



Fraunhofer
Institut
Sichere Informations-
Technologie



Institut für Informatik der
Goethe-Universität Frankfurt am Main
Professur für Graphische Datenverarbeitung

D I P L O M A R B E I T

TRUSTED COMPUTING SICHERHEIT FÜR WEBBROWSER

vorgelegt von

Gökhan Bal

Frankfurt am Main, 28. April 2009

Betreuer: Prof. Dr.-Ing. Detlef Krömker
Zweitgutachterin: Prof. Dr. Claudia Eckert

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, sind durch Quellenangaben im Text deutlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Frankfurt am Main, 28. April 2009

Gökhan Bal

Danksagung

Ich möchte mich zunächst bei Herrn Prof. Dr.-Ing. Detlef Krömker für die Bereitschaft, die Betreuung dieser Diplomarbeit zu übernehmen, bedanken. Mir ist bewusst, dass es bei externen Arbeiten nicht die Selbstverständlichkeit ist.

Weiterhin bedanke ich mich ganz herzlich bei Dr. Andreas U. Schmidt und Nicolai Kuntze vom Fraunhofer Institut für Sichere Informationstechnologie. Diese Diplomarbeit hat mir die Gelegenheit gegeben, in eine sehr interessante Themenwelt einzutauchen und mich noch mehr mit der IT Security-Welt vertraut zu machen. Dieser Dank gebührt auch Frau Prof. Dr. Claudia Eckert, die auch externen Studenten die Möglichkeit bietet, interessante Diplomarbeiten an einem namhaften Forschungsinstitut auszuarbeiten. Nicolai Kuntze danke ich zudem noch für die schier endlose Hilfsbereitschaft, das Informationsmaterial und den geliehenen Server. Ein besonderes Highlight für mich war sicherlich das Verfassen des Papers für die ICACT 09. Auch für diese Förderung möchte ich mich bedanken.

Ganz ganz herzlich bedanke ich mich bei meinen Eltern dafür, dass sie einfach da sind und mich immer in Allem unterstützen. Das gilt auch für meine beiden Schwestern, die mich bereits in sehr frühen Jahren intellektuell gefördert und gefordert haben. Danke!

Und selbstverständlich danke ich meiner Freundin dafür, dass sie von Anfang an immer dabei war und mich dazu bringt, an mich und meine Stärken zu glauben. Sie hat es während der Ausarbeitung dieser Diplomarbeit immer wieder auf unglaubliche Art und Weise geschafft, mich zu pushen und zu motivieren. Ich danke ihr zudem für die Unterstützung beim Korrekturlesen und ihre objektiv-kritischen Meinungen.

Bei Andreas Leicher und Andreas Brett möchte ich mich für das super Framework bedanken, dass sie innerhalb kürzester Zeit zusammengeschraubt haben. Es hat mir sehr dabei geholfen, den TSS zu verstehen und anzuwenden. Zusätzlichen Dank an A. Leicher für das Hosten des Entwicklungsservers.

Zum Schluß möchte ich mich bei all denjenigen auf der Welt bedanken, die so viele herrliche Tools frei verfügbar machen, die das Management dieser Diplomarbeit um so vieles leichter gemacht haben.

Zusammenfassung

Webbrowser gehören zu den wichtigsten und meistgenutzten Softwareprodukten. Durch das rasante Wachstum des World Wide Webs und den vielfältigen Technologien, die damit verbunden sind, müssen moderne Webbrowser in der Lage sein, mit der Komplexität der Aufgaben umzugehen. In Zeiten, in denen immer öfter von Datenskandalen zu hören ist, gehört es zu den wichtigsten Aufgaben eines Browsers, benutzerbezogene Credentials wie Passwörter angemessen zu schützen. Die Schutzmechanismen der meisten Webbrowser basieren auf Software-Sicherheitsmodulen. Auch wenn die verwendeten kryptographischen Algorithmen stark sind, so sind solche Module dennoch angreifbar. Eine Reihe von Angriffen auf diese Systeme zeigen, dass eine Entwendung von geheimem Datenmaterial ohne großen Aufwand möglich ist. Auf der anderen Seite bieten Trusted Computing Plattformen viele neue Möglichkeiten, sensibles Datenmaterial sicher zu speichern. Durch den Aufbau einer geschützten Schlüsselhierarchie mit Hilfe des Trusted Platform Moduls (TPM) haben Anwendungen wie Webbrowser die Möglichkeit ihre Daten hardwarebasiert zu schützen. In dieser Arbeit wird eine Architektur vorgestellt, wie dies realisiert werden kann. Der Hauptteil der Architektur ist ein zentrales Schlüssel- und Credential-Management, das Anwendungen über eine Schnittstelle die Funktionalitäten des TPM zur Verfügung stellt. Sie können sich TPM-Schlüssel erzeugen und ihre Daten mit diesen verschlüsseln lassen. Von diesen Möglichkeiten kann ein Webbrowser Gebrauch machen, um Credentials besser zu schützen. In Form einer Browser-Erweiterung wird gezeigt, wie die Sicherheit des Credential-Managements von Mozilla Firefox erhöht werden kann. Im Vergleich zu softwarebasierten Lösungen ist es bei der vorgestellten TPM-basierten Lösung schwieriger, unauthorisiert an geheimes Datenmaterial zu kommen. Zudem kann die Lösung in bestehende Systeme integriert werden und kann für diese als sinnvolle Erweiterung dienen.

Abstract

Today, web browsers are one of the most used software products. As a consequence of the rapid growth of the world wide web, it's the web browser which has to get on with a never ending variety of web technologies. In times where privacy in the web is getting more and more in focus, developers of web browsers are concentrating on security and privacy features. There are a lot of different user credentials that must be protected. A common way of how this protection is implemented is the usage of software based security modules. However, a series of simple attacks show that these protection mechanisms are not adequate. The manipulation of these modules and the revealement of secret data is effortless. On the other hand Trusted Computing offers new ways for the secure storage of secret data. Based on the TPM, a protected key hierarchy can be built. If provided to applications, TPM functionalities can be used by them to store their secret data in a more secure way. In this thesis an architecture is presented which shows how these facilities can be brought to web browsers. As the main part of the architecture, a trusted key and credential management service implements functionalities that can be used by other applications to utilize the TPM services and the protected storage facilities. It is shown how these functionalities can be used by a browser extension to protect the user credentials. Through the basis on a hardware module and its shielded locations, the data is protected in a more secure and less vulnerable way.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Vernetzung und World Wide Web	1
1.2	Privatsphäre, Datenschutz und Datensicherheit im Web	2
1.3	Ziel und Aufbau dieser Arbeit	3
2	Webbrowser und Datenschutz	5
2.1	Webbrowser: Entstehung bis heute	5
2.1.1	Entwicklung des WWW	5
2.1.2	Webbrowser Architektur	7
2.1.3	Webbrowser Credentials	9
2.2	Grundlagen technischer Schutzmaßnahmen	13
2.2.1	Grundlagen der Kryptographie	13
2.2.2	Key Management	17
2.2.3	Kryptographische Hardware	18
2.2.4	Kryptographie APIs	23
2.3	Webbrowser-Sicherheit	28
2.3.1	Firefox Credential Management	28
2.3.2	Internet Explorer Credential Management	30
2.3.3	Analyse des Credential Managements	34
2.3.4	Third-Party Ansätze	35
3	Trusted Computing	39
3.1	Grundlagen	39
3.1.1	Trusted Computing Group (TCG)	40
3.1.2	Trusted Computing Konzepte	41
3.2	Trusted Computing System	42
3.2.1	Trusted Platform Module	43
3.2.2	Die Vertrauensanker	47
3.2.3	TPM Schlüsselverwaltung	51
3.2.4	TCG Software Stack	53
3.2.5	Trusted Computing Szenarien	55
3.3	Trusted Computing Projekte	58
3.4	TPM Symbiose	63

3.5	Sicherheit von Trusted Computing	65
3.6	Trusted Computing: Chancen, Risiken und Kritik	68
4	Trusted Webbrowser	71
4.1	Sicherheitsanforderungen	71
4.2	Lösungsansatz	72
4.3	Trusted Key and Credential Management	73
4.3.1	TKCMS Komponenten	74
4.3.2	Schlüsselhierarchie und Trust Domains	75
4.3.3	TKCMS Funktionsumfang	75
4.4	Trusted Browser	78
4.4.1	Credential Verwaltung	78
4.4.2	Schlüsselverwaltung	79
5	Implementierung	81
5.1	Entwicklungsumgebung	81
5.2	Trusted Key and Credential Manager	82
5.2.1	Implementierungsziele	82
5.2.2	TKCM und Credential Database	82
5.2.3	Key und Datahandles	83
5.2.4	Kommunikationsprotokoll	84
5.2.5	Kommandos	85
5.3	TrustFox: Trusted Firefox Extension	87
5.3.1	Implementierungsziele	87
5.3.2	Eingesetzte Techniken	87
5.3.3	TrustFox Arbeitsweise	90
6	Diskussion	95
6.1	Sicherheit	95
6.2	Benutzbarkeit	98
6.3	Ähnliche Arbeiten und Architekturen	98
6.4	Fazit und Ausblick	100
	Abbildungsverzeichnis	103
	Tabellenverzeichnis	105
	Quellcodeverzeichnis	107
	Abkürzungsverzeichnis	110
	Literaturverzeichnis	111
A	Inhalt der CD	121

1. Einleitung

Datenleck, Datenpanne, Datenklau oder *Datenskandal* sind Begriffe, die in den Medien immer häufiger zu hören sind [83; 87; 92]. Können die Ausmaße der entstandenen Schäden doch stark variieren, so beschreiben alle Begrifflichkeiten im Grunde mehr oder minder stark das Versagen oder die Unangemessenheit von Mechanismen zum Schutz von bestimmten Daten. Die Gründe hierfür können unterschiedlicher Natur sein. Vom menschlichen Versagen über Vorsatz bis hin zu nicht adäquaten technischen Schutzmechanismen in den verarbeitenden Rechner-Systemen. Die Vermehrung solcher Datenlecks hängt sicherlich mit den Entwicklungen in der Informationstechnik zusammen. Niedrige Kosten für digitalen Speicher, praktisch unendliche Aufnahmekapazitäten und die weltweite Vernetzung von Rechnern sind Faktoren, die zu einer massiven Ansammlung von digitalen Daten führen. Je nach Zweck und Herkunft der Daten kann ein Verlust oder ein unbefugter Zugriff gravierende Folgen haben. Solche Vorkommnisse verdeutlichen, dass viele der eingesetzten Maßnahmen keineswegs ausreichend sind.

1.1. Vernetzung und World Wide Web

Ein weiterer wichtiger Faktor, der Datenlecks begünstigt, ist die bereits erwähnte enge Vernetzung von Rechnern. Das Internet eliminiert sämtliche ortsabhängigen Barrieren, so dass ein Informations- bzw. Datenfluss zwischen zwei beliebigen Rechnern, die sich an den unterschiedlichsten Orten der Welt befinden können, grundsätzlich möglich ist. Das *World Wide Web (WWW)* trug entscheidend zur rasanten Ausdehnung des Internets bei. Was zunächst hauptsächlich eine komfortable Möglichkeit war, entfernte Daten auf dem eigenen Rechner anzeigen zu lassen, entwickelte sich zum globalen Massenmedium. Über die Jahre ergaben sich immer mehr praktische Nutzungsszenarien. Das WWW hat sich zu

1. Einleitung

einer Allzweckplattform entwickelt, die in allen Lebensbereichen, von Freizeit bis Geschäft, genutzt werden kann. Auf der anderen Seite sind es *Webbrowser*, deren Aufgaben komplexer geworden sind. Die Bedeutung von Webbrowsern ist mit der Bedeutung des Webs gewachsen und so auch die Ansprüche, die ihnen gestellt werden, um dem modernen Web gerecht zu werden. Sie unterstützen Menschen bei Recherche, Einkauf, Kommunikation, Unterhaltung, eLearning, Banking und vielem mehr. Diese vielfältigen Anwendungsszenarien bringen vielfältige Verwaltungsaufgaben mit sich. Ein Browser übernimmt die Verwaltung und den Schutz von Passwörtern, Lesezeichen, Chronik, Zertifikaten, Cookies, um dem Nutzer die Arbeit zu erleichtern, oder aber nützliche Mechanismen zu ermöglichen. Dabei werden ihnen auch viele sicherheitsrelevante Aufgaben anvertraut. Es ist daher nicht verwunderlich, dass viele der eingangs erwähnten Sicherheitslücken auch in Zusammenhang mit Webbrowsern stehen.

1.2. Privatsphäre, Datenschutz und Datensicherheit im Web

Eine der wichtigsten Entwicklungen des Webs ist der Persönlichkeitsbezug. In den Anfangsphasen war eines der Hauptmerkmale des Webs, die Möglichkeit anonym Meinungen auszutauschen. Dies hat sich in den letzten Jahren grundlegend geändert. Der medienwirksame Begriff *Web 2.0* [66] umfasst den Wandel bei den Nutzern des Webs. Ihre Rolle ist es nicht mehr nur zu konsumieren. Die Gestaltung der Inhalte des Webs liegt nun immer mehr bei ihnen selbst. Menschen bilden Profile auf Online-Plattformen mit Daten über ihre "wahren" Identitäten, sie knüpfen Kontakte in *sozialen Netzwerken* und berichten über ihren Alltag in *Weblogs*. Dadurch erhält die Privatsphäre vermehrt Einzug in das WWW. Dieser Wandel des Webs spiegelt sich dementsprechend auch bei Webbrowsern wider. Die Entwicklung von Mechanismen zur Sicherheit und zum Schutz der Privatsphäre liegt nun mehr im Vordergrund, im Gegensatz zur Leistungs- und Geschwindigkeitsorientierung bei der Entwicklung früherer Browser. Die umgesetzten Maßnahmen fangen auf der technischen Ebene an. Sämtliche Daten, die in Bezug zur Person stehen, sei es sicherheitskritisch oder die Privatsphäre betreffend, müssen durch angemessene Mechanismen geschützt werden. Ist an den meisten Lösungen theoretisch und

algorithmisch wenig zu bemängeln, so zeigt sich in der Praxis, dass der Schutz nur so lange gewährleistet ist, wie die Implementierungen das tun, was sie sollen. Relativ einfache Manipulationen können – ungeachtet der Stärke der eingesetzten Verfahren – sämtliche Schutzmaßnahmen aushebeln.

1.3. Ziel und Aufbau dieser Arbeit

In dieser Arbeit sollen die in gängigen Webbrowsern eingesetzten Techniken zum Schutz sensibler Daten näher betrachtet werden, um ein Urteil darüber bilden zu können, in wie fern diese Mechanismen ausreichend sind, bzw. unter welchen Gesichtspunkten eine Verbesserung möglich ist. Dabei wird untersucht, ob und wie hierbei *Trusted Computing (TC)* unterstützend sein kann. TC bietet unter anderem Möglichkeiten, sensible Daten hardwarebasiert zu schützen, was potenziell einige wichtige Schwachstellen gängiger Systeme eliminieren kann.

Die vorliegende Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden zunächst einige methodische Grundlagen vorgestellt, die für die Umsetzung von Sicherheitskomponenten nötig sind. Im selben Kapitel wird auf die Architektur und die vorgesehenen Schutzmaßnahmen bei Webbrowsern eingegangen. Kapitel 3 bietet eine Einführung in Trusted Computing Systeme. In Kapitel 4 wird eine Architektur für das Credential Management von Webbrowsern vorgestellt, die von den Möglichkeiten einer Trusted Computing Plattform Gebrauch macht. In Kapitel 5 wird auf die Details der Implementierung eines Prototyps für die vorgestellte Architektur eingegangen, woraufhin in Kapitel 6 eine Analyse des Systems folgt. Im selben Kapitel werden einige ähnliche Arbeiten vorgestellt und ein Fazit gebildet.

1. *Einleitung*

2. Webbrowser und Datenschutz

Die Verantwortung von Webbrowsern liegt vermehrt im sicheren Umgang von sensiblen Daten und der Ermöglichung sicherer Kommunikation. Beim Design und der Entwicklung ist vor allem auf angemessene Schutzmechanismen rund um Sicherheit, Datenschutz und Privatsphäre zu achten. Unabhängig vom eingesetzten Kontext, basieren die meisten Lösungen der Informationssicherheit auf grundlegenden Werkzeugen, die dies ermöglichen. Nach einer kurzen Einführung in die Geschichte des WWW und Webbrowsern werden diese Werkzeuge in den nächsten Abschnitten vorgestellt. Daraufhin werden die konkreten Mechanismen in gängigen Webbrowsern dargestellt und analysiert.

2.1. Webbrowser: Entstehung bis heute

In diesem Abschnitt wird ein kurzer Überblick über die Geschichte und weitere Entwicklung von Webbrowsern gegeben. Dies soll ein besseres Verständnis für die Bedeutung von Webbrowsern vom heutigen Standpunkt aus ermöglichen. Anschließend wird die heutige Marktsituation betrachtet.

2.1.1. Entwicklung des WWW

Die Entstehungsgeschichte von Webbrowsern hängt naheliegenderweise sehr eng mit der des World Wide Webs zusammen. Im Jahr 1989 setzte nämlich hauptsächlich Tim Berners-Lee seine Ideen für das WWW in die Tat um, in dem er sowohl die Server- als auch die Client-Software – einen Webbrowser – programmierte [6]. So wenig die Erfindung der für das WWW maßgeblichen Protokolle mit der Erfindung des Internets gleichzusetzen ist, ist das Web doch diejenige Anwendung, die es für eine sehr breite Masse interessant machte. Die ersten Webbrowser waren

2. Webbrowser und Datenschutz

jedoch rein textbasiert. Charakteristisch für die Bedeutung von Webbrowsern, war es im Jahr 1993 *Mosaic*, der endgültig den Hype für das WWW auslöste. Mosaics Erfolg beruhte hauptsächlich in der Einfachheit seiner Installation und Bedienung. Seitdem ist das Web rasant gewachsen und hat sich mit der Entwicklung zahlreicher Webtechnologien auch in Funktionalität entfaltet. Die Versionierung des Webs – bezogen auf den Begriff Web 2.0 – ist zwar aus technischer Sicht nicht korrekt, jedoch umso mehr aus sozialer Betrachtungsweise. Menschen sind heute sehr stark mit dem Web verbunden und wirken an dessen Gestaltung maßgeblich mit. Ob soziale Netzwerke, Audio- und Videoportale, Weblogs, Fotoportale oder soziales Bookmarking; diese Dienste leben in gewisser Weise von ihrem Persönlichkeitsbezug. Anonymität spielt im Web 2.0 eine eher untergeordnete Rolle. Vielfältige Webanwendungen, wie bspw. Office-Tools machen klassische Desktop-Anwendungen über den Webbrowser und das Web nutzbar. Dadurch entfällt die Gebundenheit an einen bestimmten Rechner. Auf persönliche Dokumente kann von jeder beliebigen Plattform mit Internetzugang zugegriffen werden. Unterstützt und realisiert werden solche komplexe Webanwendungen von zahlreichen Technologien, wie HTML, XML, PHP, ASP, JavaScript, Flash, Java, AJAX, CSS und vielen mehr. Dies lässt erahnen, mit welchen Aufgaben moderne Webbrowser konfrontiert werden. Diese Vielfalt an Aufgaben ist umso schwieriger zu handhaben. Zahlreiche, auch sicherheitskritische Punkte müssen beachtet werden. Denkt man bspw. an Online-Banking, das in der Regel über eine Weboberfläche abläuft, so lässt sich erahnen, welchen Grad an Sicherheit Webbrowser erfüllen müssen.

