entwickler.press
shortcuts

Plattform-sicherheit

Smart Contracts und TPM

Christof Graff, Matthias Zscherp,
Helmut Stoiber

Christof Graff, Matthias Zscherp, Helmut Stoiber

Plattformsicherheit

Smart Contracts und TPM

ISBN: 978-3-86802-557-6

Ein Imprint der Software & Support Media GmbH

Inhaltsverzeichnis

1 Entwicklung von Vertragsagenten mit Codius

2 Plattformsicherheit per Kontrollchip

3 TPM-Kommandos und Programmierschnittstellen

Die Autoren

1 Entwicklung von Vertragsagenten mit Codius

Vertragsagenten (engl. „Smart Contracts“) können automatisiert durch die im Programmcode hinterlegten Bedingungen einen Vertrag (z. B. Handelsvertrag) überwachen und mit zuvor definierten Maßnahmen auf Abweichungen reagieren. Die Vorteile und die Herausforderungen bei der Entwicklung von Vertragsagenten werden am Beispiel des auf Node.js basierenden Open-Source-Frameworks „Codius“ [1] beschrieben.

Anwendungen werden stetig komplexer und automatisieren sukzessive manuelle Prozesse in den verschiedensten Arbeitsbereichen. Heutzutage werden komplexe Programme zur Gerätesteuerung oder -überwachung in vielen Gesellschaftsbereichen akzeptiert, beispielsweise der Autopilot im Passagierflugzeug. Trotz dieser Entwicklungen werden Verträge hingegen noch überwiegend in Papierform formuliert, archiviert und manuell überwacht. Vertragsagenten wiederum können durch Computersysteme auswertbare Bedingungen wie einen elektronischen Zahlungseingang automatisiert überwachen und bei Abweichungen als neutraler Vermittler in vorher abgestimmter Art und Weise agieren.

Die aktuellen Entwicklungen der kryptografischen Zahlungssysteme ermöglichen den kostengünstigen und schnellen Zahlungsverkehr und leisten damit einen grundlegenden Beitrag zur Abwicklung von elektronischen Zahlungsvorgängen innerhalb von Vertragsagenten. Die im Codius-Framework bereitgestellte Technologie unterstützt die Entwicklung und den Betrieb von sicheren Vertragsagenten und kann die Vertragsführung optimieren.

Wichtige Begriffe auf einen Blick

- **Smart Contract** (dt. Vertragsagent): Automat, der autonom Bedingungen von Verträgen auf deren Einhaltung prüfen und ggf. reaktive Maßnahmen einleiten kann; Automaten folgen einem vorher ausgehandelten Protokoll
- **Smart Oracle**: Laufzeitumgebung, die die Vertragsagenten voneinander und von der darunterliegenden Betriebssystemumgebung isoliert und ihnen über definierte APIs Informationen der Außenwelt (z. B. Währungs- und Aktienkurse, Paketlieferstatus) zur Verfügung stellt
- **kryptografisches Zahlungssystem**: kryptografische Währungen sind virtuelle Zahlungsmittel, die über dafür geschaffene verteilte Datennetzwerke verwaltet und kryptografisch verschlüsselt übermittelt werden [2]

Ursprung

Das Konzept der Vertragsagenten geht auf Nick Szabo zurück [3]. Er argumentierte bereits in den Neunzigerjahren, dass die Formalisierung von Geschäftsbeziehungen und deren Abbildung in Soft- und Hardware die Geschäftslogik und -funktionalität gleichermaßen

vereinfachen und absichern würde.

Smart Contract

Ein Vertragsagent ist ein Vertrag, dessen Bedingungen und Konditionen als Regelwerk im Programmcode formuliert werden. Er ermöglicht automatisiert die Prüfung dieser Bedingungen und führt Vereinbarungen aus, die zwischen den Vertragsparteien explizit vorab definiert wurden. Die Funktionsweise eines Vertragsagenten wird nachfolgend beispielhaft anhand der monatlichen Lieferung einer Ware zwischen zwei Vertragsparteien (Händler und Kunde) beschrieben. Der Programmcode prüft dabei zu definierten Zeitpunkten den monatlichen Geldeingang auf einem für beide Parteien unzugänglichen Konto sowie den Lieferstatus der Ware. Sollte eine der Parteien ihren Pflichten innerhalb des abgestimmten Karenzzeitraums nicht nachkommen, setzt der Vertragsagent als neutraler Vermittler die zwischen beiden Parteien zuvor definierten Maßnahmen um: Im Falle einer Vertragsverletzung in Form eines Lieferausfalls durch den Händler erhält der Kunde automatisch das Geld erstattet. Im umgekehrten Fall erhält der Händler per E-Mail eine Warnung und versendet seine Ware nicht. Die Vertragsautomatisierung ermöglicht eine verbesserte Absicherung der beteiligten Vertragspartner und optimiert die Vertragsführung. Durch die Integration komplexerer Prüfungen und die Involvierung weiterer Parteien können Steuerung und Überwachung der Vertragsabwicklung weiter optimiert werden.

Ein Vertragsagent wird durch Vertragsautoren (z. B. Softwareentwickler) entweder individuell neu entwickelt oder durch Einbeziehung vorgefertigter Vertragsmodule anderer Autoren konfiguriert. Die Vertragsautoren sollten unparteiisch und nicht persönlich von den Bedingungen des zu entwickelnden Vertrags betroffen sein. Nach Fertigstellung der Entwicklung ist die Abnahme des Programmcodes oder der Konfiguration durch technisches Personal der Vertragsparteien erforderlich – so wird sichergestellt, dass alle verpflichtenden Vereinbarungen festgelegt sind. Vorgefertigte und behördlich zertifizierte Vertragsmodule, die mit vorgefertigten traditionellen Vertragsklauseln vergleichbar sind, können diesen Prozess zukünftig deutlich vereinfachen. Die eindeutige Identifizierung und der Manipulationsschutz des Vertragsagenten werden u. a. durch deterministische Kompilation und kryptografische Hash-Funktionen sichergestellt.

Das Konzept der Vertragsagenten wurde in der Fachliteratur wiederholt beschrieben und diskutiert, dennoch hat es bisher keine breite Anerkennung gefunden. Gründe hierfür sind die noch fehlende Akzeptanz bei den involvierten Vertragsparteien, offene rechtliche Fragen, fehlende Standardisierung und die noch nicht weit genug entwickelte technische Infrastruktur. Neue Entwicklungen im Bereich der kryptografischen Zahlungssysteme und die heranwachsende technisch versierte Generation (so genannte Digital Natives) könnten

aber zur breiteren Akzeptanz dieser Technologie beitragen.

Einsatzszenarien für Vertragsagenten

Die Einsatzszenarien für Vertragsagenten sind vielfältig und entwickeln sich stetig weiter. Die folgende Auflistung stellt exemplarische Anwendungsmöglichkeiten dar:

- **Treuhandvertrag** (engl. Escrow): Als neutraler Vermittler kann ein Vertragsagent den Austausch von vorzugsweise digitalen Produkten (Content wie Software, E-Books oder Musiktitel) zwischen zwei oder mehreren Beteiligten überwachen und absichern.
- **Schwarmfinanzierung** (engl. Crowdfunding): Durch Crowdfunding finanzierte Projekte können durch Vertragsagenten als digitale Treuhänder abgesichert werden. Die Finanzierungseinlagen werden dabei auf einem nur dem Vertragsagenten zugänglichen Konto verwaltet und nur bei Einhaltung der definierten Projektkriterien an den Initiator weitergeleitet. Mangelhafte Projekte, die nicht begonnen werden oder nicht die vereinbarte Qualität liefern, können anhand implementierter Vertragsregeln identifiziert werden und zu Erstattungen der Finanzierungseinlagen führen. Ein demokratisches Abstimmungssystem durch die Investoren des Projekts ermöglicht indes die objektive Bewertung der Projektqualität.
- **SLA-Überwachung**: Ein Vertragsagent kann die Überwachung von Service-Level-Agreements (SLAs) übernehmen. Über einen Sensor (http-get) könnte entsprechend eines definierten Prüfintervalls getestet werden, ob ein Webserver verfügbar ist. Der Vertragsagent führt eine Statistik über die vom Sensor ermittelten Statuswerte und wertet diese aus. Bei Erreichen der garantierten Mindestverfügbarkeit gibt der Vertragsagent die volle Zahlung des täglichen Mietsatzes an den Anbieter frei. Bei Unterschreiten der garantierten Mindestverfügbarkeit wird der Anbieter über das Problem informiert und dem Mieter ein vorab definierter Nachlass auf den täglichen Mietsatz gewährt.
- **Auktionen für digitale Güter**: Ein Vertragsagent kann die Geschäftslogik einer Auktion von digitalen Gütern ausführen. Die automatische Lieferung erfolgt auf elektronischem Weg, sofern der Vertragsagent die vollständige Kontrolle über das digitale Produkt (Content) besitzt.

Projekt Codius

Das Open-Source-Projekt „Codius“ [4] wurde von der Firma Ripple Labs mit Sitz in San Francisco initiiert, die auch das kryptografische Zahlungsnetzwerk „Ripple“ [5] betreibt. Codius ist ein plattformunabhängiges Open-Source-Framework zur Entwicklung und zum Betrieb von Vertragsagenten. Der Projektname leitet sich aus dem Begriff „(Programm-)Code“ und dem lateinischen Wort für Recht (lat. ius) her und bedeutet frei übersetzt

„Code-Recht“.

Das im Juli 2014 von Ripple Labs veröffentlichte Konzeptdokument [6] beschreibt einen neuartigen und flexiblen Ansatz für Vertragsagenten, deren Programmausführung innerhalb spezialisierter Hostingumgebungen (Smart Oracles) erfolgt. Bei der Entwicklung des Frameworks wurde explizit auf einen flexiblen und generischen Ansatz geachtet, der die Interoperabilität zwischen heterogenen Netzwerkteilnehmern (z. B. den Zahlungsnetzwerken Ripple, Bitcoin und Informationsanbietern für Lieferstatus und Währungskurse) und unterschiedlichen Programmiersprachen durch Entkopplung ermöglicht.

Durch Codius können zukünftig verschiedene Vertragsagenten in einer beliebigen Programmiersprache kodiert werden, sofern die Bedingungen und Auswirkungen eindeutig und deterministisch formuliert vorliegen und diese von der Hostingumgebung zur Verfügung gestellt werden. Das Codius-Projekt befindet sich noch im frühen Prototypenstatus und stellt aktuell ein auf Node.js basierendes Framework und eine Hostingumgebung zur Verfügung.

Smart Oracle

Die Hostingumgebung stellt einem Vertragsagenten über definierte APIs Informationen der Außenwelt zur Verfügung, z. B. für Dateisystem- und Internetzugriffe oder den Zugriff auf kryptografische Funktionen. Diese APIs können auch als Sensorenframework verstanden werden. So wie ein Kaffeeautomat mittels Sensoren prüft, ob genügend Münzen über den Geldschlitz eingeworfen wurden oder ob Kaffeebohnen nachgefüllt werden müssen, bezieht der Vertragsagent über die APIs der Hostingumgebung die Informationen, die er als Entscheidungsgrundlage benötigt.

Die Ausführung des (aus Sicht der Hostingumgebung) potenziell nicht vertrauenswürdigen Programmcodes eines Vertragsagenten erfolgt in einer Software-Sandbox, die verschiedene Vertragsagenten voneinander sowie von der Hostingumgebung abschirmt und die nutzbaren APIs kontrolliert (**Abb. 1.1**).

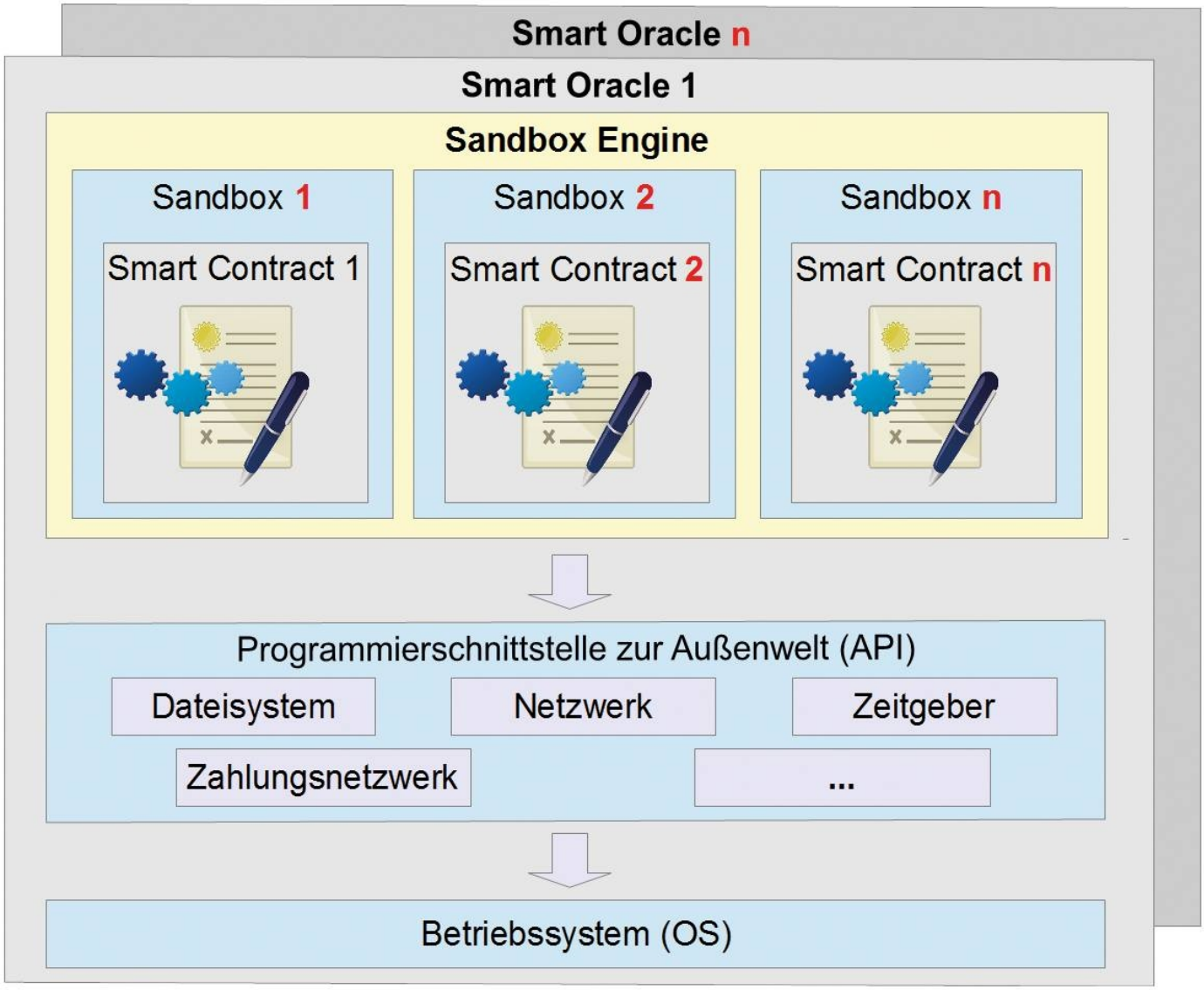

Abbildung 1.1: Übersicht über die Codius-Architektur

In der Prototypenfassung wird die Sandbox durch das gleichnamige Node.js-Modul emuliert und zukünftig durch die plattformunabhängige Sandbox des Google Native Client (NaCl) [7] ersetzt. Der Native Client ermöglicht die Ausführung von Programmcodes beliebiger Programmiersprachen und isoliert dessen Zugriffe auf andere Systembereiche (z. B. Prozesse oder Arbeitsspeicher). Durch diese Technologie können mehrere Ausführungsprozesse von Vertragsagenten parallel auf der gleichen Betriebssysteminstanz voneinander abgeschirmt betrieben werden. Ein alternativer Lösungsansatz zur Isolation des ausgeführten Programmcodes, allerdings mit einem deutlich höheren Bedarf an Systemressourcen, ist der Einsatz von virtuellen Maschinen.

