mit TNC kompatibel. Nachdem Microsoft zunächst nur eine Kooperation mit der Lösung von Cisco (NAC) angekündigt hatte, wird nun auch an der Interoperabilität mit TNC gearbeitet [Micro13].

Im Gegensatz zu TNC liefert Microsoft ein SHA/SHV-Paar gleich mit. Dieses Paar dient der Überwachung bzw. der Auswertung der vom Windows-Sicherheitscenter bekannten Komponenten (Windows Firewall, Virenscanner und Windows Updates). Im Falle des *Policy Enforcement Point* definiert Microsoft neben den 802.1x-Geräten auch die Verwendung weiterer Microsoft-Server-Komponenten (z. B. DHCP Server, VPN Server).

Microsofts NAP sieht zum aktuellen Zeitpunkt keinerlei Verwendung für das TPM vor. Durch die einfache Erweiterbarkeit des NAP-Frameworks ist die Entwicklung eines SHA/SHV-Paars mit TPM-Support aber möglich[8]. Da die Architektur von Microsoft auf die Integrität der Agenten (SHVs) angewiesen ist, die möglichen Plattformen für den *TNC Client* (Windows XP, Windows Vista und Windows Server 2008) jedoch den Anforderungen an ein *Trusted-OS* nicht genügen, ist auch NAP für den im nächsten Abschnitt beschriebenen Angriff anfällig.

8.2.3 Cisco Network Admission Control (NAC)

Ciscos *Network Admission Control* (NAC) ist eine weitere mit der TNC-Architektur verwandte Technologie. Auf eine genaue Beschreibung der Umsetzung wird daher verzichtet. Anzumerken ist jedoch, dass nach Wissen des Autors Cisco als Erster eine entsprechende Technologie vorgestellt hatte und zum aktuellen Zeitpunkt mit seiner Lösung den größten Marktanteil hat.

Allen drei Lösungen gemein ist jedoch die Möglichkeit zur Umgehung der Schutzfunktionen. Nachdem bereits im Jahr 2006 auf der Black-Hat-Konferenz einige Angriffe auf Basis gefälschter MAC- und IP-Adressen demonstriert wurden [Arkin2006], zeigten zwei Sicherheitsexperten auf der Black-Hat-Konferenz 2007 weitere Angriffsvektoren. Mittels Reverse-Engineering des Kommunikationsprotokolls zwischen dem *Cisco Trust Agent*, dem *TNC-Client*-Pendant, und der Server-Komponente *Cisco Secure ACS* ist es den Experten gelungen, einen eigenen *Trust Agent* zu entwickeln. Dieser *Trust Agent* ist in der Lage, auch im Falle einer Abweichung des Rechners von der Sicherheitsrichtlinie der Firma einen integeren Systemzustand zu attestieren [Röcher2007]. Da dieser Angriff auf einer grundsätzlichen Schwäche der Ermittlung des Systemzustands durch Softwareagenten aufsetzt,

[8] In Kapitel 11 wird näher auf die Funktionen einer solchen NAP-Erweiterung eingegangen.

sind auch sämtliche andere Systeme dieser Art (wie z. B. Microsoft NAP und TNC) für einen vergleichbaren Angriff anfällig. Obwohl eine Verwendung des TPM zur Verbesserung der Sicherheit solcher Systeme beitragen könnte, hat auch Cisco bisher auf dessen Einsatz verzichtet.

8.2.4 Property-Based Attestation

Das von der TCG definierte Konzept zur Bewertung der Vertrauenswürdigkeit eines Systems basiert auf der Übermittlung der erzeugten Prüfsummen an ein externes System. Bei diesem System könnte es sich beispielsweise um einen Dienstanbieter für hochauflösendes Videomaterial im Internet handeln. An diesem Konzept lassen sich hinsichtlich Datenschutz und Sicherheit des Kunden einige Kritikpunkte finden:

1. Durch die Übermittlung der Prüfsummen, und somit der aktuellen Konfiguration der Hard- und Software des Systems, an eine entfernte Instanz (z. B. Serviceanbieter) erhält diese umfangreiche Informationen über das System. Diese Informationen können anschließend verwendet werden, um bestimmten Kunden (z. B. mit Linux-Betriebssystem) den Zugriff auf den Service zu verweigern, und dies unabhängig von der Sicherheit des ermittelten Systemzustands.
2. Weiter können diese Informationen auch dazu verwendet werden, die bei der Auswertung der Prüfsummen gefundenen Schwachstellen des Systems gezielt auszunutzen.
3. Die Bewertung auf Basis von Prüfsummen ist aufgrund der zahlreichen unterschiedlichen Systeme und unterschiedlichen Konfigurationen nicht praktikabel. Hinzu kommt die Auswertung von bekannten Sicherheitslücken und vorhandenen Patches für jede Systemkonfiguration.

Eine mögliche Lösung ist die Bewertung auf Basis von Eigenschaften (Properties) des Systems. Denkbare Plattform-Eigenschaften hierbei sind z. B. Informationen über in das System eingebaute Sicherheitsfunktionen, der Status wichtiger Systemkomponenten oder die Unterstützung für bestimmte Zugriffskontrollsysteme (z. B. *Multi-Level-Security* – MPLS). Die Herausforderung bei diesem Ansatz ist die Ableitung dieser Eigenschaften aus dem aktuellen Systemzustand. Hierfür ist eine Transformation der auf dem System vorhandenen Prüfsummen in Eigenschaften und umgekehrt notwendig. Ein solcher Transformator hat zusätzlich die Aufgabe der Ausstellung von so genannten Property-Zertifikaten, um die Glaubwürdigkeit der ermittelten Eigenschaft zu gewährleisten.

Für die Umsetzung der notwendigen Transformator-Komponente sind zwei unterschiedliche Ansätze denkbar, eine Erweiterung der Funktionen des TPM oder die Verwendung eines externen Systems als *Trusted-Third-Party* (TTP). Eine Erweiterung des TPM wäre aus Sicht des Autors die bessere der beiden möglichen Lösungen, jedoch aufgrund der Komplexität eines solchen Prozesses und der damit verbundenen Verteuerung des TPM-Chips für die Praxis aktuell wenig relevant.

Die Umsetzung in Form einer zusätzlichen TTP erscheint somit als praktikabler. Neben der in [Sadeghi2004] vorgestellten Umsetzung existiert mit der im nächsten Abschnitt vorgestellten „WS-Attestation“ ein weiteres Konzept für die Erweiterung der *Trusted-Computing*-Infrastruktur um eine solche Komponente.

8.2.5 WS-Attestation

Im Folgenden wird ein weiteres Konzept zur *Remote-Attestation*, das von IBM Research entwickelt wurde, vorgestellt. Unter dem Namen „*WS-Attestation*“ sind eine ganze Reihe von Konzepten und Erweiterungen von bestehenden Standards zusammengefasst. Im erstem Teil werden die Designprinzipien für die Erstellung von Prüfsummen zur Laufzeit und das Design einer für die *Remote Attestation* notwendigen Infrastruktur vorgestellt. Dem folgt die Definition eines *Remote-Attestation*-Protokolls auf Basis von *WS-Security*-tandards und den dafür notwendigen Erweiterungen. Den Schluss bildet eine Beschreibung der Architektur des entwickelten Prototyps auf Basis bereits vorhandener Technologien und der OSGi-Plattform (*Open Service Gateway initiative*).

8.2.5.1 Designprinzipien und Architektur

Auch IBM erkennt die Probleme bei der Abbildung des Systemzustands über die Laufzeit des Systems. So ist eine Bewertung der Systemintegrität auf Basis von Prüfsummen über auf dem System ausgeführte Prozesse (binary measurement) alleine nicht praktikabel, da solche Messprotokolle schnell eine nicht mehr auswertbare Größe überschreiten. *WS-Attestation* verwendet deshalb für das beschriebene Konzept eine Kombination aller drei bisher bekannten Ansätze.

1. Den Aufbau einer *Vertrauenskette* (*Chain of Trust*) beim Startvorgang (z. B. mit *Trusted-Grub*) sowie der Fortsetzung dieser Kette zur Laufzeit durch die *Integrity-Measurement-Architektur* (IMA).
2. Den Einsatz von feingranularen Messprotokollen durch die auf dem System ausgeführte Middleware wie z. B. die Java VM. Angedacht ist hierbei die Erzeugung von Prüfsummen über jedes in die VM geladene Java class file. Wichtig dabei ist, dass die Middleware selbst vor dem Laden durch das Betriebssystem durch die IMA oder durch eine vergleichbare Technologie protokolliert wurde, um die aufgebaute *Vertrauenskette* nicht zu unterbrechen.
3. Die Verwendung von *Semantic Attestation* zur Erfassung von Eigenschaften, die nicht über die Erzeugung von Prüfsummen abgebildet werden können. Als Beispiel dienen hierfür Agenten, die lokal auf dem zu bewertenden System Konfigurationsdateien oder den Status der Host-Firewall oder des Virenscanners überwachen. Identisch zur Erstellung von Prüfsummen durch die Middleware muss auch bei den Agenten sichergestellt werden, dass diese Teil der *Vertrauenskette* sind.

Während eine Bewertung der Systemintegrität auf Basis des von Agenten gemeldeten Systemzustands verhältnismäßig einfach umzusetzen ist, ist die Auswertung von Messprotokollen auf Basis von Prüfsummen ein sehr komplexes Unterfangen. So führt der Aufbau einer *Vertrauenskette* auf den unterschiedlichen Versionen und Varianten gängiger Betriebssysteme zu komplett unterschiedlichen Prüfsummen, was zu einer sehr langen Liste an notwendigen Referenzwerten führt. Darüber hinaus werden für jedes Betriebssystem Informationen über bekannte Angriffe und über die Verfügbarkeit von Patches benötigt (*Vulnerability Database*). Da solche Datenbanken nicht von jedem Serviceanbieter betrieben werden können, sieht das Konzept von IBM hierfür die Verwendung einer weiteren *Trusted Third Party,* des *Validation Service* vor. Der *Validation Service* übernimmt für den Serviceanbieter die Überprüfung der Systeme durch einen Vergleich der Prüfsummen mit Referenzwerten aus seinen Datenbanken. Die Systeme erhalten nach erfolgreicher Validierung einen Nachweis (*Attestation Credentials*) über den Grad ihrer Vertrauenswürdigkeit mit Bezug auf die durchgeführten Überprüfungen[9]. Ein weiterer Vorteil dieser Aufgabenteilung ist die mögliche Geheimhaltung der für die Bewertung verwendeten Prüfsummen. Neben dem Aspekt des Datenschutzes ist dies auch hinsichtlich der Sicherheit des Systems von Vorteil, da aus den Prüfsummen Rückschlüsse auf die genaue Konfiguration des Systems und somit auf mögliche Angriffsvektoren gezogen werden können. Ein Angreifer könnte diese Informationen für einen gezielten Angriff auf das System verwenden. Die folgende Abbildung zeigt die durch die Einführung eines *Validation Service* entstehende Architektur:

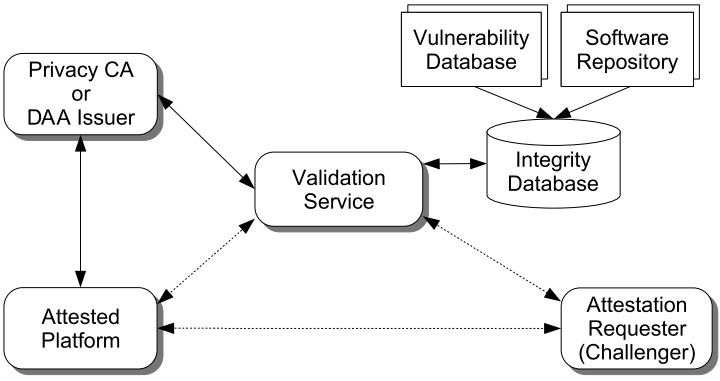

Abb. 8.8 WS-Attestation Architektur [Munetoh2005]

Zusätzlich zu der Definition der Architektur mit *Attested Platform* (AP), *Attestation Requester* (AR) und *Validation Service* (VS) enthält das Konzept vier unterschiedliche Kommunikationsmodelle bei der Überprüfung der Vertrauenswürdigkeit. Je

[9] Ein ähnlicher Ansatz ist auch für die NAP-Architektur von Microsoft verfügbar. Die TTP ist hierbei die *Health Registration Authority* (HRA), das Pendant zum *Attestation Credential* das *Health Certificate.* [Micro12]

nach Vertrauensstellung zwischen den Parteien kommt eines dieser Modelle zum Einsatz. Eine vollständige Beschreibung der Modelle liefert [Munetoh2005].

8.2.5.2 WS-Security-Erweiterungen (WS-Attestation)

Für die Kommunikation zwischen den einzelnen Parteien definiert IBM ein weiteres RA-Protokoll. Ziel hierbei war es, kein gänzlich neues Protokoll zu entwickeln, sondern bereits vorhandene und etablierte Protokolle und Datenstrukturen zu verwenden und diese nur geringfügig zu erweitern. Aus diesem Grund wurde hierfür ein neues Profil für das *WS-Security-Framework* entwickelt. Hierbei musste die Liste der in *WS-Security* vorhandenen Optionen zur Signatur von XML-Nachrichten um die Funktion des TPM (*TPM_QUOTE*) erweitert werden. Dies kann durch eine verhältnismäßig kleine Modifikation einer *WS-Security*-Implementierung[10] umgesetzt werden. Für den Datenaustausch zwischen den Parteien wurden zusätzlich drei neue Datenstrukturen (*security tokens*) definiert:

1. **Measurements (PCR)**
   Dies ist die einfachste Datenstruktur und enthält die im Rahmen der *Vertrauenskette* erzeugten Prüfsummen und somit den Inhalt der PCR des TPM.

2. **Platform Measurement Description (PMD)**
   Die PMD bietet die Möglichkeit zur Übertragung von weiteren Prüfsummen und den zugehörigen Metainformationen wie z. B. die *Measurement List* der IMA. Auch die von den Agenten ermittelten Eigenschaften werden hier hinterlegt.

3. **Attestation Credential**
   Dies ist der von *Validation Service* ausgestellte Nachweis über den Grad der Vertrauenswürdigkeit des Systems. Dieses *Token* ist vergleichbar mit den in *Single-Sign-On*-Systemen verwendeten Tickets und unterliegt ähnlichen Anforderungen hinsichtlich Gültigkeitsdauer und Vertraulichkeit. Das *Attestation Credential* kann sowohl in Form eines X.509-Zertifikats mit entsprechenden Erweiterungen als auch als *SAML-Assertions*[11] ausgestellt werden.

Für die eigentliche Kommunikation wird das *WS-Trust*-Protokoll verwendet, da es die notwendigen Funktionen zur Signatur der Nachrichten sowie ein Schlüsselaustausch-Protokoll zur Erzeugung eines Sitzungsschlüssels bereits beinhaltet. Die Abb. 8.9 beschreibt die ausgetauschten Nachrichten für eines der vier Kommunikationsmodelle (*Delegated Attestation*):

1. SecurityTokenRequest[TokenType(Attestation Credential), Challenge(n1)]
2. SecurityTokenRequest[TokenType(PCR, PMD), Challenge(n2)]
3. SecurityTokenResponse[Token(Attestation Credential), Challenge(n2)]AIK_{AP}
4. SecurityTokenResponse[Token(Attestation Credential), Challenge(n1)]KEY_{VS}

[10] Für den Prototyp wurde hierfür die SOAP/WS-Security-Implementierung von OSGi erweitert.

[11] SAML – Security Assertion Markup Language

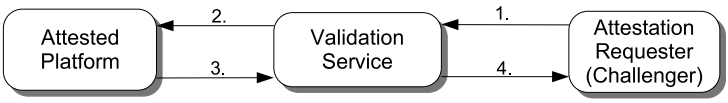

Abb. 8.9 WS-Attestation mit WS-Trust

Der Prototyp ist zum aktuellen Zeitpunkt nicht öffentlich zugänglich. Auch die Erweiterung *WS-Attestation* ist bisher noch nicht in die offiziellen Standards aufgenommen worden. Die Idee hinter der beschriebenen Architektur sowie die Verwendung offener Standards für die Umsetzung eröffnen dem Ansatz, nach Meinung des Autors, dennoch gute Chancen auf Unterstützung durch andere Projekte.

8.2.6 Sicherheit des Attestation-Protokolls

Bei der Durchführung einer*Remote Attestation* müssen die auf einem System erzeugten Prüfsummen und etwaige Metainformationen sicher an ein entferntes System übertragen werden. Hierfür wird ein entsprechendes RA-Protokoll benötigt. Die TCG selbst sieht hierfür die Verwendung der auf Seite 89 erwähnten TLS[12]-Erweiterung (*TLS-Attestation*) vor. Die Spezifikationen zum *TLS-Attestation*-Protokoll sind zum Zeitpunkt der Erstellung des Buches noch nicht öffentlich zugänglich, es ist jedoch davon auszugehen, dass das auf Seite 55 beschriebene *Challenge-Response*-Verfahren zur Signatur der Prüfsummen durch das TPM zum Einsatz kommt. Hierbei wird eine vom entfernten System erzeugte Zufallszahl zusammen mit den Werten der *Platform Configuration Register* durch das TPM signiert. Dies geschieht durch einen Aufruf des TPM_Quote-Kommandos und einen der vom TPM verwalteten *Attestation Identity Keys* (AIK). Die signierte Datenstruktur wird an das externe System übermittelt, welches durch Auswertung des AIK-Zertifikates und der digitalen Signatur feststellen kann, dass der Absender über ein TPM verfügt und die PCR-Werte bei der Übermittlung nicht modifiziert wurden.

Durch die Verwendung eines *Challenge-Response*-Verfahrens und der Signatur durch das TPM ist das Protokoll resistent gegen Manipulation der Daten und gegen das Abfangen und erneute Senden der Antwort (Replay-Angriff). In „*A Robust Integrity Reporting Protocol for Remote Attestation*“[13] wird jedoch die Möglichkeit eines Masquerading-Angriffs[14] auf dieses Verfahren beschrieben. Der Angreifer benötigt hierbei neben seinem modifizierten System (M) Zugriff auf ein weiteres System (B), welches den Sicherheitsanforderungen des entfernten Systems (A) entspricht – also ein System, welches sich in einem vertrauenswürdigen Zustand hinsichtlich der ausgeführten Applikation und deren Konfiguration befindet. Wird nun das System M des Angreifers von A zur Übermittlung der Prüfsummen auf-

[12] TLS – Transport Layer Security

[13] [Stumpf2006]

[14] Der Masquerading-Angriff ist auch als Relay-Angriff bekannt

gefordert, leitet es diesen Aufruf an das vertrauenswürdige System B weiter. Die folgende Abbildung veranschaulicht diesen Vorgang:

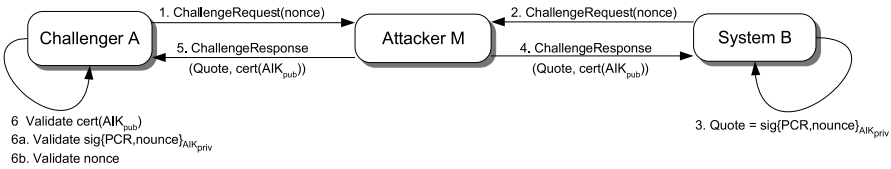

Abb. 8.10 Remote-Attestation-Protokoll

Hierbei werden die auf dem vertrauenswürdigen System B erzeugten Prüfsummen für den Nachweis der Integrität des Systems des Angreifers M verwendet. Dieser Angriffsvektor liegt in den Eigenschaften der AIKs begründet. Sie ermöglichen per Design lediglich den Nachweis, dass ein zu den TCG-Spezifikationen konformes TPM verwendet wurde um die Daten zu signieren; sie sind nicht geeignet, um eine bestimmte Gegenstelle, sprich ein Computersystem, eindeutig zu identifizieren. Grund hierfür ist die aus Sicht des Datenschutzes notwendige Entkoppelung der Beziehung zwischen dem *Endorsement Key* (EK) der Plattform und den AIKs.

Auch eine dem Verfahren vorausgehende beidseitige Authentisierung der TLS löst das Problem nur zu einem bestimmten Grad, da die hierbei verwendeten X.509-Zertifikate in der Regel einen bestimmten Benutzer und nicht ein System identifizieren. Hat der Angreifer die Kontrolle über ein vertrauenswürdiges System, ist er in der Lage, das dort hinterlegte X.509-Benutzer-Zertifikat für den Aufbau des TLS-Tunnels zum System M zu verwenden.