Webbrowser-Markt

Der Webbrowser-Markt steht sehr stark im Fokus der Fachpresse. Nicht ohne Grund sind es Meldungen über Sicherheitslücken in Webbrowsern, die das größte Aufsehen erregen. Gerade Sicherheitsfeatures stehen besonders im Vordergrund. Denkt man an den “*ersten Browser-Krieg*” Mitte bis Ende der neunziger Jahre, war es hauptsächlich der Navigator von Netscape und Microsofts Internet Explorer, die den Browser-Markt dominierten. Nachdem Microsoft Marktanteile von bis zu 90% erreichte [89], kam erst ab 2004 mit der Veröffentlichung von Mozilla Firefox wieder Dynamik in den Markt. Spätestens seit dem Markteintritt von *Google Chrome* im Dezember 2008 spricht man vom “*zweiten Browser-Krieg*”[74]. Weitere vielgenutzte Browser sind *Opera* und *Safari*. Abhängig von der Quelle

2.1. Webbrowser: Entstehung bis heute

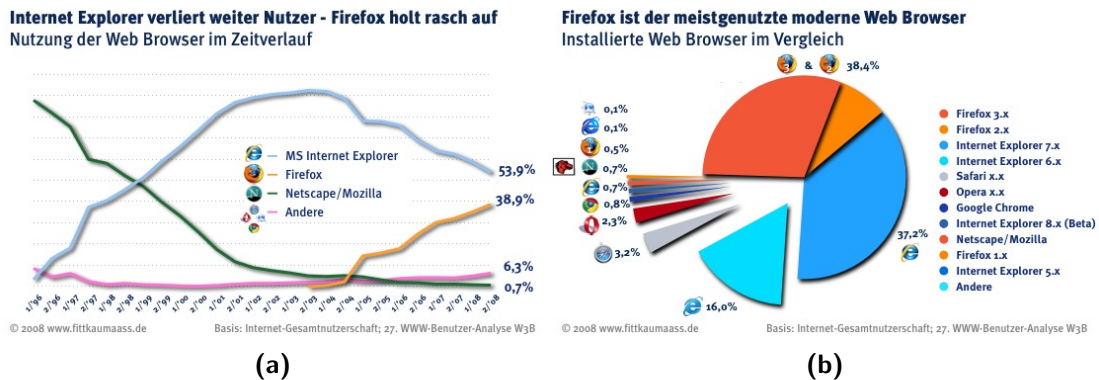


Abbildung 2.1.: Marktanteile der verschiedenen Browser. (a) zeigt die Entwicklung der Marktanteile seit 1996. (b) stellt die Verteilung der Marktanteile für Anfang 2009 dar. Quelle: www.fittkaumaas.de

variieren die Statistiken über Marktanteile sehr stark, doch Abbildung 2.1 verdeutlicht zumindest tendenziell, wie groß die jeweiligen Anteile sind. Die Anbieter von Browsern der jüngsten Generation stellen gerade die Funktionalitäten rund um Sicherheit und den Schutz der Privatsphäre in den Vordergrund, um für ihre Produkte zu werben. Schließlich sind dies die Angelegenheiten, mit denen die Nutzer in Zeiten von Datenklau und Datenpannen am ehesten aufmerksam gemacht werden können.

2.1.2. Webbrowser Architektur

Webbrowser gibt es für die unterschiedlichsten Betriebssysteme und Plattformen wie PCs, PDAs, Handys und viele mehr. Die teilweise großen Unterschiede der Systeme (Displaygröße, CPU-Leistung, Arbeitsspeicher, etc.) stellen unterschiedliche Anforderungen an die Implementierungen. Eine generische Referenzarchitektur für Webbrowser, die die grundlegenden Eigenschaften von solchen beschreibt, kann Entwicklern von Webbrowsern daher eine hilfreiche Richtlinie sein. Eine solche Referenzarchitektur wird in [30] vorgestellt. Die Architektur ist das Resultat einer Analyse der Quellcodes der Browser Mozilla und Konqueror aus dem Jahr 2005. Die Referenzarchitektur konnte jedoch bei fünf weiteren Webbrowsern bestätigt werden. Demnach bestehen Browser aus diversen Modulen, die anhand der

2. Webbrowser und Datenschutz

Aufgabenbereiche klassifiziert werden können. Die in Abbildung 2.2 dargestellten Komponenten sind:

- Das *User Interface* bildet die Schnittstelle zwischen dem Benutzer und dem Browser. Es erledigt alle Aufgaben, die mit der visuellen Darstellung von Browser-Aktivitäten zu tun haben (Toolbars, Steuerelemente, etc.)
- Die *Browser Engine* bietet eine Schnittstelle zur Rendering Engine. Zu den Aufgaben gehören das Laden von Webseiten und die Umsetzung von Navigations-Funktionen wie bspw. das *Zurück-* und *Vorwärtsnavigieren*. Zudem werden hier Funktionalitäten implementiert, die es erlauben Zustände von Browser-Sitzungen einzusehen und zu manipulieren (z.B. Status des Ladens einer Seite)
- Die *Rendering Engine* ist dafür zuständig visuelle Repräsentationen der erhaltenen Webseiten (oder anderen Daten) zu erstellen. Dazu gehört bspw. die Übersetzung von HTML-Tags und die Umsetzung von Cascading Style Sheets (CSS), oder auch die Anzeige von Bildern, Audio oder Video.
- Das *Networking*-Subsystem setzt sämtliche Kommunikationsprotokolle um, die für das Browsen benötigt werden. Dazu gehört vor allem die Implementierung des HTTP-Clients.
- Der *JavaScript* Interpreter wertet JavaScript Code aus, der in HTML-Seiten eingebettet ist und führt ggf. die entsprechenden Aktionen aus.
- Der *XML Parser* zerlegt Dokumente in einen DOM¹-Baum. DOM-Bäume erlauben eine einfache Navigation innerhalb von Dokumenten.
- Das *Display Backend* liefert sämtliche Funktionalitäten, die für das Erzeugen von UI-Elementen (Widgets, Fonts, etc.) nötig sind.
- Das *Data Persistence*-Subsystem ist für die Verwaltung und Speicherung sämtlicher nutzerbezogenen Daten zuständig. Unter anderem beinhaltet dies den Schutz sensibler Daten wie Zertifikate und Passwörter.

¹Document Object Model

2.1. Webbrowser: Entstehung bis heute

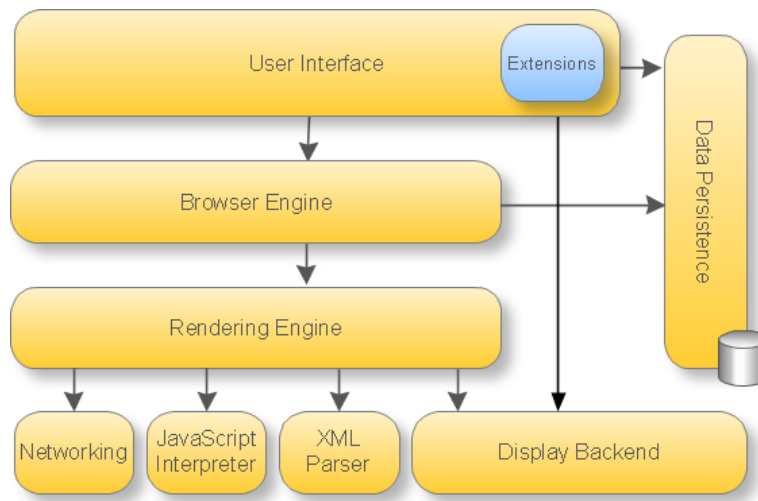


Abbildung 2.2.: Referenzarchitektur von Webbrowsern

Diese Referenzarchitektur ist keinesfalls als Vorgabe zu betrachten. Diese Struktur ist lediglich das Ergebnis der Analyse von einigen wenigen quelloffenen Browsern und spiegelt lediglich die bisherigen Vorgehensweisen bei der Entwicklung von Browsern wider ohne auf die Angemessenheit dieser einzugehen. Andererseits ist solch ein modularer Aufbau naheliegend. Dabei können sich verschiedene Implementierungen in manchen Details unterscheiden. Beispielsweise können mehrere Module je nach Umfang zusammengefasst oder weiter unterteilt werden. Zum Beispiel ist die Erweiterbarkeit eine besondere Eigenschaft von Mozilla Firefox. Sämtliche Browserelemente können durch Erweiterungen sowohl in Aussehen, als auch in Funktionalität modifiziert werden. In der Referenzarchitektur kann dies durch eine zusätzliche Komponente zwischen User Interface und Data Persistence dargestellt werden (blau in Abbildung), da sämtliche Elemente einer Erweiterungen aus dem Datenspeicher geholt werden.

2.1.3. Webbrowser Credentials

Webbrowser gehören zu den wichtigsten und meist genutzten Softwareprodukten. Im Fokus bei der Entwicklung von Browsern stehen dabei immer mehr die Mechanismen, die für Sicherheit und Privatsphäre sorgen sollen. Dies ist u.a. dadurch bedingt, dass ein Browser mit immer mehr nutzerbezogenen Daten zu tun

2. Webbrowser und Datenschutz

hat, die die Arbeit mit dem Web erleichtern sollen. Im Folgenden werden die wichtigsten dieser *Webbrowser Credentials* und ihre Funktionen näher vorgestellt.

Login Credentials

Die Vielfalt an Online-Plattformen erfordert das Verwalten von vielen Authentifizierungsinformationen. Die meistgenutzte Methode dabei ist immer noch das Verwenden von Passwörtern. Die meisten Webbrowser bieten dem Nutzer integrierte Passwort-Manager an, die die Authentifizierungsinformationen für sie verwalten und bei Bedarf in die entsprechenden Formularfelder einfügen. Dies erfordert das Speichern dieser Daten auf der Festplatte. Als gravierende Folge eines inadäquaten Schutzes dieser Daten hätten Unbefugte Zugriff auf sämtliche Konten des Benutzers auf Online-Plattformen.

Zertifikate

Viele Transaktionen über das WWW laufen über gesicherte Verbindungen, da oft sensible Daten über das (unsichere) Internet versendet werden. Um zusätzlich eine Sicherheit darüber zu erhalten, ob es sich bspw. beim Online-Banking tatsächlich um den HTTP-Server der Bank handelt, mit dem kommuniziert wird, muss der Webbrowser ein Zertifikat, das vor Beginn der Transaktion vom Server an den Client gesendet wird, auf Gültigkeit überprüfen. Die im Web für solche Zwecke aufgebaute *Public Key-Infrastruktur* erfordert, dass der Browser die gesamte Kette der Hierarchie vom Server-Zertifikat, bis hin zur Zertifizierungsstelle (*Certification Authority, CA*) durchwandert, und jeweils die digitalen Signaturen überprüft. Das Zertifikat am Ende dieser Kette muss der Browser kennen, da sonst keine Validierung durchgeführt werden kann. Bei Erfolg akzeptiert der Browser das Zertifikat und somit auch die Verbindung. Gängige Webbrowser besitzen eine Reihe von *Root-Zertifikaten* von verschiedenen *Trust Centern*, die jeweils die Wurzel einer Validierungskette darstellen. Der Eintrag eines Root-Zertifikats in der Datenbank eines Webbrowsers hat zur Folge, dass sämtliche Zertifikate, die sich in der Hierarchie unterhalb dieses Zertifikats befinden, vom Browser ohne weiteres akzeptiert werden. Somit ist der Pool der Root-Zertifikate sehr gut zu schützen. Wenn man an das obige Beispiel des Online-Bankings knüpft, hätte ein gefälschter Eintrag gravierende Folgen. Nicht jedes Server-Zertifikat kann erfolgreich vom Browser validiert werden. Das kann z.B. damit zu tun haben, dass

2.1. Webbrowser: Entstehung bis heute

der Browser kein Zertifikat der entsprechenden CA besitzt, mit dem das Server-Zertifikat signiert wurde. Eine gängige Vorgehensweise ist, den Nutzer darüber zu informieren und ihm die Option anzubieten, dem Server zu trauen und das Zertifikat manuell zu akzeptieren. In diesem Fall wird das Server-Zertifikat entweder nur für die Sitzung akzeptiert, oder dauerhaft in einer zusätzlichen Zertifikat-Datenbank abgelegt. Im zweiten Fall wird der Browser bei zukünftigen Aufrufen, bei denen der Server dieses Zertifikat vorweist, keine Warnung mehr zeigen. Eine Manipulation dieser Datenbank muss verhindert werden, da durch den Eintrag eines einzigen (gefälschten) Zertifikats viele wichtige Schutzmechanismen ausgehebelt werden können.

Cookies

Cookies[27] ermöglichen Zustandsverwaltung im an sich zustandslosen HTTP. Cookies sind *Name/Wert*-Paare, die nach einer Anfrage serverseitig erzeugt, und dem Response-Header hinzugefügt werden. Bei zukünftigen Anfragen kann der Client diesen Cookie an die Anfrage hängen. Der Server hat somit eine Möglichkeit, die Anfrage mit einer früheren Anfrage in Verbindung zu setzen, um den Anfragenden sozusagen zu “identifizieren”. Ein gängiges Anwendungsbeispiel ist ein Webshop mit dem Warenkorb-Prinzip. Während der Kunde weiterhin auf der Shop-Website navigiert, kann sich der Server mit Hilfe der Cookies *merken*, was der Kunde in seinem Warenkorb hat. Jedoch besitzt dieser Mechanismus das Potenzial, zweckentfremdet zu werden. Cookies können als Mittel zur Spurenerfolgung von Websurfern verwendet werden. Dies führte zu vielen Diskussionen über Cookies im Zusammenhang mit Datenschutz und Privatsphäre [52]. In vielen Szenarien wird ein Cookie für Sitzungsverwaltung, bspw. für eine Sitzung im Web-Interface eines E-Mail-Providers verwendet. Ein Cookie wird nach dem Einloggen erzeugt, und an den Client gesendet. Der Cookie dient für die Dauer der (verschlüsselten) Sitzung als Berechtigungsnachweis. Dies verhindert, dass der Nutzer jedes Mal sein Passwort eingeben muss, wenn er die Seite innerhalb der Web-Schnittstelle wechselt. Sollte der Cookie während der Sitzung an einen unbefugten Dritten gelangen, hätte derjenige Zugriff auf das E-Mail-Konto. *Surf Jacking* beweist, dass über Cookies sogar SSL, bzw. TLS ausgehebelt werden kann [28]. Bei dieser Attacke zwingt der Angreifer den Client, ein (Sitzungs-)Cookie über eine unverschlüsselte Verbindung zu senden. Er kann den Cookie übernehmen und für

2. Webbrowser und Datenschutz

eine eigene Anfrage benutzen. Der Server erkennt den Angreifer als den legitimen Nutzer an, womit dieser Zugriff auf das gesamte Konto des Opfers hat.

DOM Storage / Web Storage

Mit HTML 5 wird eine zu Cookies sehr ähnliche Technologie eingeführt, die vor allem webbasierte Anwendungen unterstützen soll. Unter dem Namen *Web Storage* führt die W3C einen neuen Mechanismus ein, wie clientseitig strukturierte Daten abgelegt werden können [15]. Hierzu werden die zwei DOM-Attribute *sessionStorage* und *localStorage* eingeführt. Ersteres ist ein Speicherbereich für *Name/Wert*-Paare, der für jeden Browserkontext (Fenster, Tab, etc.) getrennt erzeugt wird. Bspw. kann dadurch folgende Schwachstelle von Cookies beseitigt werden. Ein Nutzer öffnet zwei Browserfenster, weil er zwei verschiedene Flugtickets (vom selben) Online-Reisebüro kaufen möchte. Werden hierfür Cookies eingesetzt, kann dies dazu führen, dass der Nutzer am Ende zweimal das identische Ticket bestellt hat, ohne dies zu wollen. Dies hängt damit zusammen, dass in der Regel für die beiden Browserfenster keine getrennten Credentialspeicher existieren und somit dieselbe Cookie-Datenbank benutzt wird. Im Gegensatz dazu ist beim *sessionStorage* vorgesehen, dass jeder *top-level browsing context* einen eigenen Speicherbereich initiiert, der nur von diesem Kontext verwendet wird. Über das *localStorage*-Objekt erhält jede Webseite eine eigene Speicherdomäne beim Client, in dem wie auch bei Cookies *Name/Wert*-Paare abgespeichert werden können. Anders als Cookies, die eine maximale Größe von 4KB besitzen können, haben Webseiten die Möglichkeit mehrere Megabyte an Daten abzulegen. Dies soll zu einer erhöhten Leistung von Web 2.0-Anwendungen führen, indem bspw. Nutzerdaten zwischenzeitlich lokal ausgelagert werden. Aufgrund der Ähnlichkeiten zu Cookies besitzen Storage-Objekte die gleichen potenziellen Risiken (Tracking, etc.)². Konsequenterweise sollten diese Speicher mit geeigneten Mechanismen geschützt werden. Denn anders als bei Cookies kann das *localStorage* dafür verwendet werden, eine fast beliebige Menge an Daten unmittelbar beim Client abzulegen. Cookies hingegen dienen nur als "Verweis" auf eine Ansammlung von Daten, die beim Server verwaltet werden. Umso strikter müssen Richtlinien eingehalten werden, die dafür sorgen, dass Daten auch tatsächlich nur an die entsprechenden Server gesendet werden. Diese neuen Mechanismen wurden bei

²Dies führte u.a zu der Bezeichnung "Super Cookies"

2.2. Grundlagen technischer Schutzmaßnahmen

Firefox bereits mit Version 2.0 eingeführt. Internet Explorer setzt diesen Standard³ erstmals in IE 8 um. Es ist zu erwarten, dass es in naher Zukunft viele Diskussionen um diese neuen Mechanismen geben wird, was den Datenschutz und die Privatsphäre betrifft.