Hostingumgebungen für Vertragsagenten können in Abhängigkeit der Sensibilität des Vertrags entweder auf einem einzelnen Serversystem oder in verteilten Serversystemen, beispielsweise in der Cloud, betrieben werden. Zum Schutz der Vertragsausführung vor Manipulationen ist es erforderlich, dass alle Vertragsparteien dem Anbieter der Hostingumgebung vertrauen und dieser unparteiisch agiert.

Windows für Codius-Entwicklung vorbereiten

Für die Entwicklung eigener Vertragsagenten ist ein aktuelles Linux-System notwendig, das mit folgender Anleitung in einer virtuellen Umgebung unter Windows bereitgestellt werden kann.

Die Installation der Betriebsumgebung ist interessant und für Windows-Benutzer ohne tiefgehende Unix-Kenntnisse möglich, wobei der grundlegende Umgang mit der Kommandozeile vorausgesetzt wird.

Zum Ausschluss ungewünschter Seiteneffekte wird empfohlen, maximal eine Virtualisierungssoftware auf dem Host zu installieren und die Installation auf einem Testsystem vorzunehmen.

1. Installation der Virtualisierungssoftware VirtualBox für Windows [8] unter Verwendung der Standardeinstellungen. Aktuell unterstützt das Image der Betriebsumgebung nicht die Virtualisierungssoftware Hyper-V.
2. Installation von Vagrant für Windows [9]: Vagrant stellt dabei u. a. die Betriebsumgebung als Image zum Download bereit und bindet dieses zur Ausführung und Administration in VirtualBox ein.
3. Installation des SSH-Terminal Clients PuTTY für Windows [10]
4. Neustart des Betriebssystems
5. Anlegen eines Ordners zur Speicherung der Vagrant-Definition und als Arbeitsverzeichnis zum einfachen Austausch von Dateien (z. B. Quellcodes von Vertragsagenten) zwischen virtueller Maschine und lokalem System (z. B. *C:\Temp\Vagrant-Ubuntu-Codius*). Damit können Dateien lokal im Windows bearbeitet werden und einfach der virtuellen Umgebung zur Verfügung gestellt werden. Der gemeinsame Ordner wird im Linux-System unter *„/vagrant“* bereitgestellt (Befehl zum Wechsel in das Verzeichnis *„cd /vagrant“*).
6. Der Download und Start der Betriebsumgebung erfolgt durch Ausführung der nachfolgenden Befehle in der Kommandozeile im Kontext des zuvor angelegten Ordners

   - Befehl zur Initialisierung der virtuellen Betriebumgebung: *vagrant init ubuntu/trusty32*
   - Optional : Einige Vertragsagenten benötigen zur Ausführung eine Http-Portfreischaltung aus der virtuellen Maschine zum Host. Hierfür kann eine Portweiterleitung in der zuvor angelegten Vagrant-Konfigurationsdatei aktiviert werden.

Die Konfigurationsdatei enthält bereits eine passende Vorlage inklusive Beschreibung (siehe: *# config.vm.network "forwarded_port", guest: 80, host: 8080*), bei der nur noch das Kommentar „#“ zur Aktivierung entfernt und der Guest-Port auf den Port des Vertragsagenten (Standard: 8000) gesetzt werden muss.

- Start der virtuellen Maschine: *vagrant up*
  Dieser Befehl initiiert bei der ersten Ausführung den Download der Betriebsumgebung als Image (ca. 357 MB). Nachfolgend erzeugte virtuelle Maschinen erfordern nicht erneut diesen Download, sondern kopieren bzw. referenzieren bereits vorhandene lokale Daten.
- Bereitstellen einer SSH-Verbindung zur virtuellen Maschine und Ausgabe der Verbindungsdaten: *vagrant ssh*

Der Shutdown der virtuellen Maschine erfolgt mit dem Befehl „vagrant halt“. Die virtuelle Maschine kann alternativ über den VirtualBox-Manager administriert und verwendet werden.

Die virtuelle Betriebsumgebung ist nun auf dem lokalen System einsatzbereit und muss nun noch um die Codius-Entwicklungsumgebung ergänzt werden. Die Anmeldung am Terminal erfolgt durch eine SSH-Verbindung mit der Software PuTTY unter Verwendung des Benutzernamens und des Passworts „vagrant“.

Die weitere Installation der Codius-Entwicklungsumgebung (inkl. Projektbeispiele) erfolgt nun in der virtuellen Maschine unter Verwendung der vom Codius-Team bereitgestellten Installationsanleitung [11].

Zur Verifikation der erfolgreichen Installation sollte der in der Installationsanleitung beschriebene Self-Test und das Hello-World Beispiel ausgeführt werden. Das Hello-World Beispiel sollte dabei im Kontext des zuvor beschriebenen gemeinsamen Ordners heruntergeladen und ausgeführt werden, da dieses dadurch leicht unter Windows modifiziert und unter Linux ebenso einfach nach einer Anpassung ausgeführt werden kann.

Sollte bei der Ausführung von Codius die fehlende Bibliothek *„seccomp“* als Fehler gemeldet werden, kann dieser durch Ausführung des Updates „sudo apt-get install libseccomp-dev“ behoben werden.

Zur Durchführung vereinfachter Dateiverwaltungsvorgänge unter Linux wird die Installation des Midnight-Commander angeraten, der mit dem Befehl *„sudo apt-get install mc“* installiert und dem Befehl *„mc“* ausgeführt werden kann.

Ein lauffähiges Beispiel

Nach erfolgter Installation kann direkt mit der Entwicklung eines Vertragsagenten begonnen werden. Besonders eignet sich Microsoft Visual Studio Code [12] dafür. Das folgende Beispiel beschreibt die wichtigsten Komponenten (Listing 1.1).

```
/*
Dieser Miniagent simuliert die Verwendung der Zeit als Sensor.

Anhand des aktuellen Minutenwerts werden zwei Zustände abgeleitet:
Gerade Minutenzahl = Zustand 1 (gültig)
Ungerade Minutenzahl = Zustand 2 (ungültig)

Die Logausgabe simuliert die vereinbarte Reaktion auf den Zustand.
*/
var minutes = new Date().getMinutes();
var state = minutes % 2;
if (state === 0) {
  console.log('Zeitstatus gültig!');
} else {
  console.log('Zeitstatus ungültig!');
}
```

Listing 1.1: Quellcode von „contract.js“

Der Vertragsagent teilt der Hostingumgebung wichtige Schnittstelleninformationen über die Manifest-Datei *codius-manifest.json* mit:

```
{
  "name": "contract-sample-timer",
  "apis": [],
  "main": "contract.js"
}
```

Die aktuell in der Manifest-Datei verfügbaren Eigenschaften sind:

- **name**: Name des Vertragsagenten
- **apis**: Liste der APIs, die der Vertragsagent benutzen wird
- **main**: der Einsprungspunkt zum Start des Agenten
- **modules**: Liste eingebundener Module, die sich jeweils im Unterverzeichnis *codius_modules/<modulname>* befinden müssen; dies können einzelne JavaScript-Dateien, Teilagenten mit eigenem Manifest oder vollwertige Node.js-Pakete sein
- **files**: Liste verwendeter Dateien, die sich jeweils im Stammverzeichnis der Manifest-Datei oder in Unterverzeichnissen befinden müssen; ein direkter Zugriff auf Dateistrukturen außerhalb der Sandbox ist nicht möglich

Die Funktionsfähigkeit des Beispielagenten kann leicht überprüft werden. Hierzu führen sie in der Shell der virtuellen Maschine folgenden Befehl aus: *“cd /vagrant/Ordnername unter Windows”*. Der Befehl *codius run* führt den Vertragsagent in der Kommandozeile aus.

Ein weiteres, etwas komplexeres Beispiel („contract-sample-status“) befindet sich im Downloadbereich unter [13]. Dieser Vertragsagent wird als Webapplikation über den

Befehl *codius serve* gestartet. Über die Eingabe des im Kommandofenster angezeigten URLs kann die passende Statusseite im Webbrowser abgerufen werden.

Eine wachsende Community

Das Codius-Team stellt unter [1] ein Forum und einen Chat zur Verfügung, die den gegenseitigen Austausch zwischen Einsteigern und Fortgeschrittenen ermöglichen. Wie bei anderen Open-Source-Projekten kann auch hier jeder Interessierte eigene Ideen beisteuern und aktiv mitarbeiten.

Fazit und Ausblick

Schon in dieser frühen Phase zeigt das Codius-Projekt Ansätze, die zur Standardisierung der Entwicklung von Vertragsagenten führen können, und legt somit Grundlagen zur Vertrauensbildung in diese neue Technologie. Die Ziele des Projekts sind anspruchsvoll und bieten perspektivisch Potenzial für die weitere Beschäftigung mit diesem Thema.

Vertragsagenten bleiben eine interessante Herausforderung in den Bereichen Technik, Wirtschaft und Recht und bieten großes Potenzial für neue Innovationen. Vielleicht nimmt die weitere Entwicklung sogar Einfluss auf die Spezialisierungsrichtung des Informatikers mit juristischem Schwerpunkt („Rechtsinformatiker“). Viele Fragen bleiben aktuell noch offen, unter anderem:

- Wie erfolgt die Identifizierung der Vertragsteilnehmer? Wie werden diese an den Vertragsagenten gebunden?
- Wie erfolgt die Kopplung des Vertragsagenten an den zugehörigen Vertrag, z. B. in Bezug auf Laufzeit und Kündigung?
- Wie kann ein Vertragsagent effektiv gegen eine manipulierte Hostingumgebung abgesichert werden?
- Wie steht es um die Rechtssicherheit in Bezug auf die im Vertragsagenten definierten Maßnahmen? Welche Rechtsmittel haben die Beteiligten, wenn Maßnahmen aufgrund von Fehlern im Vertragscode ausgelöst werden? Was passiert, wenn Beteiligte unberechtigt gegen Maßnahmen gerichtlich vorgehen?

Links & Literatur

[1] Codius: http://codius.org

[2] Graff, Christof; Zscherp, Matthias: „Open Source für Onlinezahlungen“, in Windows Developer 9.2014

[3] Smart Contracts: http://szabo.best.vwh.net/smart_contracts_idea.html

[4] Codius Source: https://github.com/codius/codius

[5] Ripple: https://ripple.com

[6] Codius Whitepaper: https://github.com/codius/codius/wiki/Smart-Oracles:-A-Simple,-Powerful-Approach-to-Smart-Contracts

[7] Google Native Client: https://developer.chrome.com/native-client

[8] VirtualBox: https://www.virtualbox.org/wiki/Downloads/

[9] Vagrant: http://www.vagrantup.com/downloads/

[10] PuTTY: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[11] Codius npm: https://www.npmjs.com/package/codius

[12] Microsoft Visual Studio Code: https://code.visualstudio.com

[13] https://entwickler.de/press/shortcuts/plattformsicherheit-166041.html

2 Plattformsicherheit per Kontrollchip

Das zweite Kapitel beschäftigt sich mit der Funktionsweise und den Einsatzmöglichkeiten eines „Trusted Platform Module“(TPM). Eine Einführung in die Anwendungsgebiete und das Management eines TPM sowie die Darstellung der Pros und Kontras dieser Technologie sollen ein besseres Verständnis vermitteln und damit eine individuelle Einschätzung der Technologie ermöglichen. Darüber hinaus werden die für ein rudimentäres Verständnis erforderlichen kryptografischen Grundlagen erörtert.

Ein Trusted Platform Module (TPM) ist eine Hardwarekomponente in Form eines diskreten Schaltkreises oder eines voll in die Hauptplatine integrierten Bausteins. Generell betrachtet ist ein TPM ein sicherer lokaler Speicher für digitale Schlüssel und eignet sich darüber hinaus zur internen Ausführung kryptografischer Operationen. Somit ist ein TPM in der Lage, eine auf digitalen Schlüsseln und Zertifikaten basierende vertrauenswürdige IT-Plattform zu realisieren. Mittlerweile sind beinahe alle neueren Computer und Tablets mit einem TPM-Chip ausgestattet, sodass die Vorteile einer vertrauenswürdigen Plattform verwirklicht werden können. Doch es gibt auch kritische Stimmen, da ein TPM beispielsweise den Status eines Systems speichern und unter bestimmten Umständen für Dritte abrufbar machen kann (Remote-Attestation).

Standards und Ziele des TPM

Das TPM realisiert die durch ein Konsortium namens TCG (Trusted Computing Group) spezifizierte Anforderungen an eine vertrauenswürdige IT-Plattform. Nach eigenen Angaben handelt es sich bei TCG um eine nicht auf Gewinn ausgerichtete Organisation, die sich als Herausgeber internationaler, offener Standards versteht [1]. Derzeit untergliedert sich die gesamte TPM-Spezifikation in drei Teile: „Design-Principles“, „TPMStructures“ und „Commands“.

Diese offenen Standards und Spezifikationen sollen es ermöglichen, sichere IT-Plattformen zu realisieren, die sich unter anderem durch Sicherheitsmechanismen wie einer sicheren Authentifizierung, einer starken Absicherung von Benutzeridentitäten und dem Schutz von kritischen Geschäftsdaten auszeichnen. Ein weiteres Ziel ist die Sicherstellung der Integrität eines Computersystems, sowohl im Online-, als auch im Offlinemodus. Hierbei kann es sich nicht nur um einen klassischen PC, ein Notebook, ein Tablet oder um ein unternehmensweites Netzwerk handeln, sondern ebenso um Systeme mit Cloud-Anbindung.

Architektur und Sicherheit

Im Prinzip ist ein TPM eine auf Hardware basierende Funktionseinheit für das sichere Speichern von digitalen Artefakten, also beispielsweise Schlüsseln, Passwörtern oder Zertifikaten. Darüber hinaus ist ein TPM dafür ausgelegt, kryptografische Operationen intern sicher auszuführen (Crypto Processor). Das bedeutet, für die Abarbeitung der erforderlichen Algorithmen finden weder externe Systemspeicher, noch externe Bibliotheken, die über das Betriebssystem verfügbar sind, Verwendung. Damit ist sichergestellt, dass das TPM nicht durch Tools oder Schadsoftware manipuliert werden kann.