Als Lösung für das Problem ist die Verwendung zusätzlicher kryptographischer Schlüssel zur Absicherung der Kommunikation denkbar. Hierbei wird das RA-Protokoll um den Austausch eines Sitzungsschlüssels (*shared session key*) für die Verschlüsselung der nachfolgenden Kommunikation erweitert. Als Schlüssel-Austausch-Protokoll kann hierbei sowohl das Diffie-Hellman- als auch das RSA-Verfahren verwendet werden. Die folgende Abbildung zeigt den um das Diffie-Hellman-Verfahren erweiterten Ablauf:

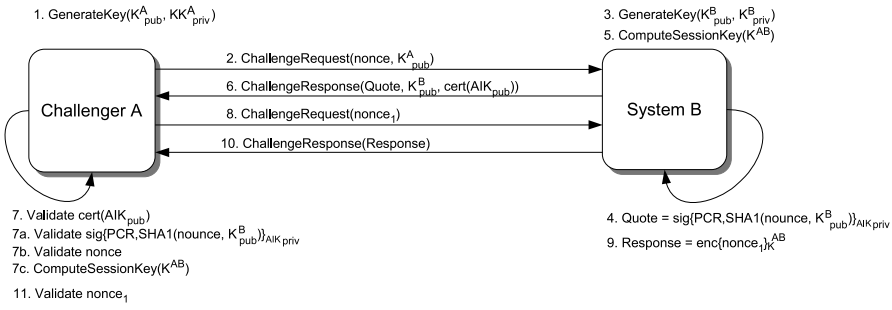

Abb. 8.11 Erweitertes Remote-Attestation-Protokoll

Die entscheidende Erweiterung des neuen Protokolls ist die Generierung eines zusätzlichen Schlüsselpaares nach dem DH-Verfahren auf beiden Seiten. Da der öffentliche Teil des Schlüssels des Systems B zusätzlich zusammen mit den PCR-Registern signiert wird, handelt es sich hierbei um eine abgeschwächte Variante, das Signed-DH-Verfahren. Abgeschwächt bedeutet hierbei, dass nur das zu überprüfende System B sich authentisiert, sprich seine Nachricht signiert. Um das Verfahren zusätzlich resistent gegen eine Fälschung des Challengers A zu machen und somit dem System B eine Identifizierung des Kommunikationspartners zu erlauben, wäre eine Signatur der Anfrage in Schritt 2 notwendig. Um jedoch auch im Falle der einseitigen Signatur einen *Man-In-The-Middle*-Angriff zu erkennen, wird mit den Schritten 8–11 zusätzlich der gemeinsame Sitzungsschlüssel auf seine Richtigkeit überprüft.

Das von [Stumpf2006] entwickelte Verfahren ist als Erweiterung des noch nicht finalen *TLS-Attestation*-Protokolls gedacht, kann jedoch auch mit einigen Anpassungen auf andere RA-Protokolle angewandt werden.

Hinweis: Da das TPM keine Unterstützung für symmetrische Kryptographie bietet, befindet sich der für die Kommunikation notwendige Sitzungsschlüssel im Arbeitsspeicher des Systems. Um ein Auslesen des Schlüssels auf dem vertrauenswürdigen System zu verhindern, muss das Trusted Computing System den Speicherbereich der für die Remote Attestation zuständigen Softwarekomponente vor dem Zugriff durch andere Prozesse schützen. Hierfür kann eine der unter „Protected Execution“ vorgestellten Lösungen verwendet werden.

8.3 Trusted Software Stack (TSS)

Ein *Trusted Software Stack* (TSS) dient als Schnittstelle zwischen Applikationen und Diensten des Betriebssystems und dem *Trusted-Platform-Modul*. Die einzelnen Schichten dieses Software Stack sowie deren Funktionalität sind durch die TCG-Spezifikation festgelegt [TCG2006-2]. Neben etlichen kommerziellen Implementierungen dieses Standards existieren auch einige quelloffene Varianten. Die Open-Source-Implementierungen werden in diesem Abschnitt näher betrachtet.

8.3.1 TrouSerS

Im Rahmen des TrouSerS[15]-Projekts wird ein zur TCG-Spezifikation konformer *Trusted Software Stack* für Linux entwickelt. Der TSS wird bereits in einigen TC-Projekten eingesetzt und im Rahmen der *OpenTC*-Initiative regelmäßig weiterentwickelt. Zum aktuellen Zeitpunkt unterstützt *TrouSerS* alle in Version 1.1 der Spe-

[15] [URL22]

zifikation geforderten Funktionen. Die Arbeiten an der Umsetzung der Version 1.2 der Spezifikation sind bisher noch nicht abgeschlossen.

8.3.2 Trusted Java

Trusted Java[16] ist ebenfalls ein Teilprojekt der *OpenTC*-Initiative. Im Rahmen dieses Projektes wurde zunächst mit jTss Wrapper eine Umsetzung des *TrouSerS* TSS auf Java verfolgt. Unter dem Namen jTSS wird nun an einer Implementierung der TCG-Spezifikation V1.2 in Java gearbeitet. Ziel hierbei ist es, sämtliche Schichten des TSS in 100% Java umzusetzen. Die Entwicklung wird vor allem an der TU-Graz vorangetrieben.

8.3.3 TPM/J

Eine weitere Implementierung eines TSS in Java existiert im TPM/J[17]. TPM/J ist jedoch absichtlich nicht konform zu den Spezifikationen der TCG. Das Ziel von TPM/J ist die möglichst vollständige Implementierung der TPM-Kommandos als Java-Klassen. Eine Umsetzung der geforderten Schichten und deren Funktionen (wie z. B. Key Cache Manager) wird zurzeit nicht angestrebt. TPM/J eignet sich daher eher für den Einsatz in Test-Anwendungen und Prototypen. Neben Linux, MacOS und Windows XP ist TMP/J auch auf Windows Vista lauffähig. Die dabei notwendigen Anpassungen wurden im Rahmen der Entstehung dieses Buches vom Autor entwickelt und sind in die aktuelle Version von TPM/J integriert.

8.4 Protected Execution

Die Kernkomponenten jedes *Trusted-Computing-Systems* ist die *Trusted Computing Base* (TCB). Oberstes Schutzziel ist daher, die TCB vor Manipulationen durch andere auf dem System ausgeführten Prozesse zu bewahren. Die hierfür notwendige Isolation der TCB wird auch als *Protected Execution* bezeichnet. Darüber hinaus erlaubt die Umsetzung dieses Konzeptes die Partitionierung des Systems in Bereiche mit unterschiedlichen Integritätsanforderungen (*Security Domains*). Dieses Kapitel liefert einen Überblick über einige im Rahmen von Forschungsprojekten entstandene Sicherheitsarchitekturen für ein *Trusted-OS*. Interessant hierbei ist, dass alle vorgestellten Projekte eine Form von Virtualisierungstechnologie zur Isolation einsetzten. Durch den Einsatz eines *Virtual Machine Monitor* (VMM) profitieren die

[16] [URL23]
[17] [URL24]

vorgestellten Architekturen von folgenden allgemeinen Eigenschaften dieser Technik:

- **Isolation**
  Ein VMM erlaubt die Ausführung mehrerer Betriebssysteme in unterschiedlichen virtuellen Maschinen (VM). Jede VM läuft hierbei auf ihrer eigenen virtualisierten Hardware unabhängig von den anderen VMs des Systems. Vor allem die Virtualisierung der Zugriffe auf den Arbeitsspeicher vereinfacht die Isolationen der einzelnen Partitionen. Eine Kompromittierung eines Betriebssystems oder einer Applikationen innerhalb einer Partition ist weniger kritisch für die Integrität des gesamten Systems. Der VMM bildet hierbei eine zusätzliche Schutzschicht (Layer of Protection).

- **Kompatibilität**
  Durch das Einschieben einer Virtualisierungsschicht zwischen das Betriebssystem und die Hardware entsteht eine Abstraktion der Eigenschaften der physikalischen Hardware. Das Betriebssystem in der virtuellen Maschine findet daher auf jedem physikalischen System dieselben definierten Schnittstellen zur Kommunikation mit der Hardware wieder. Die Erstellung von virtuellen Umgebungen für bestimmte Aufgaben (*Virtual Appliances*) wird dadurch stark vereinfacht. Ein solches VM-Image kann ohne vorhergehende Installation oder Konfiguration direkt durch den VMM ausgeführt werden. Im Internet existieren bereits mehrere kommerzielle und auch kostenlose Angebote dieser Art[18].

- **Sicherheit**
  Ein VMM ist im Vergleich zu gängigen Desktop-Betriebssystemen eine verhältnismäßig einfache Software hinsichtlich Funktion und Umfang des Sourcecodes. Ein VMM muss im Gegensatz zu gewöhnlichen Betriebssystemen nur wenige einfache Schnittstellen und Funktionen (wie z. B. CPU-Zugriff, Speicherverwaltung, I/O-Verwaltung) implementieren. Dies erlaubt die Entwicklung einer schlanken Softwarekomponente, was die Überprüfung auf Programmierfehler durch Code-Reviews deutlich vereinfacht. Im Rahmen des VFiasco[19]-Projektes wird sogar an der Erstellung eines beweisbar sicheren Mikrokernels gearbeitet.

Virtualisierungstechnologie eignet sich auf Grund dieser und weiterer Eigenschaften sehr gut zur Reduzierung der Komplexität der sicherheitskritischen Komponenten eines Systems, der *Trusted Computing Base*. [Hohmuth2004] und [Singaravelu2006] beschreiben die Reduzierung der Größe der TCB als eines der wichtigsten Designkriterien für den Bau von *Trusted Computing Systems*.

[18] z. B. http://virtualappliances.net/; http://www.vmware.com/vmtn/appliances/

[19] [URL25]

8.4.1 Terra Architecture

Terra ist die älteste dem Autor bekannte Beschreibung einer *Trusted-Operating-System*-Architektur und wurde erstmals in [Garfinkel2003] vorgestellt. Das Ziel von *Terra* ist die Partitionierung des Systems in isolierte virtuelle Maschinen (VM) durch Einsatz eines *Trusted Virtual Machine Monitor* (TVMM). *Terra* definiert hierbei drei unterschiedliche Arten von virtuellen Maschinen:

- **„Open-Box"-VM**
  Diese Art von VM erlaubt die Ausführung von vorhandenen Betriebssystemen und Standard-Applikationen in einer virtualisierten Umgebung. Diese VMs unterliegen keinerlei Einschränkungen hinsichtlich der ausgeführten Applikationen und dem Zugriff durch den Benutzer der Plattform, profitieren jedoch bereits von der durch den VMM erzeugten Isolation von anderen „Open-Box"-VMs des Systems.

- **„Closed-Box"-VM**
  Das Konzept der „Closed-Box"-VM entspricht dem von Geräten aus dem Gebiet der Unterhaltungselektronik (z. B. Spielekonsolen und Set-Top-Boxen). Sie enthalten ein System mit eingeschränkten, aber sehr speziellen Funktionen für eine bestimmte Aufgabe. Der TVMM schützt die Daten der VM hinsichtlich Vertraulichkeit und Integrität. Der Inhalt der VM kann daher nicht vom Eigentümer der Plattform eingesehen oder verändert werden. Der TVMM bietet weiter die Möglichkeit, den aktuellen Zustand einer „Closed-Box"-VM an eine entfernte Instanz zu übermitteln, und bietet somit eine weitere Implementierung der *Remote Attestation*. Identisch zu den „Open-Box"-VMs sind auch die „Closed-Box"-VMs vor Zugriffen von anderen Partitionen durch Abschirmung des Arbeitsspeichers geschützt. Beispiele für eine „Closed-Box"-Anwendung sind z. B. sicheres Online-Banking oder Personal Firewalls.

- **Management-VM**
  Die Management-VM ist verantwortlich für die Konfiguration der beiden anderen Arten von virtuellen Maschinen hinsichtlich zugeteiltem Festplattenspeicher, Arbeitsspeicher und verfügbaren Schnittstellen zur Kommunikationen mit anderen Systemen (z. B. virtuelle Netzwerkkarten). Sie erlaubt zusätzlich das Starten und Stoppen sowie die Überwachung der verwendeten Ressourcen der virtuellen Maschinen.

Ein Beispiel für die dabei entstehende Architektur beschreibt Abb. 8.12.
Neben den allgemeinen Funktionen eines VMM verfügt die Terra-Architektur über eine Umsetzung des *Integrity-Measurement*- bzw. des *Integrity-Protection*-Konzeptes zur Protokollierung des Systemzustands. Für den Aufbau dieser *Vertrauenskette* setzt *Terra* ebenfalls auf das TPM als den in Hardware verankerten *Vertrauensanker*. Die Fortsetzung der durch die *Trusted Computing Platform* erzeugte *Vertrauenskette* erfolgt nach demselben Konzept wie beim bereits beschriebenen

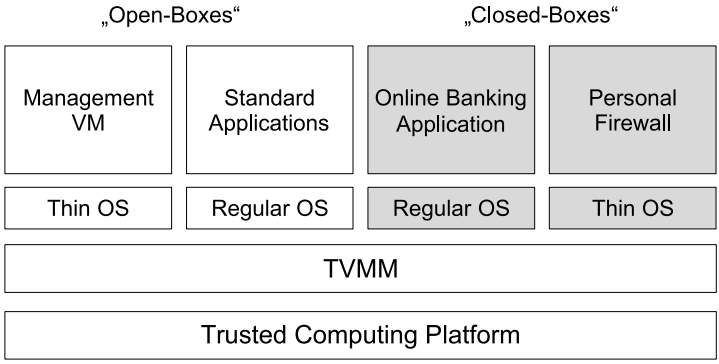

Abb. 8.12 Terra-Architektur

Trusted-Grub Bootmanager. Hierbei werden alle Komponenten bis zum erfolgreichen Start des TVMM protokolliert und in den Registern des TPM hinterlegt. Bei der Erstellung von Prüfsummen zur Laufzeit der Architektur beschränkt sich *Terra* auf das Protokollieren der einzelnen virtuellen Maschinen durch Erstellen von Prüfsummen über relevante Teile der VM-Images. Dadurch umgeht *Terra* zum Teil die beschriebene Problematik der Komplexität solcher Protokolle. Dies ist ein weiterer Vorteil beim Einsatz einer Virtualisierungsschicht, da für einen bestimmten Anwendungsfall nur die Prüfsummen des Startvorgangs des VMM und die einer virtuellen Maschine evaluiert werden müssen.

Ein solcher Anwendungsfall wäre z. B. die Verwendung einer Online-Banking-Applikation innerhalb einer „Closed-Box“-VM. Das VM-Image wird von der Bank als CD-ROM oder als Download zur Verfügung gestellt. Beim Start der VM erzeugt der TVMM nun eine Prüfsumme über das komplette VM-Image oder über Teilbereiche davon. Beim Verbinden der Online-Banking-Software mit dem Server der Bank erfragt dieser zunächst die beim Startvorgang des TVMM erzeugten Prüfsummen und überzeugt sich dadurch von der Integrität der Virtualisierungsschicht. Durch den darauf folgenden Vergleich der Prüfsumme des VM-Image mit dem auf dem Server der Bank hinterlegten Referenzwert kann sichergestellt werden, dass das Image seit der Auslieferung nicht manipuliert wurde. Als zusätzliche Schutzfunktion wäre es denkbar das VM-Image bei der Auslieferung an eine bestimmte Plattform zu binden. Hierfür kann die Sealing-Funktion des TPM verwendet werden. Somit könnte das VM-Image nur auf einem bestimmten PC und nur im Falle eines integeren Zustands entschlüsselt und verwendet werden.

Als weitere Anforderungen an ein *Trusted Computing System* identifiziert *Terra* ebenfalls eine Schnittstelle zur vertrauenswürdigen Interaktion mit dem Benutzer (Secure- bzw. Trusted-GUI[20]) und den Schutz der Ein- und Ausgaben des Systems (*Trusted-I/O*), verweist jedoch nur auf die dafür notwendige Unterstützung durch die Hardware des Systems. Der auf der Publikation basierende Prototyp adressiert

[20] Beispiele für die Umsetzung dieser Anforderung sind ab Seite 122 zu finden.

diese beiden zusätzlichen Anforderungen nicht. Nach Wissen des Autors ist der beschriebene Prototyp nicht öffentlich verfügbar.

8.4.2 Nizza Architecture

Eine weitere Architektur zum Bau eines *Trusted-OS* wird an der TU Dresden[21] unter dem Namen „*Nizza Secure-System Architecture*“ entwickelt. Identisch zur Terra-Architektur findet auch hierbei eine Verlagerung der Standard-Betriebssysteme aus dem TCB in eine virtuelle Maschine statt. Die Sicherheit des Gesamtsystems soll somit von der Sicherheit des Betriebssystems entkoppelt werden. Die Nizza-Architektur geht hierbei jedoch mit der Forderung, dass sämtliche sicherheitskritischen Applikationen ebenfalls unabhängig vom verwendeten Betriebssystem sein sollen, noch einen entscheidenden Schritt weiter. Trotzdem soll die Architektur die Möglichkeit bieten, die umfangreichen Funktionen eines Desktop-Betriebssystems weiterhin zu verwenden und Anwendungen wie gewohnt auszuführen. Hieraus ergeben sich die zwei wesentlichen Designkriterien der Nizza-Architektur:

- **Isolation**
  Alle sicherheitskritischen Teile der Architektur wie der sichere Systemkern (*Secure Kernel*) und Applikationen und Dienste (wie z. B. VPN-Software und Firewall) sind als Teil der *Trusted Computing Base* (TCB) zu verstehen und daher vom Rest des Systems zu isolieren. Hierbei sollen jedoch nur die wirklich kritischen Teile der Applikationen in die TCB integriert werden, um deren Codeumfang möglichst gering zu halten. Nichtkritische Teile der Applikationen sollen nach wie vor innerhalb der regulären Betriebssysteme in einer VM ausgeführt werden. Die Kommunikation zwischen den Komponenten der TCB und den unterschiedlichen Sicherheitsdomänen (Security Domains) soll nur mittels kontrollierter Inter-Prozess-Kommunikation (IPC) möglich sein.

- **Weiterverwendung von regulären Betriebssystemen**
  Um die volle Funktionalität des Standard-Betriebssystems anbieten zu können, bietet Nizza die Möglichkeit, dieses innerhalb eines gesicherten Containers auszuführen, um somit eine möglichst große Anzahl an Standard-Software ohne Anpassungen zu unterstützen. Darüber hinaus soll es möglich sein, auf die Funktionen dieser Betriebssysteme von außen zuzugreifen um sie dadurch auch für vertrauenswürdige Applikationen innerhalb der TCB verfügbar zu machen. Auch eine Kommunikation zwischen Applikationen der Betriebssysteme und der TCB ist wünschenswert.

Für die Umsetzung der Designkriterien verwendet Nizza ein Virtualisierungskonzept auf Basis eines schlanken Sicherheitskerns (*Secure Kernel*) und einer darauf aufbauenden Sicherheitsschicht. Die folgende Abbildung beschreibt diesen Aufbau:

[21] http://www.tudos.org

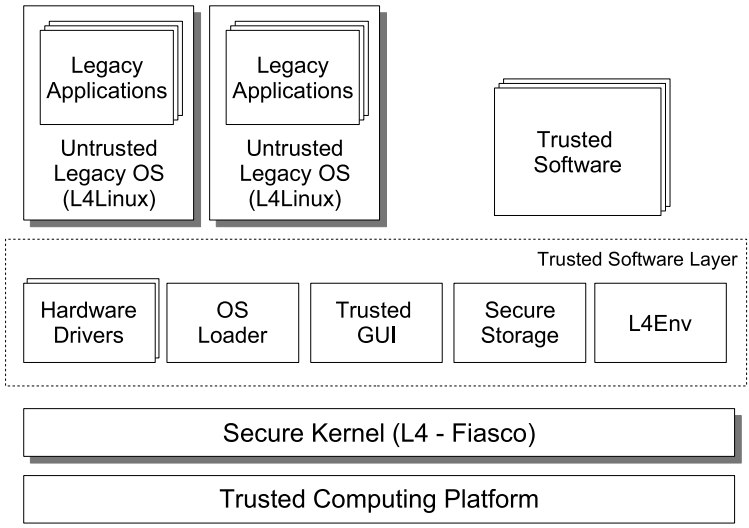

Abb. 8.13 Nizza-Architektur

Im Gegensatz zur Terra-Architektur verwendet Nizza für die Umsetzung der Virtualisierungsschicht keinen klassischen VMM (z. B. VMware ESX), sondern die Implementierung eines L4-Mikrokernels. Hierfür kommt der um wenige Funktionen erweiterte Fiasco-Kernel[22] der TU Dresden zum Einsatz. Der Vorteil dieses Ansatzes liegt in der Möglichkeit, vertrauenswürdige Applikationen (Trusted Software) mit Hilfe des *Trusted Software Layers* direkt oberhalb des *Secure Kernels* auszuführen. Ein zusätzliches Betriebssystem wird hierbei nicht benötigt, was sich positiv auf die Größe der TCB für diesen Anwendungsfall auswirkt. Weiter profitiert diese Architektur von sämtlichen Vorteilen des Mikrokernels, wie z. B. der Auslagerung der Gerätetreiber in den unprivilegierten Userspace.