Browser History, Form History, Lesezeichen

Weiterhin gibt es eine Reihe von Credentials, die die Wiederverwendung von Webseiten oder Suchbegriffen erleichtern. Dazu gehören Lesezeichen, bzw. *Bookmarks*, die *Formhistory* oder die allgemeine *Chronik*, in der alle über einen bestimmten Zeitraum besuchten Webseiten gespeichert sind. Diese Daten sind im Bezug auf die Privatsphäre sehr sensibel, da sich aus ihnen viele Rückschlüsse auf die Vorlieben und Surf-Gewohnheiten eines Nutzers ziehen lassen. Man erhält einen Überblick darüber, welche Seiten zu welcher Zeit besucht, und welche Begriffe in eine Suchmaschine eingegeben wurden.

2.2. Grundlagen technischer Schutzmaßnahmen

Unabhängig von der konkreten Anwendung basieren die meisten Produkte für Informationssicherheit auf denselben Grundlagen. Auf methodischer Ebene sind es Verfahren und Algorithmen aus der Kryptographie. Auf diesen grundlegenden Methoden bauen alle Implementierungen auf. In diesem Abschnitt wird ein kurzer Überblick über die vorhandenen Werkzeuge gegeben, die auch für die Entwicklung von Schutzmaßnahmen für Webbrowser Credentials nützlich sind. Hierbei wird nur so tief ins Detail gegangen, wie es für das Verständnis der folgenden Kapitel notwendig ist. Gute Einführungen zu diesem Thema sind in [56],[73], [21] und [80] zu finden.

2.2.1. Grundlagen der Kryptographie

Verschlüsselung

Mit Verschlüsselung wird das Ziel der *Vertraulichkeit* erreicht. Vereinfacht dargestellt ist die Vorgehensweise eines jeden Kryptosystems dieselbe. Eine Nachricht

³Zum 28. April 2009 noch ein Entwurf

2. Webbrowser und Datenschutz

M wird mit Hilfe eines Schlüssels K dermaßen transformiert, dass ein *Zifferntext* C des Textes resultiert, aus dem die ursprüngliche Information nicht ohne weiteres zu extrahieren ist. Die Entschlüsselung wird entsprechend umgekehrt durchgeführt. Wie genau die Transformation aussieht, und welche Schlüssel jeweils zur Ver- und Entschlüsselung verwendet werden, macht die Unterschiede zwischen den verschiedenen Verfahren aus. Ein Charakteristikum anhand dem sich Verschlüsselungsverfahren in zwei Klassen einteilen lassen, ist der für die Ver- und Entschlüsselung verwendete Schlüssel. Ist es jeweils derselbe, so spricht man von *symmetrischer Verschlüsselung*, was die symmetrische Informationsverteilung zwischen Sender und Empfänger beschreibt. Beide besitzen denselben Schlüssel. Werden für beide Vorgänge jeweils unterschiedliche Schlüssel verwendet, so spricht man von *asymmetrischen Verschlüsselungsverfahren*. Eine Aussage darüber, welche Variante besser ist, lässt sich nur in Abhängigkeit vom Einsatzszenario machen. So haben beide Varianten ihre Vor- und Nachteile. Allgemeiner Nachteil einer symmetrischen Verschlüsselung ist die Verteilung des Schlüssels an alle Kommunikationspartner. Wenn dies über ein Rechnernetz geschieht, besteht das Risiko, dass die Informationen von Dritten abgehört werden. Jedoch sind auch Szenarien möglich, in denen Sender und Empfänger identisch sind, bspw. verschlüsselt ein Nutzer bestimmte Daten, um sie zu einem späteren Zeitpunkt wieder verwenden zu können. Ein Schlüsselaustausch ist hier nicht nötig. Ein weiteres Manko ist die komplexe Schlüsselverwaltung. Pro Kommunikations-Paar wird ein geheimer Schlüssel benötigt. Bei Größen, wie sie im WWW zu finden sind, kann das immense Dimensionen annehmen⁴. Genau diese beiden Nachteile werden von asymmetrischen Verfahren eliminiert. Wie bereits erwähnt, sind bei diesen Verfahren der Schlüssel für Ver- und Entschlüsselung jeweils unterschiedlich. Je nach Algorithmus ist der eine Schlüssel in einer bestimmten Weise von dem anderen abgeleitet, so dass sie ein *Schlüsselpaar* bilden, bei dem der eine Schlüssel zur Verschlüsselung, und nur der andere zur Entschlüsselung herangezogen werden kann. Ohne auf Details einzugehen, sei lediglich erwähnt, dass es somit möglich ist, den Schlüssel der für die Verschlüsselung verwendet wird, öffentlich bekannt zu geben. Für die Entschlüsselung kann nur der andere (geheime) Schlüssel herangezogen werden. Asymmetrische Verfahren werden daher auch *Public Key-Verfahren* genannt. Großer Nachteil asymmetrischer Verfahren, wie *RSA* [70] ist die geringe

⁴ $\frac{n^2-n}{2}$ Schlüssel bei n Teilnehmern. Ergibt bei 1000 Teilnehmern bereits knapp 500000 Schlüssel

2.2. Grundlagen technischer Schutzmaßnahmen

Leistungsfähigkeit der Implementierungen. Wo es sinnvoll ist werden in der Praxis symmetrische und asymmetrische Verfahren kombiniert und als *hybride Lösungen* eingesetzt, um die Vorteile beider Systeme auszunutzen. So werden in Protokollen wie TLS die symmetrischen Sitzungsschlüssel zunächst über asymmetrische Verfahren ausgetauscht. Fakt ist, dass Public-Key Kryptographie Informationssicherheit revolutionierte. Durch sie sind viele Szenarien realisierbar, die mit klassischen Kryptosystemen nicht möglich waren.

Kryptographische Hashfunktionen und Digitale Signaturen

Hashfunktionen werden in den verschiedensten Szenarien eingesetzt. Sie haben auch in kryptographischen Protokollen eine wichtige Rolle. Sogenannte *kryptographische Hashfunktionen* (auch *Einweg-Hashfunktionen* genannt) sind gewöhnliche Hashfunktionen, denen jedoch besonders hohe Anforderungen gestellt werden. Diese lauten [73]:

- Gegeben eine Nachricht M , ist es einfach $H(M)$ zu berechnen.
- Gegeben $H(M)$, ist es schwer M zu berechnen.
- Gegeben M , ist es schwer M' zu finden, so dass $H(M) = H(M')$.

Letzteres ist in kryptographischen Kontexten deshalb so wichtig, weil Hashfunktionen eingesetzt werden, um bspw. die Integrität von wichtigen Daten oder Programmbibliotheken zu wahren. Kritisch ist das beim Einsatz dieser Hashfunktionen bei *digitalen Signaturen*. Hier werden die Hashwerte von zu signierenden Daten, als *Fingerabdruck* der eigentlichen Nachricht M genutzt, um die Signatur zu erstellen. Kann ein Angreifer eine Nachricht $M' \neq M$ mit $H(M') = H(M)$ erzeugen, so kann er diese mit der bereits erstellten Signatur von M versehen. Für einen Dritten wäre dies eine gültige Signatur von M' . Weitverbreitete Hash-Algorithmen sind *Message-Digest Algorithm 5 (MD5)* und die *Secure Hash Algorithm (SHA)-Familie*. MD5 gilt seit Längerem als unsicher [10]. Auch auf SHA-1 wurden bereits erfolgreiche Angriffe durchgeführt [90]. Die Existenz von starken kryptographischen Hash-Algorithmen ist auch noch in Zukunft von wichtiger Bedeutung. Die *National Institute of Standards and Technology (NIST)* hat daher im Jahr 2006 einen Wettbewerb für die Standardisierung eines neuen Hash-Algorithmus ausgesprochen. Der neue Standard soll 2012 veröffentlicht werden [65].

Generierung von Zufallszahlen

Kryptographische Algorithmen wie Verschlüsselungsverfahren benötigen diverse Parameter, die bspw. für die Erzeugung von kryptographischen Schlüsseln gebraucht werden. Diese Parameter müssen möglichst zufällig erzeugt werden, um eine Voraussage über die resultierenden Werte ausschließen zu können. Hier kommen *Zufallszahlengeneratoren* (engl. *Random Number Generators, RNG*), bzw. *Zufallssequenzen-Generatoren* ins Spiel. Die erzeugten Sequenzen sollten in keiner Weise vorhersagbar sein. Echte Zufälligkeit können deterministisch arbeitende Rechner jedoch nicht erzeugen. Daher gilt es bei der Zufallszahlengenerierung Algorithmen zu finden, die *so zufällig, wie möglich* arbeiten. Man bezeichnet sie daher als *Pseudo-Zufallszahlen-Generatoren* (PRNG). Die allgemeine Vorgehensweise besteht darin, möglichst viele Parameter aus dem Systemzustand zu sammeln und einen *Entropie-Pool* zu erzeugen. Die Daten in diesem Pool dienen dann als Grundlage für die eigentliche Generierung von Zufallszahlen. Diese Parameter können aus der aktuellen Systemzeit, bzw. den Messungen von Zeitintervallen bestimmter Ereignisse, Prozessnummern, Benutzereingaben (bspw. Mausbewegungen), oder sonstigem bestehen. Ist der Entropie-Pool hinreichend gefüllt, können mit ihm Pseudo-Zufallszahlen erzeugt werden, die für den jeweiligen Einsatzzweck zufällig genug sind. Dass es hierbei auch um eine saubere Implementierung des Verfahrens ankommt, ist die Lehre des sogenannten *Debian-Debakels* [40]. Die Schwachstelle führte letztendlich dazu, dass eine erzeugte Zufallszahl nur $2^{15} = 32768$ mögliche Werte annehmen konnte. Somit wäre der Schlüssel kryptographisch nicht stark genug, da man durch einfaches ausprobieren aller möglichen Werte, auf den Wert des Schlüssels kommen kann. Sämtliche Zertifikate, die auf solchen *schwachen Schlüssel* basieren (der Fehler blieb zwei Jahre lang unentdeckt), können ohne weiteres gefälscht werden. Welche fatalen Folgen das mit sich bringen kann, macht deutlich, wenn man daran denkt, dass CAs solche Schlüssel für ihre Zertifikate verwendet haben könnten. Die gesamte Validierungskette, ausgehend von einem schwachen Zertifikat einer CA, kann somit auch gefälscht werden. Dieser Vorfall zeigt die Wichtigkeit von guten Zufallszahlengeneratoren und Implementierungen auf. Es ist nicht ausgeschlossen, dass nicht auch weitere Implementierungen ähnliche Fehler aufweisen.

2.2.2. Key Management

Die Anwendung von guten kryptographischen Algorithmen allein ist keine Garantie für die Sicherheit. Ob *Transport Layer Security (TLS)*, *Secure Shell (SSH)*, *Pretty Good Privacy (PGP)* oder *S/MIME*. Jeder Mechanismus, der Verschlüsselung einsetzt, muss für die Geheimhaltung der Schlüssel sorgen. Auf dieser Geheimhaltung basiert letztendlich die ganze Sicherheit. Dies beginnt bereits bei der Erzeugung der Schlüssel. Ist die Umgebung, in der sie generiert werden nicht ausreichend geschützt, kann eine Entwendung schon hier stattfinden. Darüber hinaus sind Schlüssel über ihre ganzen Lebenszyklen sicher aufzubewahren. Je nach Art und Zweck des Schlüssels kann dies ein längerer Zeitraum sein, wie bspw. bei PGP-Schlüsselpaaren, oder temporär für die Dauer einer Kommunikationssitzung mit einem Webserver. Abbildung 2.3 illustriert die Aufgaben eines *Key Managements*, welche folgendermaßen aufgelistet werden können [56]:

- Erzeugung, Verteilung und Installierung von Schlüsselmaterial;
- Kontrolle der Nutzung von Schlüsselmaterial;
- Aktualisierung, Widerrufung und Zerstörung von Schlüsseln;
- Speicherung, Backup/Wiederherstellung, und Archivierung von Schlüsselmaterial.

Die Verwaltung von Schlüsseln ist somit eine komplexe Aufgabe, die auch als der *schwierigste Teil der Kryptographie* bezeichnet [73] wird. Dies zeigt sich auch darin, dass es eine Reihe von Standards und Publikationen [17; 25; 44; 69] gibt, die sich allein dieser Aufgabe widmen. Dies ist jedoch auch erforderlich, da das Key Management häufig die schwächste Stelle von Kryptosystemen bildet. Ist das zugrunde liegende Key Management schwach, so dann auch das gesamte System. Um die Problematik zu verdeutlichen, denke man an das übliche Szenario eines *Multi-User-Betriebssystems*. Verwendet Benutzer *A* einen geheimen Schlüssel für eine beliebige Operation, so muss dieser gezwungenermaßen in den Arbeitsspeicher geladen werden. *Cache Attacks* (bzw. *Side-Channel-, Timing-Attacks*) sorgen dafür, dass die Geheimhaltung des geheimen Schlüssels, nicht unbedingt gegeben ist [7; 67]. Resistenter gegen solche Angriffe sind spezielle kryptographische Hard-

2.2. Grundlagen technischer Schutzmaßnahmen

graphischen Operationen implementieren, und benutzte Schlüssel dabei innerhalb des Gerätes schützen. Vorteile von Hardware-Implementierungen sind u.a. [73]:

- **Geschwindigkeit.** In Hardware realisierte Algorithmen sind meistens leistungsfähiger, da sie für spezielle Aufgaben optimiert werden können⁵. Zudem kann durch die Auslagerung von rechenintensiven Krypto-Prozessen die CPU entlastet werden.
- **Sicherheit.** Werden sensible Prozesse vom allgemeinen System getrennt, ist eine Manipulation über fremde Prozesse nur schwer möglich, da HSM sicher vom restlichen System abgekapselt sind. Dies ermöglicht zudem die sichere Speicherung von sensiblen Daten, wie z.B. kryptographische Schlüssel.

Neben ATMs und militärischen Einsatzzwecken werden HSMs vermehrt in vielen weiteren Szenarien eingesetzt.

Smart Cards

Smart Cards sind ISO 7810-konforme Karten⁶ mit eingebauten elektronischen Funktionseinheiten. Die technischen Eigenschaften sind in ISO/IEC 7816 definiert [26]. Man unterscheidet zwei Kategorien von Smart Cards. *Speicher-Karten* enthalten hauptsächlich einen Speicher-Chip (bspw. EEPROM) und eventuell noch eine minimale Logikeinheit. *Prozessor-Karten* sind vergleichbar mit Mikrocontrollern. Sie besitzen neben Speichereinheiten noch eine Logik- bzw. Arithmetik-Einheit und eigenen Programmcode. Sämtliche Operationen können in der geschützten Umgebung der Smart Card durchgeführt werden, so dass geheime Schlüssel nicht nach außen weitergeleitet werden müssen. Die auf der Karte gespeicherten Schlüssel sind an diese Smart Card gebunden, so dass deren Benutzung nur durch den Besitz der Karte in Frage kommt.

Mit Smart Cards, ob Speicher- oder Prozessor-Karte, lassen sich vielfältige Anwendungen realisieren. So dienen sie als Datenträger bei Telefon-, Geld- oder Krankenversicherungskarte. Auch in sicherheitsrelevanten Szenarien werden sie häufig eingesetzt. *Subscriber Identity Modules (SIM)* dienen im Mobilfunk u.a. als Trä-

⁵In der Praxis sind jedoch bekannte Algorithmen, wie *Data Encryption Standard (DES)* oder RSA in Hardware nur ineffizient zu realisieren.

⁶Dieser Standard beschreibt das Format (Größe, etc.), wie man es bspw. von Kreditkarten kennt

2. Webbrowser und Datenschutz

ger von geheimen Schlüsseln, die zu Authentifizierungszwecken genutzt werden, im Bereich *Pay-TV* dienen Smart Cards als Dekoderkarte und in Zugangskontrollsystemen werden sie als Zugangsberechtigung verwendet. Auch im Internet gibt es zahlreiche Einsatzmöglichkeiten. So kann eine Smart Card als Grundlage für die Authentifizierung eines Nutzers in VPN, WiFi oder sonstigen Kommunikationsprotokollen dienen. Im *Extensible Web Authentication Framework* werden Smart Cards als Authentifizierungstoken für Webdienste vorgesehen [3]. Abbildung 2.4 zeigt den Ablauf eines solchen Protokolls, bei dem eine Kreditkarte für Online-Bezahlungen verwendet wird. Hierbei nimmt der Webbrowser die Rolle des Vermittlers zwischen Authentifizierungs-Server und Smart Card ein. Weiterhin können Smart Cards als allgemeiner Träger für beliebige sensible Daten dienen. Beispielsweise können asymmetrische Schlüsselpaare für digitale Signaturen auf ihnen gespeichert werden, so dass der Besitzer sie stets bei sich haben kann. In einigen Ländern bereits eingeführt und in Deutschland zur Zeit dieser Ausarbeitung noch in der Testphase, sind *elektronische Gesundheitskarten (eGK)*. Anders als bisherige Krankenversicherungskarten, sind eGK Prozessor-Karten, die mehr Daten speichern, und mehr Zwecke erfüllen sollen [18]. Aufgrund ihrer kleinen Größe sind sie sehr portabel, was eines der Hauptvorteile von Smart Cards ist. Inzwischen existieren auch USB-basierte Smart Cards. Hierbei ist eine Smart Card-Architektur in ein USB-Token integriert. Der Vorteil dieser Lösung liegt in der weiten Verbreitung der USB-Schnittstelle. Die Notwendigkeit für spezielle Lesegeräte entfällt dadurch.