Auf den ersten Blick ist ein TPM direkt mit einer Chipkarte [2] vergleichbar. Der signifikante Unterschied besteht jedoch darin, dass ein TPM an eine Plattform bzw. ein System gebunden ist, wogegen eine Chipkarte in aller Regel einen einzigen Besitzer hat.

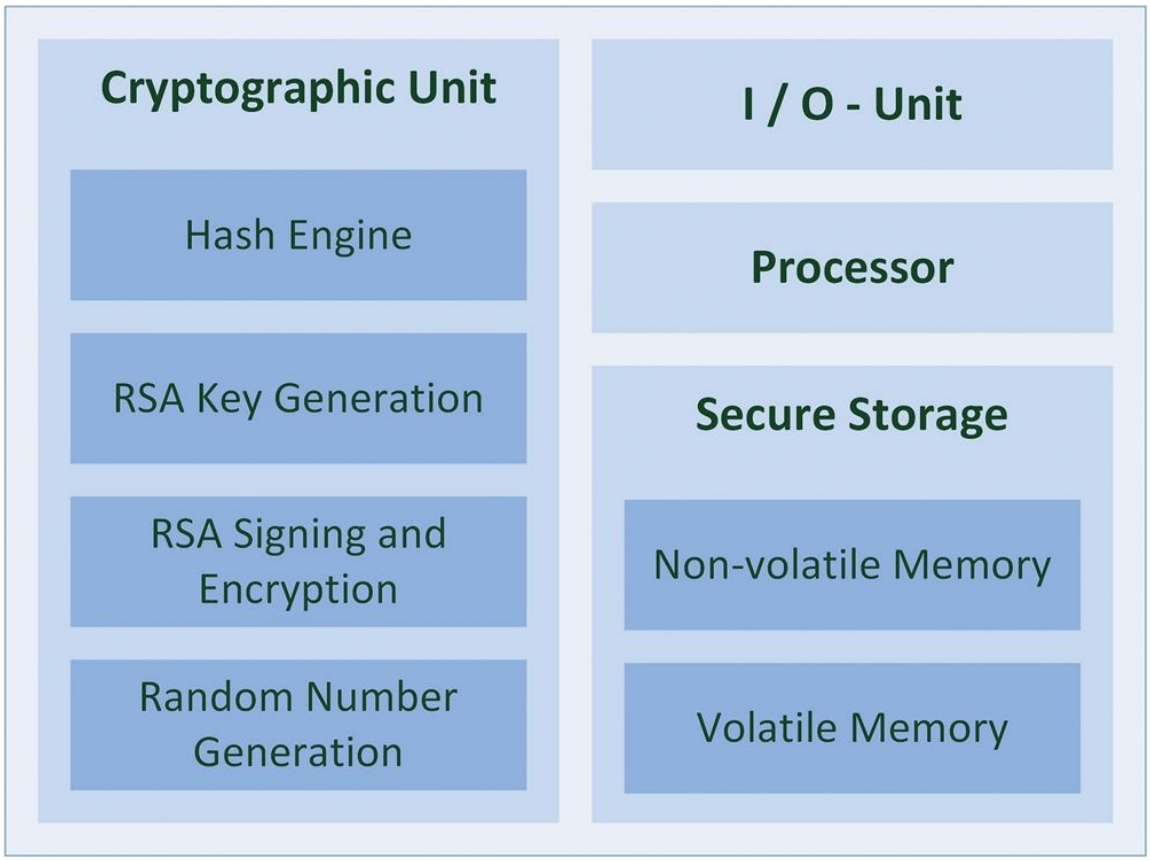

Abbildung 2.1: Komponenten eines TPM

Abbildung 2.1 zeigt die typischen Komponenten eines TPM, wobei der klassische Aufbau eines Mikrocontrollers an Prozessor (CPU), Speicher sowie Ein- und Ausgabeeinheit (I/O) zu erkennen ist. Der Speicher ist als Secure Storage ausgelegt, sodass von außen nicht direkt darauf zugegriffen werden kann. Weiterhin wird zwischen einem flüchtigen und einem nicht flüchtigen Speicher unterschieden. Der nicht flüchtige Speicher (Non-volatile Memory) ist für das persistente Speichern der beiden wichtigsten Schlüssel vorgesehen: „Endorsement Key“ und „Storage Root Key“. Die Konfigurationsdaten für die Plattform (Platform Configuration Registers) sowie via System generierte Schlüssel (Loaded Keys)

werden im „Volatile Memory“ abgelegt. Die kryptografische Einheit stellt folgende Funktionalitäten bereit: Generieren von Hash-Werten und asymmetrischen RSA-Schlüsseln, Signier- und Verschlüsselungsfunktionen, sowie Generieren von Zufallszahlen.

Eine Zertifizierung gemäß internationaler Standards (EAL, NIST etc.) gilt als allgemeiner Nachweis für die Erfüllung und Einhaltung der von der Trusted Computing Group vorgeschriebenen restriktiven Sicherheitsanforderungen für ein TPM. Hierbei ist insbesondere der Nachweis zu erbringen, dass das Modul nicht nur gegen Softwareattacken geschützt, sondern auch in hohem Maße gegen mechanische Angriffe resistent ist. So ist beispielsweise der Nachweis zu erbringen, dass der Speicherinhalt eines diskreten TPM-Schaltkreises sofort zerstört wird, sollte der Versuch unternommen werden, diesen mechanisch zu öffnen und so den Chip freizulegen. Obwohl der in **Abbildung 2.2** gezeigte TPM-Schaltkreis äußerlich nicht von einem Standardbaustein zu unterscheiden ist, erfüllt der interne Aufbau die strengen Kriterien, die für die Klassifizierung „Tamper Resistant“, also Manipulationssicherheit, spezifiziert sind.

Abbildung 2.2: TPM-Chip von Infineon

Anwendungsspektrum

Auf ein TPM kann immer dann zurückgegriffen werden, wenn es gilt, sichere kryptografische Operationen durchzuführen und/oder ein sicheres Speichermedium bereitzustellen. Aufgrund des geringen Stückpreises kann man einem TPM gegenüber teurer Zusatzhardware den Vorzug geben, sofern keine Einschränkungen durch andere Randbedingungen bestehen. Falls beispielsweise ein hoher Durchsatz bezüglich

kryptografischer Funktionen gefordert ist, oder sehr große Datenmengen hochsicher zu speichern sind, kommt man um ein HSM (Hardware Security Module) nicht herum. Ein HSM liegt entweder als PCI-Erweiterungssteckkarte für einen PC-Slot vor, oder ist als Netzwerkkomponente im 19”-Format erhältlich.

Das Anwendungsspektrum eines mit einem aktiven TPM ausgestatteten Systems erstreckt sich derzeit im Wesentlichen auf nachstehend aufgeführte Technologien.

Schutz elektronischer Daten (Data Protection): Realisierung durch symmetrische, asymmetrische oder hybride kryptografische Verfahren. Hierzu werden primär digitale Schlüssel benötigt, die sicher generiert und sicher gespeichert werden müssen. Dies setzt ein probates Schlüsselmanagement voraus, das beispielsweise auf einem TPM basieren kann. Neben dem klassischen Verschlüsseln und Signieren elektronischer Daten bzw. Dateien sind in diesem Kontext so genannte selbstverschlüsselnde Speichermedien (Self-Encrypting Drive/SED), sowie proprietäre Verschlüsselungslösungen wie „BitLocker“ von Microsoft zu nennen.

Benutzer und Systemauthentifizierung: Die für die Verifikation einer digitalen Identität erforderlichen Schlüssel und Zertifikate können ebenfalls sicher in einem TPM gespeichert werden. Allgemein betrachtet können die für die Authentifizierung von Benutzern und Systemen benötigten Berechtigungen („Credentials“) durch ein TPM geschützt werden und stehen unter anderem für den sicheren Kommunikationsaufbau von VPN, Drahtlosnetzwerken oder Cloud Computing zur Verfügung. Auch wenn sich hier erneut der Vergleich mit einer Smartcard anbietet, muss man sich sofort ins Gedächtnis zurückrufen, dass ein TPM an eine Plattform, bzw. ein System gebunden ist, und nicht an einen Benutzer („Card-Holder“).

Plattformbescheinigung (Remote-Attestation): Dieses Feature wird bereits zu Beginn des Boot-Prozesses aktiv. Schon in diesem frühen Stadium wird hierbei überprüft, ob die Systemintegrität gewahrt ist. Das Ergebnis wird sicher gespeichert und kann von autorisierten Dritten online abgefragt werden. Weiterhin kann kontrolliert werden, welche Softwarekomponenten innerhalb der Plattform zugelassen sind und ausgeführt werden können.

Der Fokus liegt eindeutig auf Analyse und Protokollierung des Status einer Plattform, d. h. das aktive Verhindern von Manipulationen und schädlichen Eingriffen durch Malware ist nicht Sache des TPM. Hierfür können andere Mechanismen eingesetzt werden, die auf das Ergebnis der Plattformüberprüfung zurückgreifen.

Kritikpunkte und Privatsphäre

Wie bereits erwähnt, haben TPMs Einzug in nahezu alle neueren Computer, Tablets und

sogar Smartphones erhalten. Auf den ersten Blick erscheint das mit einem TPM angestrebte Ziel, eine vertrauenswürdige Plattform und restriktive Sicherheitsrichtlinien (Security Policies) realisieren zu können, äußerst erstrebenswert. Kritiker melden sich allerdings dahingehend zu Wort, dass der Einsatz der TPM-Technologie an einige rudimentäre Bedingungen geknüpft sein müsste. Die nachstehende Aufstellung gibt diesbezüglich einen minimalen Anforderungskatalog mit den wichtigsten Kriterien wieder:

- TPM ist nicht aktiviert im Auslieferungszustand eines Systems
- Aktivierung ist freiwillig und nicht verpflichtend
- Ohne Aktivierung ergeben sich keine funktionellen Einschränkungen
- Deaktivierung muss jederzeit uneingeschränkt möglich sein
- Intuitive Managementsoftware zur Verwaltung des TPM
- Automatische Analysen im Hintergrund müssen für den Anwender transparent sein
- Gespeicherte Informationen müssen im Klartext einsehbar sein
- Durch Dritte abgerufene Informationen (Remote-Attestation) müssen protokolliert werden

Die beiden erstgenannten Kardinalforderungen, dass ein TPM im Auslieferungszustand nicht aktiviert ist und ein System mit integriertem TPM auch weiterhin ohne Aktivierung keinen Einschränkungen unterworfen sein darf, erfüllt die aktuelle „TPM-Spezifikation-1.2“. Dies sieht allerdings mit der Version 2.0 etwas anders aus, hier soll nämlich alleinig das Betriebssystem die Aktivierung und die Kontrolle über das TPM übernehmen.

Zudem ist es ein grundsätzliches Ziel von Softwareherstellern sowie Anbietern von Apps und Medien (wie Musikdateien), auf die in einem internen Speicher des TPM erfassten Daten im Rahmen einer Remote-Kommunikation zyklisch zuzugreifen und diese auszuwerten. Hiermit ließen sich Raubkopien erkennen und es wäre kein aufwändiger Schutz in Form eines Dongle oder ähnlicher Hardware notwendig. Der wesentliche Unterschied besteht allerdings darin, dass ein Dongle und die korrespondierende Software gemeinsam auf ein anderes System übertragen werden können, ohne dass eine neue Lizenz beschafft werden müsste. Ein TPM kann jedoch im Gegensatz zu einem Dongle nicht einfach von einem System abgezogen und in ein anderes System eingesteckt werden. Das hat zur Konsequenz, dass die Lizenzierung bzw. Aktivierung beispielsweise nach einem Systemcrash erneut durchlaufen werden müsste.

Nachfolgend sind die mit einem aktivierten TPM realisierbaren Anwendungs- und Einsatzszenarien auszugsweise zusammengestellt (ohne Wertung und Anspruch auf

Vollständigkeit):

- Pre-Boot-Systemintegritätstests und Schutz vor Schadsoftware
- Plattformbescheinigung (Remote-Attestation)
- Kontrolle des Computers durch Dritte aus der Ferne
- Unbemerkter Zugriff auf das TPM und dessen Daten durch Dritte möglich
- Bindung einer aktivierten bzw. freigeschalteten Software an eine Plattform (System)
- Aufzeichnung von Benutzeraktivitäten
- Kontrolle (durch Dritte) darüber, welche Software auf einer Plattform ausgeführt werden darf, Einschränkung auf bestimmte Applikationen, also beispielsweise Blockierung von Open-Source-Software
- Bestimmte Anwendungen haben nur mit einem TPM den vollen Funktionsumfang (BitLocker etc.)
- Kopierschutz für Software und digitale Medien
- Musikdateien, oder anderes digitales Eigentum nicht ohne Weiteres auf andere Plattform übertragbar, obwohl rechtmäßig erworben
- Authentifizierung einer Plattform und nicht eines Benutzers
- Zugriff auf TPM-Schlüssel via Hintertürchen (OS-Hersteller, NSA etc.); Gefahr von Backdoor-Attacken demzufolge nicht auszuschließen
- Nichtübertragbarkeit geschützter Datenobjekte (Schlüssel), daher sehr problematische Backup- und Recovery-Strategien
- Letztendlich könnte der Wechsel auf ein anderes System erschwert oder nahezu unmöglich gemacht werden, oder aufgrund zu hoher Kosten unzumutbar sein

Obwohl die einzelnen Aspekte von den diversen Lagern naturgemäß völlig konträr gesehen und bewertet werden, sollte sich doch dahingehend ein gemeinsamer Konsens herbeiführen lassen, dass die Aktivierung des TPM auf freiwilliger Basis ausschließlich durch den Systemeigentümer (Benutzer) vorgenommen werden kann und der Einzelne somit sein Maß an Privatsphäre selbst definieren kann.

Dass die Realität etwas anders aussehen kann, stellt Microsoft unter Beweis, indem es das aktuelle Windows Phone 8.1 bereits mit einem aktivierten TPM ausstattet; und zwar schon bevor es der Käufer das erste Mal in seinen Händen hält. Gemäß einer Publikation von Microsoft [3] ist das TPM ein wichtiges Securityfeature der aktuellen Windows-Betriebssysteme. Ein Standardtool zur Deaktivierung ist auf einem Windows Phone 8.1 derzeit nicht installiert und steht für diese Plattform auch nicht im Online-Store zum Download bereit.

Zustände und Zustandsänderungen

Bevor ein TPM verwendet werden kann, muss es zunächst initialisiert werden. Im Anschluss daran kann das TMP in einen bestimmten Zustand versetzt werden. Grundsätzlich handelt es sich dabei um einen der folgenden primären Zustände:

- Das TPM ist nicht einsatzbereit.
- Das TPM ist deaktiviert, und der Besitz des TPM wurde nicht übernommen.
- Das TPM ist einsatzbereit.

Weist das TPM den Status „einsatzbereit“ auf, ist eine Reihe von weiteren Aktionen verfügbar (Tabelle 2.1).