Dieser Ansatz hat jedoch den Nachteil, dass diese *Trusted Software* neu entwickelt bzw. auf die API des *Trusted Software Layer* portiert werden muss. Für die Ausführung von regulären Betriebssystemen oberhalb des *Secure Kernels* müssen diese ebenfalls an die Schnittstelle des L4-Mikrokernels sowie an die Ausführung im Userspace angepasst werden. Daher ermöglicht die Nizza-Architektur in ihrer aktuellen Version nur die Ausführung von quelloffenen Gast-Betriebssystemen. Dieses als *Paravirtualisierung* bezeichnete Konzept kommt u. a. auch beim Xen Hypervisor zum Einsatz und ersetzt die ansonsten notwendige Emulation von Hardware wie z. B. der Grafikkarte. Mit L4Linux[23] existiert bereits eine Portierung von Linux auf die L4-API. Jede L4Linux-Domäne wird hierbei wie eine gewöhnliche L4-Applikationen als Prozess ausgeführt und durch den Mikrokernel verwaltet. L4Linux bleibt trotz der Anpassungen binär-kompatibel zu Linux und somit zu sämtlichen unter Linux lauffähigen Applikationen. Zusätzlich verfügt eine

[22] [URL26]
[23] [URL27]

L4Linux-Applikation jedoch über die Möglichkeit, auf die API des *Trusted Software Layer* und dessen Funktionen zuzugreifen. Dadurch wird auch die Kommunikation zwischen L4Linux-Applikationen und der *Trusted Software* ermöglicht.

Ein wichtiger Bestandteil der Architektur ist der *Trusted Software Layer*, der unter anderem die folgende Funktionalität liefert:

- **Loader**
  Der Loader ist verantwortlich für das Laden und Installieren von *Trusted Software*. Er ist zuständig für die Einhaltung der Zugriffsrechte der einzelnen Partitionen und stellt sicher, dass nur definierte Kommunikationskanäle zwischen den Domänen der Betriebssysteme und der *Trusted Software* verwendet werden können. Zusätzlich ermöglicht der Loader auch die Protokollierung des Startvorgangs durch Einsatz des TPM zur späteren Verwendung bei der *Remote Attestation*.

- **Trusted-GUI**
  Auch die Nizza-Architektur identifiziert die Notwendigkeit eines vertrauenswürdigen Kanals zwischen der Sicherheitsarchitektur und dem Benutzer. Unter dem Namen *Nitpicker*[24] entwickelt das Projekt eine umfangreiche Implementierung des *Trusted-GUI*-Ansatzes.

- **Secure Storage**
  Die Secure-Storage-Komponente dient der sicheren Speicherung von kritischen Daten (wie z. B. Passwörter und Schlüsselmaterial) der *Trusted Software* und der L4-Linux Partitionen. Die enthaltenen Daten werden hierfür unter Verwendung des TPM verschlüsselt abgelegt. Diese Implementierung ist vergleichbar mit der im TSS enthaltenen Schlüsselverwaltung.

Neben der Verschlüsselung des *Secure Storage* und der Protokollierung des Zustandes der *Trusted-Software*-Komponenten verwendet Nizza die TCP auch für den Aufbau einer *Vertrauenskette* vom Start des Systems bis zum erfolgreichen Laden des *Secure Kernels.* Wie diese Funktionalität umgesetzt wurde und welches Konzept für die *Remote Attestation* zum Einsatz kommt, wird jedoch nicht näher beschrieben.

Die Nizza-Architektur wird aktuell im Rahmen eines von der Europäischen Kommission geförderten Projektes[25] weiterentwickelt. Ein Teil der dabei entstandenen Resultate kann in Form einer Demo-CD bereits heruntergeladen werden. Weitere technische Details zur Nizza-Architektur ist in [Haertig2005] zu finden.

[24] Nitpicker wird detailliert ab Seite 123 beschrieben.

[25] Die OpenTC Initiative wird auf Seite 128 vorgestellt.

8.4.3 Perseus Architecture

Zeitgleich mit der Nizza-Architektur startete an der Ruhr-Universität Bochum die Entwicklung der Perseus-Architektur[26]. Die den beiden Entwicklungen zu Grunde liegende Idee wurde bereits 1999 an der Universität des Saarlands in Kooperation mit IBM Research Zürich entwickelt. Die Nähe der beiden Projekte schlägt sich auch in der jeweiligen Entwicklung nieder; so verwenden beide Implementierungen zum Großteil die gleichen Komponenten. So setzt auch die Perseus-Architektur für ihren *Secure Kernel* auf den Fiasco-Mikrokern der TU Dresden und auf die Verwendung von L4Linux als Gast-Betriebssysteme. Einige wenige Unterscheidungsmerkmale lassen sich in den Diensten des *Trusted Software Layer* erkennen. Hier verweisen die Publikationen zur Perseus-Architektur auf einige zusätzliche Komponenten. Die folgende Abbildung beschreibt diese zusätzlichen Bausteine innerhalb der Architektur:

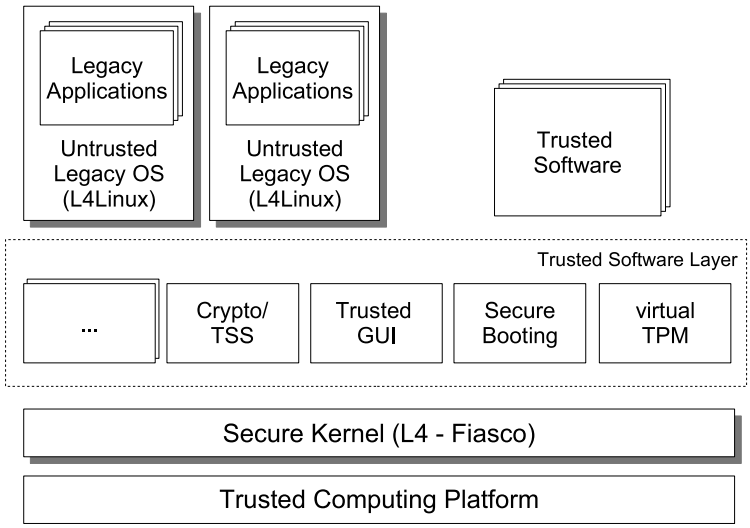

Abb. 8.14 Perseus-Architektur

Für die Umsetzung des *Trusted-GUI* setzt das Projekt auf eine mit *Nitpicker* verwandte Lösung[27]. Über die Crypto-Services bietet die Architektur zusätzlich Zugriff auf die kryptographischen Funktionen des TPM (z. B. Sealing, Binding, Signing und Sealed-Signing). Für die Fortsetzung der *Vertrauenskette* sorgt die Secure-Booting-Komponente; sie ermöglicht neben dem protokollierten Start von *Trusted Software* auch die Erzeugung von Prüfsummen beim Start der Gast-Betriebssysteme. Für

[26] http://www.perseus-os.org

[27] Vermutlich handelt es sich hierbei um Nitpicker was jedoch in den Publikationen nicht direkt bestätigt wird.

Secure Booting kommt der ebenfalls an der Ruhr-Universität Bochum entwickelte *Trusted-Grub* Bootmanager zum Einsatz.

Um auch innerhalb der Gast-Betriebssysteme von den Funktionen eines TPM zu profitieren, besteht die Möglichkeit, jeder Betriebssystem-Partition ein eigenes virtuelles TPM (vTPM) zur Verfügung zu stellen. Für die Gast-Betriebssysteme ist kein Unterschied zu einem Hardware-TPM zu erkennen. Die Integrität und Vertraulichkeit der virtuellen TPMs wird durch das TPM der *Trusted Computing Platform* sichergestellt.

Ein weiterer Unterschied zwischen den Projekten liegt in der Option der Perseus-Architektur, die Virtualisierungsschicht mit Hilfe des Xen-Hypervisors zu realisieren, um auch nicht-quelloffene Betriebssysteme wie Microsofts Windows ausführen zu können. Hierbei gibt es im Wesentlichen zwei unterschiedliche Möglichkeiten für die Positionierung des Xen-Hypervisors: zum einen als eine weitere Virtualisierungsschicht oberhalb des *Trusted Software Layer*.

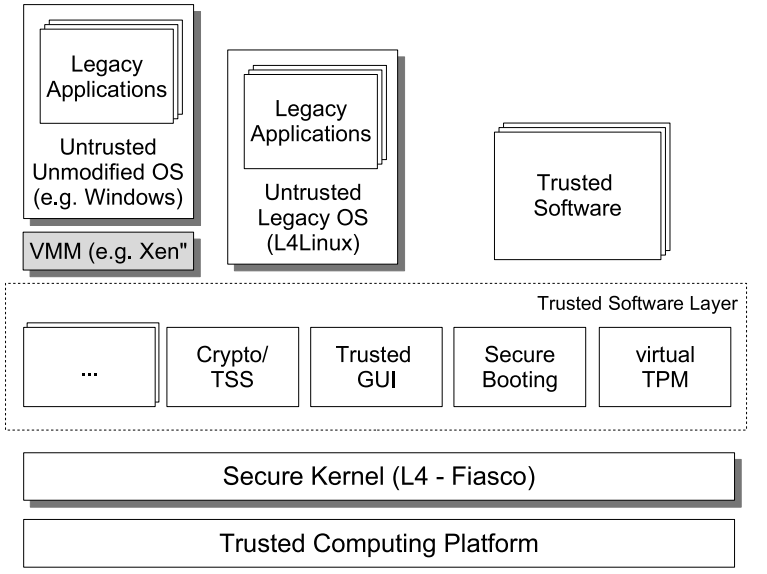

Abb. 8.15 Perseus-Architektur mit Xen

Hierbei könnten anstelle von Xen auch andere VMM-Implementierungen wie z. B. Qemu oder VMware zum Einsatz kommen. Der Nachteil dieser Anordnung ist die notwendige Portierung des VMM auf die Schnittstellen des L4-Mikrokerns. Zusätzlich ist mit einer Verschlechterung der Performance durch die zusätzliche Abstraktionsschicht zu rechnen. Die Vorteile sind die unveränderte Größe der *Trusted Computing Base,* da der VMM für dieses Szenario nicht unbedingt Teil der TCB sein muss, und die Möglichkeit, den VMM nur bei Bedarf zu laden und anschließend wieder aus dem Speicher zu entfernen.

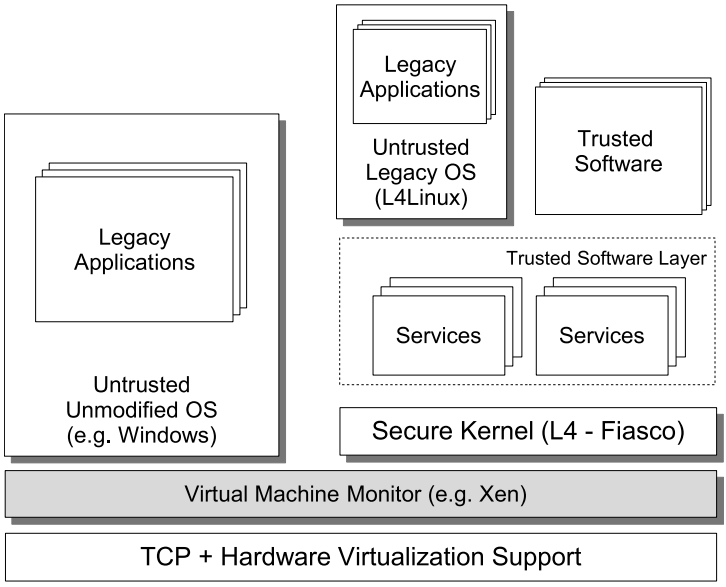

Abb. 8.16 Perseus-Architektur mit Xen II

Zum anderen kann der Xen-Hypervisor als Virtualisierungsschicht direkt über der Hardware des Systems positioniert werden. Der VMM teilt hierbei das System in zwei Bereiche: einen Bereich für unmodifizierte Gast-Betriebssysteme und einen Bereich für die ursprüngliche Perseus-Architektur. Der *Secure Kernel* übernimmt hierbei aus Sicht des Xen Hypervisors die Rolle der privilegierten Domäne-0 und enthält nach wie vor die Gerätetreiber und die Möglichkeit zur Steuerung des Systems. Der Vorteil dieser Anordnung ist die bessere Performance des unmodifizierten Betriebssystems und die Verwendbarkeit einer unmodifizierten Version des Xen-Hypervisors. Der Nachteil ist jedoch eine enorme Vergrößerung der TCB um die Codebasis von Xen. Eine Linderung dieses Problems ist durch die im nächsten Abschnitt beschrieben Erweiterungen für Xen jedoch bereits in Aussicht.

Da für dieses Szenario der *Hardware-Virtual-Machine*-Modus (HVM) von Xen zum Einsatz kommt, muss die *Trusted Computing Platform* über Hardware-Support für die Virtualisierung der CPU verfügen (z. B. Intel VT oder AMD SVM).

8.4.4 Xen-Hypervisor-Erweiterungen

Nachdem Xen vor allem im Bereich der Virtualisierung und Konsolidierung von Serversystemen unter Linux bekannt geworden war, machen die aktuellen Erweiterungen diese Technologie auch interessant für den Einsatz im *Trusted-Computing*-Umfeld. Besondere Relevanz haben hierbei die folgenden Entwicklungen:

- **HVM Support**
  Auf Plattformen mit aktuellen CPUs von Intel und AMD ist Xen ab der Version 3.0 nun auch in der Lage, unmodifizierte Betriebssysteme als Gäste auszuführen. Hardwarekomponenten ohne Unterstützung für Virtualisierung (z. B. Grafik- und Netzwerkkarten) werden mit Hilfe einer angepassten Version von QEMU[28] emuliert.

- **sHype**
  Im Rahmen des sHype-Projekts von IBM Research wurde ein *Mandatory Access Control System* für Xen entwickelt. Damit ist es möglich, die einzelnen Domänen der Xen-Architektur zusätzlich voneinander zu isolieren. Durch die Definition von Richtlinien (MAC Policies) kann die Kommunikation zwischen den Domänen und der Zugriff auf gemeinsamen Speicher kontrolliert werden. Die Ergebnisse der Arbeit sollen in den offiziellen Zweig des Xen-Sourcecodes einfließen. Details zur Implementierung liefern [Sailer2005] und [Sailer2005-2].

- **vTPM Manager**
  Ebenfalls im Rahmen des sHype-Projektes entstanden ist die Umsetzung eines virtual-TPM-Managers zur Bereitstellung von virtuellen TPM-Instanzen für die Gast-Betriebssysteme. Die Verwaltung und Sicherung der virtuellen TPMs erledigt der vTPM-Manager, der entweder innerhalb der Domäne-0 oder direkt oberhalb der Virtualisierungsschicht ausgeführt wird. Die Bereitstellung eines vTPM wird bisher nur für paravirtualisierte Gäste wie Linux möglich. An einer Erweiterung des QEMU-Gerätemodells zur Bereitstellung von vTPMs an HVM-Gäste wie Windows wird bereits gearbeitet. Mehr Details liefert [Berger2006].

- **Support für Intel TXT und AMD SEM**
  Die wohl bedeutendste Erweiterung für *Trusted Computing* ist die kürzlich angekündigte Unterstützung der *Trusted-Computing*-Funktionen der aktuellen CPU- und Chipsatz-Generation von Intel und AMD. Wie auf Seite 65 beschrieben, erlauben die Erweiterungen u. a. die Abschirmung einzelner Bereiche des Arbeitsspeichers vom Zugriff durch andere Prozesse sowie die Ausführung eines vertrauenswürdigen *Virtual-Machine-Monitors* (MVMM) innerhalb einer durch Hardware geschützten Umgebung. Neben einer stärkeren Isolation des VMM ermöglicht dies die Reduzierung der Größe der TCB, da nur noch die innerhalb der geschützten Umgebung ausgeführten Teile von Xen Teil der TCB sind. Eine erste Version eines Patches von Intel ist bereits im Entwickler-Zweig von Xen enthalten.

[28] http://www.qemu.org/

8.5 Trusted Graphical User Interface (Trusted-GUI)

8.5.1 Dynamic Security Skins

Ein interessanter Ansatz zur sicheren Interaktion zwischen Benutzern und dem Betriebssystem wird in [Dhamija2005] beschrieben. Dieses Konzept wurde eigentlich für den sicheren Zugriff auf Internetseiten entwickelt, um gängige Phishing-Angriffe zu verhindern, es lässt sich aber auch für die Implementierung eines Trusted-GUI verwenden. Wie bereits unter „Anforderungen an vertrauenswürdige Betriebssysteme“ angedeutet, haben die klassischen GUIs heutiger Betriebssysteme die folgenden Schwächen:

- Die Grafikschnittstelle erlaubt das Anzeigen beliebiger Grafiken und erlaubt dadurch die Imitation sämtlicher durch das Betriebssystem dargestellter Informationen.
- Benutzer können nicht zwischen einem Bild (Bitmap) eines Fensters und einem funktionalen Fenster einer Applikation unterscheiden. Ein Beispiel hierfür sind Werbebanner in Form von Windows-Dialogen auf Internetseiten.
- Die Zugehörigkeit eines Fensters zu einer Applikation ist nicht transparent für den Benutzer. Erscheint ein neues Fenster auf dem Desktop, ist nicht zu erkennen zu welcher Applikation es gehört.

Diese Schwächen können von schadhafter Software ausgenutzt werden, ohne das Betriebssystem zu modifizieren. Hierfür werden frei zugängliche Schnittstellen verwendet, daher verhaltenen sie sich wie reguläre Applikationen und können nur schwer durch Virenscanner erkannt werden.

Durch den Einsatz von *Dynamic Security Skins* sollen diese Probleme abgeschwächt werden. Die Idee hierbei ist die Einführung eines dem Benutzer und dem Betriebssystem bekannten Geheimnisses (shared secret). Dieses *shared secret* existiert in der Form einer visuellen Information (z. B. ein Foto, ein Logo oder ein Muster) und wird z. B. während der Installation des Betriebssystems vom Benutzer hinterlegt. Das Betriebssystem speichert diese Information verschlüsselt auf dem Datenträger ab. Diese Verschlüsselung wird bei der Anmeldung des Benutzers durch Eingabe seines Passworts rückgängig gemacht und im Speicherbereich des Betriebssystems hinterlegt. Jeder sicherheitskritische Dialog, der zur Laufzeit des Betriebssystems angezeigt wird, enthält dieses *shared secret*. Der Benutzer hat dadurch die Möglichkeit zu erkennen, ob der angezeigte Dialog Teil des Betriebssystems ist oder nicht.

Um zu verhindern, dass dieses *shared secret* mittels einer Bildschirmkopie (Screenshot) abgefangen und gefälscht wird, muss dafür gesorgt werden, dass der Zugriff auf eine solche Schnittstelle nur mit Systemrechten möglich ist. Eine Alternative hierzu ist, diesen Teil des Desktops im Screenshot zu schwärzen.

Hinweis: Dieses Verfahren ist nur erfolgreich, solange die Integrität des Betriebssystems nicht kompromittiert wurde. Ist dies der Fall, hat die schadhafte Soft-

ware in der Regel Zugriff auf den Speicherbereich des Betriebssystemkerns und somit auch auf das gemeinsame Geheimnis. Die Sicherstellung der Systemintegrität ist Aufgabe des restlichen Trusted-Computing-Systems.

8.5.2 Nitpicker – Overlay Window Management

Eines der beschriebenen Konzepte zur Umsetzung des *Protected-Execution*-Ansatzes ist das der Virtualisierung oder Partitionierung des Systems und der dadurch erzeugten Isolation der unterschiedlichen Sicherheitsdomänen. Die Ausgabe der graphischen Informationen dieser Domänen wurde bisher nicht berücksichtigt. Hierfür sind unterschiedliche Verfahren denkbar:

1. Das Host-Betriebssystem verwaltet den Zugriff auf die Grafikkarte des Systems. Gast-Systeme erhalten keinen Zugriff auf Grafikschnittstellen und sind somit nur über ein Netzwerkprotokoll (z. B. SSH) in Form einer Kommandozeile zu bedienen.
2. Das Host-Betriebssystem verwaltet den Zugriff auf die Grafikkarte des Systems. Gast-Systeme verwenden Remote-Desktop-Lösungen (z. B. Microsoft RDP oder VNC) durch die Installation entsprechender Server-Software. Die grafische Ausgabe der Gast-Systeme kann dann über ein Netzwerk auf entfernten Rechnern oder auf Host-Systems dargestellt werden.
3. Das Host-Betriebssystem wird ausschließlich über eine Kommandozeile gesteuert und die Verwaltung der Grafikkarte wird an eines der Gast-Systeme übergeben. Die restlichen Gast-Systeme verwenden 1. oder 2. als Kommunikationspfad[29].
4. Das Host-Betriebssystem verwaltet den Zugriff auf die Grafikkarte des Systems und emuliert für jeden seiner Gäste eine weitere Grafikkarte. Diese virtuellen Grafikkarten werden auf dem Host-System als virtuelle *Framebuffer*[30] repräsentiert. Jeder virtuelle *Framebuffer* erhält hierfür vom Grafiksubsystem des Host ein eigenes Fenster auf dessen Desktop[31].