Kryptographische Co-Prozessoren

Bereits 1980 gab es Vorschläge, wie Software-Piraterie mit Hilfe von speziellen *Crypto-Mikroprozessoren* verhindert werden könnte [8]. Programmcode soll nur verschlüsselt abgelegt, und *on-demand* vom Krypto-Prozessor entschlüsselt werden. Es wurde relativ früh erkannt, dass sicherheitskritische Prozesse auch in nicht-militärischen Umgebungen zusätzlich geschützt werden müssen. Es gibt eine Reihe von Arbeiten, unter anderem Open Source-Varianten, die ähnliche Hardwarearchitekturen vorstellen [37; 55]. Sogenannte *kryptographische Co-Prozessoren* sind im Allgemeinen parallel zur CPU existierende Bausteine, die nur für sicherheitskritische Prozesse eingesetzt werden. Diese können als eigenständige Boards oder als integrierte Schaltkreise realisiert werden [88]. Wissend, dass die Existenz eines

2.2. Grundlagen technischer Schutzmaßnahmen

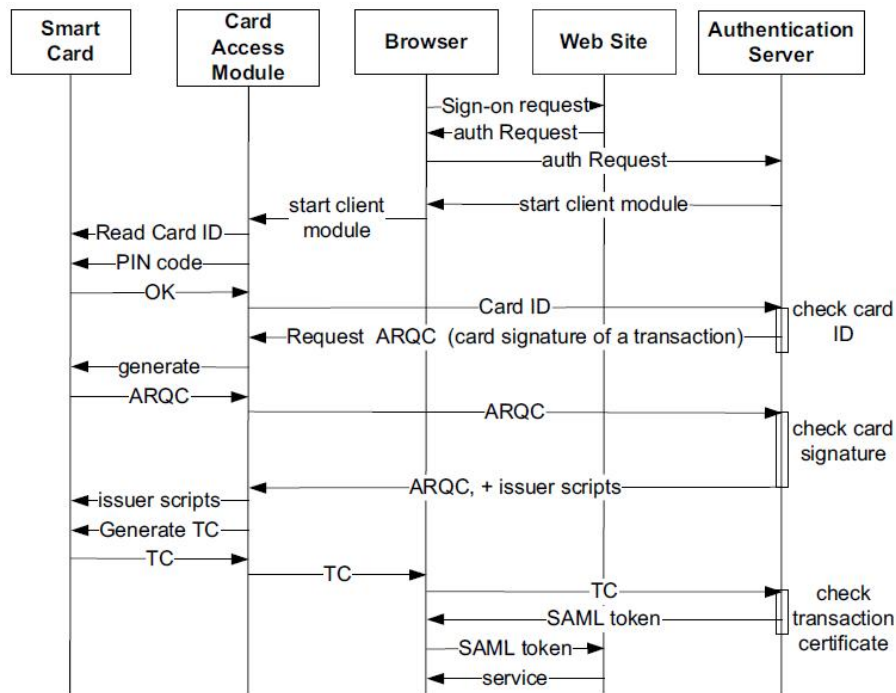


Abbildung 2.4.: Ende-zu-Ende-Authentifizierung mit Kreditkarte [3]

kryptographischen Co-Prozessors alleine nicht ausreichen kann, um die gewünschte Sicherheit zu gewährleisten, werden dabei Gesamtarchitekturen vorgestellt, die das Zusammenspiel von Software und Hardware beschreiben, um sichere Plattformen zu schaffen. Dies beinhaltet unter anderem eine sichere *Bootstrap-Architektur*, in dem der Boot-Vorgang überwacht wird, oder ein *Concealed Execution Mode*, das eine geschützte Laufzeitumgebung für Programme bietet [20; 54]. In Abbildung 2.5 ist eine von IBM vorgestellte Architektur abgebildet. Hier werden u.a. Funktionseinheiten für Verschlüsselung und Zufallszahlengenerierung vorgesehen.

Sicherheit kryptographischer Hardware

Hardwarebasierte Sicherheitsmodule bieten aufgrund ihrer Manipulationssicherheit einen erhöhten Schutz. Dieser ist auf physikalischer Ebene gegeben, so dass direkte Angriffe auf Programmroutinen und sensibles Datenmaterial nur schwer möglich sind. Jedoch gibt es eine Reihe von Angriffsmethoden für Krypto-Hardware, die gerade die physikalischen Eigenschaften ausnutzen. Dazu zählt die Klasse der *Seitenkanal-Attacken*. Anders als bei der klassischen Kryptoanalyse, wo Algorithmen

2. Webbrowser und Datenschutz

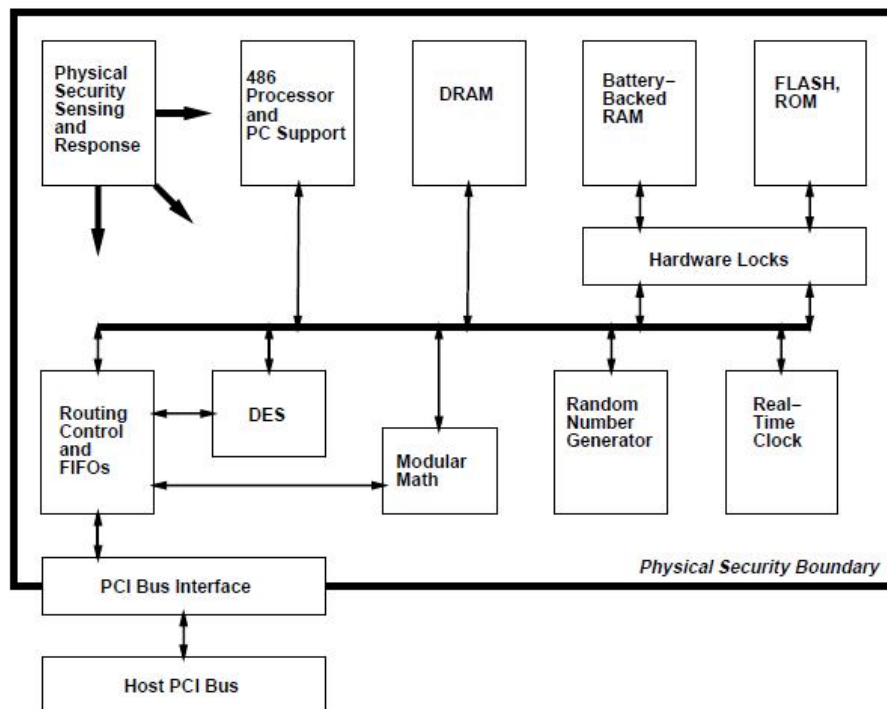


Abbildung 2.5.: Hardware-Architektur einer von IBM entwickelten sicheren Plattform [20]

men und Protokolle selbst die Angriffsobjekte sind, wird bei Seitenkanal-Attacken die konkrete Implementierung angegriffen. Dazu zählen vor allem *Timing Attacks* [48], die Analysen auf Basis der verbrauchten Rechenzeit durchführen, die *Differential Power Analysis* [47], die entsprechend mit der verbrauchten Leistung vorgeht und die *Differential Fault Analysis* [9], bei der durch absichtliches Hinzufügen von Fehlern (z.B. Veränderung der Spannung) und dem anschließenden Beobachten der Resultate bspw. versucht wird, auf die Klartexte eines Zifferntextes zu kommen. Seitenkanal-Attacken wurden erfolgreich in einer Reihe von Implementierungen, wie z.B. Diffie-Hellman, RSA, und DSS. Betroffen sind auch Smart Card-basierte Lösungen [46; 57]. Dies verdeutlicht, dass auch kryptographische Hardware prinzipiell angreifbar ist.

Nachteile

Dass kryptographische Hardware nicht weitverbreitet ist, zeigt dass es einige Hindernisse für eine breitere Akzeptanz solcher Technologien gibt. Hauptsächlich

2.2. Grundlagen technischer Schutzmaßnahmen

hängt das mit der Unflexibilität von Hardware-Implementierungen zusammen. Sie können nicht ohne weiteres verändert oder erweitert werden. Dies birgt auch Sicherheitsrisiken in sich, da sich Implementierungsfehler im Nachhinein nicht beheben lassen. Während eine Software-Bibliothek gepatcht werden kann, wird eine Hardware-Lösung evtl. durch solch einen Fehler unbrauchbar. Weiterhin spielen hohe Herstellungskosten eine Rolle. Krypto-Prozessoren, wie im letzten Abschnitt beschrieben, sind High-End-Geräte, deren Herstellung zu teuer ist, als dass sie in viele Rechner eingebaut werden könnten. Die große Hürde für die Smart Card-Technologie ist die Abhängigkeit von zusätzlicher Hardware, wie z.B. einem Lesegerät. Im privaten Bereich wird sie daher selten eingesetzt.

2.2.4. Kryptographie APIs

Produkte, die Kryptographie einsetzen, bauen häufig auf dieselben Algorithmen. Diese Algorithmen für jede Anwendung neu zu implementieren, bedeutet einen großen Aufwand. Das beansprucht viel kostbare Entwicklungszeit, die man für die Umsetzung der eigentlichen Kernkomponenten verliert. Daher gibt es eine Reihe Bibliotheken, die häufig benötigte Algorithmen von Verschlüsselung über Hashing bis hin zu Zufallszahlengenerierung, implementieren, und diese über eine API zur Verfügung stellen. Solche *Krypto-APIs* gibt es in verschiedenen Formen und mit unterschiedlichem Umfang. Manche sind als Programmbibliotheken oder Pakete in die eigenen Programme integrierbar, andere laufen als Systemdienst und sind über Systemschnittstellen ansprechbar. Krypto-Pakete können wiederum auf anderen Krypto-Paketen aufbauen, um höhere Funktionalitäten zu implementieren. Ein Argument für solche allgemeingültigen Krypto-APIs ist, dass sie über Jahre hinweg weiterentwickelt und stabilisiert werden können. Fehler in den Implementierungen können aufgedeckt und beseitigt werden. Im Gegensatz dazu ist bei Neuimplementierungen das Risiko Programmierfehler einzubauen, zu groß. Dies kann gerade bei Komponenten, die für Sicherheit sorgen sollen, gravierende Folgen haben. Im Folgenden werden einige dieser APIs vorgestellt.

Common Data Security Architecture

Ursprünglich von den *Intel Architecture Labs* entwickelt, und in der Zwischenzeit von der *Open Group* standardisiert [31], ist die *Common Data Security Archi-*

2. Webbrowser und Datenschutz

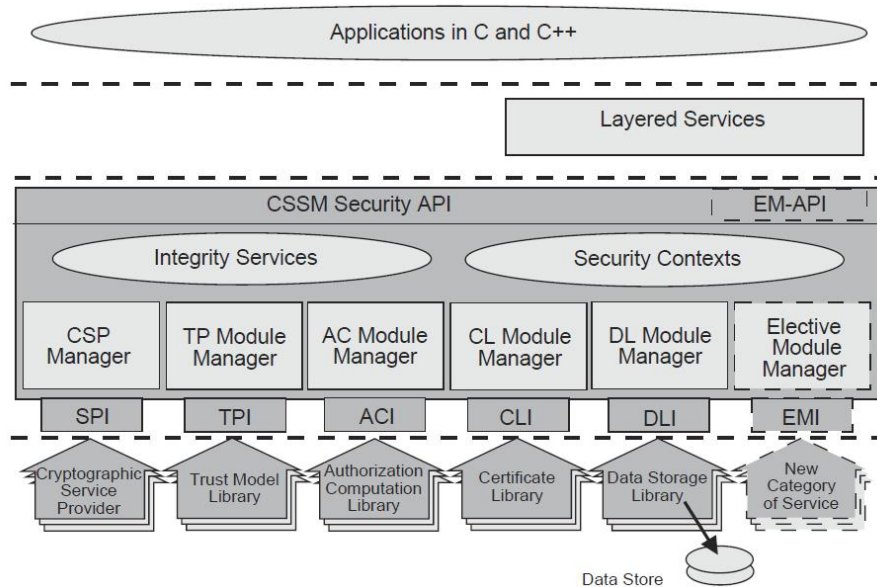


Abbildung 2.6.: Übersicht des Common Data Security Architecture-Frameworks [31]

ecture (CDSA) ein komplexes Sicherheits-Framework. Die aus mehreren Schichten bestehende Architektur von CDSA beinhaltet sowohl grundlegende kryptographische Funktionen, als auch spezifische *high-level*-Funktionalitäten, wie z.B. Schlüssel- und Zertifikatsverwaltung oder E-Mail-Dienste. In Abbildung 2.6 sind die verschiedenen Schichten und deren Beziehungen untereinander dargestellt. Ein Hauptmerkmal von CDSA ist die Modularität. Jedes der Module kann ersetzt werden, und hard- oder softwarebasiert sein. So kann bspw. ein beliebiger Krypto-Prozessor die *low-level*-Prozesse der CDSA-Architektur übernehmen. CDSA bietet somit viele Gestaltungsmöglichkeiten. Diese Flexibilität bringt jedoch eine enorm hohe Komplexität mit sich, was sicherlich einer der Gründe ist, warum CDSA nicht weitverbreitet ist. Außer einer Referenzimplementierung und der Veröffentlichung des Standards existiert kaum Informationsmaterial über CDSA, das ein konkretes Einsatzszenario beschreibt. Aus dem Informationsstand der Open Group Homepage kann geschlossen werden, dass dieser Standard nicht weiterentwickelt wird, jedoch ist dies ein sehr gutes Beispiel dafür, dass Komplexität auch ein Hindernis für sicherheitskritische Komponenten sein kann.

Microsoft CryptoAPI / CNG

Microsoft Cryptographic Application Programming Interface (CryptoAPI, CAPI) ist eine Sammlung von Diensten, die Anwendungen in Windows-Umgebungen über eine Abstraktionsschicht die Möglichkeit bietet, kryptographische Funktionen zu benutzen [62]. Realisiert werden die konkreten Funktionen, wie Verschlüsselung, digitale Signaturen oder Zufallszahlengenerierung von so genannten *Cryptographic Service Providern (CSPs)*. Als Programmbibliotheken können sie in Anwendungen integriert werden. Die seit Windows Vista existierende *Cryptography API: Next Generation (CNG)* API soll CryptoAPI langfristig ersetzen. Zu den Erneuerungen gehört u.a. die Ersetzbarkeit von Kryptographiemodulen, wie z.B. den Zufallszahlengenerator. Im Unterschied zu CryptoAPI werden kryptographische Dienste von Schlüsselverwaltungsdiensten getrennt. So genannte *key storage providers (KSPs)* können benutzt werden, um Schlüssel zu erzeugen, löschen, exportieren, importieren oder zu speichern [61]. Anwendungen werden diese Funktionalitäten über den *Key Storage Router (KSR)* verfügbar gemacht. Der KSR kümmert sich um die Interaktion mit den jeweiligen KSPs (z.B. Smart Card oder Hardwaremodul). Außerdem bietet CNG volle Unterstützung für *Elliptische Kurven Kryptographie (ECC)* ⁷.

PKCS#11

Die verbreitete Akzeptanz und der Einsatz von Kryptographie erfordert Interoperabilität, wenn auch die praktischen Anwendungen erfolgreich sein sollen. Dieses Ziel versuchen die *RSA Laboratories* mit der Veröffentlichung von Standards rund um public-key Kryptographie zu erreichen. Unter dem Namen *Public Key Cryptography Standards (PKCS)* wurden so bisher eine Reihe von Standards veröffentlicht. Einer davon ist *PKCS#11*. Dieser Standard spezifiziert eine API, auch *Cryptographic Token Interface (Cryptoki)* genannt, das eine vom Typ unabhängige logische Sicht auf kryptographische Hardware bieten soll [50]. Entwickler müssen sich somit nicht mit den herstellereigenen Schnittstellen der Hardware aus-

⁷Elliptische Kurven Kryptographie ist ein asymmetrisches Krypto-Verfahren, das auf der Schwierigkeit der Berechnung des diskreten Logarithmus in der Gruppe der Punkte der elliptischen Kurve basiert. ECC bietet bei gleicher Schlüssellänge eine höhere Sicherheit als andere asymmetrische Verfahren, wie z.B. RSA. Zudem ist die Berechnung von ECC-Verfahren ressourcenschonender, was ECC vor allem für Umgebungen mit geringen Kapazitäten, wie Smart Cards qualifiziert.