Aktion	Beschreibung
TPM vorbereiten	Die Option „TPM vorbereiten“ leitet das Initialisieren ein, d. h. das TPM wird aktiviert und einem Besitzer zugewiesen. Das Initialisieren erfordert einen Neustart des Computers. Nach Neustart und Anmeldung erscheint ein Dialog, der den Status des TPM ausweist und das Speichern des TPM-Besitzerkennworts in eine Datei ermöglicht. Bei dieser Datei handelt es sich um eine XLM-Datei, in der innerhalb des Tags „tpmOwnerData“ die beiden Tags „tpmInfo“ sowie „ownerAuth“ enthalten sind.
TPM ausschalten	Diese Option deaktiviert das TPM. Falls das TPM-Besitzerkennwort bekannt ist, oder eine TPM-Kennwortdatei verfügbar ist, kann die Deaktivierung direkt vorgenommen werden, ansonsten kann das TPM über das BIOS deaktiviert werden.
Besitzerkennwort ändern	Das Besitzerkennwort kann entweder durch die direkte Eingabe eines Kennworts erfolgen, oder ein Assistent generiert ein Besitzerkennwort, wobei die letztgenannte Methode empfohlen wird.
TPM löschen	Hierdurch wird das TPM in einen besitzerlosen Zustand versetzt und die vormals gespeicherten Daten sind nicht mehr verfügbar. Falls das TPM nur temporär nicht aktiv sein soll, ist es empfehlenswert, das TPM lediglich auszuschalten.
TPM-Sperre zurücksetzen	Falls die TPM-Logik eine Manipulation oder einen Angriff „vermutet“, sperrt sich das TPM selbst. Die Sperre kann mit dieser Aktion zurückgesetzt werden.

Tabelle 2.1: Verfügbare Aktionen eines einsatzbereiten TPM

Unter den Windows-Betriebssystemen ist für die Administration eines TPM ein Plug-in für eine Microsoft-Management-Konsole (MMC) verfügbar. Zunächst ist die MMC mit *mmc.exe* zu starten, anschließend unter DATEI | SNAP-IN hinzuzufügen. Das Snap-in kann über „TPM-Verwaltung“ ausgewählt werden. **Abbildung 2.3** zeigt neben den verfügbaren Aktionen den TPM-Status sowie die TPM-Herstellerinformationen eines einsatzbereiten TPM. Optional kann durch das Starten von *tpm.msc* direkt eine Administrationskonsole mit TPM-Snap-in geöffnet werden.

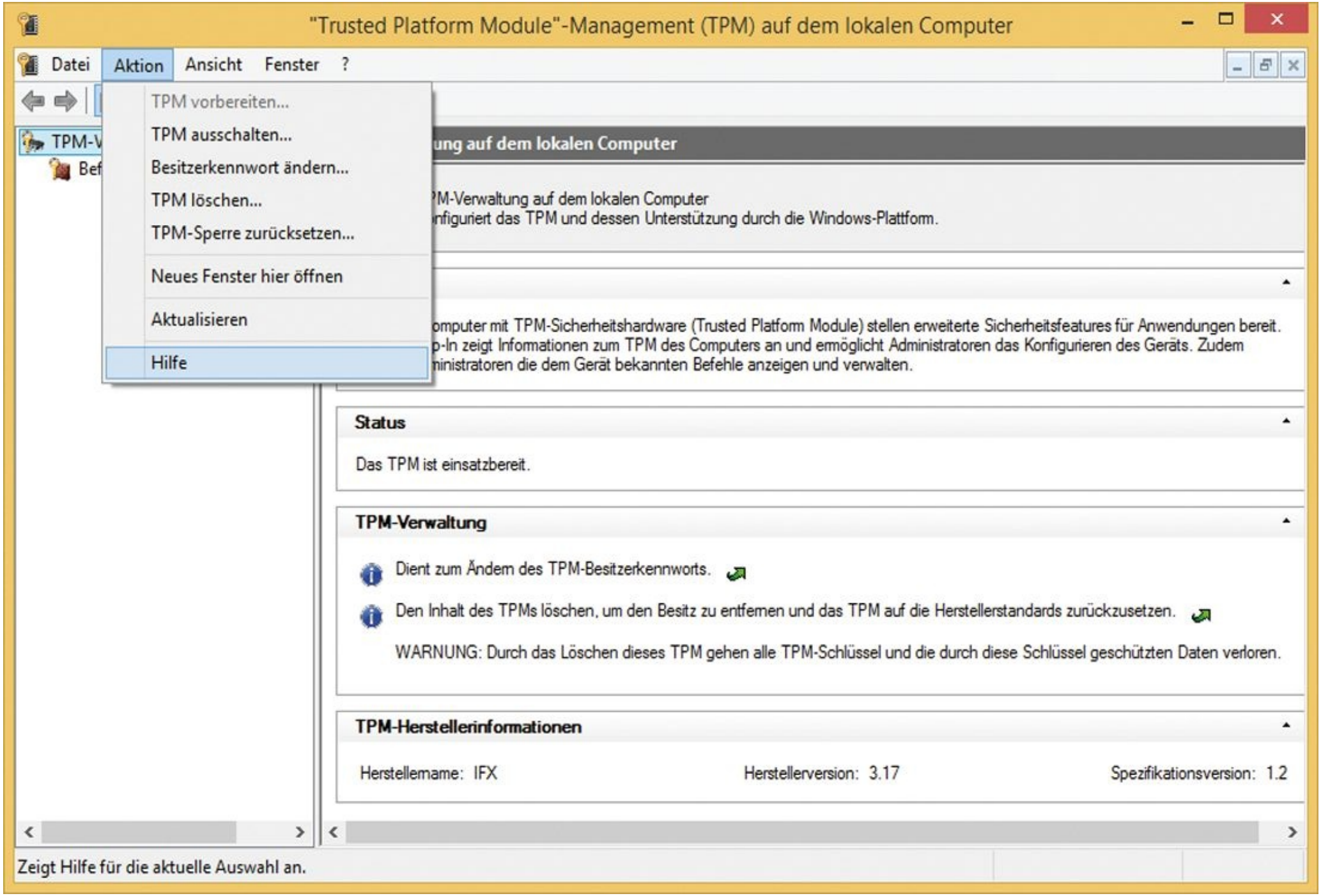
"Trusted Platform Module"-Management (TPM) auf dem lokalen Computer
Datei
Aktion
Ansicht
Fenster
?
TPM vorbereiten...
TPM ausschalten...
Besitzerkennwort ändern...
TPM löschen...
TPM-Sperre zurücksetzen...
Neues Fenster hier öffnen
Aktualisieren
Hilfe
TPM-V
Bef
ung auf dem lokalen Computer
PM-Verwaltung auf dem lokalen Computer
nfiguriert das TPM und dessen Unterstützung durch die Windows-Plattform.
omputer mit TPM-Sicherheitshardware (Trusted Platform Module) stellen erweiterte Sicherheitsfeatures für Anwendungen bereit.
p-In zeigt Informationen zum TPM des Computers an und ermöglicht Administratoren das Konfigurieren des Geräts. Zudem
ministratoren die dem Gerät bekannten Befehle anzeigen und verwalten.
Status
Das TPM ist einsatzbereit.
TPM-Verwaltung
Dient zum Ändern des TPM-Besitzerkennworts.
Den Inhalt des TPMs löschen, um den Besitz zu entfernen und das TPM auf die Herstellerstandards zurückzusetzen.
WARNUNG: Durch das Löschen dieses TPM gehen alle TPM-Schlüssel und die durch diese Schlüssel geschützten Daten verloren.
TPM-Herstellerinformationen
Herstellername: IFX
Herstellerversion: 3.17
Spezifikationsversion: 1.2
Zeigt Hilfe für die aktuelle Auswahl an.

Abbildung 2.3: TPM-Verwaltung

Systemintegration (OS)

Grundsätzlich fungiert das Betriebssystem als Interface zwischen einem TPM und der Applikation, die auf das TPM zugreift. Genauer gesagt liegt ein proprietärer Treiber zwischen dem Betriebssystem und der TPM-Hardware. Welche Befehle für eine Applikation zugelassen oder blockiert sind, kann mit der in **Abbildung 2.4** gezeigten TPM-Befehlsverwaltung festgelegt werden. Zur Änderung des Status ist zunächst der Befehl zu selektieren. Anschließend kann entweder über den Hauptmenüpunkt „Aktion“ oder via rechte Maustaste und Kontextmenü der Status des Befehls zugelassen oder blockiert werden. Hierfür stehen kontextsensitiv die Optionen „Ausgewählten Befehl zulassen“ oder „Ausgewählten Befehl blockieren“ zur Auswahl.

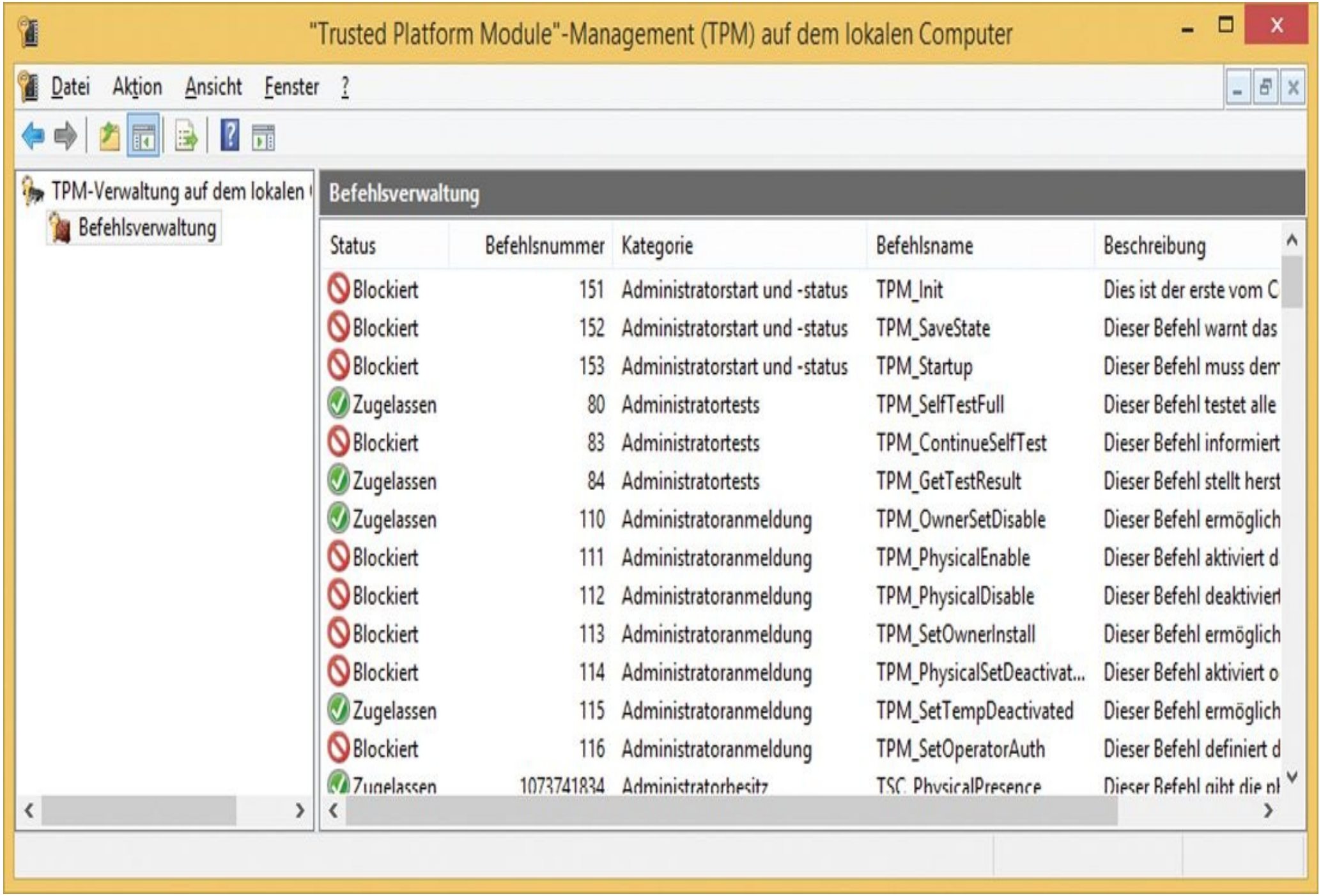
"Trusted Platform Module"-Management (TPM) auf dem lokalen Computer
Datei
Aktion
Ansicht
Fenster
?
TPM-Verwaltung auf dem lokalen
Befehlsverwaltung
Befehlsverwaltung
Status
Befehlsnummer
Kategorie
Befehlsname
Beschreibung
Blockiert 151 Administratorstart und -status TPM_Init Dies ist der erste vom C
Blockiert 152 Administratorstart und -status TPM_SaveState Dieser Befehl warnt das
Blockiert 153 Administratorstart und -status TPM_Startup Dieser Befehl muss dem
Zugelassen 80 Administratortests TPM_SelfTestFull Dieser Befehl testet alle
Blockiert 83 Administratortests TPM_ContinueSelfTest Dieser Befehl informiert
Zugelassen 84 Administratortests TPM_GetTestResult Dieser Befehl stellt herst
Zugelassen 110 Administratoranmeldung TPM_OwnerSetDisable Dieser Befehl ermöglich
Blockiert 111 Administratoranmeldung TPM_PhysicalEnable Dieser Befehl aktiviert d
Blockiert 112 Administratoranmeldung TPM_PhysicalDisable Dieser Befehl deaktiviert
Blockiert 113 Administratoranmeldung TPM_SetOwnerInstall Dieser Befehl ermöglich
Blockiert 114 Administratoranmeldung TPM_PhysicalSetDeactivat... Dieser Befehl aktiviert o
Zugelassen 115 Administratoranmeldung TPM_SetTempDeactivated Dieser Befehl ermöglich
Blockiert 116 Administratoranmeldung TPM_SetOperatorAuth Dieser Befehl definiert d
Zugelassen 1073741834 Administratorbesitz TSC_PhysicalPresence Dieser Befehl gibt die pl

Abbildung 2.4: TPM-Befehlsverwaltung

Ein Befehl wird einer bestimmten Kategorie zugeordnet und trägt einen Namen in Form einer Mnemonik. Weiterhin ist jeder Befehl durch einen Status gekennzeichnet, der einen der beiden Werte „Zugelassen“ oder „Blockiert“ annehmen kann. In Tabelle 2.2 sind exemplarisch einige Befehle und deren Status für diverse Windows-Betriebssysteme zusammengestellt. Bei den hier aufgeführten Status handelt es sich um Default-Werte, wie sie nach der Betriebssysteminstallation vorliegen, d. h. es wurde keine Veränderung durch einen Anwender oder eine Softwarekomponente vorgenommen.