Jedoch hat selbst der vierte Ansatz einen entscheidenden Nachteil: Sämtliche Grafikinformationen eines Gast-Systems werden in einem Fenster des Host-Systems angezeigt, somit auch der Desktop des Gast-Systems. Die Tatsache, dass Applikationen in unterschiedlichen Sicherheitsdomänen ausgeführt werden, sollte die Benutzbarkeit des Systems nicht beeinträchtigen. Hierfür sollte nur das jeweilige Applikationsfenster[32] auf dem Desktop dargestellt werden, nicht jedoch der Desktop dieser Domäne. Die folgende Abbildung veranschaulicht diese Problematik.

---

[29] Dieser Ansatz wird z. B. von Xen bei der Paravirtualisierung verwendet.

[30] Der physikalische Framebuffer befindet sich in der Regel direkt auf der Grafikhardware.

[31] Dies ist der übliche Ansatz moderner Virtualisierungsysteme wie z. B. VMware oder Parallels.

[32] Diese Technik wird häufig als Coherence-Modus bezeichnet.

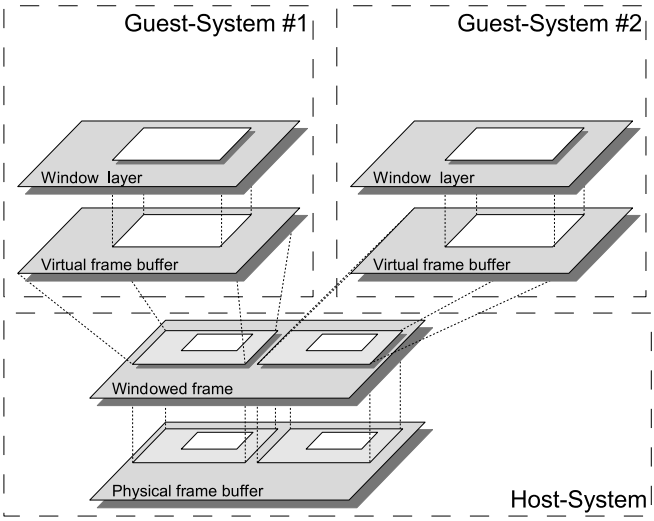

Abb. 8.17 Overlay Window Management

Ein Ansatz zur Lösung dieser Problematik beschreibt das an der TU Dresden entwickelte *Nitpicker* in [Feske2004] und [Feske2005]. *Nitpicker* entstand im Rahmen der Entwicklung der Nizza-Architektur. Eine weitere entscheidende Anforderung war hierbei, die unterschiedlichen Möglichkeiten zur Implementierung eines Fenstermanagers (X11, GEM, DopE und Windows) der Gast-Systeme zu unterstützen. Hierfür sollten keine oder nur minimale Modifikationen der Fenstermanager nötig sein. Das Problem des unter 4. beschriebenen Verfahrens ist der Verlust von semantischen Informationen über das vom Gast-System erzeugte Bild. Schreibt das Gast-System die Grafikausgabe in den bereitgestellten virtuellen Framebuffer, steht für das Host-System nur noch eine Bitmap-Grafik zur Verfügung, die Informationen über die darin enthaltenen Fenster (wie z. B. Lage und Größe) gehen hierbei verloren.

Um diese Metainformationen zu erhalten, muss zwischen dem Host-System und den Gast-Systemen ein weiterer Datenaustausch stattfinden. Im Falle der Implementierung für das X11-Window-System wurde hierfür ein spezieller Grafiktreiber und eine Erweiterung für den Fenstermanager entwickelt, der diese Informationen an das Host-System übergibt. Hierbei handelt es sich um einen bidirektionalen Kommunikationskanal, um Veränderungen des Fensters im Host-System (Änderung der Größe, Minimieren, Maximieren und Schließen) zurück an das Gast-System zu übertragen. Zusätzlich zu der Übertragung von Bildinformationen müssen auch die Eingabegeräte mit dem Gast-System verbunden werden. Dies wird ebenfalls in Form spezieller Maus- und Tastaturtreiber im Gast-System erreicht. Diese Informationen werden dann vom Fenstermanager des Host-Systems dazu verwendet die Fenster der Gast-Systeme auf lokale Fenster abzubilden. Die folgende Abbildung veranschaulicht dieses Konzept:

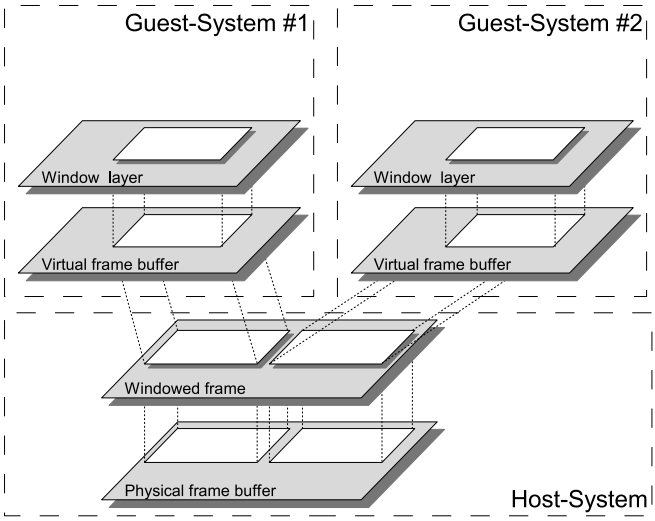

Abb. 8.18 Overlay Window Management #2

Für den Benutzer des Systems ist es nun irrelevant, in welcher Domäne seine Applikation gestartet wurde, da sich die Bedienung nicht von Fenstern des Host-Systems unterscheidet. Als zusätzliche Schutzfunktion befindet sich auf dem Desktop des Host-Systems eine Informationsleiste, die den Namen der zum aktuell aktiven Fenster zugehörigen Domäne anzeigt. Basierend auf dieser Information kann der Benutzer entscheiden, welche Daten er an diese Domäne übergibt.

Für die Umsetzung des Konzeptes verwendet *Nitpicker* u. a. das *Desktop Operating Environment* (DopE) des L4-Mikrokerns Fiasco, beschrieben in [Feske2003]. Weitere interessante Informationen zum Bau eines *Trusted-GUI* liefern [Shapiro2004] und [Yee2002].

Kapitel 9
Trusted-Computing-Systeme

Dieses Kapitel gibt einen Überblick über die aktuell verfügbaren Implementierungen eines *Trusted-Computing-Systems*. Es wird erwähnt, welche im vorhergehenden Kapitel beschriebenen Komponenten verwendet werden und inwiefern die Systeme bereits für den produktiven Einsatz geeignet sind.

9.1 European Multilaterally Secure Computing Base

Im Rahmen des Projekts *European Multilaterally Secure Computing Base* (EMSCB)[1] wird ein frei verfügbares *Trusted-Computing-System* auf Basis offener Standards entwickelt. Ein wichtiges Ziel des Projektes ist es, bereits bestehende Betriebssysteme und Applikationen weiterverwenden zu können um eine Migration in kleinen Schritten zu ermöglichen.

Für die Umsetzung des *Trusted-Computing-Systems* setzt EMSCB auf eine zentral gesteuerte und gezielte Weiterentwicklung der beiden vorgestellten Sicherheitsarchitekturen der TU Dresden und der Ruhr-Universität Bochum (*Nizza-Architektur und Perseus-Architektur*) sowie einiger weiterer in dieser Arbeit vorgestellter Projekte. Die bestehenden Technologien wurden unter dem neuen Produktnamen *Turaya* zusammengeführt. Neben den beiden genannten Hochschulen sind aktuell die folgenden Einrichtungen an der Weiterentwicklung im Rahmen des EMSCB-Projekts beteiligt:

- Fachhochschule Gelsenkirchen
- Sirrix AG security technologies, Saarbrücken
- escrypt GmbH, Bochum
- Infineon Technologies AG
- SAP AG, Walldorf
- Blaupunkt GmbH, Hildesheim

[1] http://www.emscb.de

Darüber hinaus wird das Projekt vom *Bundesministerium für Wirtschaft und Technologie*[2] finanziell gefördert. Eine weitere Aufgabe des EMSCB-Projektes ist es, das Interesse an *Trusted-Computing*-Konzepten zu wecken und mögliche Anwendungsfälle zu beschreiben. Hierfür wurden bereits einige Demo-Applikationen auf Basis der *Turaya-Plattform* entwickelt und auch als Demo-CD veröffentlicht. Hierzu zählen die Implementierung einer Festplattenverschlüsselungssoftware (Turaya.Crypt), ein VPN-Client (Turaya.VPN) und die Umsetzung einer Lösung für *Digital Rights Management* (Turaya.DRM). Ebenfalls im Rahmen des Projektes wurde eine erste Implementierung des *Trusted-Network-Connect* (TNC)-Standards vorgestellt. Auch wenn bisher noch keine Produkte auf Basis von *Turaya* erhältlich sind, hat die Plattform nach Meinung des Autors durchaus das Potential, auch im kommerziellen Umfeld in absehbarer Zeit zum Einsatz zu kommen.

9.2 Open Trusted Computing

Das zweite große Projekt auf diesem Gebiet ist die *Open Trusted Computing Initiative* (OpenTC). Im Rahmen des *OpenTC*-Projektes wird ebenfalls an der Entwicklung offener Standards und offener Software für *Trusted Computing* gearbeitet. Die Mehrzahl der am EMSCB-Projekt beteiligten Einrichtungen ist auch als Mitglied beim *OpenTC*-Projekt vertreten. Die Ergebnisse der beiden Projekte fließen daher auch meist in die Entwicklung des jeweils anderen Projekts ein. Eine genaue Beschreibung der Kooperation und der Ziele der beiden Projekte findet sich auf deren Internetseiten. Als Plattform für Technologie-Demos setzt auch das *OpenTC*-Projekt auf die *Turaya*-Plattform. Während bei EMSCB hauptsächlich Einrichtungen aus Deutschland zu den Mitgliedern zählen, sind bei *OpenTC* auch Firmen und Hochschulen aus dem europäischen Ausland vertreten. Unter anderem auch Branchengrößen wie IBM, Infineon, AMD und Hewlett-Packard. Im Rahmen des OpenTC-Projektes ist bisher noch kein Prototyp veröffentlicht worden, vielmehr ist anzunehmen, dass Entwicklungen direkt in die Turaya-Plattform von EMSCB integriert werden.

9.3 Intel Virtual Appliances/RedHat Embedded IT Software (EIT)

Unter dem Namen *„Virtual Appliances“* hat auch Intel ein Konzept für die Partitionierung eines PCs in unterschiedliche Sicherheitsdomänen auf der Roadmap. Auf Business-PCs auf Basis der vPro-Plattform sollen neben dem eigentlichen Betriebssystem zusätzliche virtuelle Maschinen laufen können, die unter anderem etwa den Netzwerkverkehr analysieren oder Fernwartungsfunktionen ermöglichen. Die-

[2] http://www.bmwi.de

se virtuellen Maschinen wären vom Betriebssystem aus unsichtbar und damit gegen feindliche Angriffe durch in das Betriebssystem eingeschleppte Trojaner oder gegen Fehlbedienungen ungeschickter Anwender immun. Die hierfür notwendige Virtualisierungsschicht bezeichnet Intel als *Lightweight Virtual Machine Monitor* (LVMM). Um die Integrität dieses VMM sicherzustellen setzt Intel auf die bereits beschriebene *Trusted-Execution-Technologie* (TXT) aus dem eigenen Hause.

Eine Umsetzung des Konzeptes wird in Kooperation mit der Firma Red Hat unter dem Namen *„Embedded Information Technology“* (EIT) entwickelt. Red Hat verwendet hierfür zum Großteil dieselben Lösungen wie die anderen bereits vorgestellten Projekte. Die im Dezember 2007 erstmals veröffentlichte Architektur des Systems beschreibt die folgende vereinfachte Darstellung:

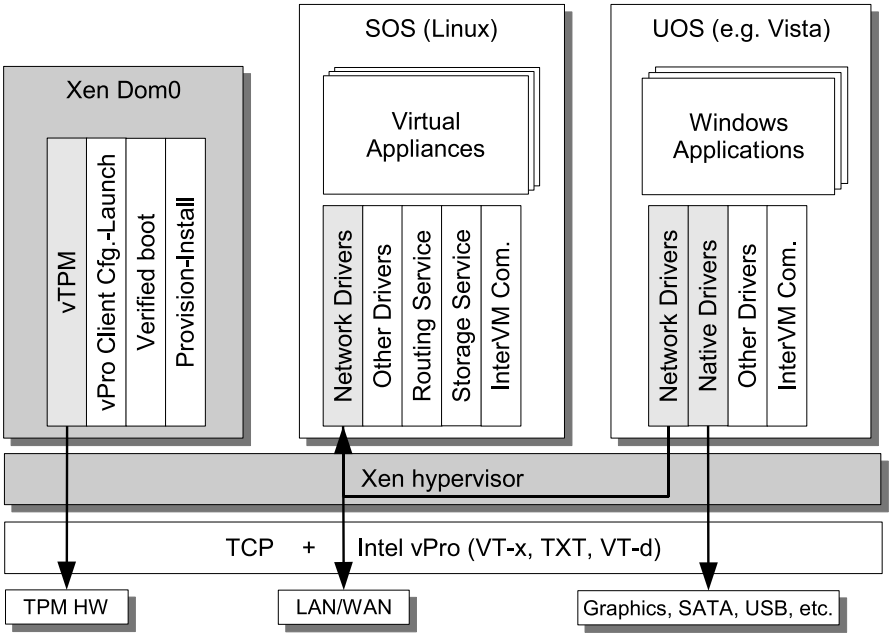

Abb. 9.1 Intel Virtual Appliances

Red Hat verwendet Xen als VMM, hierbei kommen bereits die auf Seite 120 erwähnten Erweiterungen zur Unterstützung der Intel-vPro-Technologie zum Einsatz. Sie ermöglicht das Starten des VMM im so genannten *Protected Execution Mode*[3]. Dieser stellt sicher, dass der vom VMM verwendete Speicherbereich vor Zugriffen durch die anderen Komponenten beschützt ist. Ebenfalls durch diese Technik geschützt ist die privilegierte Domäne-0. Sie dient in erster Linie der Steuerung und Überwachung des VMM und dem protokollierten Starten von weiteren virtuellen Maschinen (*Verified Boot*). Eine weitere Aufgabe der Domäne-0 ist die Verwaltung

[3] Siehe Seite 65

des Hardware-TPM und die Bereitstellung von virtuellen TPMs. Der VMM und die Domäne-0 bilden zusammen den *Secure Kernel* der Architektur und somit die *Trusted Computing Base* (TCB).

Neben der Domäne-0 besteht das System aus zwei weiteren virtuellen Maschinen, dem *Service-OS* (SOS) und dem *User-OS* (UOS). Das SOS ist identisch zur Domäne-0 ein an Xen angepasstes Linux und bildet die Laufzeitumgebung für die *Virtual Appliances*. Das SOS erhält als einzige VM direkten Zugriff auf die Netzwerkschnittstellen des PCs. Alle anderen VMs greifen über virtuelle Netzwerkschnittstellen auf die Netzwerke zu. Somit läuft sämtlicher Netzwerkverkehr über das SOS und erlaubt dadurch die Implementierung einer Firewall oder eines Host-IDS[4] außerhalb des vom Benutzer verwendeten Betriebssystems. Das UOS ist ein unmodifiziertes Betriebssystem (z. B. Windows XP oder Windows Vista). Durch die in modernen Prozessoren vorhandene Unterstützung für Hardware-Virtualisierung (z. B. Intel VT-x) kann nahezu jedes Betriebssystem ohne Modifikation auf der EIT-Plattform ausgeführt werden. Auch kann ein Großteil der vorhandenen Treiber des Betriebssystems wiederverwendet werden, da der VMM dem UOS direkten Zugriff auf Hardwareressourcen wie Grafikkarte, Festplatten und USB-Controller gestattet.

Der Aspekt eines *Trusted GUI* wird von EIT dabei jedoch nicht adressiert. Das UOS erhält, nach Abschluss des Startvorgangs, exklusiven Zugriff auf die Grafikhardware des Systems. Die Domäne-0 sowie das SOS incl. der darauf laufenden *Virtual Appliances* haben somit keine Möglichkeit der grafischen Ausgabe. Unklar ist auch, wie und ob der Benutzer bzw. das UOS mit den Appliances kommunizieren kann. Denkbar wäre hierbei eine reine Remote-Management-Lösung, bei der nur über die Netzwerkschnittstelle auf das SOS zugegriffen werden kann. Ebenfalls nicht erwähnt ist der Aspekt der *Remote Attestation*; so werden während des *Measured Launch* unter Einsatz von Intel TET auch Prüfsummen über den VMM und die Domäne-0 erzeugt und im TPM hinterlegt. Wie und ob diese Prüfsummen zu einem späteren Zeitpunkt wieder ausgewertet werden, ist nicht bekannt. Eine Option hierfür ist sicherlich die Entwicklung eines TNC-Client als *Virtual Appliance* und der Übermittlung der Prüfsummen bei der Authentisierung im einem Firmennetzwerk.

Für Entwickler von *Virtual Appliances* haben Intel und Red Hat die Verfügbarkeit eines SDK im Laufe des Jahres 2008 angekündigt. Ein öffentlicher Beta-Release der EIT steht bereits auf den Internetseiten von RedHat zur Verfügung[5]. Mit Symantec will Intel außerdem bereits einen ersten Anbieter für *Virtual Appliances* gewonnen haben. Unter dem Namen *Virtual Security Solution* (VSS) will Symantec künftig eine Sicherheitslösung auf der Basis von EIT anbieten.

Die Lösung von Red Hat erinnert nicht nur stark an den Aufbau der *Perseus-Architektur* mit Xen, sondern teilweise auch an die von Microsoft unter dem Namen NGSCB[6] vorgestellten Konzepte zur Unterteilung des Betriebssystems in zwei Sicherheitsdomänen. Auch wenn EIT aktuell die besten Chancen auf einen Einsatz im produktiven Umfeld hat, muss sich zeigen, ob die Komplexität der Architektur nicht zu viele Einbußen auf Seiten der Benutzbarkeit mit sich bringt.

[4] Intrusion Detection System

[5] http://eit.et.redhat.com/

[6] Next Generation Secure Computing Base

Kapitel 10
Fazit

Für viele der am Anfang des Buches identifizierten Ziele des *Trusted Computing* existieren bereits praktische Lösungsansätze. Am weitesten fortgeschritten ist hierbei definitiv der Teilbereich der durch die TCG spezifizierten Umsetzung einer *Trusted Computing Platform*. Nahezu alle aktuell verkauften Business-Notebooks verfügen über ein TPM und die entsprechenden Erweiterungen des BIOS. Durch die in immer mehr Rechner eingebauten erweiterten Sicherheitsfunktionen im Chipsatz und der CPU werden in naher Zukunft größtenteils vollständige *Trusted-Computing*-Plattformen in Form von Standard-PCs zur Verfügung stehen.

Mindestens ebenso wichtig für die Umsetzung der Konzepte des *Trusted Computing* wie die Verfügbarkeit von *Trusted-Computing*-Plattformen sind Betriebssysteme, die von den gebotenen Funktionen Gebrauch machen. In diesem Bereich befindet sich *Trusted Computing* nach wie vor in einem frühen Entwicklungsstadium. Zwar ist der Zugriff auf das TPM über einen *Trusted Software Stack* bereits unter allen gängigen Desktop-Betriebssystemen möglich, jedoch werden die Funktionen des TPM in erster Linie von Anwendungsprogrammen aus dem Bereich der Datenverschlüsselung oder von Programmen zur Verwaltung von Passwörtern verwendet. Das TPM erfüllt hierbei dieselben Aufgaben wie eine zu diesem Zweck eingesetzte *SmartCard*, mit *Trusted Computing* und dessen Konzepten hat dies jedoch wenig zu tun.

Bei der Umsetzung der Kernkonzepte des *Trusted Computing* ist der Bereich der *Protected Execution* und hier vor allem die *Turaya*-Plattform hervorzuheben. Sie bündelt einen Großteil der aktuell verfügbaren Technologien zu einer Plattform für weiterführende Konzepte bzw. deren Prototypen. Sie unterstreicht auch den aktuellen Trend der Verschmelzung von *Trusted Computing* und der Virtualisierungstechnologie und der daraus entstehenden Möglichkeit zur Partitionierung eines Systems in unterschiedliche Sicherheitsdomänen. Noch einen Schritt weiter in Richtung Isolation von Sicherheitsdomänen gehen Intel und Red Hat mit EIT unter Einsatz von Intels vPro. Zusätzlichen Auftrieb erhält dieser Trend durch die immer bessere Unterstützung von Virtualisierung direkt in der Hardware der Systeme. Der nächste bereits angekündigte Schritt ist hierbei die Virtualisierung der Eingabe- und Ausgabekanäle des Systems (IOMMU) sowie die Partitionierung der Ressourcen von weiteren Systemkomponenten (wie z. B. Netzwerkkarte und Grafikkarte).