2. Webbrowser und Datenschutz

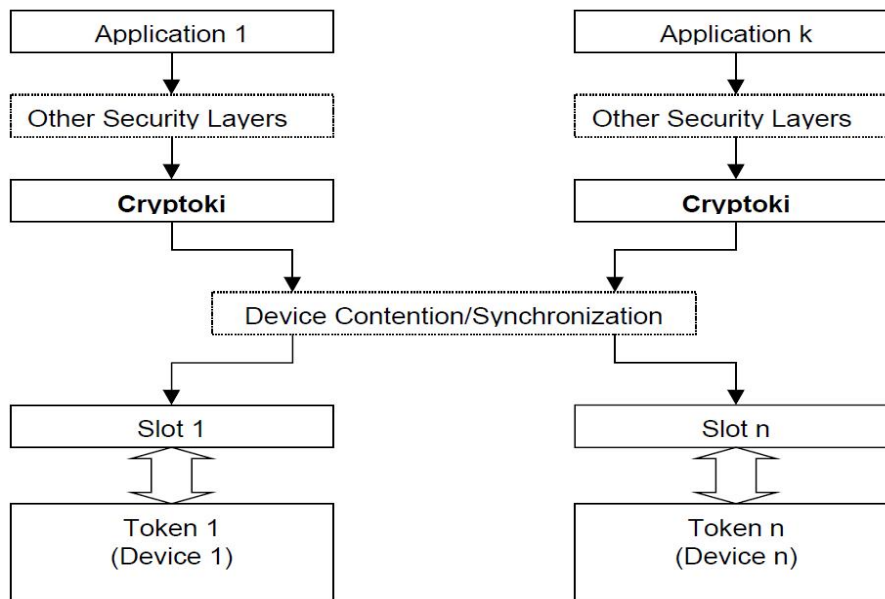


Abbildung 2.7.: Cryptoki Funktionsweise [50]

einandersetzen. Cryptoki bildet die Schnittstelle zwischen Anwendungen und beliebigen kryptographischen Geräten. Dabei wird von den Implementierungsdetails des jeweiligen Gerätes abstrahiert. Ein weiteres Ziel beim Design von Cryptoki ist, dass in Mehrbenutzer-Betriebssystemen die Geräte von mehreren Anwendungen gleichzeitig benutzt werden können (*resource sharing*). Abbildung 2.7 verdeutlicht das Arbeitsprinzip von Cryptoki. Anwendungen, die kryptographische Operationen durchführen wollen, sprechen die jeweilige Cryptoki-Schnittstelle an. Ein zentraler Synchronisationsdienst regelt konkurrierende Zugriffe auf die entsprechenden Token, wo die Operationen letztendlich ausgeführt werden. Die Geräte sind für die Anwendungen dabei nicht sichtbar. In das Cryptoki-System werden kryptographische Token über sogenannte *Slots* integriert. Diese Slots sind rein logischer Natur und sind nicht mit den physischen Schnittstellen, wie z.B. dem Lesegerät für Smart Cards gleichzusetzen. Dabei können z.B. zwei unterschiedliche Slots auf dieselbe Hardware zugreifen. Zudem können die Module auch in Software realisiert sein. Ein Token kann Objekte speichern und kryptographische Funktionen ausführen. Objekte können Daten, Zertifikate oder kryptographische Schlüssel sein. Es werden zwei Benutzertypen unterschieden. Zum einen der *Security Officer (SO)*, der administrative Operationen, wie die Initialisierung eines Token oder das Setzen von PINs durchführen kann. Normale Benutzer dürfen nach

Tabelle 2.1.: Wichtige Cryptoki-Funktionen

Name	Funktion
C_Initialize	Initialisiert Cryptoki
C_GetFunctionList	Liefert eine Übersicht über alle von der konkreten Implementierung unterstützten Cryptoki-Funktionen.
C_GetSlotList	Liefert eine Liste der verfügbaren Slots
C_InitToken	Initialisiert ein Token.
C_OpenSession	Startet eine Verbindung (Session) zwischen der aufrufenden Anwendung und einem Token
C_Login	Loggt einen Benutzer in das Token ein.
C_Encrypt	Verschlüsselt Daten
C_Digest	Erzeugt Hashwert von Daten
C_Sign	Signiert Daten
C_GenerateKey	Erzeugt symmetrischen Schlüssel, bzw. asymmetrische Schlüsselpaare
C_GenerateKeyPair	
C_WrapKey	Verschlüsselt einen Schlüssel mit einem anderen Schlüssel
C_GenerateRandom	Erzeugt zufällige Sequenzen

erfolgreicher Authentifizierung auf private Objekte auf dem Token zugreifen und auf ihnen operieren. Möchte eine Anwendung Cryptoki nutzen, muss sie zunächst die Funktion `C_Initialize` aufrufen. Danach startet eine *Session* und sämtliche Funktionen der API können genutzt werden. Tabelle 2.1 gibt einen Überblick über wichtige Cryptoki-Funktionen. PKCS#11 wird in vielen Anwendungen unterstützt und ist weitverbreitet. So wird Cryptoki bspw. in VPN-Clients eingesetzt, um Smart Card-basierte Authentifizierungsmechanismen zu realisieren. Viele Hersteller kryptographischer Hardware liefern gleich einen Cryptoki-Treiber für das Gerät mit. Mit *OpenCryptoki* existiert auch eine quasi Standardimplementierung von PKCS#11 für Linux [77]. Das Paket kommt mit einer Reihe von Treibern für IBM-Hardware und auch für das Trusted Platform Module (TPM). Im Vergleich zu CDSA ist Cryptoki relativ einfach aufgebaut. Jedoch ist eine Vergleichbarkeit nicht unbedingt gegeben. CDSA ist, wie bereits erwähnt, ein komplexes Framework, das auch high-level-Funktionalitäten definiert. Eine Kombination beider Technologien ist auch möglich, indem Cryptoki als low-level-Implementierung in CDSA als Modul eingegliedert wird [85].

2.3. Webbrowser-Sicherheit

Der Begriff *Webbrowser-Sicherheit* umfasst viele Aspekte von Schutzmechanismen. Dazu gehört eine sichere Datenübertragung, Schutz von sensiblen Daten, Phishing-Schutz, etc. Jedes dieser Aspekte erfordert entsprechende Maßnahmen. In diesem Abschnitt werden die Mechanismen der gängigsten Webbrowser im Bezug auf das Credential Management betrachtet. Dies beinhaltet sowohl die Umgebung, in der diese Daten gespeichert werden, als auch die Mechanismen, mit denen auf sie zugegriffen wird.

2.3.1. Firefox Credential Management

Unabhängig vom eingesetzten Betriebssystem, wird bei der Nutzung von Firefox für jeden Nutzer (genauer: für jedes Profil) ein Profildatensatz innerhalb Benutzer-Verzeichnisses im Betriebssystem angelegt, unter dem alle Credentials abgelegt werden. Im Folgenden werden die wichtigsten Credentials im Profildatensatz einer Standardinstallation betrachtet werden.

cert8.db. In dieser Datei werden sämtliche Zertifikate von Personen oder Zertifizierungsstellen gespeichert. Dies betrifft nicht die Root-Zertifikate, die bei einer Standardinstallation von Firefox bereits enthalten sind.

cert_override.txt. Wurde ein Server-Zertifikat, das von Firefox nicht validiert werden konnte, vom Benutzer dauerhaft akzeptiert, so wird für dieses Zertifikat eine Ausnahme erstellt. Dies geschieht in der Form eines Eintrages in die Datei *cert_override.txt*. Eine Manipulation dieser Textdatei, bspw. durch einen zusätzlichen Eintrag, hat zur Folge, dass sämtliche Verbindungen zu Server, die das entsprechende Zertifikat vorweisen, akzeptiert werden, ohne dass eine Warnung angezeigt wird. Dadurch können vertrauenswürdige Webseiten vorgetäuscht werden.

***.sqlite.** Ab Version 3 wurde bei Firefox das SQLite (Referenz)-System für die Verwaltung diverser Credentials eingeführt. Dies ist ein kleines, einfaches SQL-Datenbank-Managementsystem. Die betroffenen Credentials werden in einfachen SQL-Tabellen gehalten. Eine dieser Dateien ist die *cookies.sqlite*, worin in der

Tabelle *moz_cookies* alle Cookies gespeichert sind. Die *formhistory.sqlite* enthält alle gespeicherten Formulareingaben. *places.sqlite* enthält mehrere Tabellen. Hier sind u.a. Lesezeichen und Chronik in getrennten Tabellen abgelegt. Daten aus dem DOM Storage werden in *webappstore.sqlite* abgelegt.

key3.db. *key3.db* enthält kryptographische Schlüssel, mit denen gespeicherte Passwörter ver- und entschlüsselt werden [60].

signons3.txt. Diese Datei enthält alle Login-Credentials, die vom Firefox-internen Passwort-Manager verwaltet werden. Passwörter und Benutzernamen sind dabei mit Schlüsseln aus *key3.db* verschlüsselt.

Mozilla Security Architecture

Der Firefox-Sicherheitsarchitektur liegen die *Network Security Services (NSS)* von Mozilla zu Grunde. NSS ist eine API, wie sie in Abschnitt 2.2.4 vorgestellt wurden und soll vor allem die Entwicklung von *Cross-Platform*-Architekturen im Bezug auf Sicherheit unterstützen [59]. NSS wird in vielen Mozilla-Produkten wie Firefox oder Thunderbird eingesetzt, können aber auch in beliebige Anwendungen integriert werden. Zu den von NSS unterstützten Standards gehören unter anderem SSL/TLS, PKCS-Standards, Cryptography Message Syntax, X.509, OCSP⁸, RSA, DSA, AES, SHA und MD5.

Firefox und PKCS#11. Durch die PKCS#11-Unterstützung in NSS ist es auch in Firefox möglich, kryptographische Token einzusetzen, um sensible Daten zu speichern. Die entsprechenden Module können im Firefox-Setup geladen und aktiviert werden (Abbildung 2.8). Um Cryptoki auch einsetzen zu können, wenn keine kryptographische Hardware vorhanden ist, wird mit NSS ein Software-Modul geliefert, das als Krypto-Token eingesetzt wird. Cryptoki ist, wie in Abschnitt 2.2.4 bereits erwähnt, nicht auf Hardware-Module beschränkt. Das in NSS enthaltene Modul ist die erste Open Source Krypto-Bibliothek, die eine *Federal Information Processing Standards (FIPS)* 140-2 Validierung erhalten hat. Die FIPS werden von der *National Institute of Standards and Technology (NIST)* beschrieben, und haben hauptsächlich den Zweck eine Richtlinie für die Verwendung von sicherheitskritischen Komponenten in der Verwaltung der US-Behörden und des Militärs zu

⁸Online Certificate Status Protocol

2. Webbrowser und Datenschutz

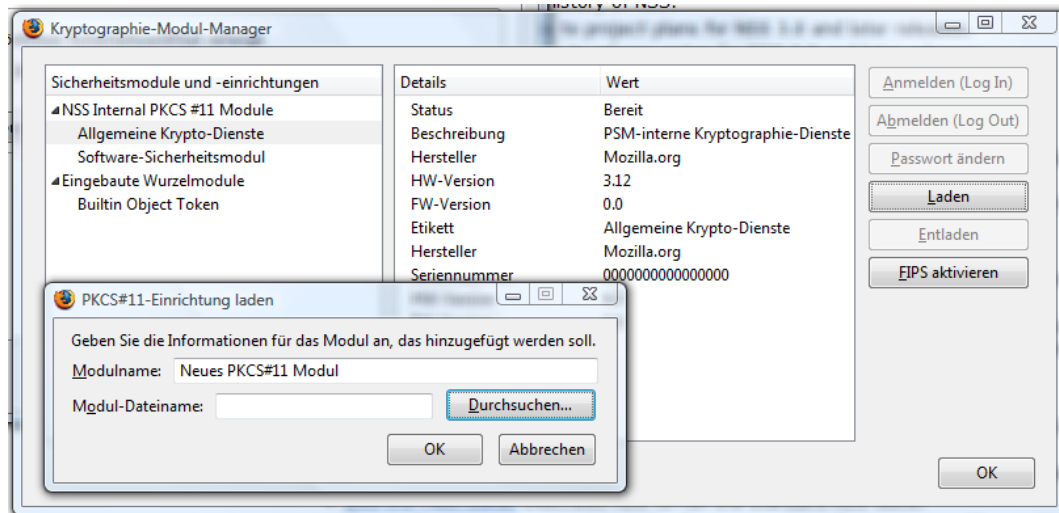


Abbildung 2.8.: Kryptographie Modul-Manager in Firefox

definieren. Das Modul für Firefox befindet sich im Hauptverzeichnis unter dem Namen *softokn3.dll* bzw. *softokn3.def* in Linux. Unter anderem kann im Modul-Manager von Firefox der *FIPS-Modus* aktiviert werden, was so viel bedeutet, dass zum Beispiel ein *Master-Passwort* zwingend gesetzt werden muss, da dies Vorgabe der FIPS 140-2 Richtlinie ist [64]. Ein Master-Passwort dient dazu, den Zugriff auf die geschützten Credentials, wie zum Beispiel Login-Informationen durch Passwort-Abfrage zu verschlüsseln. Eine Entschlüsselung der Daten folgt nur bei Eingabe des korrekten Passwortes, bspw. jedesmal, wenn Login-Daten in ein Formularfeld eingetragen werden müssen. Alternativ kann Firefox so konfiguriert werden, dass eine Eingabe nur einmal pro Sitzung, oder in regelmäßigen Zeitabständen gefordert wird.

2.3.2. Internet Explorer Credential Management

Die Schutzmaßnahmen im Internet Explorer sind sehr eng mit dem Betriebssystem verbunden. Im Allgemeinen werden vom IE Dienste in Anspruch genommen, die das Betriebssystem über eine Schnittstelle anbietet. Browserdaten wie Cookies, Favoriten, etc. werden im Profildrucker des jeweiligen Nutzers abgelegt. Für sicherheitskritische Daten wie Passwörter und Zertifikate wird die Windows Registry genutzt. In IE 7 werden Login Credentials bspw. in der Registry unter `HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2` ab-

gespeichert. Für die Verschlüsselung und den Zugriff auf die Daten existieren eine Reihe von APIs in Windows Betriebssystemen, die von den unterschiedlichen IE Versionen in Anspruch genommen werden.

Protected Storage (PStore). Die *Protected Storage API (PStore)* bietet Anwendungen über eine Schnittstelle die Möglichkeit sensible Nutzerdaten sicher abzuspeichern. Einzelne Daten werden dabei als *Item* bezeichnet. Die Struktur der zu speichernden Daten ist für PStore transparent. Es kann sich also um beliebige Daten handeln. Die Daten werden mit dem Algorithmus TripleDES verschlüsselt. Der benutzte Schlüssel ist dabei stets mit den Login Credentials für das Betriebssystem verknüpft. Das bedeutet, dass sämtliche verschlüsselten Daten, die mit PStore abgespeichert werden, entschlüsselt werden, sobald sich der Nutzer am Betriebssystem anmeldet. PStore wird von Microsoft Produkten wie IE (bis Version 6) oder auch Outlook verwendet.

Data Protection API. Die *Data Protection API (DPAPI)* ist Teil der Windows CryptoAPI und wurde mit Windows 2000 eingeführt. DPAPI soll die veraltete PStore API langfristig ersetzen. Ähnlich wie PStore bietet DPAPI Funktionalitäten, die es Anwendungen erlauben sensible Daten zu verschlüsseln. Anders als bei PStore werden die verschlüsselten Daten nicht vom Dienst verwaltet. Nach einer Verschlüsselungsoperation werden die Daten der aufrufenden Anwendung zurückgegeben. Diese Vorgehensweise ist in Abbildung 2.9 dargestellt. Die zu verschlüsselnden Daten werden mit dem Aufruf der Funktion `CryptProtectData` in Klartext an die API übergeben. DPAPI ruft daraufhin die entsprechenden Funktionen der Kryptobibliothek auf, um die Daten zu verschlüsseln. Die verschlüsselten Daten werden nach der Operation an die Anwendung übergeben. Entsprechend umgekehrt ist der Vorgang für die Entschlüsselung mit der Funktion `CryptUnprotectData`. Wie auch bei PStore ist der verwendete Schlüssel an die Login Credentials des Betriebssystems gekoppelt. Jedoch haben Anwendungen die Möglichkeit, ein zusätzliches Passwort anzugeben. Für die Verschlüsselungsoperationen erzeugt DPAPI zunächst einen *Master Key*. Dieser Master Key wird mit einem Schlüssel, der aus dem Betriebssystem-Passwort des Benutzers abgeleitet wird, verschlüsselt, und im Profilverzeichnis des Nutzers abgelegt. Die Verschlüsselung der zu schützenden Daten erfolgt dann mit symmetrischen *Session Keys*, die wiederum aus dem Master Key und Zufallsdaten hergeleitet werden. Die Session

2. Webbrowser und Datenschutz

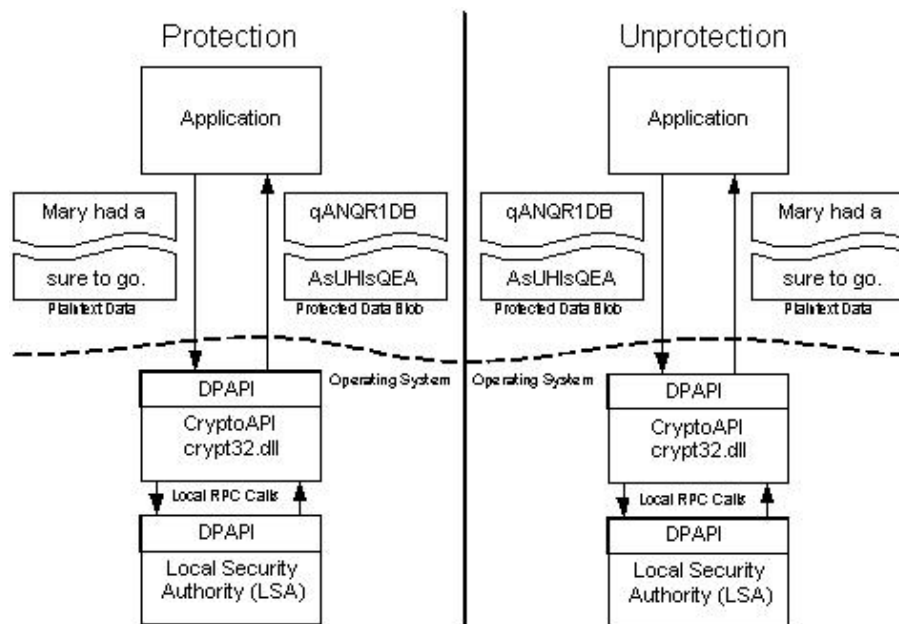


Abbildung 2.9.: Arbeitsweise von DPAPI [63]

Keys werden nicht auf der Festplatte abgelegt. Stattdessen werden die benutzten Zufallsdaten abgespeichert. Diese und der Master Key reichen aus, um denselben Session Key erneut zu erzeugen. Dies macht wiederum deutlich, dass der Schutz der Daten, die mit DPAPI geschützt werden allein auf der Fähigkeit des Betriebssystems beruht, die benutzten Schlüssel (z.B. den Master Key) angemessen zu schützen. Seit Version 7 setzt der Internet Explorer auf die DPAPI, um Login Credentials zu speichern. Dabei wird folgendes Schema angewandt, um bspw. das Passwort für eine Webseite zu speichern [22]:

1. Speichere die Adresse der Webseite und verwende sie als *Encryption Key (EK)*.
2. Berechne *Record Key (RK)*: $RK = \text{SHA}(EK)$.
3. Berechne Checksumme des RK, um Integrität zu gewährleisten. Die Integrität der eigentlichen Daten wird von DPAPI gewährleistet. $\text{RecordKeyCrc} = \text{CRC}(RK)$.
4. Verschlüsse Daten (Passwörter) mit EK:
 $\text{EncryptedData} = \text{DPAPI_Encrypt}(\text{Data}, EK)$.
5. Speichere $\text{RecordKeyCrc} + \text{RecordKey} + \text{EncryptedData}$ in der Registry

6. Zerstöre Encryption Key EK

Die Entschlüsselung erfolgt dann folgendermaßen:

1. Wenn die Webseite geöffnet ist, nehme die Adresse als Encryption Key EK und berechne Record Key RK: $RK = \text{SHA}(EK)$.
2. Gehe sämtliche Einträge in der Registry durch, um einen passenden Eintrag zu finden.
3. Falls RecordKey gefunden wird, entschlüssele Daten:
 $\text{Data} = \text{DPAPI_Decrypt}(\text{EncryptedData}, EK)$.

Die Entschlüsselung hängt also im Wesentlichen davon ab, ob die originale Adresse der Webseite bekannt ist. Weitere Browserdaten, wie Cookies oder Favoriten werden in unterschiedlichen Ordnern des Benutzerverzeichnisses abgelegt.