Windows 7	Windows 8	S2008R2	S2012			
Status	**Status**	**Status**	**Status**	**Command Number**	**Kategorie**	**Befehlsname**
Zugelassen	Blockiert	Zugelassen	Blockiert	151	Administratorstart und -status	TPM_Init
Zugelassen	Blockiert	Zugelassen	Blockiert	83	Administratortests	TPM_ContinueSelfTest
Zugelassen	Blockiert	Zugelassen	Blockiert	111	Administratoranmeldung	TPM_PhysicalEnable
Zugelassen	Blockiert	Zugelassen	Blockiert	114	Administratoranmeldung	TPM_PhysicalSetDeactivated
Zugelassen	Blockiert	Zugelassen	Blockiert	116	Administratoranmeldung	TPM_SetOperatorAuth
Zugelassen	Blockiert	Zugelassen	Blockiert	13	Administratorbesitz	TPM_TakeOwnership

Tabelle 2.2: Default-Status unterschiedlicher Windows-Betriebssysteme

Interessant ist hierbei die Tatsache, dass Windows 7 und Server 2008R2 Befehle mit den Status „Zugelassen“ aufweisen, während sich die äquivalenten Befehle der neueren Betriebssysteme Windows 8 und Server 2012 jeweils im Status „Blockiert“ befinden. Beispielsweise konnte der Befehl „TPM_Init“ unter Windows 7 ausgeführt werden, da sein Status auf „Zugelassen“ stand, während derselbe Befehl unter Windows 8 blockiert wird.

Je nach Bedarf können die Status der einzelnen TPM-Befehle mit dem Tool tpm.msc modifiziert werden, also beispielsweise von „Zugelassen“ auf „Blockiert“ gesetzt werden. Obwohl eine derartige Modifikation leicht vonstattengeht, sollte man sich dennoch darüber im Klaren sein, dass durch das Blockieren eines Befehls eine das TPM verwendende Anwendung nicht mehr korrekt arbeiten wird.

Neben den mit dem Tool tmp.msc möglichen Veränderungen der einzelnen Status können diverse Einstellungen des TPM mit den lokalen Gruppenrichtlinien modifiziert werden, indem das entsprechende Snap-in durch die Eingabe von *gpedit.msc* geöffnet und die Konsolenstruktur wie folgt erweitert wird: COMPUTERKONFIGURATION | ADMINISTRATIVE

VORLAGEN | SYSTEM | TPM-DIENSTE.

Zudem lassen sich die TPM-Einstellungen über das Gruppenrichtlinienmanagement für Domänen verwalten, indem auf einem Domain-Controller das Snap-in *gpmc.msc* gestartet und auf die zuvor beschriebene Konsolenstruktur erweitert wird. Hierbei sollte man allerdings sehr vorsichtig zu Werke gehen, da aufgrund der Mächtigkeit von auf Domänen bezogenen Group Policy Objects eine komplette Domäne bis hin zu einem Forrest lahmgelegt werden kann.

Lokal sind die TPM-Parameter in der Registry von Windows gespeichert, und zwar unter dem Schlüssel *HKEY_LOCAL_MACHINE\Software\Microsoft\Tpm*. Unterhalb dieses Schlüssels existieren genau vier Subschlüssel (**Abb. 2.5**).

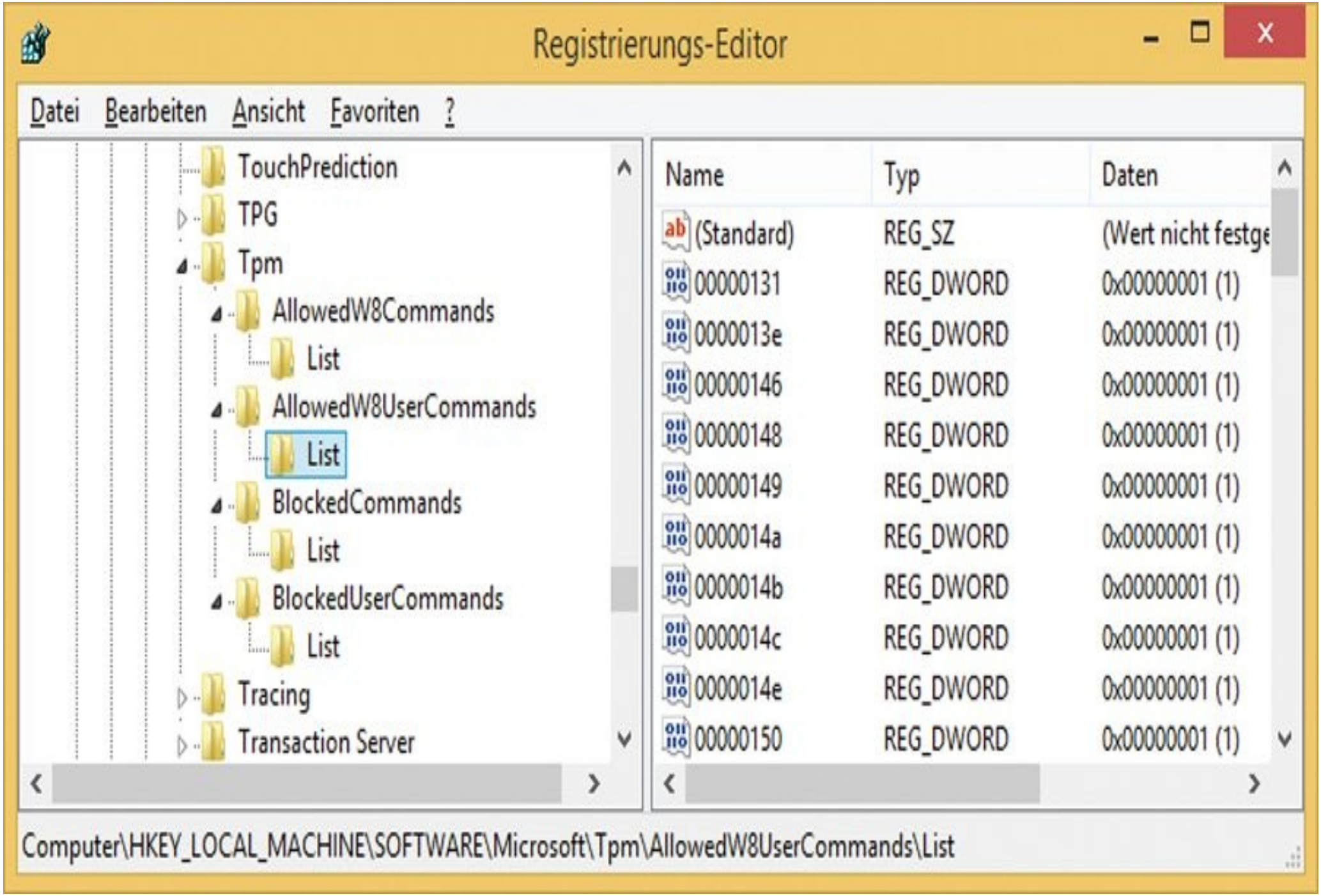
Registrierungs-Editor
Datei Bearbeiten Ansicht Favoriten ?
TouchPrediction
TPG
Tpm
AllowedW8Commands
List
AllowedW8UserCommands
List
BlockedCommands
List
BlockedUserCommands
List
Tracing
Transaction Server
Name
Typ
Daten
(Standard)
REG_SZ
(Wert nicht festge
00000131
REG_DWORD
0x00000001 (1)
0000013e
REG_DWORD
0x00000001 (1)
00000146
REG_DWORD
0x00000001 (1)
00000148
REG_DWORD
0x00000001 (1)
00000149
REG_DWORD
0x00000001 (1)
0000014a
REG_DWORD
0x00000001 (1)
0000014b
REG_DWORD
0x00000001 (1)
0000014c
REG_DWORD
0x00000001 (1)
0000014e
REG_DWORD
0x00000001 (1)
00000150
REG_DWORD
0x00000001 (1)
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tpm\AllowedW8UserCommands\List

Abbildung 2.5: Registrierungs-Editor regedit.exe

Kryptografie

Grundsätzlich unterscheidet man zwischen symmetrischen und asymmetrischen kryptografischen Verfahren. Bei den symmetrischen Verfahren wird für das Ver- und Entschlüsseln von Daten ein und derselbe Schlüssel verwendet. Dieser so genannte „geheime“ Schlüssel muss entsprechend geschützt, d. h. in einer sicheren Umgebung generiert und gespeichert werden. Falls das Ver- und Entschlüsseln von verschiedenen Personen oder von unterschiedlichen Orten aus vorgenommen werden soll, muss der hierfür erforderliche Schlüsselaustausch bzw. Schlüsseltransport unbedingt unter Einhaltung strenger Sicherheitsvorgaben abgewickelt werden. Aktuell finden die in Tabelle 2.3 aufgeführten symmetrischen Algorithmen Anwendung, wobei AES der neueste Standard ist.

Symmetrische Algorithmen	Asymmetrische Algorithmen
AES (Advanced Encryption Standard)	RSA (Rivest Shamir Adleman)
DES (Data Encryption Standard)	DSA (Digital Signature Algorithm)
3DES (Tripple Encryption Standard)	ECDSA (Elliptic Curve Digital Signature Algorithm)

Tabelle 2.3: Kryptografische Standardalgorithmen

Asymmetrische Verfahren basieren auf einem „privaten“ Schlüssel (Private Key) und einem „öffentlichen“ Schlüssel (Public Key), wobei beide Schlüssel mathematisch korreliert sind. Zusätzlich zu den verfügbaren Verschlüsselungsfunktion eignen sich asymmetrische Verfahren auch zum Generieren und Verifizieren von digitalen Signaturen. Der private Schlüssel wird zum Generieren einer Signatur und zum Entschlüsseln von verschlüsselten Daten verwendet. Der öffentliche Schlüssel hingegen zum Verifizieren einer Signatur und zum Verschlüsseln von Daten. Die derzeit aktuellen Standards sind in Tabelle 2.3 zusammengestellt. Der traditionelle RSA-Algorithmus ist noch immer am weitesten verbreitet und wird von nahezu allen Systemen unterstützt. Der neuere auf elliptischen Kurven basierende EC-Algorithmus bietet einige Vorteile gegenüber RSA, unter anderem eine signifikant kleinere Schlüssellänge, bei gleichem Security-Level. In Tabelle 2.4 sind symmetrische und asymmetrische Algorithmen und die zugehörigen Security-Level gegenübergestellt.

Security-Level (Bit)				
Symmetrisch	112	128	192	256
Elliptic Curve	224	256	384	512
RSA	2048	3072	8192	15360

Tabelle 2.4: Security-Level

Hash-Algorithmen erzeugen aus einer beliebig langen Eingangssequenz (Bytes) eine Ausgangssequenz mit genau festgelegter Länge. Weiterhin muss jede unterschiedliche Eingangssequenz eine unterschiedliche Ausgangssequenz, den so genannten Hash-Wert, erzeugen. Ist diese Forderung erfüllt, wird der Hash-Algorithmus als kollisionsfrei bezeichnet. Hash-Algorithmen werden zum Beispiel vor jedem Signaturvorgang angewendet, da die Eingangssequenz der einzelnen Signaturalgorithmen genau definierte Längen aufweisen muss. In **Abbildung 2.6** ist eine Verschlüsselungssignatursequenz veranschaulicht. Die Eingangssequenz in Form einer Klartextdatei wird im oberen Zweig mit einem symmetrischen Schlüssel verschlüsselt. Hierfür kann beispielsweise ein Tripple-DES-Algorithmus verwendet werden. Im unteren Zweig wird zunächst der Hash-Wert der Eingangssequenz berechnet und im Anschluss daran erfolgt die Generierung einer Signatur. Der Geheimtext und die Signatur werden im letzten Schritt konkateniert, sodass letztendlich ein verschlüsseltes, signiertes Dokument vorliegt.

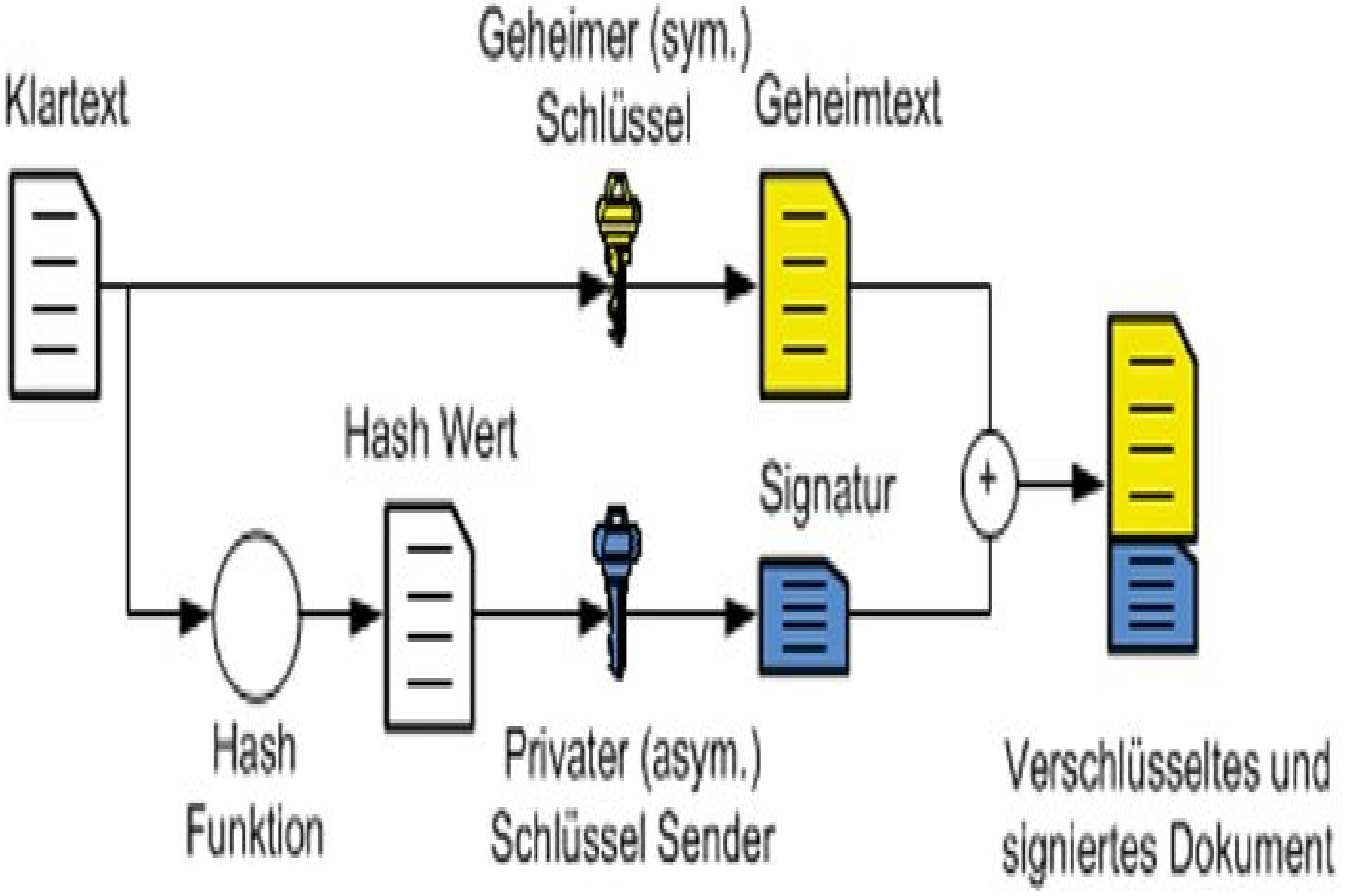
Geheimer (sym.)
Schlüssel
Klartext
Geheimtext
Hash Wert
Signatur
Hash
Funktion
Privater (asym.)
Schlüssel Sender
Verschlüsseltes und
signiertes Dokument

Abbildung 2.6: Verschlüsseln und Signieren eines Klartexts

Ein digitales Zertifikat enthält primär einen Aussteller (Issuer) und einen Antragsteller (Subject), sowie einen Gültigkeitszeitraum und einen öffentlichen Schüssel. Somit kapselt ein digitales Zertifikat einen öffentlichen Schlüssel und ordnet diesen einem Aussteller und einem Antragssteller genau zu. Der ansonsten völlig anonyme öffentliche Schlüssel in Form einer Bytesequenz wird somit greifbar, und es kann entschieden werden, ob er für eine Verifikation benutzt werden soll. Ob der Aussteller des Zertifikats vertrauenswürdig ist, kann nachweislich nur mit der PKI (Public Key Intrastructure), von der das Zertifikat ausgegeben wurde, validiert werden. Genauer betrachtet handelt es sich bei dem Aussteller um eine Zertifizierungsstelle (Certificate Authority) als zentrales Element jeder PKI [6]. Ein digitales Zertifikat, das internationalen Standards genügt, ist gemäß RFC 5280 strukturiert und kodiert, sodass die Interoperabilität zwischen verschiedenen Systemen gewährleistet ist. Als Kodierung kommt entweder Base64 oder ASN.1 in Frage.