Nach Meinung des Autors ist der produktive Einsatz von *Trusted-Computing-Systemen* wie *Turaya* und EIT bis in 1–2 Jahren durchaus denkbar, wenn auch zunächst eher im Firmen- und Embedded-Umfeld. Auch auf dem Gebiet der kommerziellen Betriebssystem-Entwicklung ist mit entsprechenden Produkten zu rechnen. Eine Umsetzung der von Microsoft mit NGSCB vorgestellten Partitionierung des Systems in einen normalen und einen vertrauenswürdigen Bereich ist für die nächste oder spätestens übernächste Generation von Betriebssystemen aus dem Hause Microsoft nicht grundsätzlich auszuschließen.

Weniger ausgereift sind die Bereiche *Integrity Measurement* und *Remote Attestation*. Während die Erzeugung von Prüfsummen beim Startvorgang durch Implementierungen wie *Trusted-Grub* und zu Teilen auch mit *Secure Startup*[1] in Windows Vista sinnvoll umgesetzt wurde, steht die Erzeugung von Prüfsummen zur Laufzeit aktuell noch vor einigen ungelösten Problemen. Eine Bewertung des Systemzustands auf Basis von Prüfsummen über alle auf dem System ausgeführten Applikationen (*Binary Attestation*) ist in der Praxis nicht anwendbar. Da erscheinen Ansätze wie „*WS-Attestation*“ zur Kombination der Bewertung auf Basis von Prüfsummen sowie auf Basis der Eigenschaften des Systems (*Semantic-* oder *Property-based Attestation*) deutlich praktikabler. Auch eine Kombination von Agentensystemen wie Microsofts NAP oder Ciscos NAC mit einer *Binary Attestation* ist denkbar.

Während sich im Firmen-Umfeld durchaus Szenarien zum sinnvollen Einsatz von *Remote Attestation* identifizieren lassen, ist ein erfolgreicher Einsatz dieses Konzeptes im privaten Bereich sehr fraglich. Zum einen ist der Aufbau der dafür notwendigen *Public-Key-Infrastruktur* ein komplexes und langwieriges Unterfangen, zum anderen dürfte eine derartige Überprüfung von privaten PCs auf wenig Akzeptanz auf Seiten der Nutzer treffen. Die Einführung einer weiteren *Trusted Third Party*, wie z. B. die mit „*WS-Attestation*“ beschriebenen *Validation Services*, könnte sich hierbei positiv auf die Akzeptanz auswirken. Den ersten Schritt beim Aufbau der nötigen Infrastruktur müssen zunächst die Hersteller von TPMs und *Trusted-Computing-Plattformen* machen, indem sie die beschriebenen Plattform-Zertifikate für die Endnutzer zugänglich machen.

Trusted Computing hat trotz einiger Baustellen das Potential zur Verbesserung des Sicherheitsniveaus heutiger Computersysteme. Viel versprechend sind hierbei u. a. die Bestrebungen von Firmen wie Intel und AMD, immer mehr Sicherheitsfunktionen direkt in der Hardware der Systeme zu verankern. Wirklich sicherer werden Systeme jedoch erst durch die Unterstützung solcher Funktionen durch die Betriebssysteme werden. Dies erfordert teilweise tiefe Eingriffe in die Architektur und Funktionsweise der Betriebssysteme; dies mit der von den Kunden geforderten Abwärtskompatibilität zu älteren Versionen desselben Systems zu vereinbaren, stellt die Hersteller vor große Probleme. Daher ist nicht mit dem Einzug sämtlicher Konzepte innerhalb einer Generation eines Betriebssystems zu rechnen. Auch werden die Umsetzungen immer ein Kompromiss zwischen dem erreichbaren Sicherheitsniveau und der Benutzbarkeit des Systems sein. Eine Beschreibung der Bestrebungen, Microsofts *Trusted-Computing*-Konzepte in Windows Vista umzusetzen, liefert das folgende Kapitel.

[1] Secure Startup wird auf Seite 137 beschrieben.

Kapitel 11
Trusted Computing mit Windows Vista

Am 09.11.2006 veröffentlichte Microsoft im Rahmen seines *Customer Preview Program* (CPP) die *Release-to-Manufacturing* (RTM)-Version des neuen Betriebssystems Windows Vista™. Windows Vista ist der Nachfolger von Windows XP™ und steht in einer 32-Bit- und 64-Bit-Version zur Verfügung. Auch die Server-Variante Windows Server® 2008 wird noch im Laufe des Jahres 2008 auf den Markt kommen. Vor allem im Bereich Sicherheit will Microsoft mit den beiden Betriebssystemen einen neuen Maßstab setzen. Speziell der Vorgänger Windows XP ist bekannt für seine Anfälligkeit für Manipulationen durch Schadsoftware (Viren, Würmer, Trojanische Pferde und Rootkits). Als Hauptkritikpunkte am Sicherheitsdesign von Windows XP gelten die Rechte- und Benutzerkonten-Verwaltung sowie die einfache Manipulation des Betriebssystemkerns durch Nachladen von schadhaften Systemtreibern. Microsoft hat beide Betriebssysteme mit zahlreichen neuen Sicherheitsfunktionen ausgestattet und viele der bisherigen überarbeitet. Einige der neuen Funktionen bleiben jedoch den beiden 64-Bit-Varianten der Betriebssysteme vorbehalten. Die Entwicklung der Desktop- und der Server-Variante lief größtenteils parallel; sie basieren weitestgehend auf derselben Codebasis und enthalten somit dieselben Sicherheitsfunktionen. Für den restlichen Teil des Kapitels wird stellvertretend für beide Betriebssysteme nur der Begriff Windows Vista verwendet.

Nach einem Überblick über die Geschichte der Entwicklung von Windows Vista wird der unter „Anforderungen an vertrauenswürdige Betriebssysteme“ definierte Anforderungskatalog für ein *Trusted-OS* mit der Umsetzung im Betriebssystem Windows Vista verglichen. Hierfür werden die für die Untersuchung relevanten Sicherheitsfunktionen kurz beschrieben und falls möglich einem der Bereiche des Anforderungskatalogs zugeordnet. Dem folgt eine Analyse, welche Teile des Anforderungskatalogs abgedeckt werden und welche Angriffe gegen die Sicherheitsfunktion denkbar bzw. verfügbar sind. Am Ende der Untersuchung werden die nicht adressierten Anforderungen aufgezeigt und die daraus resultierenden Angriffsvektoren beschrieben.

11.1 Die Geschichte von Windows Vista

Mit Palladium, das im Jahre 2003 in *Next Generation Secure Conputing Base* (NGSCB) umgetauft wurde, hatte Microsoft sich das ehrgeizige Ziel gesetzt ein *Trusted Operating System* zu entwickeln. Als eines der zentralen Konzepte von NGSCB wurde die Unterteilung des Betriebssystems in einen vertrauenswürdigen Bereich für sicherheitskritische Anwendungen und einen Bereich für nicht vertrauenswürdige Software vorgestellt. Die im Rahmen der NGSCB entwickelten Konzepte sollten in Windows Vista einfließen, welches zu diesem Zeitpunkt noch unter dem Namen „Longhorn" bekannt war. Die von Microsoft präsentierte Architektur entsprach weitestgehend der nachfolgenden Abbildung:

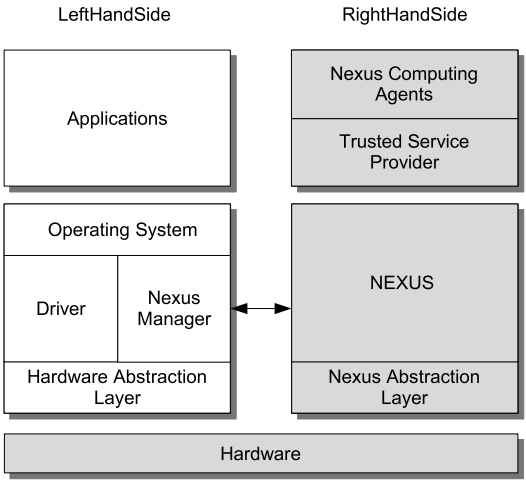

Abb. 11.1 NGSCB

So sollte das Betriebssystem neben dem normalen Kernel (LeftHandSide) zur Ausführung von Windows-Anwendungen und zur Sicherstellung der Abwärtskompatibilität einen zusätzlichen *Secure Kernel* besitzen (RightHandSide). Der als *NEXUS* bezeichnete Kernel sollte die Laufzeit für sicherheitskritische Applikationen bereitstellen. *NEXUS* sollte dazu vollständig vom normalen Betriebssystem isoliert ausgeführt werden. Als die einzige Möglichkeit zur Kommunikation zwischen den beiden Kernen war der *NEXUS-Manager* vorgesehen. Auch der Einsatz eines TPM zur Protokollierung des Systemstarts durch die Erzeugung von Prüfsummen war geplant. Interessant war auch die Anforderung den *NEXUS*-Kernel nur bei Bedarf und während der Laufzeit der Betriebssystems nachzuladen. Diese sollte jedoch keinen negativen Einfluss auf die Integrität der *NEXUS*-Umgebung haben. Für die Umsetzung der Konzepte wollte Microsoft neben dem TPM auch die damals noch nicht verfügbare *Trusted Execution Extention* (TXT) von Intel verwenden. Zum Zeitpunkt der Vorstellung von der NGSCB war diese Erweiterung noch unter dem Namen Intel *LaGrande* bekannt.

Was genau die Gründe dafür waren, dass Microsoft einen Großteil dieser Pläne für Windows Vista wieder verworfen hat, ist nicht bekannt. Denkbar ist jedoch, dass speziell die für das sichere Nachladen des *NEXUS*-Kernel notwendige Intel TET zum geplanten Releasetermin von Vista nicht oder nur auf sehr wenigen Plattformen verfügbar gewesen sein könnte. Andere Quellen berichten, dass Microsoft an der Implementierung des Kommunikationskanals zwischen den beiden Bereichen gescheitert sei.

11.2 Sicherheitsfunktionen in Windows Vista

Windows Vista erhielt von Microsoft eine ganze Reihe neuer und verbesserter Sicherheitsfunktionen. Diese Funktionen lassen sich grob in die drei Bereiche *Kernel-Integrität*, *System-Integrität* und *Funktionen zum Schutz vor Exploits* einteilen. Die folgende Abbildung veranschaulicht diese Einteilung:

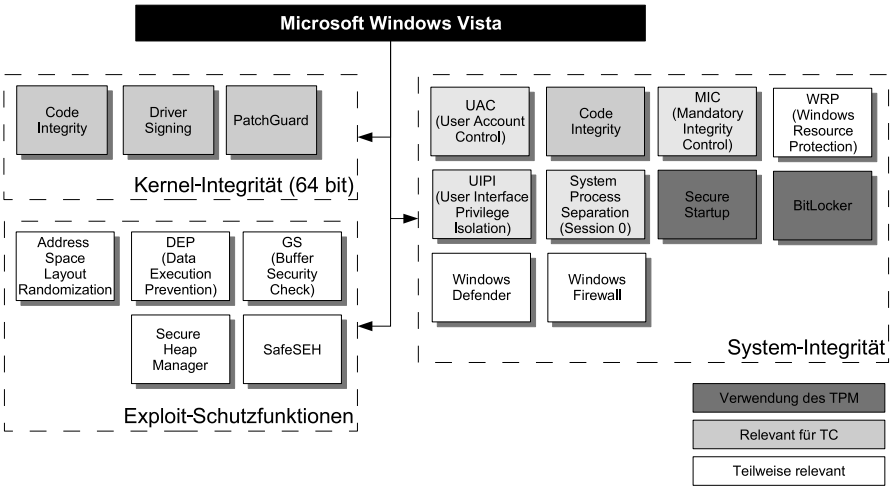

Abb. 11.2 Sicherheitsfunktionen in Windows Vista

Nicht alle Schutzfunktionen lassen sich direkt mit *Trusted-Computing*-Konzepten in Verbindung bringen. Relevant für die folgende Betrachtung von Windows Vista sind jedoch in erster Linie Funktionen mit direktem Bezug zu einem oder mehreren Konzepten und Funktionen mit Verwendung für das TPM (grau). Einige Funktionen wie z. B. die *User Account Control* haben keinen direkten Bezug, sind jedoch von zentraler Bedeutung für die Sicherheit des Betriebssystems und finden deshalb ebenfalls Erwähnung (weiß/grau).

11.3 Windows Vista TPM Support

Als erste Windows-Version enthält Vista Unterstützung für *Trusted-Platform-Module* ohne zusätzliche Treiber des TPM-Herstellers. Das TPM kann unter Vista mit Hilfe eines Assistenten verwaltet werden. Er bietet die Möglichkeit das TPM vom Betriebszustand *Deactivated* in den Zustand *Activated* zu überführen. Ist es jedoch im Zustand *Disabled*, kann es nur im BIOS aktiviert werden. Da für den Zustandswechsel von *Deactivated* nach *Activated* und für das Einrichten des TPM-Eigentümers der Beweis des physikalischen Zugriffs auf das System erbracht werden muss, informiert Vista das BIOS über die gewünschte Befehlsausführung. Das BIOS präsentiert dem Benutzer daraufhin beim nächsten Start des Systems einen Dialog, dessen Bestätigung den Beweis des physikalischen Zugriffs darstellt und den Zugriff auf den Befehl freigibt. Befindet sich das TPM im Zustand *Activated*, konfiguriert der Assistent das TPM für die spätere Verwendung durch die *BitLocker*-Festplattenverschlüsselung[1], indem der Assistent den Besitz des TPM übernimmt (*TPM_TakeOwnership*). Um das TPM im Folgenden vor unberechtigtem Zugriff zu schützen, erfordert der Zugriff auf sicherheitskritische Funktionen des TPM die Angabe des Eigentümer-Passworts, welches automatisch durch den Assistenten erzeugt werden kann. Dieses Kennwort wird bei entsprechender Konfiguration der Gruppenrichtlinien im *Active Directory* der Windows-Domäne abgelegt. Auf die Möglichkeit, den Zugriff auf den im TPM erzeugten *Storage Root Key* (SRK) ebenfalls durch ein Passwort zu schützen, hat Microsoft verzichtet, was in erster Linie ein Zugeständnis an die Benutzerfreundlichkeit ist. Wäre dieses Passwort gesetzt, müsste der Benutzer bei Verwendung des SRK das Passwort angeben, im Falle einer aktivierten *BitLocker*-Festplattenverschlüsselung somit bei jedem Systemstart.

Bei der Implementierung des *Trusted Software Stack* (TSS) für den Zugriff auf das TPM hat sich Microsoft für eine von der TCG-Spezifikation[2] abweichende Umsetzung entschieden. Der Zugriff auf die Funktionen des TPM erfolgt entweder über eine WMI-Schnittstelle[3] oder über die Microsoft Krypto-API. Die folgende Abbildung beschreibt das Zusammenspiel der einzelnen Schnittstellen:

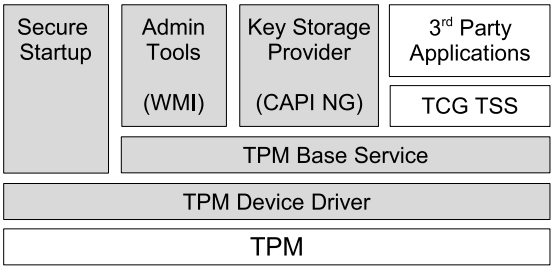

Abb. 11.3 TPM-Dienste in Windows Vista

[1] Beschrieben im Abschnitt „Secure Startup und Full Volume Encryption“.

[2] TCG-TSS Spezifikation [TCG2006-2]

[3] Windows Management Instrumentation

Die beiden Schnittstellen bilden jedoch nur eine Teilmenge der verfügbaren Funktionen des TPM ab. Der TPM Base Service verwaltet hierbei den simultanen Zugriff auf das TPM und bietet eine Schnittstelle an, auf die vollständige TSS-Implementierungen aufsetzen können, um somit den vollen Funktionsumfang des TPM anzubieten. Der TPM Base Service bietet weiterhin die Möglichkeit, die Verwendung bestimmter TPM-Befehle unter Zuhilfenahme einer Sperrliste zu blockieren. Die Sperrliste kann sowohl über die lokale Windows-Sicherheitsrichtlinie als auch zentral über Gruppenrichtlinien erstellt und verteilt werden. Es können lokal keine Kommandos freigegeben werden, die per Gruppenrichtlinie gesperrt sind. Es ist jedoch möglich, zusätzliche Kommandos über die lokale Sicherheitsrichtlinie zu sperren. Derselbe Weg in die andere Richtung ist jedoch gangbar; Microsoft hat eine zusätzliche Option in den Gruppenrichtlinien vorgesehen, die verhindert, dass Kommandos gesperrt werden, die in den Gruppenrichtlinien als aktiv markiert sind.

11.4 Secure Startup und Full Volume Encryption (FVE) – BitLocker

Die *BitLocker*-Funktion in Windows Vista besteht aus zwei Komponenten, dem *Secure Startup* und der *Full Volume Encryption* (FVE). Notwendige Voraussetzung für die Nutzung dieser Komponenten ist die Verfügbarkeit eines TPM[4]. Auf einem System mit aktiviertem TPM[5] dient der *Secure Startup* der Fortsetzung der *Vertrauenskette* (*Dynamic Chain of Trust*). Hierfür erweitert Microsoft den MBR, die letzte von der *Trusted Computing Platform* protokollierte Komponente, um die Funktionalität, die nächste Stufe des Startvorgangs, den NTFS-Bootsektor (VBR), zu protokollieren. Die dabei erzeugte Prüfsumme wird im PCR-8 des TPM hinterlegt. Der NTFS-Bootsektor wiederum speichert eine Prüfsumme über den NTFS Boot Block (BPB) im PCR-9. Zuletzt protokolliert der NTFS Boot Block den Bootmanager und speichert dessen Wert im PCR-10. Dieser Vorgang wird bei jedem Startvorgang wiederholt. Die nachfolgende Abbildung beschreibt diesen Ablauf:

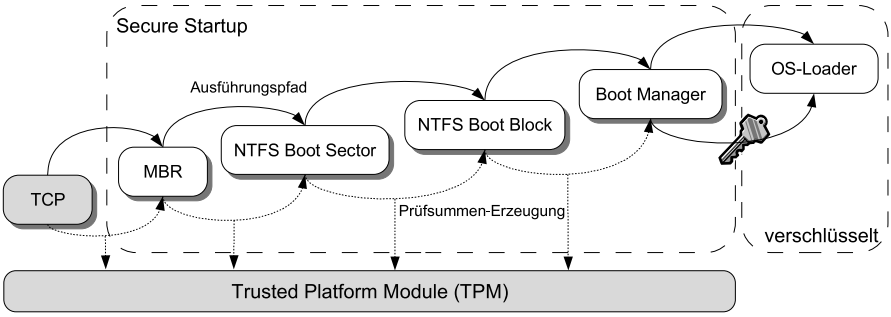

Abb. 11.4 Secure Startup

[4] Die FVE kann mit Einschränkungen auch ohne TPM verwendet werden.

[5] Windows Vista unterstützt ausschließlich TPM Chips in der Version 1.2.

Die *Full Volume Encryption* bietet, zusätzlich zur bereits vorhandenen Verschlüsselung auf Dateiebene (*Encrypted File System* – EFS), die Möglichkeit, die komplette Systempartition zu verschlüsseln. In der Server-Edition 2008 soll die FVE zusätzlich auch für weitere Datenpartitionen aktiviert werden können. Diese Verschlüsselung ist im I/O-Stack des Betriebssystems als Filter-Treiber unterhalb des Dateisystems implementiert und findet dadurch für den Benutzer transparent statt. Hierdurch werden neben Daten der Benutzer auch sämtliche System-Dateien, temporäre Dateien, die Auslagerungsdatei und die Dateien des Ruhezustands (*Hibernation-Mode*) verschlüsselt. Um die Partition beim Startvorgang entschlüsseln zu können, muss eine weitere NTFS-Partition angelegt werden. Sie enthält neben dem MBR den *Bootmanager*, der die Entschlüsselung startet und danach den *OS-Loader* (*%Windows%\system32\WINLOAD.EXE*) aufruft. Als Verschlüsselungsalgorithmus kommt eine abgewandelte[6] Version von AES zum Einsatz, wahlweise mit 128 Bit oder 256 Bit Schlüssellänge. Der Schlüssel wird als *Full Volume Encryption Key* (FVEK) bezeichnet und ist nochmals mittels AES-256 durch den *Volume Master Key* (VMK) gesichert. Dieses zweistufige Konzept hat den Vorteil, dass bei einer notwendigen Änderung des Schlüssels nicht die komplette Partition ent- und anschließend wieder verschlüsselt werden muss. Sowohl der FVEK als auch der VMK werden in einem Bereich der Partition hinterlegt, auf die der Bootmanager Zugriff hat. Die genaue Lage der Schlüssel wird von Microsoft nicht dokumentiert, ein möglicher Speicherort ist der NTFS-Header der Systempartition.