Google Chrome

Google Chrome ist ein relativ junger Browser. Aufgrund der Dominanz von Google in anderen Bereichen hat sich dieser Browser aber relativ schnell verbreitet. Chrome ist der erste Browser, der einen Modus für anonymes Surfen anbietet (*Inkognito Browsing*)⁹. Sicherheit und Privatsphäre waren die Gesichtspunkte, die bei der Vermarktung im Vordergrund standen. Was die technischen Schutzmaßnahmen angeht ist Chrome vergleichbar zu Firefox und IE. Lesezeichen, History, Cookies u.ä. werden wie bei Firefox in Datenbanken im SQLite-Format gespeichert. Passwörter werden unter Windows über DPAPI Funktionen (`CryptProtectData`, `CryptUnprotectData`) ver- und entschlüsselt und ebenfalls in Datenbank-Dateien abgelegt. Auch für die Speicherung von Zertifikaten werden die entsprechenden Schnittstellen des Betriebssystems genutzt.

⁹Wird dieser Modus aktiviert, werden keine Daten auf der Festplatte gespeichert, die während dem Browsen entstehen. Dies betrifft die Chronik, Formulardaten, Cookies, etc. Mit IE 8 bietet Microsoft einen ähnlichen Modus (InPrivate-Browsing) Auch für Firefox 3.5 ist dieser Modus geplant (Private Browsing).

2.3.3. Analyse des Credential Managements

Ein erster Schutzmechanismus für die Credentials ist die Betriebssystem-Zugriffskontrolle, da außer dem Besitzer selbst kein anderer Nutzer Zugriff auf die Daten hat. Dieser Schutz wirkt jedoch nur auf Betriebssystem-Ebene. Hat jemand Zugriff auf den Rechner, kann er diesen Schutz ohne Probleme aushebeln, bspw. durch Mounten der Windows-Partition unter Linux. Dieser Mechanismus wirkt daher höchstens gegen Gelegenheitsangriffe und bietet keinen hohen Schutz. Aus diesem Grund ist es wichtig, dass die Daten zusätzlich gesichert werden. So werden bspw. die Login-Credentials zusätzlich durch Verschlüsselung geschützt. Credentials, wie Cookies, Lesezeichen, Chronik u.ä. sind jedoch unverschlüsselt abgelegt. Unter Privacy-Aspekten wäre es jedoch empfehlenswert, auch diese Daten durch Verschlüsselung zu sichern. Im Unterschied zu Firefox werden beim Internet Explorer kryptographische Schlüssel oder Passwörter nicht in Dateien, sondern in der Windows-Registry abgelegt. Im Endeffekt sind die Teile der Registry in einzelnen Dateien gespeichert. So sind die benutzerspezifischen Daten, genauso wie auch bei Firefox, im Benutzerprofilverzeichnis abgelegt.

Risiken und Schwachstellen

Der Einsatz eines Softwaremoduls als kryptographische Einheit resultierte aus den bereits genannten Gründen. Hardware-Krypto-Module einzusetzen erfordert zusätzlichen Aufwand oder Kosten. So sehr das Softwaremodul die Richtlinien und Standards einhält, kann es nicht den gleichen Grad an Sicherheit bieten, wie echte Hardwaremodule. Die Besonderheit an diesen ist gerade ihre Manipulationsicherheit, d.h. die Schwierigkeit und der Aufwand, der damit verbunden ist, unautorisiert an die geschützten Daten zu kommen. Module, die als Software laufen, besitzen keine eigenen Speicherbereiche, auf denen sie arbeiten können. Sämtliche sensiblen Daten befinden sich zur Laufzeit im Arbeitsspeicher. Weiterhin zeigen in Browsern integrierte Passwort-Managementsysteme große Schwächen. In [22] wird beschrieben, wie einfach es ist den Schutz, trotz Benutzung eines Master-Passworts zu umgehen, und an alle gespeicherten Passwörter zu kommen. Es existieren frei verfügbare Tools, die innerhalb wenigen Sekunden sämtliche Passwörter, inklusive dem Master-Passwort entschlüsseln [84]. Diese Angriffe zeigen die Grenzen der gängigen Schutzmechanismen klar auf. Einen besonders hohen Schutz können die-

se Mechanismen nicht bieten, weshalb teilweise von der Nutzung der integrierten Passwort-Managern abgeraten wird.

2.3.4. Third-Party Ansätze

Es gibt zahlreiche Ansätze, die versuchen, die genannten Schwachstellen von Standard-Lösungen zu beseitigen. Diese Ansätze erfolgen meistens in Form einer Browser-Erweiterung. Viele der Lösungen betreffen die Passwortverwaltung. Die Schwierigkeit hierbei liegt in der Schaffung einer ausgewogenen Balance zwischen Sicherheit und Benutzerfreundlichkeit. Zum einen müssen bestimmte Sicherheitsrichtlinien, wie z.B. lange und unterschiedliche Passwörter für unterschiedliche Portale eingehalten werden. Zum anderen kann nicht erwartet werden, dass der Nutzer die ganze Verwaltung dieses *Passwort-Dschungels* übernimmt. Eine der größten Gefahren im Web ist das *Phishing*, bei dem ein Angreifer versucht dem Nutzer eine vertraute Webseite vorzutäuschen, um an geheime Informationen von Nutzer zu kommen. Viele der Lösungen für das Passwortmanagement-Problem sind vor allem durch Phishing-Angriffe motiviert. In [93] beschreiben Wu und Kollegen eine Methode gegen Phishing, bei der eine zusätzliche Sidebar im Browser angezeigt wird. In dieser Sidebar wird für eine Webseite, die ein Login-Formular enthält dieses Formular ersetzt. Das Formular auf der Website wird gesperrt, so dass Eingaben nur über die Sidebar gemacht werden können. Vor Senden der Informationen findet eine Überprüfung der Ziel-URL statt. Phishing-Attacken sollen verhindert werden, indem ermittelt wird, wohin der Nutzer die Daten schicken möchte. Daraufhin bietet die Extension eine Auswahl an zuvor gespeicherten URLs, die dem gewünschten Ziel entsprechen. Die Schwäche dieses *Web Wallet* genannten Systems, ist wie auch im Paper erklärt, dass es selbst durch Phishing angreifbar ist. Die Sidebar kann durch eine Kopie imitiert werden, woraufhin sie der Nutzer nicht durch das Original unterscheiden kann. Eine weitere Arbeit [39] geht auf das Problem der Passwort-Vielfalt ein. Die Lösung besteht darin, mit nur einem einzigen Passwort und einer Hashfunktion unterschiedliche Passwörter für verschiedene Websites zu erzeugen. Das Passwort wird jeweils mit der Ziel-URL gehasht, so dass im Endeffekt für jede Website ein unterschiedliches Passwort entsteht. Der Benutzer muss sich dabei nur ein einziges merken. Ist dieses eine Passwort und die Hashfunktion jedoch bekannt, so können sämtliche Passwörter offen-

2. Webbrowser und Datenschutz

bart, und der Schutz umgangen werden. Es gibt auch zahlreiche Erweiterungen, die erweiterte Cookie-, History- oder Cache-Management Funktionalitäten bieten (BetterPrivacy, Cache-Status, Clear-Private-Data, Cookie Button, Cookie Monster, etc.). Bei den meisten solcher *Privacy Enhancing Technologies (PETs)* geht es lediglich um den Aspekt der Benutzerfreundlichkeit. So ist zwar das Löschen sämtlicher Cookies eine Lösung für die in Abschnitt 2.1.3 genannten Risiken, jedoch werden auch sämtliche Vorteile von Cookies eliminiert. Eine Erweiterung, die erhöhten Schutz für Authentifizierungsdaten bietet, ist z.B. *Fingerfox*, welches eine Authentifizierung über einen Fingerabdruck-Sensor erlaubt. Login-Credentials werden nur bei erfolgreicher Authentifizierung freigeschaltet.

Wallet-basierte Lösungen

Eine weitere Klasse von Credential-Management-Lösungen sind *Wallet-basierte* Systeme. Charakteristisch für diese Ansätze ist, dass es sich um zusätzliche Software handelt, entweder als Browser-Erweiterung oder als eigenständiges Programm, die in gewisser Weise einen zentralen Speicher und Schutz für Credentials bieten, indem das Credential-Management vom Browser-Kontext gelöst wird. *CardSpace* von Microsoft zielt darauf ab, die Verwendung von Passwörtern abzulösen. Der Benutzer kann sich mehrere Identitätskarten anlegen, die er Webseiten zur Authentifizierung senden kann. Es wird hierbei versucht Benutzerfreundlichkeit zu erhöhen, indem man die Identitätskarten-Metapher umsetzt. Diese Lösung benötigt jedoch auch serverseitige Anpassungen, was einer der Gründe für die geringe Verbreitung ist. Die Karten, ähnlich wie Browser-Credentials verschlüsselt auf der Festplatte abgelegt. Eine sehr interessante Lösung ist *KWallet* [79]. Dieses KDE-Linux Programm dient als zentrale "Brieftasche" für User Credentials. KDE-basierte Anwendungen, wie z.B. der Webbrowser *Konqueror* können die Funktionalitäten von *KWallet* über eine API nutzen. So können Browser-Credentials, wie Passwörter, Cookies oder Formulardaten in einer digitalen Brieftasche gespeichert werden. Ein sehr ähnliches Konzept besitzt *GnomeKeyring*[53] für GNOME-basierte Linux-Systeme. Unter anderem unterstützt *GnomeKeyring* PKCS#11. Weiterhin gibt es eine Reihe von Credential-, bzw. Passwort-Management-Programme, die es erlauben sämtliche Passwörter zentral zu verwalten. Dazu gehören u.a. *Password Safe*[68] und *KeePass*[45].

2.3. Webbrowser-Sicherheit

Alle hier vorgestellten Lösungen versuchen die Probleme der integrierten Managementsysteme zu beheben. Der Vorteil dieser Lösungen liegt darin, dass sie speziell für den Schutz der Credentials entwickelt wurden. Somit sind diese Anwendungen als “Spezialisten” für Credential-Management zu betrachten, da alle Lösungen starke Kryptographie einzusetzen, um höchstmöglichen Schutz zu bieten. Wie bereits im Zusammenhang mit softwarebasiertem Cryptoki erläutert, können auch diese softwarebasierten Anwendungen nur Schutz bieten, wenn sie so funktionieren, wie es für sie vorgesehen ist. Eine Kompromittierung der Software könnte die Offenlegung der Geheimnisse mit sich bringen.

2. *Webbrowser und Datenschutz*

3. Trusted Computing

Das Bestreben nach Sicherheit in der Informationstechnologie ist ständig neuen Herausforderungen gestellt. Traditionsgemäß wurden per se unsichere Systeme erst nachträglich um Sicherheitsmaßnahmen erweitert. Die Zahl der kritischen Sicherheitslücken zeigen deutlich, dass diese Vorgehensweise nicht in jedem Szenario für die Zukunft geeignet ist. Es gab bereits einige Ansätze, die versuchten Systemarchitekturen grundlegend zu ändern, um Sicherheitskomponenten zu einem festen Bestandteil von Rechnersystemen zu machen. Einer dieser Ansätze wurde in Abschnitt 2.2.3 vorgestellt. Die vorgestellten Lösungen waren mehr oder weniger erfolgreich, sinnvoll, anerkannt oder realisierbar. Hürden waren oft zu hohe Kosten oder ein zu großer Aufwand für die Anpassungen. Ein aktueller Ansatz, der gleichermaßen vielversprechend ist und rege diskutiert wird, ist die *Trusted Computing-Initiative*. Von der *Trusted Computing Group (TCG)* erarbeitet, beschreiben diverse Spezifikationen, wie IT-Systeme zukünftig mit Unterstützung von Hardware-Komponenten sicherer gemacht werden können. Dieses Kapitel bietet eine Einführung in diese Technologie, wobei die wichtigsten Spezifikationen vorgestellt werden. Anschließend werden mögliche Szenarien vorgestellt, woraufhin eine Diskussion über Trusted Computing folgt.

3.1. Grundlagen

Jede neue Technologie muss sich zunächst am Markt durchsetzen. Dies gilt auch für Security-Lösungen. Wichtige Voraussetzungen hierfür sind neben einem lückenlosen Design auch die Förderung und Unterstützung durch wichtige Institutionen und Unternehmen. In diesem Abschnitt werden kurz die Akteure und Förderer der Trusted Computing-Initiative vorgestellt. Dabei wird auch auf die grundlegenden Konzepte von TC eingegangen.

3. Trusted Computing

3.1.1. Trusted Computing Group (TCG)

Die Geschichte der Trusted Computing Group beginnt im Jahr 2003 mit der Auflösung der *Trusted Computing Platform Alliance (TCPA)*, die im 1999 gegründet wurde. Die TCPA hatte sich zum Ziel gesetzt, einen Industriestandard für vertrauenswürdige Computersysteme zu entwickeln. Gründe für die Auflösung waren zum einen die komplizierte Organisationsstruktur des Konsortiums, die eine produktive Arbeit verhinderte, und zum anderen wesentliche Kritiken, die an den bis dahin vorgestellten Ideen ausgeübt wurden. So entstand relativ früh eine starke Gegenbewegung, was u.a. damit zu begründen ist, dass das Konsortium hauptsächlich aus Vertretern der Industrie bestand (u.a. Microsoft, IBM, Hewlett-Packard, Compaq). Befürchtet wurde, dass die Gestaltung von Computersystemen in Zukunft sich nach den Wünschen der Unternehmen richten würde. Vor allem wurden TCPA-Konzepte nur als Motor für die Umsetzung von *Digital Rights Management* gesehen. Im Jahr 2003 organisierten sich einige der Gründer der TCPA neu unter dem Namen Trusted Computing Group (TCG). Die Ziele und bisherigen Arbeiten wurden im Prinzip von der TCPA übernommen. Seit der Gründung wurden durch die in mehrere Arbeitsgruppen unterteilte TCG, diverse Spezifikationen erarbeitet. Ziel dabei ist die Veröffentlichung von Standards für Architekturen, Funktionen und Schnittstellen, die als Referenzen für eine Implementierung von *Trusted Computing Plattformen* dienen sollen [34]. Die Standards sollen die unterschiedlichsten Computerplattformen von mobilen Endgeräten, bis hin zu eingebetteten Systemen berücksichtigen und von den jeweiligen Arbeitsgruppen erarbeitet werden. Da der Begriff *Trust* im Zusammenhang mit Computersystemen verwirrend sein kann, sei im weiteren Verlauf dieser Arbeit von der folgenden Definition auszugehen.

“Trust is the expectation that a device will behave in a particular manner for a specific purpose.”

Sinngemäß übersetzt bedeutet dies, dass von einer Komponente erwartet wird, dass sie sich so verhält, wie es für sie vorgesehen ist. Das Schlüsselwort ist hierbei der Begriff *particular* (deutsch: *bestimmt*). Ein “vertrauenswürdiges” System befindet sich demnach zu jedem Zeitpunkt in einem definierten Zustand, bzw. ein unerwünschter Zustand wird erkannt und verhindert.

3.1.2. Trusted Computing Konzepte

Die allgemeinen Konzepte, die von der TCG vorgestellt werden, können mit den in Kapitel 2 vorgestellten Architekturen verglichen werden. Auch hier soll die Sicherheit von Computersystemen aufbauend auf einem Hardwaremodul erhöht werden. Genauso wie in der vorgestellten Architektur von IBM wird eine Gesamtarchitektur, bestehend aus Hard- und Softwarekomponenten beschrieben, die dies ermöglichen soll. Im Unterschied zu bisherigen Konzepten steht jedoch der Faktor Interoperabilität im Vordergrund. Es geht bei Trusted Computing nicht nur darum ein einzelnes Computersystem zu schützen, sondern eine Infrastruktur zu schaffen, in der Systeme untereinander sicher kommunizieren und sich gegenseitig ihre Vertrauenswürdigkeit beweisen können. In der stark vernetzten Computerlandschaft von heute ist das durchaus ein sinnvoller Ansatz. Die TCG definiert grundlegende Eigenschaften, die eine Trusted Plattform besitzen muss. *Protected Capabilities* sind ein Satz an Kommandos, die als einziger das Recht haben auf bestimmte geschützte Bereiche zuzugreifen. Diese sogenannten *Shielded Locations* sind geschützte Umgebungen, in denen die Operation auf kritischen Daten sicher ist. *Integrity Measurement* ist dafür zuständig, Systemzustände zu messen und die Messwerte in bestimmten Registern abzulegen. Diese Messwerte repräsentieren die Integrität und somit die Vertrauenswürdigkeit eines Systems. Das *Integrity Reporting* ist die vertrauenswürdige Berichterstattung über die gemessenen Werte. Sinn hinter dem Integrity Measurement und Reporting ist, dass ein System zwar jeden Zustand einnehmen darf, jedoch keine Falschaussagen darüber machen kann, in welchen Zuständen es sich befand oder befindet. Weitere Mechanismen haben dann die Möglichkeit ggf. Maßnahmen zu ergreifen, falls ein unerwünschter Systemzustand entdeckt wird. Aufbauend auf diesen Eigenschaften sollen Mechanismen implementiert werden können, die es u.U. ungleichen Computersystemen erlauben ihre Vertrauenswürdigkeit anderen Plattformen zu attestieren, eine vertrauenswürdige Verbindung von Kommunikationsendpunkten herzustellen, sicherheitskritische Komponenten zu überwachen oder einen sicheren Speicher für sicherheitskritische Daten zur Verfügung zu stellen. Diese Konzepte und deren Realisierung werden in den folgenden Abschnitten näher beschrieben.