Kryptografische Artefakte eines TPM

Nachstehend sind die wichtigsten kryptografischen Artefakte eines TPM wie Schlüssel, Zertifikate und Passphrasen kurz beschrieben. Eine detaillierte Beschreibung ist in [5] gegeben.

Endorsement Key (EK): Die wohl wichtigste Komponente eines TPM ist der Endorsement Key (EK), der bereits während der Fertigung generiert und in das TPM eingebrannt wird und somit nicht verändert werden kann. Genauer betrachtet handelt es sich um ein asymmetrisches RSA-Schlüsselpaar (2 048 Bit), wobei der private Schlüssel mit „PRIVEK“ und der öffentliche mit PUBEK bezeichnet werden. Auf diesen privaten Schlüssel kann zu keinem Zeitpunkt von außen zugegriffen werden, weder von einer anderen Hardwarekomponente, noch von einem Tool oder einer Applikation. Beispielsweise wird der private Schlüssel „PRIVEK“ bei der Generierung von Signaturen verwendet. Das Generieren erfolgt allerdings intern in der Cryptographic Unit des TPM (vgl. **Abb. 2.1**), sodass gemäß Forderung der Schlüssel PRIVEK niemals das TPM verlässt. Eine generierte Signatur kann mithilfe des korrespondierenden öffentlichen Schlüssels verifiziert werden.

Storage Root Key (SRK): Der Storage Root Key ist im TPM eingebettet, und auf die private Schlüsselkomponente kann ebenfalls nicht von außen zugegriffen werden. Die Aufgabe des Storage Root Key ist der Schutz (Verschlüsselung) von TPM-Schlüsseln, die von Applikationen generiert wurden und im TPM gespeichert sind. Im Gegensatz zum Endorsement Key wird der Storage Root Key nicht bereits bei der Herstellung generiert, sondern erst dann, wenn das TPM zum ersten Mal initialisiert, oder einem neuen Benutzer zugewiesen wird.

Endorsement Certificate: Das Endorsement Certificate wird vom Hersteller generiert und enthält den zu dem privaten Endorsement Key PRIVEK korrespondierenden öffentlichen Schlüssel PUBEK. Weiterhin enthält dieses Zertifikat den Herausgeber (Issuer) und den Antragsteller (Subject), sowie eine Seriennummer und eine Gültigkeitsangabe (not before und not after). Der im Endorsement Certificate enthaltene öffentliche Schlüssel ermöglicht somit die Verifikation von mit dem privaten Schlüssel signierten Daten. Somit kann beispielsweise eine Attestation auf Echtheit verifiziert werden.

Das PowerShell Commandlet „Get-TpmEndorsementKeyInfo“ liefert die Zertifikatsdaten eines aktivierten TPM (**Abb. 2.7**). Optional wurde hier der Parameter *HashAlgorithm* sha256 angegeben, sodass zusätzlich der Hash-Wert des öffentlichen Schlüssels mit ausgegeben wird.

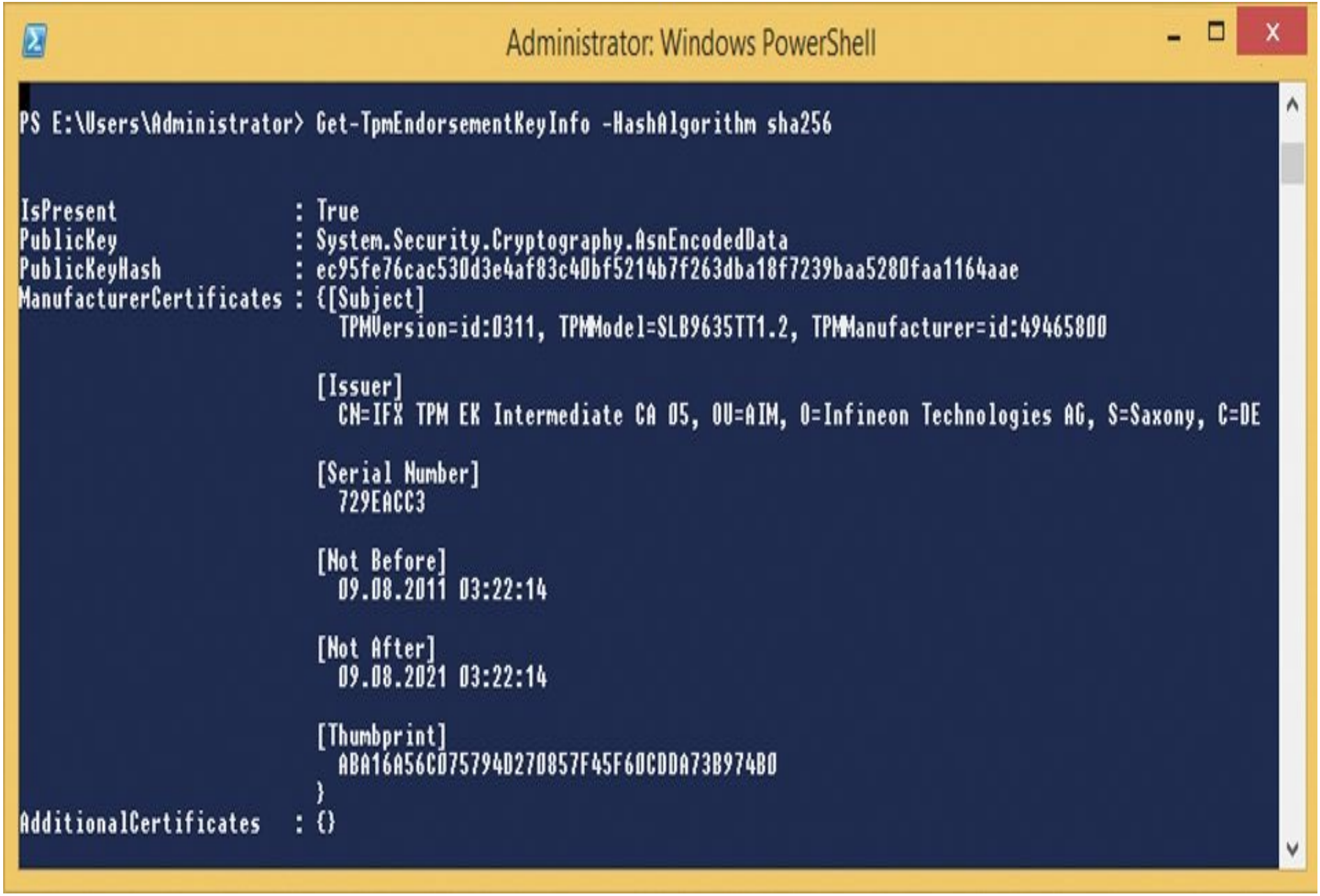
Administrator: Windows PowerShell
PS E:\Users\Administrator> Get-TpmEndorsementKeyInfo -HashAlgorithm sha256
IsPresent : True
PublicKey : System.Security.Cryptography.AsnEncodedData
PublicKeyHash : ec95fe76cac530d3e4af83c40bf5214b7f263dba18f7239baa5280faa1164aae
ManufacturerCertificates : {[Subject]
TPMVersion=id:0311, TPMModel=SLB9635TT1.2, TPMManufacturer=id:49465800
[Issuer]
CN=IFX TPM EK Intermediate CA 05, OU=AIM, O=Infineon Technologies AG, S=Saxony, C=DE
[Serial Number]
729EACC3
[Not Before]
09.08.2011 03:22:14
[Not After]
09.08.2021 03:22:14
[Thumbprint]
ABA16A56CD75794D270857F45F60CDDA73B974BD
}
AdditionalCertificates : {}

Abbildung 2.7: TPM-Zertifikatsdaten

Platform Configuration Register (PCR): Ein Platform Configuration Register ist ein Speicherbereich, in dem die Momentaufnahmen der Zustände von Konfigurationen von Hard- und Software einer Trusted-Plattform gespeichert werden können. Genauer betrachtet werden Hash-Werte von Systemzuständen und Softwarekomponenten, wie beispielsweise der des Betriebssystemkernels, gespeichert. Die gespeicherten Daten können zur Validierung der Systemintegrität verwendet werden. Im Gegensatz zu einem externen Crypto Token (Smart Card etc.) garantiert ein fest eigebautes TPM, dass es in jedem Fall bereits während des Boot-Vorgangs präsent ist und diesen mitverfolgen kann.

TPM Owner Password: Per Definition gilt derjenige als Besitzer (Owner) des TPM, der das Owner Password kennt. Pro TPM existiert nur ein einziges Owner Password. Der Besitzer kann von der vollen TPM-Funktionalität Gebrauch machen. Es bleibt ausschließlich dem Besitzer vorbehalten, ein TPM-Remote zu manipulieren, also freizugeben, zu sperren oder zu löschen. Das TPM Owner Password kann während der Initialisierung in einer Datei gespeichert oder gedruckt werden.

Fazit

Solange es einem Besitzer eines Computersystems bzw. einer Plattform möglich ist, ein fest integriertes TPM abzuschalten, ohne dass das Gesamtsystem oder Teilsysteme unbenutzbar werden, kann jeder individuell entscheiden, ob das TPM aktiviert werden soll oder nicht. Nichtsdestotrotz mutiert ein TPM zu einem nicht ganz unumstrittenen Begleiter in fast allen neueren Computern und Tablets und löst im Rahmen von Remote-Attestation von Zeit zu Zeit Diskussionen bezüglich des Datenschutzes aus. Inwieweit sich die Unterhaltungsbranche ein aktiviertes TPM zunutze macht, um illegale Raubkopien oder Downloads zu blockieren, wird erst die Zukunft zeigen. Außerdem bleibt abzuwarten, ob die Version 2.0 der TPM-Spezifikation weiterhin eine gewisse Wahlfreiheit zulässt oder ob drastische Restriktionen umgesetzt werden.

Links & Literatur

[1] Trusted Computing Group: www.trustedcomputinggroup.org

[2] „Grundlagen und Standards von Chipkarten“, in Windows Developer 07.2012

[3] Windows Phone 8.1 Security Overview, Microsoft April 2014

[4] Stellungnahme des BSI zur aktuellen Berichterstattung zu MS Windows 8 und TPM: https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2013/Windows_TPM_Pl_21

[5] TPM Main-Part 1 Design Principles, Trusted Computing Group

[6] „Microsoft Certificate Authority“, in dot.NET Magazin 06.2010

3 TPM-Kommandos und Programmierschnittstellen

Das dritte Kapitel gibt zunächst einen Überblick über die wichtigsten TPM-Kommandos. Daran anschließend werden diverse Programmierschnittstellen vorgestellt, nämlich PowerShell, WMI und .NET. Die Beschreibung und Realisierung kompletter Beispiele mit lauffähigem Programmcode rundet dieses Kapitel ab, wobei sowohl ein universelles C-API als auch eine Bibliothek von Microsoft Research vorgestellt werden.

Wie bereits im zweiten Kapitel beschrieben, ist ein Trusted Platform Module (TPM) ein integrierter Baustein, der in die Hauptplatine eines Computersystems integriert ist und hardwarebasierte Sicherheitsfeatures bereitstellt. Gegenüber Sicherheitsmechanismen, die auf Software basieren, erhöht sich das erzielbare Sicherheitslevel damit erheblich. Insbesondere die wichtigsten TPM-Funktionalitäten, wie Authentifikation von Nutzern, Berechtigungen für Netzwerkzugriffe und Schutz von Daten und Plattformen, tragen zu einer erhöhten Systemsicherheit bei. Über diverse Programmierschnittstellen können die einzelnen TPM-Kommandos via Anwendungsprogramm ausgeführt werden, sofern die erforderliche Berechtigung vorliegt und das jeweilige Kommando den Zustand „Zugelassen“ aufweist.

Software Stack

Die verfügbaren TPM-Kommandos der aktuellen Version 1.2 sind als separates Teildokument der gesamten Spezifikation verfügbar [1]. Diese Kommandos können über eine Programmierschnittstelle an ein TPM geschickt werden, um von dem so genannten TCG Software Stack (TSS) verarbeitet zu werden. Der TSS ist ein vom jeweiligen Hersteller unabhängiges High-Level-API, das die interne Komplexität des TPM vom Anwendungsprogrammierer fernhält. Beispielsweise werden die für eine Autorisierung erforderlichen Details oder das Generieren von Bytesequenzen sowie die von einem Applikationsprogrammierer schwer zu durchschauenden Aspekte in überschaubare, kompakte TPM-Kommandos abgebildet.

Grundsätzlich werden verschieden Arten von Kommandos unterschieden. Das Kommando *TPM_Init* wird beispielsweise nicht von einer Software aufgerufen, sondern durch die Hardware getriggert. In diesem Fall kommt der Trigger von dem LPC-Bus der Hauptplatine (Plattform). Weiterhin werden Kommandos dahingehend unterschieden, ob sie zwingend sicher ausgeführt werden müssen, oder ob die sichere Ausführung nicht unbedingt notwendig ist. Sicher auszuführende Kommandos sind in Hardware realisiert

und mit dem Präfix TPM gekennzeichnet.

TPM Device Driver befinden sich auf der untersten Ebene, d. h. auf der gleichen Ebene wie die Kernkomponenten des Betriebssystems (**Abb. 3.1**). Die hier laufenden Prozesse starten im so genannten Kernel Mode und haben somit hohe Ausführungsprivilegien.