Für den Schutz des VMK gibt es mehrere Optionen, neben der Möglichkeit des Schutzes durch einen externen Schlüssel auf einem USB-Stick kann auch das TPM verwendet werden. Hierzu kommt die *sealing*-Funktion des TPM zum Einsatz, sie ermöglicht es, beliebige Daten an die aktuelle Konfiguration des Systems zu binden. Dabei fließen die Inhalte der PCR in den Verschlüsselungsvorgang mit ein. Die aktuelle Konfiguration wird durch die von der *Trusted Computing Platform* und der *Secure-Startup*-Komponente hinterlegten Prüfsummen in den Registern PCR-0 bis PCR-11 repräsentiert. Zusätzlich lässt sich der VMK noch durch ein numerisches Kennwort (PIN) oder einen zusätzlichen externen Schlüssel schützen. In der Standardkonfiguration verwendet die FVE nur ein Subset dieser Prüfsummen PCR-(0,2,4,8–11) und verzichtet auf den zusätzlichen Schutz durch eine PIN oder weitere Schlüssel auf einem USB-Stick[7]. Die Abb. 11.5 veranschaulicht die unterschiedlichen Optionen bei der Verschlüsselung des VMK.

Beim Systemstart verwendet der Bootmanager die *unseal*-Funktion des TPM, um den VMK zu entschlüsseln. Dies gelingt jedoch nur dann, wenn die verwendeten PCR denselben Zustand wie zum Zeitpunkt der Aktivierung von *BitLocker* aufweisen. Modifikationen an Systemkomponenten wie z. B. dem MBR oder dem BIOS führen somit zu einer Unterbrechung des Startvorgangs, welche nur durch die Eingabe des Wiederherstellungskennworts, das bei der Aktivierung von BitLocker erzeugt wird, fortgesetzt werden kann. Das PCR-11, das ebenfalls beim *sealing* und *unsealing* verwendet wird, hat dahingehend eine Sonderstellung, dass Micro-

[6] Microsoft beschreibt dies als AES + Elephant Diffusor [Micro01].

[7] Die Optionen (TPM+PIN,TPM+Key) sind erst nach Aktivierung in den Gruppenrichtlinien verfügbar.

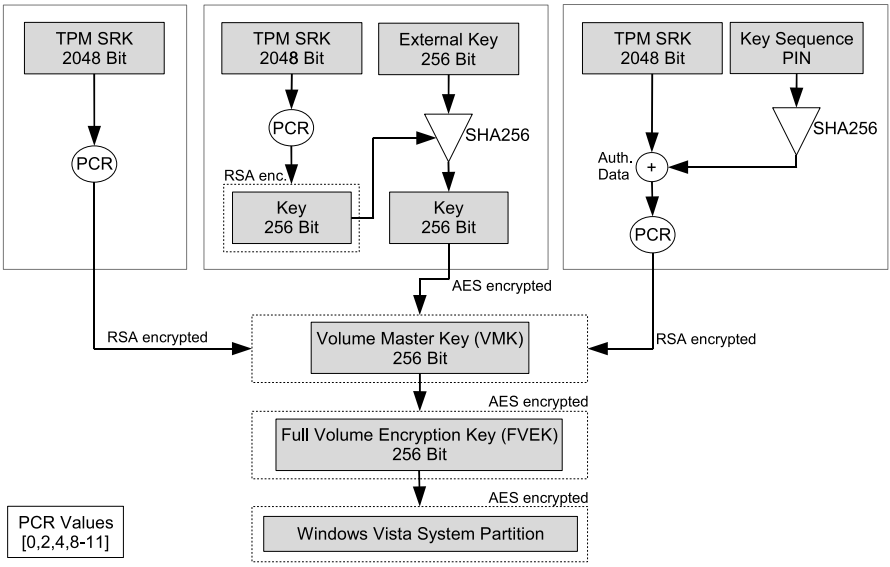

Abb. 11.5 BitLocker-Schlüsselverwaltung

soft dieses Register als „BitLocker Access Control“ beschreibt und im Falle einer aktiven FVE nach der erfolgreichen Entschlüsselung des VMK auf einen anderen Wert abändert. Dadurch soll verhindert werden, dass die Entschlüsselung zur Laufzeit des Betriebssystems wiederholt werden kann und dass ein parallel installiertes Betriebssystem Zugriff auf den VMK erhält.

Während die *Secure-Startup*-Komponente eindeutig der *Integrity Measurement* zuzuordnen ist – sie dient der Fortsetzung der *Vertrauenskette* (*Chain of Trust*) – fällt die Einordnung der Festplattenverschlüsselung etwas schwer. Ist sie aktiviert, werden zwar Modifikationen an den am Systemstart beteiligten Komponenten erkannt, es gibt jedoch keine Möglichkeit die geänderte Komponente zu identifizieren, und somit fällt es schwer, geeignete Maßnahmen zu ergreifen. Das entscheidende Problem besteht jedoch in der Tatsache, dass auch eine erfolgreiche Entschlüsselung des VMK keinen integeren Systemzustand garantiert, da hierbei lediglich sichergestellt wird, dass sich das System im selben Zustand wie bei der Aktivierung von *BitLocker* befindet. War das System jedoch bereits zu diesem Zeitpunkt kompromittiert, gibt es keine Möglichkeit, dies beim Einsatz der *Full Volume Encryption* (FVE) zu erkennen. Da die FVE neben ihrer eigentlichen Funktion, der Geheimhaltung der Daten, auch verhindert, dass Dateien des Betriebssystems durch ein parallel installiertes Betriebssystem modifiziert werden können, und somit ebenfalls zur Erhaltung der Systemintegrität beiträgt, kann jedoch auch unter dem Gesichtspunkt des Integritätsschutzes eine Aktivierung der Funktion empfohlen werden.

Die *Secure-Startup*-Komponente setzt die durch die *Trusted Computing Platform* begonnene *Vertrauenskette* sinnvoll bis zum Bootmanager von Windows Vista fort. Jedoch lassen sich auch hier zwei Kritikpunkte finden. Zum einen handelt

es sich bei *Secure Startup* um keine Implementierung des *Secure-Boot*-Konzeptes, da *Secure Startup* selbst keine Integritätsprüfungen anhand der PCR vornimmt und die Integritätsprüfung der FVE auf Grund des beschriebenen Umstands ebenfalls den Anforderungen nicht gerecht wird. Somit handelt es sich nur um eine Implementierung des *Trusted-Boot*-Konzeptes. Zum anderen wird die Vertrauenskette nur bis zum Bootmanager erweitert; die beiden nächsten Stufen, der *OS-Loader*(*%SystemRoot%\system32\WINLOAD.EXE*) und der Betriebssystemkern (*%SystemRoot%\system32\NTOSKRNL.EXE*), werden hierfür nicht mehr berücksichtigt. Zwar unterliegen diese beiden Komponenten einer weiteren Integritätsprüfung (*Kernel Integrity Checks*), jedoch hätte dadurch eine Schwachstelle, die später im Dokument beschrieben wird, verhindert werden können.

Weiter ist es zu empfehlen, im Falle der *Full Volume Encryption* den zusätzlichen Schutz mittels PIN zu aktivieren, da die alleinige Verwendung des TPM einen entscheidenden Nachteil hat. Wird ein PC mit dieser Konfiguration gestohlen, ist es für den Angreifer zwar nicht möglich, auf die Systemdateien zuzugreifen, in denen die Benutzerpasswörter hinterlegt sind, was einen Brute-Force-Angriff auf die Passwörter verhindert. Da die PCR jedoch nach wie vor die richtigen Werte enthalten, entschlüsselt das TPM auf Anfrage des Bootmanagers den Volume Master Key, welcher diesen in den Arbeitsspeicher lädt. Mit speziellen PCI-Erweiterungskarten kann nun der Inhalt des Arbeitsspeichers und somit auch der VMK durch einen zweiten PC ausgelesen werden. Solche DMA-Karten werden u. a. in der Computer-Forensik eingesetzt. Weitere Informationen hierzu liefern [Petroni2004] und [Carr2004].

11.5 Kernel Integrity Checks/Driver Signing (nur 64-Bit-Versionen)

Eine weitere Maßnahme zur Sicherung der Systemintegrität ist die Überprüfung von Treiber-Signaturen. Hierfür müssen in den 64-Bit-Versionen der Betriebssysteme alle Treiber mit einer digitalen Signatur versehen sein [Micro05]. In den 32-Bit-Versionen ist eine Signatur nur für die am Startvorgang beteiligten Treiber zwingend vorgeschrieben. Die Überprüfung der Treiber-Signaturen findet zu mehreren Zeitpunkten im Lebenszyklus des Betriebssystems statt.

Für die erste Überprüfung ist der *OS-Loader* zuständig. Die Abb. 11.6 veranschaulicht den Zusammenhang der einzelnen Signaturprüfungen. Dieser überprüft zunächst die korrekte Funktionsweise der für die Überprüfung eingesetzten Crypto-Library (*MinCrypt*) durch den Aufruf eines Selbsttests. Die *MinCrypt*-Bibliothek wird beim Übersetzen des *OS-Loaders* statisch an diesen gebunden und ist somit vor Manipulationen durch die digitale Signatur des *OS-Loaders* geschützt. Danach lädt er eine ebenfalls digital signierte Liste von gültigen Prüfsummen (*%SystemRoot%\System32\catroot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\nt5.cat*) in seinen Speicherbereich. Nun überprüft der *OS-Loader* seine eigene Integrität, indem er eine Prüfsumme über sich selbst erzeugt und mit der Prüfsumme seiner im

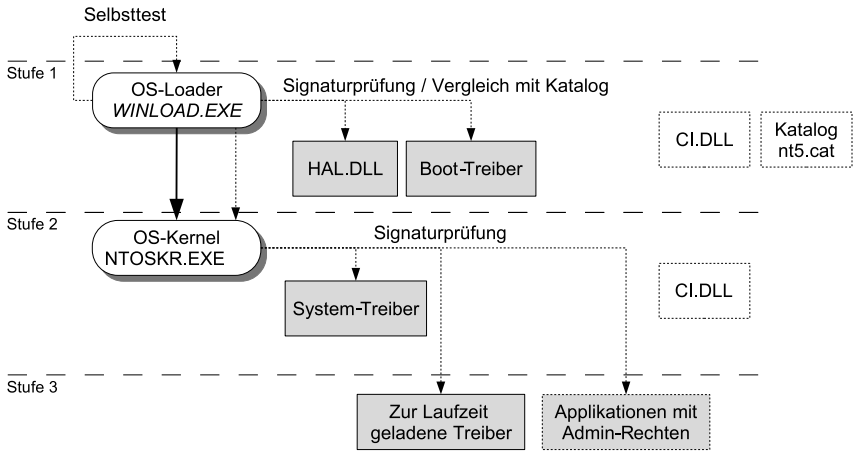

Abb. 11.6 Driver Signing in Windows Vista

Header der Datei hinterlegten Signatur vergleicht. Ist diese Überprüfung positiv, stellt er zusätzlich sicher, dass die errechnete Prüfsumme auch in der Liste der gültigen Prüfsummen enthalten ist. Genau dasselbe Verfahren wird im Folgenden auf sämtliche Boot-Treiber angewandt, also auf alle Treiber, die zum Laden des eigentlichen Betriebssystemkerns nötig sind. Dies gilt ebenso für die nächste Instanz des Startvorgangs, den Betriebssystemkern. Dadurch werden zwei Anforderungen überprüft. Zum einen kann durch die Verifizierung der Signaturen sichergestellt werden, dass die Treiber aus einer vertrauenswürdigen Quelle stammen und seit dem Zeitpunkt der Signatur nicht verändert wurden. Zum anderen stellt der Vergleich mit der Liste der gültigen Prüfsummen zusätzlich sicher, dass nur von Microsoft für dieses Stadium des Systemstarts vorgesehene Treiber geladen werden. Die für die Überprüfung der digitalen Signaturen benötigten Root-CA-Zertifikate sind fest in der *MinCrypt*-Bibliothek verankert, weswegen Angriffe scheitern, die darauf basieren, die Liste der vertrauenswürdigen Root-CA-Zertifikate des Betriebssystems um CA-Zertifikate zu erweitern.

Wurden alle Überprüfungen positiv abgeschlossen, kommen der Betriebssystemkern und die verifizierten Boot-Treiber zur Ausführung. Der Betriebssystemkern ist verantwortlich für die Überprüfung der System-Dateien, hierzu zählen die einzelnen Komponenten des Betriebssystems sowie die Treiber für systemspezifische Hardware. Hierfür verwendet er einen der geladenen Boot-Treiber (*\%SystemRoot%\system32\CI.DLL*), diese Bibliothek greift ebenfalls auf die Funktionen der MinCrypt-Bibliothek zurück. Der Betriebssystemkern überprüft nun nach demselben Verfahren wie zuvor bereits der *OS-Loader* sämtliche Treiber, bevor er sie in den Speicherbereich des Betriebssystems lädt. Zusätzlich wird eine weitere Überprüfung von Treiber-Signaturen zur Laufzeit des Systems durchgeführt. Dieser Vorgang wird unter anderem beim Verbinden von neuer Hardware mit dem System ausgelöst. Hierfür ist ebenfalls der Betriebssystemkern bzw. die *CI.DLL* verantwortlich. Eine weitere interessante Funktion der *CI.DLL* ist die Überprüfung der digitalen Si-

gnatur einer Applikation, die in ihrem Applikations-Manifest administrative Rechte anfordert oder explizit vom Administrator mit vollen Rechten gestartet wird. Verfügt eine Applikation nicht über ein gültiges Zertifikat, erscheint ein Windows-Dialog, der den Benutzer über dieses Defizit informiert. Zusätzlich kann mittels Gruppenrichtlinie die Ausführung solcher Applikationen grundsätzlich gesperrt werden. Im Unterschied zur Überprüfung der Treiber-Signaturen zur Laufzeit ist diese Funktion auch in den 32-Bit-Versionen verfügbar.

Die *Driver-Signing*-Komponente lässt sich mit der *Integrity-Validation*-Funktion des Anforderungskatalogs vergleichen, auch wenn hierbei keine der Prüfsummen aus dem TPM ausgewertet wird. Durch die Überprüfung der Treiber-Signaturen leistet sie einen Beitrag zum Erhalt der Integrität, indem zur Laufzeit hinzugefügte Treiber ebenfalls einer Prüfung unterzogen werden. Jedoch hat Microsoft die Chance nicht genutzt, die im Falle eines vorhandenen TPM durch die *Trusted Computing Platform* und den Secure Startup aufgebaute *Vertrauenskette* (Chain of Trust) fortzusetzen.

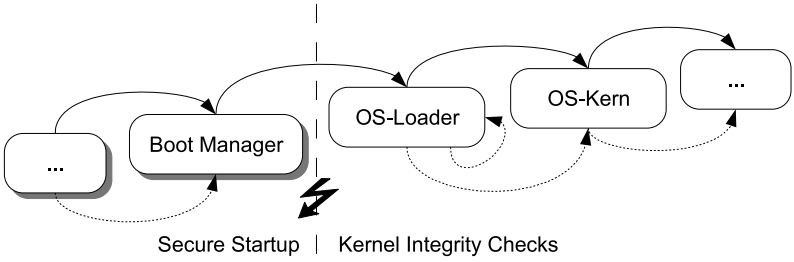

Abb. 11.7 Unterbrechung der Chain of Trust

So werden zwar sämtliche am Startvorgang beteiligte Komponenten, beginnend mit dem *OS-Loader*, anhand ihrer Signatur überprüft und nur im Falle eines positiven Ergebnisses in die nächste Betriebsstufe gewechselt, jedoch beginnt die Überprüfung mit einem Selbsttest des *OS-Loaders*. Das Konzept einer Vertrauenskette sieht jedoch vor, dass jede Komponente vor ihrer Ausführung durch ihren Vorgänger geprüft bzw. zumindest protokolliert werden muss. Dieses Versäumnis ermöglicht einen Angriff[8] auf die *Driver-Signing*-Funktion, indem die Datei des Betriebssystemkerns (*%SystemRoot%\system32\NTOSKRNL.EXE*) modifiziert wird. Da dies dazu führt, dass die Signatur der Datei ungültig wird, muss zusätzlich die Datei des *OS-Loaders* (*%SystemRoot%\system32\WINLOAD.EXE*) modifiziert werden. Dabei kann neben der Überprüfung der Treiber-Signaturen auch der Selbsttest des *OS-Loaders* deaktiviert und somit eine Erkennung der ungültigen Signaturen verhindert werden. Würde der Bootmanager (*%BitLockerPartition%\bootmgr*) vor der Ausführung des *OS-Loaders* eine Prüfsumme über diesen in einem weiteren PCR hinterlegen und somit die Vertrauenskette erweitern, gäbe es die Möglichkeit, diesen Angriff zu erkennen. Die Abb. 11.7 veranschaulicht die unterbrechung der Vertrauenskette.

[8] [Conover2006-2]

Eine weitere Möglichkeit, die Verifikation der Treiber-Signaturen zu umgehen, wird in [Rutkowska2006] beschrieben. Für diesen Angriff wird die Auslagerungsdatei manipuliert, um schadhaften Code in signierte Treiber einzuschleusen. Um diesen Angriff zu verhindern, ist es seit dem Release Candidate 2 von Windows Vista nicht mehr möglich, direkt auf die Datenstrukturen der Datenträger zuzugreifen. Zusätzlich kann über die Gruppenrichtlinien die Verschlüsselung der Auslagerungsdatei aktiviert und somit deren Manipulation verhindert werden.

11.6 Windows Resource Protection (WRP)

Die *Windows Resource Protection* ersetzt die *System File Protection* (SFP) von vorhergehenden Windows-Versionen. Das Ziel von WRP ist es, die Integrität der Systemdateien zu erhalten, und gehört somit zu der dritten Kategorie des Anforderungskatalogs. Hierzu schützt WRP kritische Teile des Betriebssystems vor unerlaubten Veränderungen, so z. B. kritische Teile der Registrierung und Dateien, die Komponenten des Betriebssystems darstellen. Dieser Schutz wird über das Setzen restriktiver Zugriffsrechte auf den entsprechenden Ressourcen erreicht. Somit handelt es sich nicht direkt um eine zusätzliche Sicherheitsfunktion, sondern eher um den automatischen Einsatz bereits vorhandener Funktionen.

Der einzige Prozess, der die nötigen Rechte besitzt, diese Objekte zu ändern, ist der *TrustedInstaller*-Dienst. Muss eines dieser Objekte aufgrund eines Updates geändert werden, wendet sich der *Windows-Update*-Prozess an diesen Dienst, um schreibenden Zugriff auf das Objekt zu erhalten. Der *TrustedInstaller* installiert ausschließlich Updates, die über eine digitale Signatur von Microsoft verfügen. Dies soll verhindern, dass wichtige Teile des Betriebssystems gegen manipulierte Versionen ausgetauscht werden können.

Auch Mitglieder der Administratorengruppe unterliegen der WRP und erhalten nicht den vollen Umfang der Zugriffsrechte auf diese Ressourcen. Zum Beispiel verfügt ein Mitglied der Benutzergruppe Administratoren nicht über Schreibrechte im Verzeichnis *%SystemRoot%\Windows*. Dadurch soll verhindert werden, dass der Administrator versehentlich wichtige Dateien beschädigt, bzw. dass Schadsoftware, die administrative Rechte erlangt hat, diese Dateien modifizieren kann. Es ist jedoch möglich, sich als Administrator die nötigen Rechte über Umwege zu beschaffen. Die WRP hat nämlich dem Administrator nicht das Recht zur Änderung des Besitzers eines Objekts entzogen. Somit kann der Administrator sich als neuen Besitzer der Datei eintragen und erhält dadurch sämtliche Zugriffsrechte für das Objekt. Somit kann auch die im vorhergehenden Kapitel beschriebene Modifikation des *OS-Loader* und des Betriebssystemkerns durchgeführt werden.

11.7 PatchGuard (nur 64-Bit-Versionen)

Obwohl es sich bei *PatchGuard* um keine neue Funktion handelt – sie war bereits in Windows Server 2003 SP1 und Windows XP Professional x64 Edition enthalten – , lohnt sich dennoch eine nähere Betrachtung.