3.2. **Trusted Computing System**

Grundlage für die Realisierung von Trusted Computing ist ein *Trusted Computing System (TCS)*. Dieses ist im Wesentlichen ein um bestimmte Hard- und Softwarekomponenten erweitertes Rechnersystem. Bisherige Architekturen werden nicht von Grund auf verändert, sondern die vorhandene Basis lediglich um ein zusätzliches Subsystem erweitert. Dieser sogenannte *Trusted Building Block (TBB)* enthält dabei die hardwarebasierten Vertrauensanker, die die Basis eines TCS bilden. Die Verankerung dieser Komponenten in Hardware ist von hoher Wichtigkeit, da sämtliche weiteren Mechanismen eines TC von deren Integrität abhängig ist. Der TBB umfasst zum einen das *Trusted Platform Module (TPM)*, ein Hardwarebaustein, der grundlegende kryptographische Funktionalitäten implementiert und einen geschützten Speicher zur Verfügung stellt. Die zweite Komponente des TBB ist der *Core Root of Trust for Measurement (CRTM)*, welches als eine Erweiterung des BIOS wichtige Funktionalitäten bereitstellt, um TC-Mechanismen bereits zu einem frühen Stadium des Bootvorgangs wirken zu lassen. Dabei macht die CRTM von den Funktionalitäten des TPM gebrauch. Insgesamt entsteht eine *Trusted Computing Platform (TCP)*. Die TCP bildet somit die Basis eines *Trusted Computing Systems*, das noch aus weiteren, auf dem TBB aufbauenden Schichten besteht. Dazu gehört bspw. ein *Trusted Operating System (TOS)*, das TC-Unterstützung auf Anwendungsebene bringen, und die vertrauenswürdige Ausführung von Anwendungen erlauben soll. Die Abhängigkeit einer Schicht von der Vertrauenswürdigkeit der unteren Schichten verdeutlicht einen wichtigen Mechanismus von Trusted Computing, den *Transitive Trust*. Das Vertrauen, das man auf unterster Ebene in die Hardwaremodule hat, wird sukzessive an die höheren Schichten weitergegeben. Dadurch wächst die *Trust Boundary*. In den folgenden Abschnitten werden die Komponenten des TBB näher vorgestellt. Im Anschluss werden unterschiedliche Konzepte und Szenarien vorgestellt, die von den Eigenschaften von Trusted Computing Plattformen gebrauch machen, um bestimmte Sicherheitsziele zu erreichen.

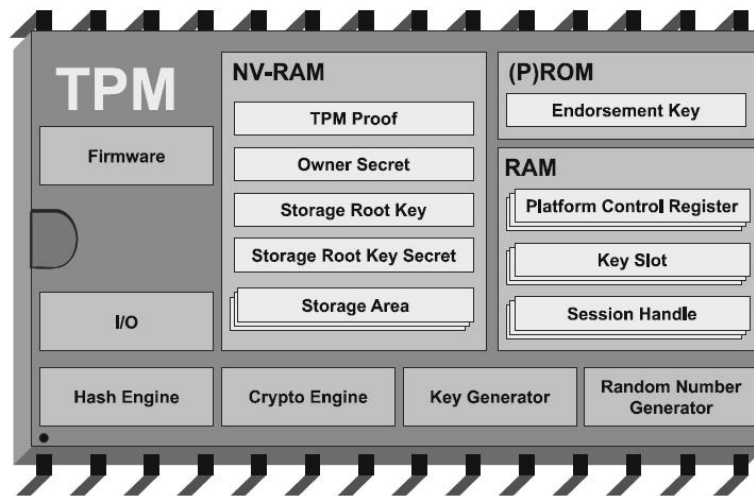


Abbildung 3.1.: Innerer Aufbau eines Trusted Platform Moduls [82]

3.2.1. Trusted Platform Module

Das Trusted Platform Module (TPM) ist die Realisierung der Protected Capabilities und Shielded Locations. Der TCG-Spezifikation nach kann ein TPM sowohl in Soft- als auch in Hardware umgesetzt werden, da die Spezifikation diesbezüglich allgemein gehalten worden ist. Sinnvoll ist jedoch nur eine Umsetzung als fest mit der Plattform verbundener Hardwarebaustein, um auch die geforderte Manipulationssicherheit zu gewährleisten. TPMs besitzen eine zu Smart Cards sehr ähnliche Mikroprozessorarchitektur, enthalten also u.a. logische Einheiten, Speicherbereiche und einen Ein-/Ausgabekanal. Sie implementieren kryptographische Funktionalitäten, die für die Realisierung von TCS erforderlich sind und dient als sicherer Speicher für sensible Daten wie kryptographische Schlüssel. Die zu implementierenden Funktionen sind in der TPM-Spezifikation beschrieben. Die wichtigsten Komponenten werden im Folgenden vorgestellt.

RSA Engine. Die Komponente implementiert RSA-Algorithmen für Schlüsselerzeugung oder die Erstellung von digitalen Signaturen. Diese Komponente muss in jeder TPM-Implementierung enthalten sein, wobei die Unterstützung der Schlüssellängen 512, 1024 und 2048 Bit Pflicht sind. Empfohlen wird die Nutzung von mindestens 2048 Bit. Die RSA-Algorithmen können auch extern über eine Software API genutzt werden. Die RSA-Engine ist die einzige Komponente für asymme-

3. *Trusted Computing*

trische Kryptographie, die implementiert werden muss. Es steht den Herstellern jedoch offen, weitere Module für asymmetrische Kryptographie einzubauen.

SHA-1 Engine. Die SHA-1 Komponente ist die vertrauenswürdige Implementierung eines Hash-Algorithmus. Die SHA-1 Komponente ist verpflichtend und muss FIPS-180-1 genügen. Hauptsächlich wird sie von der HMAC-Komponente verwendet, um Messwerte von kritischen Komponenten zu erstellen. Jedoch kann sie auch von weiteren Anwendungen in Anspruch genommen werden, indem die entsprechenden TPM-Kommandos aufgerufen werden. Da ein TPM nicht als Krypto-Beschleuniger zu sehen ist, werden keine Anforderungen an die Leistungsfähigkeit der SHA-1-Implementierung gestellt.

Zufallszahlen-Generator. Wie in Kapitel 2 beschrieben, ist die Generierung von Zufallszahlen für kryptographische Algorithmen eine besonders kritische Aufgabe. So schreibt die TCG auch die Existenz eines RNG für das TPM vor. Dieser RNG besteht aus einem Zustandsautomaten, der unvoraussagbare Daten akzeptiert, um sie dann von einem *Post-Prozessor* verarbeiten zu lassen. Die Daten hierfür kommen von einem *Entropy Collector* und können bspw. durch thermisches Rauschen beobachtet werden. Bevor sie an den Zustandsautomaten geliefert werden, müssen evtl. noch Verzerrungen aus den Daten entfernt werden. Auf Anfrage muss der RNG mindestens 32 Byte an zufälligen Daten liefern können. Bei der Implementierung muss darauf geachtet werden, dass der jeweilige neue Zustand des Automaten von außen nicht sichtbar ist. Werden die erzeugten Daten für TPM-interne Zwecke benötigt, so dürfen die Daten die geschützte Umgebung des TPM verlassen. Dies gilt nicht für Aufrufe über eine Software-API.

Platform Configuration Register. Ein *Platform Configuration Register (PCR)* ist ein 160 Bit-breiter flüchtiger Speicherbereich, der zur Speicherung von Werten aus Integritätsmessungen benutzt wird. Die Spezifikation schreibt vor, dass mindestens 16 dieser PCR existieren müssen, wobei diese unbedingt im geschützten Bereich des TPM liegen müssen.

Endorsement Key. Der *Endorsement Key (EK)* ist ein 2048 bit RSA- Schlüssel-paar, das sich im nicht-flüchtigen Speicherbereich des TPM befindet. Der EK kann benutzt werden, um vom TPM ausgehende Nachrichten und Daten zu signieren, um bspw. einer anderen Plattform seine Vertrauenswürdigkeit zu beweisen. So

können Inhalte von bestimmten PCR-Registern als Repräsentation des Systemzustands signiert und zur Überprüfung gesendet werden. Somit dient der EK als Identifizierungsmerkmal der Plattform und sollte für jedes TPM einzigartig sein. Der EK darf ein TPM niemals verlassen, da eine Verwendung nur im Zusammenhang mit dem einen TPM in Frage kommt. Generiert wird er in der Regel vom Hersteller, bevor es an den Endkunden geht. Dabei wird auch gleich ein *Endorsement Key Credential* erstellt, und vom Hersteller signiert. Das EK Credential enthält den Namen des Herstellers, die Modellbezeichnung des TPM und den öffentlichen Teil des EK. Es besteht jedoch die Möglichkeit, den EK zu erneuern. Hierzu wird das Kommando `TPM_CreateEndorsementKey` bereitgestellt. Für den nachträglich erzeugten EK kann das TPM jedoch kein signiertes EK Credential vorweisen.

Storage Root Key. Der *Storage Root Key (SRK)* ist neben dem EK das zweite 2048 bit RSA-Schlüsselpaar, das im nicht-flüchtigen Speicher des TPM liegt. Er wird erzeugt, sobald das TPM von einem Nutzer aktiviert wird (*TakeOwnership*). Der SRK dient als Wurzel für eine Schlüsselhierarchie, in der weitere Schlüssel verwaltet werden können.

RSA Key Slots. Die *RSA Key Slots* sind flüchtige Speicherbereiche im TPM, die als Zwischenspeicher für RSA-Schlüssel dienen. Die Schlüssel werden in RSA-Slots geladen, sobald sie verwendet werden sollen. Die Existenz solcher Slots ist dadurch bedingt, dass vom TPM erzeugte Schlüssel das Modul nicht ungeschützt verlassen dürfen.

Betriebszustände und Betriebsmodi

Startend aus einem *Power-Off*-Zustand, wird mit dem `TPM_Init`-Signal die Initialisierung des TPMs angestoßen. Nach der Initialisierung führt das TPM einige Selbsttests durch, um zu überprüfen, ob die Funktionalitäten korrekt arbeiten. Nach Abschluss der Tests gelangt das TPM in einen Betriebszustand, der von drei weiteren Zustandsvariablen abhängt:

- *Enabled/Disabled.* Ist das TPM im Zustand *disabled*, so können keine Kommandos ausgeführt werden, die die Ressourcen des TPM benutzen. Funktionen, wie SHA-1 können weiterhin genutzt werden, jedoch können bspw.

3. Trusted Computing

keine Schlüssel geladen oder die Sealing-Funktion in Anspruch genommen werden.

- *Activated/Deactivated.* Der Zustand deactivated ist identisch zu disabled, bis auf die Tatsache, dass im Zustand deactivated das `TPM_TakeOwnership` ausgeführt werden kann.
- *Owned/Un-owned.* Das TPM ist im Zustand owned, wenn ein EK existiert und der Besitzer das Passwort kennt. Der Besitzer der Plattform kann alle Operationen des TPM ausführen.

TPM-Kommunikationsprotokolle

TPM-Kommandos können grob in zwei Kategorien eingeteilt werden. Kommandos, die in irgendeiner Weise Sicherheit, Privatsphäre oder Geheimnisse betreffen und Kommandos, die eher informativer Natur sind. Für erstere gibt die TCG vor, dass eine in Anspruchname eine Authentifizierung und Authorisierung der aufrufenden Instanz (bspw. die Software API) erfordert. Dies erfolgt mit dem Vorweisen eines 160 Bit-Geheimnisses. Daraufhin wird ein sicherer Kommunikationskanal zwischen dem TPM und der aufrufenden Instanz erzeugt, über den die Kommandos übertragen werden können. Die TCG definiert für diese Zwecke fünf verschiedene Protokolle. *Authorization Data Insertion Protocol (ADIP)*, *Authorization Data Change Protocol (ADCP)* und *Asymmetric Authorization Change Protocol (AACP)* sind Protokolle zum Erstellen und Verwalten von Authentifizierungsdaten für TPM-Objekte (Schlüssel, BLOBs, etc.). Die folgenden beiden Protokolle ermöglichen den Aufbau von Sitzungen, die für den Aufruf von mehreren Kommandos benutzt werden können.

- *Object-Independent Authorization Protocol (OIAP)* bildet einen von TPM-Objekten unabhängigen Kanal. Dieser kann somit für mehrere Kommandos genutzt werden, die unterschiedliche Objekte betreffen. Gestartet wird eine OIAP-Sitzung mit dem Aufruf `TPM_OIAP()`.
- *Object-Specific Authorization Protocol (OSAP)* baut einen Kanal auf, der zwar für mehrere Kommandos, jedoch nur für ein bestimmtes TPM-Objekt genutzt werden kann. Gestartet wird eine OSAP-Sitzung mit dem Aufruf `TPM_OSAP()`.

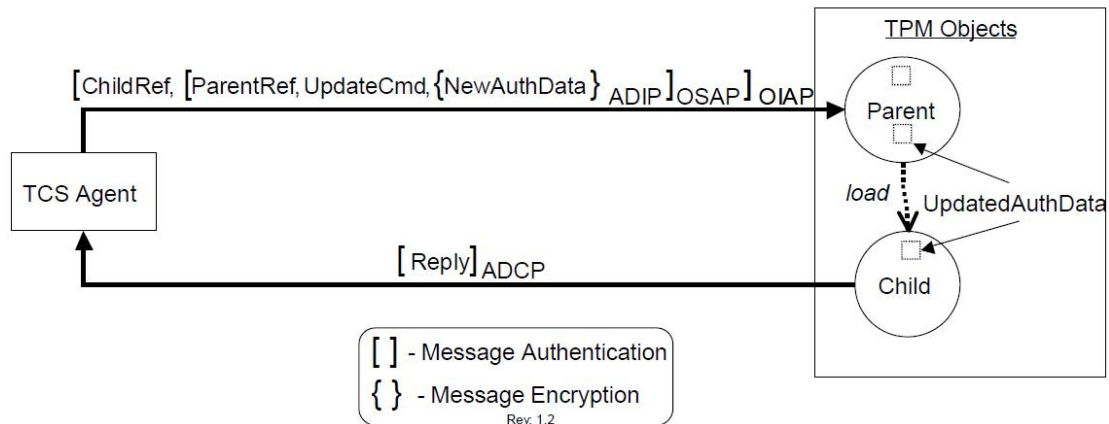


Abbildung 3.2.: Einsatz der Kommunikationsprotokolle bei Änderung von Authentifizierungsdaten eines TPM-Objektes [34]

Abbildung 3.2 verdeutlicht das Zusammenspiel der Protokolle, wenn das Passwort für ein TPM-Objekt (z.B. den SRK) geändert werden soll. Hierzu wird ADCP eingesetzt, welches wiederum ADIP nutzt, um Vertraulichkeit zu gewährleisten. OSAP und OIAP werden eingesetzt, um den Zugriff auf die entsprechenden Objekte (SRK und EK als Elternteil) zu autorisieren.

3.2.2. Die Vertrauensanker

Das Vertrauen, das man in ein Trusted Computing System setzt, basiert auf den bereits vorgestellten Komponenten des Trusted Building Blocks. Ausgehend von diesen, definiert die TCG drei sogenannter *Roots of Trust* (*Vertrauensanker*), die jeweils als *Quelle des Vertrauens* für eine bestimmte Funktionalität dienen sollen. Eine Fehlfunktion oder Manipulation dieser Komponenten muss ausgeschlossen werden, da sämtliche weitere Mechanismen, die *Vertrauen* in ein System bringen sollen, von der ordnungsmäßigen Arbeitsweise dieser Komponenten abhängen.

Root of Trust for Measurement. Die *Root of Trust for Measurement (RTM)* ist eines dieser Vertrauensanker. Es ist dafür zuständig, Prüfsummen von wichtigen Systemkomponenten zu erstellen und in den PCR abzulegen. Die in den PCR enthaltenen Werte können dann mit Referenzwerten, die ein integriertes System aufweisen sollte, verglichen werden, um unerwünschte Zustände zu erkennen. Die Messungen müssen dabei bereits zu einem sehr frühen Stadium beginnen.

3. Trusted Computing

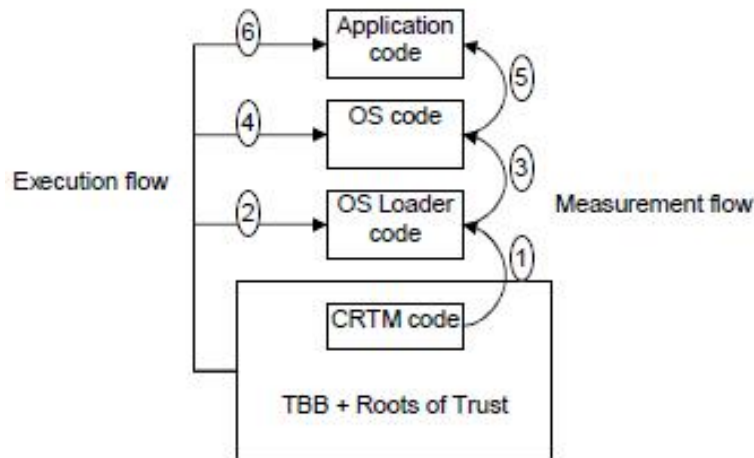


Abbildung 3.3.: Das Prinzip des transitive Trust [34]

Umgesetzt wird die RTM vom *Core Root of Trust for Measurement (CRTM)*, ein Programm, das bspw. als Erweiterung des BIOS realisiert werden kann. Abbildung 3.3 verdeutlicht das Prinzip des transitive Trust für ein *Trusted Booting*. Das CRTM sorgt dafür, dass der Code, der das Betriebssystem lädt (OS Loader) auf Integrität überprüft wird (1). Wurde der OS Loader erfolgreich gemessen, kann er in das *Trust Boundary* aufgenommen werden (2), d.h. dass er nun "Trusted" ist und die Vertrauenskette bei ihm angekommen ist. Als vertrauenswürdige Komponente, besitzt es dann die Fähigkeit, die Vertrauenskette zu erweitern. So kann der OS Loader nun den Betriebssystem-Code auf Integrität überprüfen (3), bevor er geladen wird. Nach demselben Prinzip kann das Betriebssystem in die Trust Boundary aufgenommen werden (4). Dies kann so weitergeführt werden bis zur Anwendungsebene, wo unterschiedliche Anwendungen vom Trusted OS gemessen werden (5), bevor sie gestartet werden (6). Die Komponenten, die die Messungen durchführen, z.B. das Betriebssystem, müssen dabei um Funktionalitäten erweitert werden, die dies unterstützen. Wird bei einem dieser Messungen eine Abweichung von Referenzwerten festgestellt, so können durch weitere Komponenten entsprechende Gegenmaßnahmen eingeleitet werden, um einen unvertrauenswürdigen Zustand zu verhindern. Da u.U. sehr viele Messwerte vorliegen und verwaltet werden müssen, wird ein Mechanismus benötigt, der es erlaubt trotz beschränkter Anzahl an Registern, unendlich viele Messwerte zu verwalten. Andererseits dürfen gespeicherte Werte nicht einfach gelöscht werden, da es dann möglich wäre einen

3.2. Trusted Computing System

Messwert, der eine Manipulation an einer Systemkomponente signalisiert durch einen zu ersetzen, der den bekannten Hashwert beinhaltet. Der Mechanismus, der von der TCG hierfür vorgesehen ist, sieht eine Speicherung von neuen Messwerten nach folgendem Schema vor:

$$PCR[n] = SHA - 1(PCR[n] + gemesseneDaten) \quad (3.1)$$

Ein neuer Eintrag zu Register n wird hinzugefügt, indem der neue Messwert zum alten Wert in Register n angehängt, und anschließend ein Hash auf diese Daten angewandt wird. Diese einfache Vorgehensweise garantiert, dass bei jeder neuen Messung stets sämtliche alte Werte mitberücksichtigt werden. Um zusätzlich noch einen Überblick über die Entstehung und Quelle der Messwerte zu erhalten, wird ein sogenanntes *Stored Measurement Log (SML)* protokolliert.