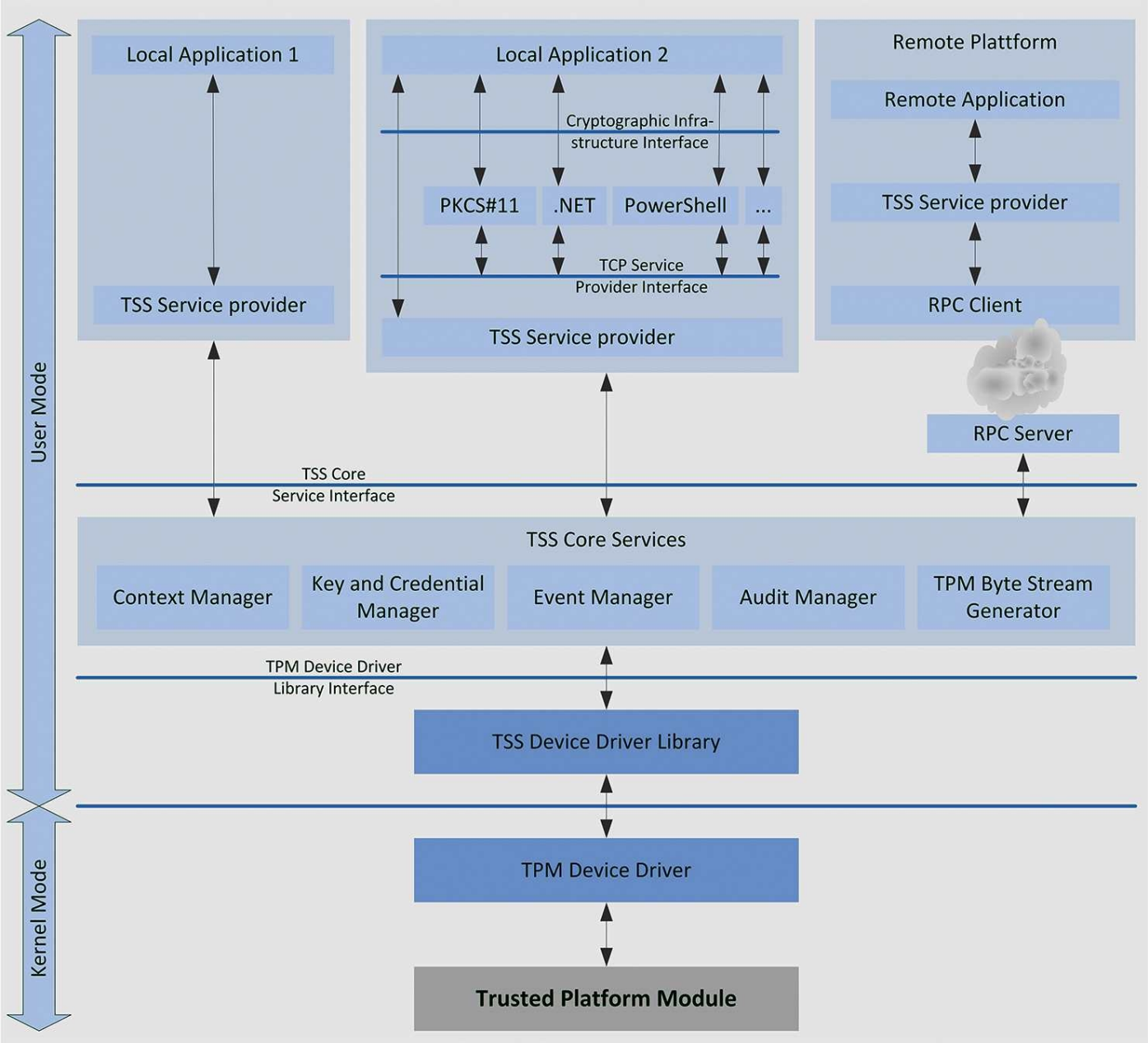

Abbildung 3.1: TSS – Architekturübersicht

Neben einer TSS Device Driver Library realisiert der User-Mode in der mittleren Schicht zahlreiche Manager, die über das TSS-Core-Service-Interface den Link zu der Applikationsschicht herstellen. Unterstützt werden nicht nur lokale Applikationen, sondern auch via RPC-Server geeignete Remote-Plattformen. Ein Cryptographic-Infrastructure-Interface fungiert als proprietäre Applikationsschnittstelle und ermöglicht den Zugriff über diverse Implementierungen wie beispielsweise .NET, PKCS#11 (*cryptoki.dll*) oder PowerShell. Generell laufen die von einem Anwender gestarteten Prozesse im User-Mode ab. Ein typischer Userprozess ist beispielweise das Verschlüsseln

von Daten, das folgendermaßen abläuft (wobei der Terminus *Seal* in dem Kontext eines TPM nichts anderes als Verschlüsselung bedeutet):

1. Ein Userprozess schickt das Kommando *TPM_CreateWrapKey(Keyinfo)* an das TPM.
2. Das TPM schickt einen *Keyblob* an den Userprozess.
3. Der Userprozess schickt das Kommando *TPM_LoadKey2(Keyblob)* an das TPM.
4. Das TPM schickt ein Handle an den Userprozess.
5. Der Userprozess schickt das Kommando *TPM_Seal(Handle, Data)* an das TPM.
6. Das TPM schickt einen *SealedBlob* an den Userprozess.

Neben dem oben vorgestellten Snap-in *tpm.msc* besteht die Möglichkeit, administrative Aufgaben abzuwickeln, indem innerhalb eines Windows-Systems per Kommandozeile bzw. Script (PowerShell etc.) auf ein TPM zugegriffen wird. Weiterhin existieren diverse APIs, die den Zugriff via Programmiersprache bereitstellen.

PowerShell – Programmierschnittstelle

PowerShell stellt in der Version 4 ein paar grundlegende Commandlets für ein TPM-Objekt zur Verfügung. Gemäß Tabelle 3.1 können hiermit allgemeine Informationen über den Status des TPM abgefragt werden. Darüber hinaus lassen sich auch Operationen durchführen, die den Status des TPM beeinflussen.

Zunächst sollte jedoch festgestellt werden, ob die installierte Version von PowerShell TPM-Commandlets unterstützt. Hierfür kann das Commandlet *Get-Command-Module TrustedPlatformModule* verwendet werden. Die installierte PowerShell-Version lässt sich durch Ausgabe der Variablen *$PSVersionTable.PSVersion* ermitteln [2], wobei die Verwendung dieser TPM-Commandlets die Major-Version 4 voraussetzt.

Wie aus Tabelle 3.1 hervorgeht, handelt es sich bei den verfügbaren TPM-Commandlets primär um Kommandos für die Administration, d. h. für die Initialisierung, Statusabfrage oder Besitzerverwaltung. Die Ausführung von kryptografischen Operationen ist derzeit offenbar nicht möglich. **Abbildung 3.2** zeigt die Ausgabe von PowerShell exemplarisch für zwei Statusabfragen. Im gezeigten Beispiel ist das TPM für Schlüsselbestätigung (Key Attestation) konfiguriert.

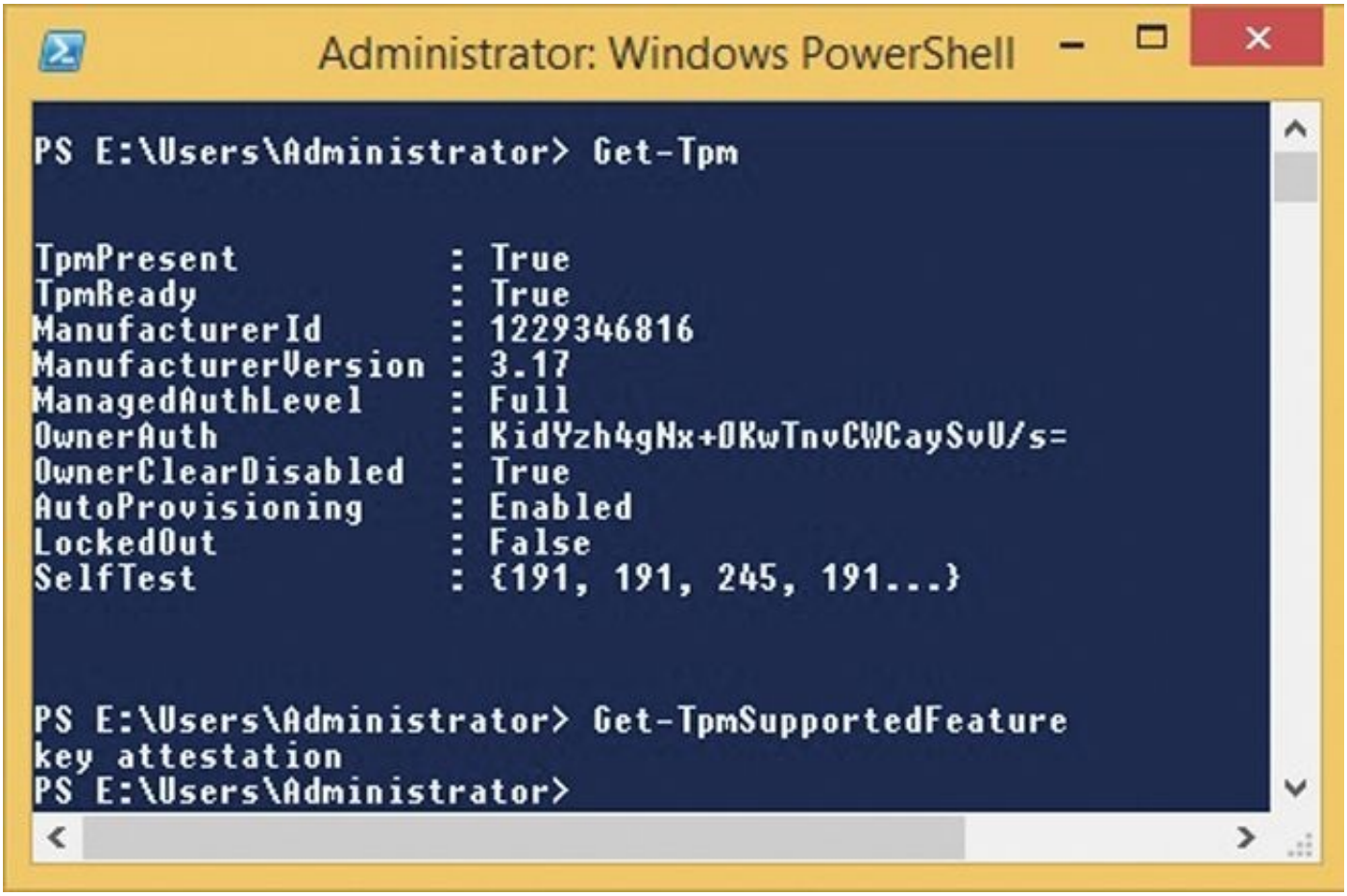

Abbildung 3.2: PowerShell TPM-Commandlets

Manage-bde.exe

Ein weiteres Tool für die Konfiguration eines TPM ist *Manage-bde.exe*. Hierbei handelt es sich um ein Kommandozeilentool, das primär für die Administration von BitLocker-Laufwerken ausgelegt ist. Da diese verschlüsselten Laufwerke auf der Funktionalität eines TPM basieren, existieren auch etliche Kommandos für die Manipulation eines TPM.

In der oberen Hälfte von **Abbildung 3.3** ist die Statusausgabe von *Manage-bde.exe* für ein mit BitLocker verschlüsseltes Laufwerk I: zu sehen. Weder die Bezeichnung noch ein Verschlüsselungsstatus ist aufgelistet. Nachdem das Laufwerk entsperrt wurde, werden alle relevanten Informationen ausgegeben. Der Verschlüsselungsalgorithmus wird in beiden Fällen ausgegeben, wobei es sich um den symmetrischen AES-Algorithmus handelt, und zwar um die schwächste Variante mit 128 Bit. AES (Advanced Encryption Standard) bietet neben der hier verwendeten Verschlüsselung mit 128 Bit auch die stärkeren Varianten mit 192 bzw. 256 Bit.

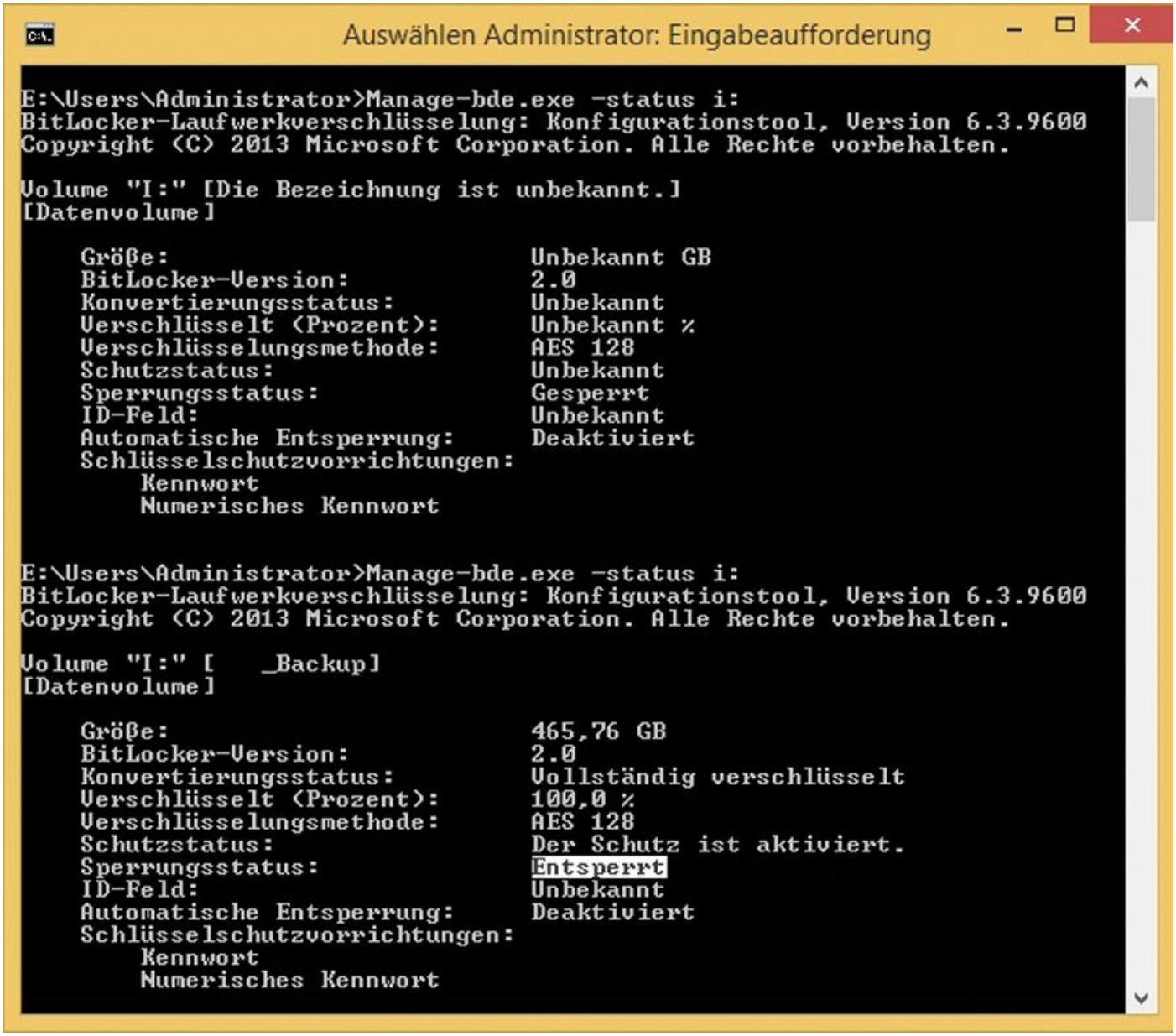

Abbildung 3.3: „Manage-bde.exe“-Laufwerkstatus

PowerShell-Commandlet	Beschreibung
Clear-Tpm	TPM in den Default-Zustand versetzen
ConvertTo-TpmOwnerAuth	Generieren einer TPM-kompatiblen Passphrase
Disable-TpmAutoProvisioning	Sperren von Auto-Provisioning
Enable-TpmAutoProvisioning	Entsperren von Auto-Provisioning
Get-Tpm	Genereller TPM-Status, z. B. Herstellername, Herstellerversion
Get-TpmEndorsementKeyInfo	Herstellerzertifikat, öffentlicher Schlüssel des Herstellers -HashAlgorithm sha256
Get-TpmSupportedFeature	Schlüsselbestätigung (Key Attestation)
Import-TpmOwnerAuth	Import einer Besitzerautorisierung
Initialize-Tpm	Initialisieren eines TPM