Neben den beschriebenen Funktionen *Secure Startup* und *Driver Signing* ist *PatchGuard* eine weitere wichtige Komponente zur Erhaltung der Systemintegrität. Im Gegensatz zur den beiden anderen Funktionen, die ihre Integritätsprüfung während des Systemstarts bzw. beim Laden von Treibern und wichtigen Betriebssystemdiensten durchführen, versucht *PatchGuard*, die Integrität des Betriebssystemkerns während der Laufzeit zu überwachen.

PatchGuard schützt das System durch regelmäßige Überprüfung von wichtigen Datenstrukturen (ca. alle 5–10 Minuten) im Speicherbereich des ausgeführten Betriebssystems. Hierfür wird unter anderem die *System Service Descriptor Table* (SSDT), welche die Adressen der Betriebssystemfunktionen enthält, auf Änderungen überprüft. Durch eine Manipulation dieser Tabelle ist es möglich, Aufrufe von Betriebssystemfunktionen auf andere Implementierungen umzuleiten. Diese Technik ist als „kernel patching“ oder „kernel hooking“ bekannt und wird häufig von Rootkits, in Form von Systemtreibern, verwendet.

Jedoch verwenden auch Hersteller von Sicherheitssoftware (vor allem bei On-Access-Viren-Scannern und Personal-Firewalls) diese Technik. Aufgrund dieser Tatsache und weil bereits Verfahren zur Umgehung[9] des Schutzes bekannt sind, fordern einige Hersteller die Entfernung der Funktion. Als Reaktion auf die Kritik hat Microsoft ein weiteres API für Sicherheitssoftware angekündigt[10]. Damit sollen digital signierte Applikationen auch ohne Manipulationen der Datenstrukturen des Betriebssystemkerns Zugriff auf die notwendigen Funktionen erhalten.

Gegen die nächste Generation der Rootkits bietet jedoch auch *PatchGuard* keinen effektiven Schutz. Diese Rootkits verwenden Virtualisierungs-Technologie, um sich vor dem Betriebssystem zu verstecken. Durch die Verwendung von Befehlsatzerweiterungen aktueller Prozessoren von AMD oder INTEL ist es einem Rootkit sogar möglich, ein Betriebssystem zur Laufzeit in eine virtuelle Maschine zu verschieben, was seine Erkennung nahezu unmöglich macht [Rutkowska2006].

11.8 User Account Control

Eine weitere Neuerung in Windows Vista ist die *User Account Control* (UAC), eine weitere Maßnahme zu Erhaltung der Integrität des Betriebssystems. Unter dem Begriff UAC sind mehrere Sicherheitsfunktionen zusammengefasst, die eine Erweiterung der Zugriffskontrolle (Access Control) in Windows implementieren. Einleitend wird deshalb zunächst die bisherige Windows-Zugriffskontrolle beschrieben.

---

[9] [Skape2005, Skape2006]

[10] Diese Schnittstelle wird voraussichtlich mit dem ersten Service Pack für Vista nachgeliefert.

Unter Microsoft Windows (NT, 2000, XP) wird jedem Betriebssystemobjekt (Dateien, Prozesse etc.) ein *Security Descriptor* zugeordnet, der u.a. eine DACL (*Discretionary Access Control List*) und eine SACL (*System Access Control List*) enthalten kann. Ist keine DACL vorhanden, so erhält jeder Benutzer Vollzugriff auf das Objekt. Ist die DACL vorhanden aber leer, so erhält kein Benutzer Zugriff. Eine DACL besteht aus einem Header und maximal 1.820*Access Control Entries* (ACE). Ein ACE enthält die Information, ob einem Benutzer oder einer Benutzergruppe eine bestimmte Zugriffsart erlaubt oder verweigert werden soll. DACLs werden dynamisch vererbt. Wird die DACL eines übergeordneten Verzeichnisses geändert, so hat dies Auswirkungen auf die darunterliegende Verzeichnisstruktur. Einträge innerhalb der SACL dienen nicht der Zugriffssteuerung, sie regeln, welche Zugriffsversuche auf das Objekt im Windows-Ereignisprotokoll aufgezeichnet werden sollen.

Benutzer und Benutzergruppen werden eindeutig durch ihren *Security Identifier* (SID) identifiziert. Windows unterscheidet zwischen Benutzern bzw. Benutzergruppen, die sich interaktiv am System anmelden können (User Accounts), und denen, die für die Ausführung des Betriebssystems und dessen Diensten verwendet werden (System Accounts). Meldet sich ein Benutzer am System an, so wird ein für diese Sitzung gültiges *Access-Token* generiert. Dieses Token enthält die SID des Benutzers, die SIDs seiner Benutzergruppen, eine Liste der Systemrechte des Benutzers und einige Metainformationen. Startet ein Benutzer einen Prozess, so erhält dieser eine Referenz auf das *Access-Token* des Benutzers und erbt somit seine Rechte. Startet ein Prozess weitere Prozesse, so erhalten auch diese die Rechte des Benutzers. Die Zugriffskontrolle in Windows ist formell als eine Kombination aus *Role Based Access Control* (RBAC) und *Descretionary Access Control* (DAC) beschreibbar[11].

11.8.1 User Account Protection (UAP)

Die *User Account Protection* ist auch unter den Namen *Least-Privilege User Accounts* oder *Limited User Accounts* (LUA) bekannt. Das Konzept, das hinter dieser Entwicklung steht, ist unter dem Namen *Principle Of Least Authority* (POLA) bekannt[12]. Jeder Nutzer und jeder Prozess eines Systems soll nur die Rechte erhalten, die für die Ausführung unbedingt benötigt werden.

Die Hauptaufgabe der UAP ist es, die potentielle Gefahr, die durch das Arbeiten unter dem Administrator-Benutzerkonto entsteht, abzuschwächen. Jeder Benutzer, der Mitglied der Benutzergruppe der Administratoren ist, erhält bei seiner Anmeldung am System zwei unterschiedliche *Access-Token*, ein *Administrator-Token* und ein *Standard-User-Token*. Das *Administrator-Token* enthält sämtliche Rechte des Benutzers, wohingegen das *Standard-User-Token* nur einen Teil dieser Rechte widerspiegelt. Startet ein Mitglied dieser Benutzergruppe nun ein Programm, wird dieses in einem Kontext ausgeführt, der nur einen kleinen Teil der Rechte des Benut-

[11] [Bishop2005]

[12] [Saltzer1975]

zers enthält, indem diesem Prozess nur eine Referenz auf das *Standard-User-Token* übergeben wird[13]. Benötigt ein Programm zur korrekten Ausführung administrative Rechte, kann der Administrator den Prozess ohne Einschränkungen ausführen. Hierbei erhält der Prozess eine Referenz auf das *Administrator-Token* und somit alle Rechte des Benutzers.

Die UAP ist standardmäßig für alle Administratoren aktiv und kann über die lokalen Sicherheitsrichtlinien oder über die Gruppenrichtlinien konfiguriert werden. Hier kann auch definiert werden, wie im Falle eines notwendigen Zugriffs auf das *Administrator-Token* verfahren werden soll. Es stehen hierbei drei Optionen zu Auswahl:

- Keine Interaktion des Administrators nötig. Es erscheint kein Dialog und der Zugriff wird automatisch genehmigt. Dies kommt einer Deaktivierung der UAP gleich und sollte nicht gewählt werden.
- Der Administrator muss den Vorgang genehmigen. Es erscheint ein Dialog mit der Aufforderung, den Vorgang zu bestätigen. Dies entspricht der Standardeinstellung, sie sollte aber nur in Verbindung mit dem *Secure Desktop*[14] verwendet werden.
- Der Administrator muss den Vorgang genehmigen. Es erscheint ein Passwort-Dialog, den der Administrator mit seinem Benutzernamen und Passwort bestätigen muss. Dies entspricht dem UAP-Dialog, der einem Standard-Benutzer beim Aufruf eines Prozesses, welcher administrative Rechte benötigt, angezeigt wird. Dies ist die sicherste der drei Optionen.

Um die Abwärtskompatibilität zu älteren Windows-Versionen gewährleisten zu können, enthält auch Windows Vista das Benutzerkonto des *Built-In-Administrators*. Dieses Konto ist standardmäßig nicht aktiviert und nicht durch UAP geschützt, somit erhält jeder von diesem Benutzer gestartete Prozess ein Access-Token, das sämtliche Rechte des Administrators enthält. Dieses Konto ist ebenfalls Mitglied in der Benutzergruppe der Administratoren, somit verfügt das Access-Token des *Built-In-Administrators* über dieselben Systemrechte wie das Administrator-Token eines durch UAP geschützten Administrators[15]. Dies wird teilweise innerhalb der Dokumentation von Microsoft anders dargestellt, möglicherweise war eine zusätzliche Beschränkung der unter der UAP stehenden Konten geplant, wurde aber für die finale Version von Vista wieder verworfen.

Das mit der UAP eingeführte Konzept des eingeschränkten Administrators ist auf jeden Fall ein sinnvoller Ansatz, der über das Potential verfügt, Schadsoftware, die administrative Rechte benötigt, einzuschränken. Ob diese Schutzfunktionen greifen, hängt jedoch ganz davon ab, ob die neuen UAP-Dialoge von den Benutzern verstanden und entsprechend genutzt werden. Ein versehentlich bestätigter UAP-Dialog eröffnet auch unter Vista ausgeführter Schadsoftware Angriffsvektoren auf das System. Einen nicht unwesentlichen Einfluss auf den Erfolg der UAP haben

---

[13] [Micro04]

[14] Eine weitere UAC-Funktion, später im Buch beschrieben.

[15] Zugriffsrechte eines Benutzers können mittels *whoami/all* ausgelesen werden.

die Software-Hersteller, die ihre Applikationen an die Ausführung mit Rechten des Standard-Users anpassen sollten, um somit unnötige UAP-Dialoge zu vermeiden. Warum jedoch der bei der Installation angelegte Benutzer nach wie vor Mitglied der Benutzergruppe „Administratoren“ ist und somit erneut die Chance vergeben wurde, den Benutzer zur Verwendung eines Standard-User-Kontos zu motivieren, ist nicht ersichtlich.

Zusätzlich sei erwähnt, dass die UAP streng genommen keine Umsetzung des POLA-Konzeptes darstellt. So hat auch eine mit dem *Standard-User-Token* arbeitende Applikation in der Regel mehr Rechte als sie benötigt. Ein Beispiel hierfür ist der Windows-Taschenrechner, er verfügt durch das *Standard-User-Token* über das Recht zum Herunterfahren des Betriebssystems. Eine vollständige Umsetzung einer *Mandatory Access Control* (MAC), wie sie z. B. SELinux[16] bietet, ist auf einem Desktop-Betriebssystem jedoch auch nahezu unmöglich.

11.8.2 Mandatory Integrity Control (MIC)

In Windows Vista besitzt jedes Objekt (z. B. eine Datei oder ein Eintrag in der Registrierung) eine Integritätsstufe (*integrity level*). Prozesse besitzen ebenfalls eine Integritätsstufe und jeder Kindprozess erbt diese von seinem Elternprozess. Hierfür wurden neue Einträge im *Security Descriptor* von Objekten und den *Access-Tokens* der Benutzer bzw. deren Prozesse definiert. Die folgende Tabelle listet die möglichen Integritätsstufen[17]:

Tabelle 11.1 Vista-Integritätsstufen

Integritätsstufe	**Systemrechte (vereinfacht)**
System	Schreibzugriff auf Dateien des Betriebssystems. Wird vom *Trusted Installer* verwendet um Windows Updates zu installieren
High (Administrative)	Schreibzugriff auf das Verzeichnis C:\Programme und auf schützenswerte Bereiche der Registrierung wie z. B. *HKEY_LOCAL_MACHINE*
Medium (User)	Erstellen und Ändern von Dateien im Verzeichnis Dokumente des aktuellen Benutzers und an benutzerspezifischen Bereichen der Registrierung (z. B. *HKEY_CURRENT_USER*)
Low (Untrusted)	Ausschließlich Schreibzugriff auf Bereiche mit Integritätsstufe „low“ (*Temporäre Internet Dateien\Low* und *HKEY_CURRENT_USER\Software\LowRegistry*)

So enthält z. B. das Administator-Token die Integritätsstufe *high* und somit schreibenden Zugriff auf kritische Teile der Registrierung. Da der Administrator

[16] [URL28]

[17] MIC überwacht ausschließlich Schreibzugriffe.

bei aktiver UAP standardmäßig nur mit einem Standard-User-Token arbeitet, erhält ein vom ihm ausgeführter Prozess nur die Integritätsstufe *medium*. Der Internet Explorer 7 (*iexplore.exe*) wurde zusätzlich eingeschränkt; wird er gestartet, erhält er nur die Stufe *low* und kann somit nur auf einen sehr eingeschränkten Bereich schreibend zugreifen [Micro03]. Lädt ein Benutzer eine Datei aus dem Internet, welche er in seinem Benutzerverzeichnis speichern möchte, wird ein Hilfsprozess (*ieuser.exe*) gestartet, der über dieselbe Integritätsstufe wie das Benutzerverzeichnis verfügt (*medium*). Durch die Einschränkung des Internet-Explorer-Prozesses soll verhindert werden, dass durch Schwachstellen in kritischen Teilen des Browser (z. B. Rendering-Engine, Javascript-Interpreter) schadhafte Dateien ohne Benutzerinteraktion (*Drive-By-Downloads*) auf den Rechner kopiert werden können.

Da MIC für schreibende Zugriffe eine zusätzliche Zugriffskontrolle zu der bisherigen Windows-Zugriffskontrolle (*Access Control Lists* – ACLs) darstellt, müssen nun beide Überprüfungen ein positives Resultat liefern, um schreibenden Zugriff auf das Objekt zu erhalten. Das heißt, der Benutzer bzw. der Prozess muss über einen entsprechenden Eintrag in der ACL des Objektes verfügen (Schreibrecht), und er benötigt mindestens dieselbe Integritätsstufe.

Um sicherzustellen, dass ein Prozess zur Laufzeit seine Rechte nicht ausweiten kann (privilege escalation attacks), sind bestimmte Systemaufrufe von einem Prozess mit niedriger Integritätsstufe auf einen Prozess mit höherer Integritätsstufe deaktiviert.

Die *Mandatory Integrity Control* ist eine Implementierung des *Biba-Sicherheitsmodells* und dient dem Schutz vor unautorisierter Änderung von Informationen (Datenintegrität). Microsoft hat darauf verzichtet, zusätzlich Vertraulichkeitsstufen einzuführen, lesende Zugriffe werden nach wie vor ausschließlich über ACL geregelt. Dies entspräche dem *Bell-LaPadula-Sicherheitsmodell*[18], welches die Umkehrung des *Biba-Modells* darstellt und die Geheimhaltung von Informationen regelt. Da in dieser Untersuchung der Fokus auf der Erhaltung der Systemintegrität und nicht auf der Vertraulichkeit von Daten liegt, ist dieser Umstand zu vernachlässigen.

11.8.3 Secure Desktop (Trusted Path)

Zum Schutz des Windows-Dialogs bei einer Erhöhung der Benutzerrechte wird ein zweiter Desktop in einer separaten Benutzersitzung ausgeführt (*Session-0*). Dies soll verhindern, dass andere Applikationen auf das Fenster des Dialogs zugreifen können und dadurch die erforderliche Bestätigung durch den Benutzer selbst erbringen. Die Verwendung des so genannten *Secure Desktop* kann über Gruppenrichtlinien konfiguriert werden und ist standardmäßig aktiviert. Als zusätzlichen Schutz bietet Windows Vista die Möglichkeit, alle UAC-Dialoge mit einem vorgeschalteten Strg+Alt+Entf-Dialog zu sichern. Diese Option ist ebenfalls nur über die Gruppenrichtlinien zu erreichen. Erst nach dem Betätigen dieser Tastenkombination wech-

[18] [Bishop2005]

selt Windows Vista zum *Secure Desktop*. Microsoft spricht hierbei vom Aufbau eines gesicherten Kommunikationspfades (*Trusted Path*) und verwendet die Tastenkombination als „*Secure Attention Sequence*“. Diese Funktion entspricht der mit Windows NT eingeführten Sicherung für den Windows-Anmelde-Dialog. Ist dieser Schutz aktiviert, wird automatisch der *Secure Desktop* verwendet, unabhängig von der entsprechenden Option in den Gruppenrichtlinien.

Der Mehrwert an Sicherheit resultiert aus der Tatsache, dass die Tastenkombination Strg+Alt+Entf einen Hardwareinterrupt auslöst, welcher nur vom Betriebssystem selbst interpretiert werden kann und somit verhindert, dass Software, die außerhalb des Betriebssystemkerns läuft, dieses Signal empfangen bzw. unterbinden kann. Das Simulieren des Signals mittels Software ist jedoch möglich (z. B. Windows Remote Desktop). Der Wechsel zum UAC-Dialog nach Drücken der Tastenkombination entspricht somit einer Authentifizierung des Betriebssystems gegenüber dem Benutzer vor dem Bildschirm und nicht wie oft angenommen der Authentifizierung des Benutzers gegenüber dem Betriebssystem.

Der *Secure Desktop* entfaltet in der Standardkonfiguration leider nur einen Teil seines Schutzpotentials. Er verhindert, dass Dialoge von Prozessen außerhalb der *Session-0* angesprochen werden können und somit, dass schadhafte Programme diesen Dialog selbst bestätigen können. Dies ist aber ohnehin nur möglich, wenn der aktive Benutzer Mitglied der Gruppe „Administratoren“ ist und eine zusätzliche Eingabe des Passwortes nicht mittels Gruppenrichtlinie aktiviert wurde. Er verhindert jedoch nicht, dass ein schadhaftes Programm einen gefälschten UAP-Dialog präsentiert und den Benutzer damit zur Eingabe seines Passwortes verleitet. Eine zusätzliche Aktivierung des Strg+Alt+Entf-Schutzes führt zur Verbesserung dieser Situation. Zwar ist es immer noch möglich, den Strg+Alt+Entf-Dialog zu fälschen, jedoch ist es wie erwähnt nicht möglich, das bei dieser Tastenkombination ausgelöste Signal ohne Manipulation des Betriebssystems abzufangen. Wird dem Benutzer bei aktiviertem Strg+Alt+Entf-Schutz ein gefälschter Dialog präsentiert und er betätigt die Tastenkombination, wird der Anmeldebildschirm von Windows Vista aktiv und entlarvt den Angriff, da nicht der erwartete UAP-Dialog erscheint.

Da Microsoft jedoch die aus Sicht des Autors unglückliche Entscheidung getroffen hat, dem Dialog nochmals einen weiteren Windows-Dialog vorzuschalten[19], wird der Benutzer mit zwei zusätzlichen Dialogen konfrontiert, bevor er den eigentlichen UAP-Dialog erreicht, was der Benutzerakzeptanz nicht unbedingt förderlich ist. Eine Alternative hierfür wäre der auf Seite 122 beschriebene Ansatz zur Authentifizierung von Betriebssystem-Dialogen.

Es sei zusätzlich erwähnt, dass weder die Kombination aus *Secure Desktop* und Strg+Alt+Entf-Dialog noch die eben beschriebene Maßnahme einen wirkungsvollen Schutz vor Key-Loggern im Betriebssystemkern oder dem Abhören der Kommunikation zwischen Tastatur und Rechner darstellt.

Ein weiterer Kritikpunkt ist, dass Dialoge, die nicht bei der Anmeldung am lokalen System oder für die Erhöhung der Systemrechte verwendet werden, dennoch aber vertrauliche Daten enthalten können, nicht mit der UAP gekoppelt sind und

[19] Dieser Dialog enthält lediglich den Hinweis, dass aus Sicherheitsgründen im nächsten Schritt die Betätigung von Strg+Alt+Entf notwendig ist.

somit nicht zu einer Verwendung des *Secure Desktop* führen. Beispiele hierfür sind der Dialog beim Verbinden von Netzwerklaufwerken und die Anzeige des TPM-Eigentümerpassworts während der Ausführung des Assistenten zur Einrichtung des TPM.

Der *Secure Desktop* adressiert die Konzepte des *Trusted GUI*, erreicht aber auf Grund der beschriebenen Schwächen nicht den notwendigen Grad an Sicherheit. Lösungsansätze wie z. B. *Nitpicker*[20] liefern ein deutlich höheres Maß an Isolation und somit Schutz der Betriebssystemdialoge.

11.8.4 UI Privilege Isolation (UIPI)

In bisherigen Windows-Versionen war es für einen Prozess möglich, Nachrichten an Fenster eines anderen Prozesses zu schicken (SendMessage und PostMessage APIs). Dies ermöglicht unter bestimmten Umständen die Ausführung von beliebigem Code im Kontext eines Prozesses mit höheren Rechten (Shatter Attack[21]).

Deshalb ist es in Windows Vista nicht mehr möglich, dass ein System-Prozess (Windows-Dienst) über eine grafische Oberfläche (GUI) verfügt. Ebenso wird verhindert, dass ein Prozess mit geringerer Integritätsstufe an einen Prozess mit höherer Integritätsstufe derartige Nachrichten senden kann. Hierfür wurde zusätzlich zu den bereits beschriebenen Integritätsstufen der UAC eine weitere Integritätsstufe eingeführt (*UIAccess-Integritätsebene*). Benötigt eine Applikation auf Grund ihrer Funktionsweise diese Integritätsstufe – ein Beispiel hierfür ist eine Bildschirmtastatur –, kann sie dies in ihrem Applikations-Manifest angeben (*UIAccess=True*) und erhält nach der Bestätigung durch den Administrator die nötigen Rechte. Als zusätzlichen Schutz kann mittels Gruppenrichtlinie die Vergabe dieser Integritätsstufe auf Applikationen beschränkt werden, deren Dateien im Windows-Verzeichnis (*%SystemRoot%*; *%SystemRoot%\system32*) abgelegt sind und über eine gültige digitale Signatur verfügen.

11.9 Windows Service Hardening

Im Gegensatz zu bisherigen Windows-Versionen laufen Systemdienste und Applikationen in unterschiedlichen Betriebssystemsitzungen und sind somit voneinander isoliert. Alle Applikationen des ersten am System angemeldeten Benutzers werden in der *Session-1* ausgeführt und sind somit nicht in der Lage, *Window Messages* (z. B. SendMessage und PostMessage) an Prozesse in der *Session-0* zu senden. Eine solche Kommunikation kann in Vista nur mittels Client/Server-Mechanismen (z. B. RPC oder Named Pipes) implementiert werden.

---

[20] Siehe „Nitpicker – Overlay Window Management“

[21] [Moore01]

Weiter ist es Prozessen in der *Session-0* nicht erlaubt auf den Grafik-Treiber zuzugreifen und somit Windows-Dialoge anzuzeigen. Will ein Windows-Dienst mit dem Benutzer interagieren, muss er dies über einen Hilfsprozess in der Sitzung des Benutzers bewerkstelligen. Die Kommunikation zwischen Windows-Dienst und Hilfsprozess muss ebenfalls über RPC oder Named Pipes realisiert werden.

Um die Abwärtskompatibilität zu Windows XP zu gewährleisten hat Microsoft in Vista den *Interactive Service Detection Service* eingeführt. Dieser überwacht alle Windows-Dienste und informiert den Benutzer über Versuche einen Windows-Dialog anzuzeigen. Dieser kann sich den unterdrückten Dialog anzeigen lassen, indem er in die *Session-0* wechselt[22] um diesen zu bearbeiten. Sind keine Dialoge mehr vorhanden, wechselt Windows automatisch zurück zur Sitzung des Benutzers.

11.10 Zusammenfassung und Schlussfolgerung

Microsoft hat mit Windows Vista zahlreiche neue Sicherheitsfunktionen eingeführt, die alle ihren Teil zum Aufbau bzw. zur Erhaltung der Systemintegrität beitragen. Neben vielen kleinen Detailverbesserungen sind hierbei vor allem die beschriebenen Funktionen *BitLocker*, *Driver Signing*, *PatchGuard* und die *User Account Control* hervorzuheben. *BitLocker* bietet die Möglichkeit die gesamte Systempartition zu verschlüsseln und den verwendeten Schlüssel durch den Einsatz eines TPM an einen bestimmten Zustand des Systems zu binden. Die *Driver-Signing*-Funktion stellt sicher, dass nur digital signierte Treiber vom Betriebssystem verwendet werden. Und *PatchGuard* überprüft kritische Datenstrukturen des Betriebssystems um eine Manipulation zur Laufzeit zu erkennen. Die UAP versucht durch Erweiterung der bereits in Windows XP vorhandenen Zugriffskontrollmechanismen die Angriffsfläche des Betriebssystems zu verkleinern.

So ist unter dem Aspekt der Betriebssystem-Sicherheit eine deutliche Verbesserung zu erkennen und viele der Schwächen von Windows XP wurden adressiert. Obwohl bereits für einige Funktionen Maßnahmen zur Umgehung bekannt geworden sind und auch mit der Aufdeckung weiterer Schwächen zu rechnen ist, ist es Microsoft durchaus gelungen das Sicherheitsniveau deutlich zu erhöhen.

Allerdings ist ein Teil der hier vorgestellten Sicherheitsfunktionen der Produktpolitik von Microsoft zum Opfer gefallen. So sind sowohl die Überprüfung von Treibersignaturen als auch die *PatchGuard*-Komponente nur in den 64-Bit-Versionen enthalten. Im Falle des *Driver Signing* ist dies laut Microsoft ein Zugeständnis an die Hardwarehersteller. Auch von den Vorteilen der *BitLocker*-Festplattenverschlüsselung und somit dem Schutz vor Offline-Angriffen werden wohl die wenigsten der Vista-Benutzer profitieren, sie ist nämlich nur in der Enterprise und der Ultimate Version enthalten.

Eine Bewertung von Windows Vista anhand des definierten Anforderungskatalogs für ein *Trusted-OS* fällt schwer; so sind zwar Teile des Anforderungskatalogs

[22] Dies entspricht einem Wechsel auf den Secure Desktop.

umgesetzt, ein durchgängiges Konzept ist jedoch nicht zu erkennen. So ist die *BitLocker*-Festplattenverschlüsselung die einzige Komponente, die von der *Trusted Computing Platform* bzw. vom TPM Gebrauch macht; sie verwendet diese aber in erster Linie für die Verschlüsselung der Systempartition und verzichtet auf die Fortsetzung der *Vertrauenskette* bis zum vollständigen Systemstart. Genau dies wäre jedoch eine der Grundlagen für den Aufbau eines *Trusted-Computing-Systems* gewesen. Microsoft vertraut in diesem Bereich alleine auf die Überprüfung der digitalen Signaturen wichtiger Systemkomponenten. Die *Network Access Protection* (NAP) ist ebenfalls eine interessante Sicherheitskomponente, sie verfolgt vergleichbar mit *Trusted Network Connect* (TNC) den Ansatz der *Remote Attestation*, jedoch verzichtet NAP auf die Verwendung des TPM.

Einflüsse der im Rahmen der NGSCB vorgestellten Konzepte auf die finale Version von Vista sind daher nur in der Form des *Secure Startup* und der für den Schutz der Systemdienste verwendeten *Session-0* zu erkennen.

Literaturverzeichnis

[Abadi2004] M. Abadi (August 2004)
„Trusted Computing, Trusted Third Parties, and Verified Communications“

[Arbaugh1997] A. Arbaugh et al. (1997)
„A Secure and Reliable Bootstrap Architecture“

[Arbaugh1997-2] A. Arbaugh et al. (1997)
„Automated Recovery in a Secure Bootstrap Process“

[Arkin2006] O. Arkir (September 2006)
„Bypassing Network Access Control Systems“

[Bell1976] D. Bell, L. LaPadula (1976)
„Secure Computer Systems: Unified Exposition and MULTICS Interpretation“

[Berger2006] S. Berger et al. (2006)
„vTPM: Virtualizing the Trusted Platform Module“

[Bishop2003] M. Bishop (2003)
„Computer Security – Art and Science“

[Bishop2005] M. Bishop (2005)
„Introduction to Computer Security“

[Buchmann2003] J. Buchmann (Mai 2003)
„Einführung in die Kryptographie“

[Brunette2005] G. Brunette (Juni 2005)
„Toward Systemically Secure IT Architectures“

[Böhmer01] F. Böhmer (2001)
„Efficient Revocation in Group Signatures“

[Camenisch001] J. Cammenisch (2001)
„Better Privacy for Trusted Computing Platforms“

[Carr2004] B. Carrier, J. Grand (Februar 2004)
„A Hardware-Based Memory Acquisition Procedure for Digital Investigation“

[CIST1999] Committee on Information Systems Trustworthiness (1999)
„Trust in Cyberspace“

[Clark1987] D. Clark, David R. Wilson (1987)
„A Comparison of Commercial and Military Computer Security Policies“

[Clark1994] Paul C. Clark (1994)
„BITS: a smartcard protected operating system“

[Conover2006] M. Conover (März 2006)
„Analysis of the Windows Vista Security Model“

[Conover2006-2] M. Conover (August 2006)
„Assessment of Windows Vista Kernel-Mode Security,“

[Dhamija2005] R. Dhamija, J.D. Tygar (Juli 2005)
„The Battle Against Phishing – Dynamic Security Skins“

[DoD1983] Department of Defense (1983)
„Trusted Computer System Evaluation Criteria“

[Feske2003] N. Feske, H. Härtig (Dezember 2003)
„DopE – a Window Server for Real-Time and Embedded Systems“

[Feske2004] N. Feske, C. Helmuth (März 2004)
„Overlay Window Management: User interaction with multiple security domains“

[Feske2005] N. Feske, C. Helmuth (2005)
„A Nitpicker’s guide to a minimal-complexity secure GUI“

[Flick2004] C. Flick (Juni 2004)
„The Controversy over Trusted Computing“

[Garfinkel2003] T. Garfinkel et al. (Oktober 2003)
„Terra: A Virtual Machine-Based Platform for Trusted Computing“

[Garfinkel2003-2] T. Garfinkel et al. (2003)
„Flexible OS Support and Applikations for Trusted Computing“
[Govi2006] S. Govindavajhala, A. Appel (Januar 2006)
„Windows Access Control Demystified“
[Grawrock2006] D. Grawrock (März 2006), Intel Press
„The Intel Safer Computing Initiative – Building Blocks for Trusted Computing“
[Gutmann2007] P. Gutmann (Februar 2007)
„A Cost Analysis of Windows Vista Content Protection“
[Haldar2004] V. Haldar et al. (2004)
„Semantic Remote Attestation – A Virtual Machine directed approach to Trusted Computing“
[Haug2002] P. Haug (2002) Universität Tübingen
„Kryptologie und Datensicherheit“
Vorlesungsskript WS 2002/2003
[Haeger1998] T. Jaeger et al. (1998)
„Security Architecture for Component-based Operating Systems“
[Haertig2002] H. Haertig (2002)
„Security Architectures Revisited“
[Haertig2005] H. Haerig et al. (2005)
„The Nizza Secure-System Architecture“
[Hohmuth2004] M. Hohmuth et al. (2004)
„Reducing TCB size by using untrusted components“
[Hohmuth2005] M. Hohmuth et al. (2005)
„The VFiasco approach for a verified operating system“
[Intel2003] Intel Corporation (2003)
„Intel Trusted Execution Technology – Architectural Overview“
[Intel2004] Intel Corporation (2004)
„Trusted Platform Module based Security on Notebook PCs“
[Intel2006] Intel Corporation (September 2006)
„LaGrande Technology – Preliminary Architecture Specification“
[Intel2006-2] Intel Corporation (2006)
„Intel Virtualization Technology for Directed I/O“
[Intel2006-3] Intel Corporation (2006)
„Intel Virtualization Technology VT-x“
[Intel2006-4] Intel Corporation (2006)
„Extending Xen with Intel Virtualization“
[Kuhlmann2006] D. Kuhlmann (Juni 2006)
„An Open Trusted Computing Architecture“
[Kursawe2003] K. Kursawe et al. (September 2003)
„Improving End-user Security and Trustworthiness of TCG-Platforms“
[Kursawe2005] K. Kursawe et al. (2005)
„Analyzing trusted platform communication“
[Lemke01] K. Lemke et al. (2001)
„Seitenkanal-Analysen: Stand der Forschung in der Methodik“
[Li2006] X. Li et al (2006)
„Autonomic and Trusted Computing Paradigms“
[NSA1998] National Security Agency - NSA (1998)
„NSA Glossary of Terms Used in Security and Intrusion Detection“
[Petroni2004] N. Petroni et al. (August 2004)
„Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor“
[Pitcher2006] C. Pitcher, J. Riely (2006)
„Dynamic Policy Discovery with Remote Attestation“
[Reid2005] J. Raid, W. Caelli (2005)
„DRM, Trusted Computing and Operating System Architecture“

[Röcher2007] D. Röcher, M. Thumann (März 2007)
„Sicherheitsanalyse des Cisco NAC Framework“
[Rutkowska2006] J. Rutkowska (2006)
„Subverting Vista Kernel For Fun And Profit“
[Sadeghi2004] A. Sadeghi et al. (2004)
„Property-based Attestation for Computing Platforms“
[Sadeghi2004-2] A. Sadeghi et al. (September 2004)
„ European Multilateral Secure Computing Base - Open Trusted Computing for You and Me“
[Sadeghi2006] A. Sadeghi et al. (Mai 2006)
„TCG Inside? A Note on TPM Specification Compliance“
[Sadeghi2007] A. Sadeghi et al. (März 2007)
„Compartment Security for Browsers“
[Safford2004] D. Safford et al. (2004)
„A Trusted Linux Client (TLC)“
[Sailer2004] R. Sailer et al. (2004)
„Design and Implementation of a TCG-based Integrity Measurement Architecture“
[Sailer2005] R. Sailer et al. (2005)
„Building a MAC-based Security Architecture for the Xen Opensource Hypervisor“
[Sailer2005-2] R. Sailer et al. (2005)
„sHype: Secure Hypervisor Approach to Trusted Virtualized Systems“
[Sailer2006] R. Sailer et al. (2006)
„PRIMA: Policy-Reduced Integrity Measurement Architecture“
[Saltzer1975] J. Salzer, M. Schroeder (April 1975)
„The Protection of Information in Computer Systems“
[Sarmenta2006] L. Sarmenta et al. (2006)
„Virtual Monotonic Counters and Count-Limited Objects using a TPM without a Trusted OS“
[Schmeh2001] K. Schmeh (2001)
„Kryptographie und Public-Key-Infrastrukturen im Internet“
[Schoen01] S. Schoen
„Trusted Computing: Promise and Risk“
[Schwenk2005] J. Schwenk (August 2005)
„Sicherheit und Kryptographie im Internet“
[Shapiro2004] S. Shapiro et al. (2004)
„Design of the EROS Trusted Window Systems“
[Shi2005] E. Shi et al. (2005)
„BIND: A Fine-grained Attestation Service for Secure Distributed Systems“
[Singaravelu2006] L. Singaravelu et al. (2006)
„Reducing TCB Complexity for Security-Sensitive Applications “
[Skape2005] Skape, Skywing (Dezember 2005)
„Bypassing PatchGuard on Windows x64“
[Skape2006] Skape, Skywing (Dezember 2006)
„Subverting PatchGuard Version 2“
[Smith2005] S. Smith (2005)
„Trusted Computing Platforms – Design and Applications“
[Song2005] Z. Song et al. (2005)
„Trusted Web Services“
[Stallmann2002] R. Stallmann (2002)
„Can you trust your computer?“ The Selected Essays of Richard M. Stallman
[Stumpf2006] F. Stumpf et al. (Dezember 2006)
„A Robust Integrity Reporting Protocol for Remote Attestation“

[TCG2004] Trusted Computing Group (April 2004)
„TCG Specification – Architecture Overview – Revision 1.2“
[TCG2005] Trusted Computing Group (Juli 2005)
„TCG PC Client Specific Implementation for conventional BIOS“
[TCG2005-1] Trusted Computing Group (Juli 2005)
„TCG PC Client Specific TPM Interface Specification (TIS)“
[TCG2005-2] Trusted Computing Group (Juni 2005)
„Reference Architecture for Interoperability (Part I)“
[TCG2005-3] Trusted Computing Group (Juni 2005)
„Subject Key Attestation Evidence Extension“
[TCG2006] Trusted Computing Group (Mai 2006)
„TNC Architecture for Interoperability“
[TCG2006-2] Trusted Computing Group (Januar 2006)
„TCG Software Stack (TSS) Specification Version 1.2“
[TCG2006-3] Trusted Computing Group (März 2006)
„TPM Main – Part 1 Design Principles – Revision 94“
[TCG2006-4] Trusted Computing Group (Januar 2006)
„TCG Credential Profiles“
[TCG2006-5] Trusted Computing Group (März 2006)
„TPM Main – Part 3 Commands – Revision 94“
[TCG2006-6] Trusted Computing Group (November 2006)
„Architecture Part II – Integrity Management“
[TCG2006-7] Trusted Computing Group (November 2006)
„Integrity Report Schema Specification“
[Yee1994] B. Yee (1994)
„Using Secure Coprocessors“
[Wikipedia2005] http://en.wikipedia.org/wiki/Trusted_Computing (Juni 2005)
[Maruyama2003] H. Maruyama et al. (Januar 2003)
„Linux with TCPA Integrity Measurement“
[Mehnert2005] F. Mehnert (Juli 2005)
„Kapselung von Standard-Betriebssystemen“
[Micro01] Microsoft Corporation (August 2006)
„AES-CBC + Elephant diffusor – A Disk Encryption Algorithm for Windows Vista“
[Micro02] A. Ben-Menahem, A. Tucker (2005) Microsoft Corporation
„Windows Vista and ‘Longhorn’ Server: Understanding, Enhancing and Extending Security End-to-end“
[Micro03] Microsoft Corporation (Januar 2006)
„Understanding and Working in Protected Mode Internet Explorer“
[Micro04] Microsoft Corporation
„Restricted Tokens“. MSDN [Online]
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/restricted_tokens.asp
[Micro05] Microsoft Corporation (Januar 2006)
„Digital Signatures for Kernel Modules on x64-based Systems Running Windows Vista“
[Micro06] Microsoft Corporation (April 2005)
„Secure Startup – Full Volume Encryption: Technical Overview“
[Micro07] Microsoft Corporation (Mai 2006)
„BitLocker Drive Encyption: Technical Overview“
[Micro08] Microsoft Corporation (Mai 2006)
„Windows Vista Beta 2 – Trusted Platform Module Services Step-by-Step Guide“
[Micro09] Microsoft Corporation (April 2005)
„Trusted Platform Module Services in Windows Longhorn“

[Micro10] Microsoft Corporation (2006)
„Step-by-Step Guide to Device Driver Signing“
[Micro11] Microsoft Corporation (März 2006)
„Impact of Session-0 Isolation on services and Drivers in Windows Vista“
[Micro12] Microsoft Corporation (Dezember 2006)
„Network Access Protection Platform Architecture“
[Micro13] Microsoft Corporation (Mai 2007)
„Standardizing Network Access Control: TNC and Microsoft NAP to Interoperate“
[Munetoh2004] S. Munetoh et al. (2004)
„Pratical Integrity Measurement and Remote Verification for Linux Platform“
[Munetoh2005] S. Munetoh et al. (2005)
„WS-Attestation: Efficient and Fine-Grained Remote Attestation on Web Services“
[Moore01] B. Moore. (Oktober 2003)
„Shattering by Example“
[OtcEmscb] OpenTC, EMSCB (2005)
„OpenTC and EMSCB: Status Quo and Cooperation Planning“
[URL01] http://en.wikipedia.org/wiki/Underground_economy
[URL02] http://www.netzwelt.de/news/73085-cybercrime-lukrativer-als-drogenhandel.html
[URL03] http://research.microsoft.com/os/singularity/
[URL04] http://os.inf.tu-dresden.de/vfiasco/
[URL05] http://www.trustedcomputinggroup.org
[URL06] http://tools.ietf.org/html/rfc2104
[URL07] http://opentc.iaik.tugraz.at
[URL08] http://de.wikipedia.org/wiki/Zero_Knowledge
[URL09] http://trustedjava.sourceforge.net
[URL10] http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20060410_security.html
April 2006
[URL11] http://msdn2.microsoft.com/en-us/library/aa380255.aspx
[URL12] http://www.rsa.com/rsalabs/node.asp?id=2133
[URL13] http://research.microsoft.com/os/singularity/
[URL14] http://www.ietf.org/rfc/rfc2511.txt
[URL15] ftp://ftp.isi.edu/in-notes/rfc2986.txt
[URL16] ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-7.asc
[URL17] http://www.w3.org/TR/xkms2/
[URL18] http://www.missl.cs.umd.edu/sebos.html
[URL19] http://www.prosec.rub.de/trusted_grub.html
[URL20] http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html
[URL21] http://sourceforge.net/projects/linux-ima
[URL22] http://trousers.sourceforge.net/
[URL23] http://trustedjava.sourceforge.net/
[URL24] http://projects.csail.mit.edu/tc/tpmj/
[URL25] http://os.inf.tu-dresden.de/vfiasco/
[URL26] http://os.inf.tu-dresden.de/fiasco/
[URL27] http://os.inf.tu-dresden.de/L4/LinuxOnL4/
[URL28] http://www.nsa.gov/selinux/
[URL29] http://bluepillproject.org/
[Wang2005] X. Wang et al. (2005)
„Finding Collisions in the Full SHA-1“
[Yee2002] K. Yee (2002)
„User Interaction Design for Secure Systems“

Sachverzeichnis

V

Z