Set-TpmOwnerAuth	Setzen einer Besitzerautorisierung
Unblock-Tpm	Entsperren eines TPM

Tabelle 3.1: TPM-Commandlets (PowerShell Version 4)

C# – WMI-Klasse

Obwohl C# kein eigenes API für die Interaktion mit einem TPM bereitstellt, gibt es die Möglichkeit, rudimentäre Abfragen mit der Klasse *ManagementClass* zu programmieren. Diese Klasse ist ein CIM (Common Information Model). Genauer betrachtet ist eine Verwaltungsklasse eine WMI-Klasse (Windows Management Instrumentation). Der Zugriff auf die WMI-Daten erfolgt durch einen vordefinierten Klassenpfad, also beispielsweise *Win32_ComputerSystem* oder *Win32_OperatingSystem*. Für den Zugriff auf ein TPM lautet der Klassenpfad */root/CIMv2/Security/MicrosoftTpm:Win32_Tpm*.

```
using System.Management;

ManagementClass mc = new
ManagementClass("/root/CIMv2/Security/MicrosoftTpm:Win32_Tpm");
string wmiDtd20 = mc.GetText(TextFormat.WmiDtd20); // -> Speicherung in XML-Datei
ManagementObjectCollection moc = mc.GetInstances();
ManagementObjectCollection.ManagementObjectEnumerator moe = moc.GetEnumerator();
moe.MoveNext();
if (mo != null)
{
  // Verfügbare Properties aus XML-Datei
  string sProperty = "IsActivated_InitialValue: " + mo["IsActivated_InitialValue"].ToString();
  ...
  // Verfügbare Methods aus XML-Datei
  object[] wmiParams = new object[1];
  wmiParams[0] = false;
  status = (UInt32)mo.InvokeMethod("IsEnabled", wmiParams);
  string sMethod = "IsEnabled: " + wmiParams[0];
  ...
}
```

Listing 3.1: TPM – WMI-Interface

Wie in Listing 3.1 zu sehen, muss zunächst ein Objekt *ManagementClass* instanziiert werden, und zwar mit dem entsprechenden Klassenpfad für das TPM. Anschließend können alle verfügbaren Eigenschaften und Methoden abgefragt werden. Der Parameter *TextFormat.WmiDtd20* unterstützt eine XML-Formatierung, sodass das Ergebnis übersichtlich visualisiert werden kann. Hierfür eignet sich zum Beispiel der Internet Explorer. Zur Auswertung der von der Klasse *ManagementClass* bereitgestellten Daten kann ein Enumerator vor das erste Element positioniert und dieses dann ausgewertet werden.

Abbildung 3.4: TPM-WMI-Eigenschaften und -Methoden

In **Abbildung 3.4** sind die verfügbaren Eigenschaften und Methoden eines TPM zu sehen, wobei die in Listing 3.1 gezeigten Codestrecken verwendet und in ein einfaches C#-Programm einprogrammiert wurden. Die Ausführung eines auf der Abfrage von WMI-Daten basierenden Programms bedarf keiner besonderen Berechtigungen, d. h. es sind keine Admin-Rechte erforderlich. Allerdings ist die verfügbare Funktionalität auf Parameter- und Statusabfragen limitiert. Diese Limitierung lässt sich umgehen, indem die DLL *tbs.dll* in ein C#-Projekt eingebunden wird und die benötigten Funktionen durch einen Wrapper bereitgestellt werden. Die Verwendung dieser DLL entspricht der nachstehend beschriebenen Vorgehensweise, sodass neben den erwähnten Abfragen auch kryptografische Operationen möglich sind.

C – API

Das SDK von Microsoft für Windows stellt ein API bereit, basierend auf einem Header plus Library sowie einer DLL. Ein Blick in die Header-Datei *tbs.h* lässt sowohl Konstantendefinitionen als auch vordefinierte Funktionen für den Zugriff auf ein TPM erkennen. Um auf diese Definitionen verweisen zu können, ist laut Listing 3.2 zunächst die Header-Datei zu inkludieren.

Listing 3.2 zeigt ausschnittsweise, wie eine Zugriffszahl in Hardware generiert werden kann. Obwohl die Realisierung eines Zufallszahlengenerators im Allgemeinen keine große Herausforderung darstellt, gibt es dennoch ein generelles Problem, nämlich die Implementierung eines geeigneten Startwerts. Ein ausschließlich in Software realisierter Zufallszahlengenerator basiert auf einem festen Algorithmus und startet daher prinzipiell immer mit dem gleichen Startwert. Außerdem liefert er sich stetig wiederholende Zufallszahlenreihen, wenn kein geeigneter Startwert vorgegeben wird. Ein auf Hardware basierender Generator (TPM) hingegen kann einen variablen Startwert selbst erzeugen, indem beispielsweise die Zeit ab dem Einschaltzeitpunkt des Systems (Power On), oder andere physikalische Parameter, wie beispielsweise Signale auf dem Systembus, zugrunde gelegt werden.

```
#include <Tbs.h>

void GetRandom(UINT nCount)
{
  TBS_CONTEXT_PARAMS pContextParams;
  TBS_HCONTEXT hContext;
  TBS_RESULT result;

  BYTE data[] = {0x00, 0xc1,       // Kommando ohne Authentifizierung
      0x00, 0x00, 0x00, 0x0e,       // Anzahl Bytes
      0x00, 0x00, 0x00, 0x46,       // Kommando: TPM_ORD_GetRandom
      0x00, 0x00, 0x00, nCount }; // Anzahl zurückzugebender Bytes
  BYTE buf[512] = { 0 };
  UINT32 buf_len = sizeof(buf);
  UINT32 offset = sizeof(data);

  // TPM Version
  pContextParams.version = TBS_CONTEXT_VERSION_ONE;
  // Context-Handle erzeugen
  result = Tbsi_Context_Create(&pContextParams, &hContext);
  // Kommano ausführen
  result = Tbsip_Submit_Command(hContext,
    TBS_COMMAND_LOCALITY_ZERO, TBS_COMMAND_PRIORITY_NORMAL,
    data, offset, buf, &buf_len);
  // Context-Handle schliessen
  result = Tbsip_Context_Close(hContext);
  // Zufallszahl ausgeben: nCout Bytes
  for (unsigned int i = 0; i < nCount; i++)
  printf("%02X ", buf[offset + i]);
}
```

Listing 3.2: True Random Number Generator (TRNG)

Bevor das API verwendet werden kann, ist zunächst die TPM-Version vorzugeben (Listing 3.2), wobei in *tbs.h* ausschließlich die Version „One“ definiert ist, also für TPM-1.2. Danach ist immer ein Kontext-Handle zu erzeugen, das jedem nachfolgenden TPM-Kommando als Parameter mitzugeben ist. Die gesamte Sequenz ist stets mit einem Kommando zum Schließen des Kontexts zu beenden. Darüber hinaus sind nach jedem TPM-Kommando entsprechende Fehlerabfragen erforderlich (*result== TBS_SUCCESS*). Diese wurden in Listing 3.2 jedoch zugunsten der Übersichtlichkeit weggelassen.

Ein Kommando ohne Authentifizierung läuft grundsätzlich nach dem gezeigten Schema ab: Handle erzeugen, TPM-Kommando ausführen und Handle schließen. Der wesentliche

Unterschied einzelner TPM-Kommandos besteht in der Zusammenstellung der Bytesequenz (*data[]*), die im Submit-Kommando übergeben wird. Die erforderlichen Informationen hierfür liefert [3]. Zunächst ist ein so genanntes „Command and Response Tag“, kurz *TPM_TAG*, erforderlich. Diese Tags sind in Abschnitt sechs von [3] spezifiziert. Im Falle der Zufallszahlengenerierung ist keine Authentifizierung erforderlich, sodass hierfür das Tag *TPM_TAG_RQU_COMMAND* mit dem hexadezimalen Wert 0x00C1 geeignet ist. Die Anzahl der Bytes ergibt sich als Summe der die Bytesequenz umfassenden Bytes. Die benötigten Informationen für die Kodierung des eigentlichen Kommandos können Abschnitt 17 von [3] entnommen werden. Hier sind unter der Überschrift „Ordinals“ unter anderem die verfügbaren kryptografischen Kommandos tabellarisch spezifiziert. Für die im vorliegenden Beispiel zu erzeugende Zufallszahl handelt es sich um das Kommando *TPM_ORD_GetRandom*, wobei sich aus der entsprechenden Tabellenzeile ergibt, dass für die Ausführung keine Authentifizierung und kein TPM-Besitzer erforderlich sind. Weiterhin geht der benötigte hexadezimale Wert in Form von vier Bytes hervor, nämlich 0x00000046.

Nach erfolgreicher Ausführung des Submit-Kommandos enthält die Variable *buf[]* die angeforderte Anzahl von Random-Bytes, wobei diesen sämtliche im Kommando übergebenen Datenbytes vorangestellt sind. Deshalb ist bei der Auswertung von *buf[]* ein Offset erforderlich, um auf die Random-Bytes zugreifen zu können.

Mit der gezeigten Vorgehensweise und unter Zuhilfenahme von [3] können die spezifizierten TPM-Kommandos ausgeführt werden, allerdings immer unter der Prämisse, dass dem auszuführenden Kommando im Rahmen der TPM-Administration der Status „Zugelassen“ zugewiesen wurde.

Obwohl mit dem vorgestellten C-API die in [3] spezifizierten Kommandos realisierbar sind, ist es dennoch etwas mühsam und zeitaufwändig, die adäquate Bytesequenz zusammenzustellen und zu testen. Dies wird sehr deutlich, wenn man sich den Umfang der Spezifikation [3] von derzeit über 200 Seiten vor Augen führt. Alternativ können geeignete Bibliotheken eingesetzt werden. Diese kapseln das jeweilige Submit-Kommando und machen somit anstelle der Zusammenstellung einer speziellen Bytesequenz ein spezielles Kommando, wie beispielsweise *GetRandom()*, verfügbar. Nachstehend wird eine solche Bibliothek kurz vorgestellt.

TPM – Simulator

Die unter der Bezeichnung „TPM Software Stack from Microsoft Research“ veröffentlichte Bibliothek [4] ist als CPP-Solution und als NET-Solution mit Quelldateien und Samples zum Download verfügbar. Nach dem Entpacken und Kompilieren der NET-Version liegt eine DLL mit der Bezeichnung *TSS.NET.dll* vor, die in eigene .NET-Projekte

eingebunden werden kann.

Nach dem Öffnen von *TSS.sln* mit Visual Studio sind im Solution Explorer neben der Bibliothek TSS.NET zahlreiche Samples aufgelistet. Hierbei handelt es sich um Konsolenanwendungen, die jeweils als Startprogramm festgelegt werden können und somit separat ausführbar sind.

Auch für die oben beschriebene Zufallszahlengenerierung liegt ein eigenes Sample mit der Bezeichnung *GetRandom* vor. Ohne Kenntnisse der umfangreichen TPM-Spezifikation kann mit diesen Bibliotheksfunktionen eine Zufallszahl generiert werden, indem ein Device-Objekt instanziiert und mit diesem die Methode *GetRandom(AnzahlBytes)* aufgerufen wird.

Wie in den einzelnen Samples gezeigt, stellt diese Bibliothek nicht nur ein Interface zu einem realen TPM bereit, sondern bietet darüber hinaus auch die Möglichkeit, einen TPM-Simulator zu realisieren. Diese Option ist hervorragend für die Entwicklung und den Test geeignet. Ob es sich um ein Interface zu einem TPM-Device oder zu einem TPM-Simulator handelt, kann problemlos beim Instanziieren der Bibliotheksklasse *Tpm2Device* festgelegt werden.

Fazit

Wir befinden uns mitten in einer digitalen Revolution. Die abgeschottete Einzelplatzumgebung im Offlinemodus gehört offensichtlich der Vergangenheit an. Während sich derzeit die Cloud-Technologien zu etablieren scheinen, wird weiterhin versucht, die Akzeptanz für eine hardwaregestützte Kontrolle via Internet durch Dritte zu erreichen, die so genannte Remote-Attestation.

Aus technischer Sicht ist natürlich jedes Mittel recht, das zur Steigerung der Systemsicherheit beiträgt. Allerdings nimmt der Zugriff auf das eigene System durch Dritte gleichermaßen zu. Das kann bedeuten, dass die totale Kontrolle bzw. Überwachung von Personal Computern, Notebooks, Tablets, Smartphones oder Unterhaltungselektronik (z. B. Xbox) droht. Eines scheint jedoch klar zu sein: Aus der Sicht des Softwareentwicklers mit Fokus auf kryptografische Applikationen ist ein TPM allemal ein willkommenes Feature, stellt es doch ein breites Spektrum an Verschlüsselungs- und Schlüsselspeicherungsfunktionen ohne zusätzliche Hardware oder Smartcard bereit.

Links & Literatur

[1] TPM Main Specification Part 3 Commands: http://www.trustedcomputing group.org/resources

[2] Stoiber, Helmut: „Zurück in die Zukunft“, in dot.NET Magazin 12.2010

[3] TPM Main Part 2 TPM Structures: http://www.trustedcomputinggroup.org/resources

[4] TPM Software Stack from Microsoft Research: https://tpm2lib.codeplex.com/

Die Autoren

Christof Graff ist dreißig Jahre alt und gelernter Fachinformatiker für Anwendungsentwicklung. Er arbeitet als leitender Senior Softwarearchitekt bei der Firma Comparex. Hauptsächlich beschäftigt er sich mit den Themen Kassensysteme, Anwendungsintegration, hoch skalierbare und performante Architekturen. Sie erreichen ihn unter *https://www.xing.com/profile/Christof_Graff*.

Matthias Zscherp ist Diplommineraloge und arbeitet als Senior Softwarearchitekt bei der Firma Comparex. Hauptsächlich beschäftigt er sich mit den Themen Anforderungsmanagement, Sicherheit und Anwendungsintegration. Sie erreichen ihn unter *https://www.xing.com/profile/Matthias_Zscherp*.

Dipl.-Ing. **Helmut Stoiber** ist externer Berater für Software- und Systemarchitektur mit Schwerpunkt Sicherheitstechnologie sowie EDV-Sachverständiger für Systeme und Anwendungen der Informationsverarbeitung. Sie erreichen ihn unter *hs@itgutachten.net*.