Root of Trust for Reporting. Das *Root of Trust for Reporting (RTR)* ist dafür zuständig, Nachweise über die in den PCR enthaltenen Messwerte zu erstellen und deren Echtheit zu beglaubigen. Hierfür werden signierte Berichte über gemessene Systemzustände erstellt, die dann bspw. im Attestierungsprozess an eine weitere Plattform gesendet werden können. Die Plattform hat somit eine vertrauenswürdige Quelle, die den Systemzustand der Plattform darlegt. Als Basis für das RTR dient der Endorsement Key. Das Prinzip der Attestierung wird im nächsten Abschnitt näher beschrieben.

Root of Trust for Storage. Dieser Vertrauensanker ist für die Bereitstellung einer Basis für einen sicheren Speicher zuständig. Mit dem Storage Root Key als Basis wird hierfür eine geschützte Schlüsselhierarchie aufgebaut. Wie bereits erwähnt ist der Storage Root Key ein 2048 Bit Schlüsselpaar, das im TPM erzeugt, und im nicht-flüchtigen Speicher abgelegt wird. Er darf die geschützte Umgebung des TPM niemals verlassen. Sämtliche Operationen, den SRK benutzen, werden innerhalb des TPM durchgeführt. Das hat die Auswirkung, dass Daten, die mit dem SRK verschlüsselt werden, auch nur in diesem einen TPM entschlüsselt werden können. Möchten verschiedene Nutzer oder Anwendungen von dieser Eigenschaft Gebrauch machen, so wird klar, dass ein einziger Schlüssel nicht ausreicht. Ein TPM hat jedoch nur einen begrenzten Speicherplatz. Die Lösung besteht darin, den SRK als Grundlage für eine beliebig große Schlüsselhierarchie zu verwenden.

3. Trusted Computing

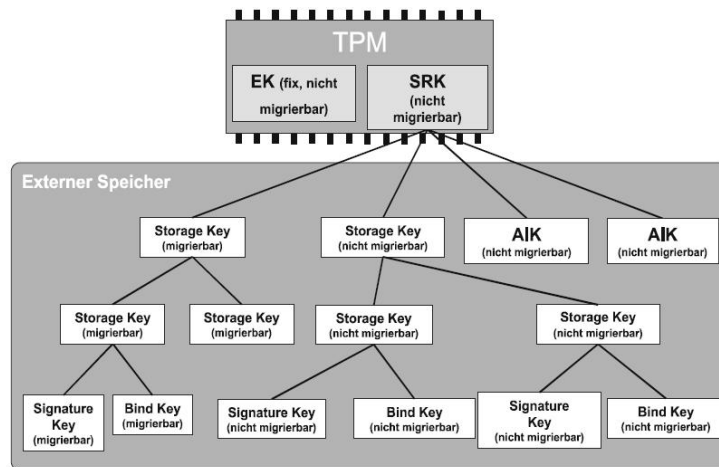


Abbildung 3.4.: TPM Schlüsselhierarchie und Trusted Storage [82]

Es können beliebig viele neue Schlüssel erzeugt und als Nachfolger des SRK in der Hierarchie eingetragen werden. Dies kann auch rekursiv auf den unteren Ebenen weitergeführt werden. Sämtliche Schlüssel unterhalb des SRK werden ausgelagert, indem jeder Schlüssel mit seinem Vorgängerschlüssel *ummantelt* wird. Die Nutzung eines Schlüssels erfordert im Endeffekt immer den SRK, welches sich bekanntermaßen im TPM befindet. Stellt man sich diese Hierarchie, wie in Abbildung 3.4 verdeutlicht, als Baum vor, bildet der SRK als Wurzel die *Root of Trust for Storage (RTS)*. Das Vertrauen wird sukzessive vom SRK bis zu den Blättern weitergereicht. Die Schlüssel des Baums ermöglichen ein *Trusted Storage*, in dem sensible Daten TPM-basiert abgesichert werden können.

Trust Domains. Die TPM-Schlüsselhierarchie hat den Effekt, dass die einzelnen Bereiche innerhalb des Schlüsselbaumes isoliert sind. Für einen Schlüssel K des Schlüsselbaumes bezeichne dom_K die Menge aller Schlüssel, die sich im Teilbaum mit Wurzel K befinden. Dann kann ein Zugriff auf einen Schlüssel in dom_K nur erfolgen, wenn das Passwort für K bekannt ist. Sind also k_1 und k_2 beliebige Schlüssel in der Schlüsselhierarchie mit $k_1 \notin dom_{k_2}$ und $k_2 \notin dom_{k_1}$, so erfordert die Nutzung für die beiden Schlüssel zwei unterschiedliche Passwörter. k_1 und k_2 befinden sich also in zwei voneinander unabhängigen *Trust Domains*. Dies ist eine ideale Ausgangslage, um voneinander getrennte, sichere Speicherbereiche für unterschiedliche Anwendungen und Benutzer zu erzeugen. So reicht ein TPM, um getrennte Domänen für mehrere Nutzer der Plattform zu erzeugen. Eine Domäne

kann auch weitere Subdomänen enthalten, bspw. um Credentials verschiedener Anwendungen zu speichern.

3.2.3. TPM Schlüsselverwaltung

Bei TPM-Schlüsseln handelt es sich um asymmetrische RSA- Schlüsselpaare. Die Schlüssellängen können bei der Erzeugung definiert werden. Unterstützt werden mindestens die Schlüssellängen 512, 1024 und 2048 Bit. Von der TCG empfohlen wird die Verwendung von 2048 Bit Schlüsseln. Beim Endorsement Key und dem Storage Root Key handelt es sich 2048 Bit Schlüsselpaare. Ein wichtiges Merkmal vom TPM-Schlüsseln ist die Migrierbarkeit. Schlüssel können als *migrierbar* oder *nicht-migrierbar* definiert werden. Nicht-migrierbare Schlüssel dürfen das TPM nicht ungeschützt verlassen. Sämtliche Daten, die mit so einem Schlüssel geschützt sind, sind für immer an das eine TPM gebunden, können also nicht auf eine andere Plattform migriert werden, genauer gesagt, kann eine Migration nicht erzwungen werden. Migrierbare Schlüssel können durch bestimmte Mechanismen auf andere Plattformen transferiert werden. Der TSS hält entsprechende Funktionalitäten bereit. Der EK und SRK sind Beispiele für nicht-migrierbare Schlüssel. Weiterhin definiert die TCG folgende Schlüsseltypen:

- *Signing Keys* werden für die Erzeugung von digitalen Signaturen und Message Digests benutzt. Sie können migrierbar oder nicht-migrierbar sein.
- *Storage Keys* haben den Zweck weitere Schlüssel in der Schlüsselhierarchie zu ummanteln oder beliebige andere Daten zu verschlüsseln. Der SRK ist solch ein Storage Key.
- *Identity Keys* sind die Attestation Identity Keys. Sie dürfen lediglich für die Signierung von Daten vom TPM (z.B. PCR Werte) benutzt werden.
- *Binding Keys* können benutzt werden um kleine Datenmengen oder symmetrische Schlüssel an das TPM zu binden
- *Attestation Identity Keys (AIK)* wurden mit Version 1.1 der TCG Spezifikation eingeführt. Sie sollen anstelle vom EK für Attestierungs-Signaturen verwendet werden. Dies resultierte aus datenschutzrechtlichen Bedenken, da zwei verschiedene Attestierungsvorgänge anhand des verwendeten EK

3. Trusted Computing

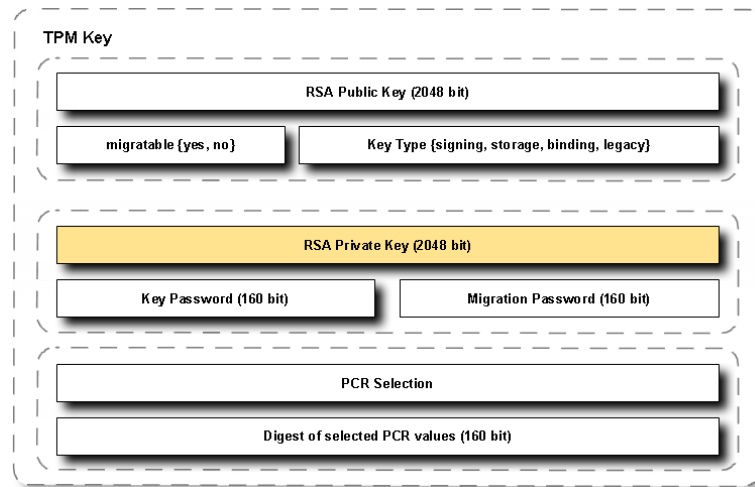


Abbildung 3.5.: Interne Struktur eines TPM-Schlüssels [58]

in Verbindung gebracht werden können. AIKs werden dagegen nur einmalig verwendet und werden von einer vertrauenswürdigen dritten Instanz signiert, damit sie dem EK gleichwertig sein können.

- *Legacy Keys* sind Schlüssel, die außerhalb des TPM erzeugt, und importiert wurden. Sie können für Signaturen oder Verschlüsselungen benutzt werden und sind naturgemäß migrierbar.
- *Authentication Keys* sind symmetrische Schlüssel, die benutzt werden, um Kommunikation mit dem TPM abzusichern.

Abbildung 3.5 zeigt die interne Struktur eines TPM-Schlüssels. Im Wesentlichen enthält die Datenstruktur eines TPM Schlüssels den 2048 Bit großen öffentlichen Teil des Schlüssels und Flags, die angeben, um welchen Typ von Schlüssel es sich handelt (z.B. migrierbar/nicht-migrierbar, storage/signing/binding, etc.). Weiterhin enthalten ist der 2048 Bit private Teil des Schlüssels. Dieser muss in verschlüsselter Form vorliegen, da ein TPM-Schlüssel ggf. extern gespeichert wird. Ein für die Entschlüsselung notwendiges Authentifizierungspasswort ist bei der Erzeugung des Schlüssels anzugeben. Der Hashwert des Passwortes ist ebenfalls in der Datenstruktur enthalten. Für migrierbare Schlüssel ist ebenfalls ein Passwort anzugeben, das auch als Hashwert in der Datenstruktur enthalten ist. Wurde ein Schlüssel an PCR-Register gebunden (sealing), so sind Informationen über die

gewählten Register und die Fingerabdrücke der Werte in einer `TPM_PCR_INFO`-Datenstruktur im Schlüssel hinterlegt.

Key Cache Manager

Wie bereits erwähnt, sind alle Schlüssel in der TPM-Schlüsselhierarchie mit einem anderen Schlüssel aus der Hierarchie ummantelt, damit sie extern gespeichert werden können. Soll ein Schlüssel verwendet werden, müssen sämtliche Schlüssel von diesem Schlüssel, bis zum SRK in das TPM geladen werden. Da die Anzahl der zu ladenden Schlüssel die Anzahl der vorhandenen Key-Slots übersteigen kann, sieht die TCG einen *Key Cache Manager* vor, der das Laden und die Verwaltung von Schlüsseln, während dem Ladeprozess übernimmt, und dabei ggf. Schlüssel temporär in einen Zwischenspeicher auslagert. Der TSS bietet die Möglichkeit, TPM-Schlüssel in einem *Persistent Storage* zu registrieren. Dem Schlüssel wird dabei ein *Universally Unique Identifier (UUID)* zugewiesen, der als Parameter angegeben werden kann, wenn der Schlüssel geladen werden soll.

3.2.4. TCG Software Stack

Der Umfang der TPM-Kommandos ist beschränkt und bietet nur grundlegende Funktionalitäten. Um diese auch sinnvoll in Anwendungen verwenden zu können, hat die TCG eine mehrschichtige API definiert, die die TPM-Kommandos ausnutzen, um höhere Funktionalitäten zu implementieren. Über den *TCG Software Stack (TSS)* haben Entwickler die Möglichkeit TC-Funktionalitäten, wie Verschlüsselung, Hashing in ihre Anwendungen einzubauen. Anwendungen können dabei an unterschiedlichen Ebenen der TSS-Hierarchie ansetzen. Ausgehend vom TPM sind das folgende Schichten:

- **TDD/TDDI.** Der TPM Device Driver bildet die unterste Ebene und ist die einzige Komponente, die direkt mit dem TPM kommuniziert. Er kann als Treiber für das herstellereigenspezifische TPM gesehen werden, und muss auch von Hersteller geliefert werden. Von weiteren Komponenten kann der TDD nur über den TDD Interface angesprochen werden. Pro TSS darf es nur eine Verbindung zu dem TPM Device Driver geben.
- **TDDL/TDDLI.** Die TPM Device Driver Library (TDDL) sorgt zum einen

3. Trusted Computing

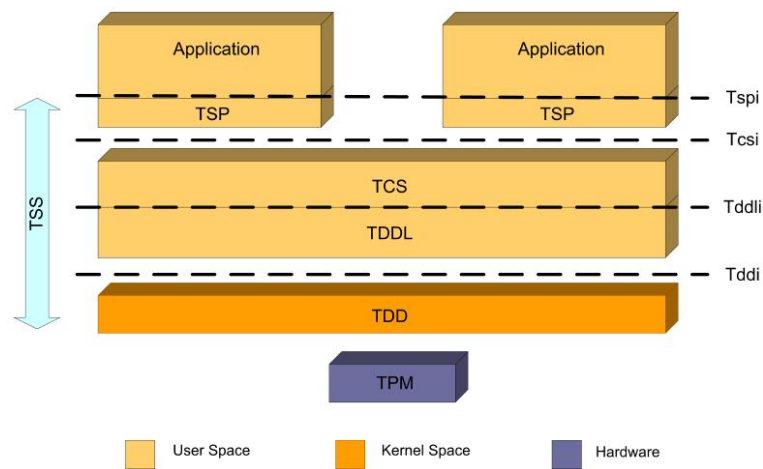


Abbildung 3.6.: Die TCG Software Stack Hierarchie

dafür, dass ein TPM, unabhängig vom Modell, von außen identisch aussieht und somit über die gleiche Schnittstelle (TDDL) ansprechbar ist. Zum anderen stellt sie den Übergang vom Kernel Mode zum User Mode dar. Pro TPM existiert nur eine TDDL-Instanz.

- **TCS/TCSI.** Die TSS Core Services (TCS) stellen sämtliche grundlegenden und höheren TPM-Funktionalitäten über die TCS Schnittstelle (TCSI) zur Verfügung. Die TCS ermöglichen somit die beschränkten Mittel eines TPMs effizient zu verwalten. Zudem bieten die TCS Möglichkeiten, um die Kommunikation und Anfragen an ein TPM zu verwalten. Eine weitere Aufgabe des TCS ist es, die Anfragen von verschiedenen TSP-Instanzen zu verwalten. Die TCS-Instanz wird als Systemdienst ausgeführt.
- **TSP/TSPI.** Auf der höchsten Schicht befinden sich die TSS Service Provider (TSP). Die TSP definieren einen Satz an höheren Funktionen, die es Entwicklern ermöglichen komplexe TC-unterstützte Anwendungen zu erstellen. Die TSP stützen sich dabei auf die TCS, und bieten ihre Funktionen wiederum über die TSPI-Schnittstelle. Für jede Anwendung existiert eine eigene Instanz des TSP im selben Prozess.

3.2.5. Trusted Computing Szenarien

Nachdem die technischen Grundlagen für Trusted Computing Systeme vorgestellt wurden, sollen in diesem Abschnitt eine Auswahl von allgemeinen Szenarien vorgestellt werden, die von ihnen profitieren können, um Sicherheit, Integrität und Datenschutz zu erhöhen. Es wird kein Anspruch auf Vollständigkeit erhoben.

Remote Attestation. Ein Kernkonzept von Trusted Computing ist *Remote Attestation*. Hierbei geht es darum, dass eine Plattform einer entfernten Plattform (bspw. ein Service Provider) seine Vertrauenswürdigkeit beweisen möchte, bzw. muss. Wie bereits erläutert repräsentieren die Inhalte der PCR die Vertrauenswürdigkeit, bzw. den Systemzustand einer Plattform. Bei Remote Attestation geht es daher im Prinzip darum, die Inhalte ausgewählter PCR an den Dienstleister zu senden, so dass dieser die Vertrauenswürdigkeit verifizieren kann. Hierzu signiert die anfragende Plattform die Daten mit einem AIK und sendet sie zusammen mit dem *AIK Credential* an die entfernte Plattform. Diese kann daraufhin die Konfiguration und somit die Vertrauenswürdigkeit des Systems überprüfen. Die Herausforderung an diesem Protokoll ist, dass der Dienstleister sich sicher sein muss, dass der verwendete AIK tatsächlich von einer Plattform mit einem konformen TPM stammt, und der private Teil des AIK nur der anfragenden Plattform bekannt ist. Hierfür ist eine weitere Partei notwendig, der sowohl die anfragende Plattform, als auch der Dienstleister vertraut. *Privacy Certification Authorities (PCA)* haben die Aufgabe, auf Anfrage einer Plattform ein signiertes AIK-Zertifikat (bzw. Attestation-Identity-Zertifikat) zu erstellen und dem Requestor zu senden. Dieses Zertifikat kann die anfragende Plattform mit seiner Anfrage an den Service Provider senden, woraufhin er das AIK-Zertifikat analysieren kann, um zu erfahren, ob die Plattform TCG-konform ist. Abbildung 3.7 verdeutlicht das Protokoll für einen AIK Credential Request. Die anfragende Plattform muss sich für diesen Prozess zunächst ein AIK-Schlüsselpaar erzeugen. Der öffentliche Teil des AIK wird zusammen mit dem *EK Credential* und weiteren Plattform Credentials in einer Anfrage an eine ausgewählte PCA gesendet. Die Daten werden vorher mit dem öffentlichen Schlüssel der PCA verschlüsselt. Die PCA überprüft die Plattform-Zertifikate und kann somit bestimmen, ob sie tatsächlich von einer TCG-konformen Plattform mit dem angegebenen EK stammen. Ist die Validierung

3. Trusted Computing

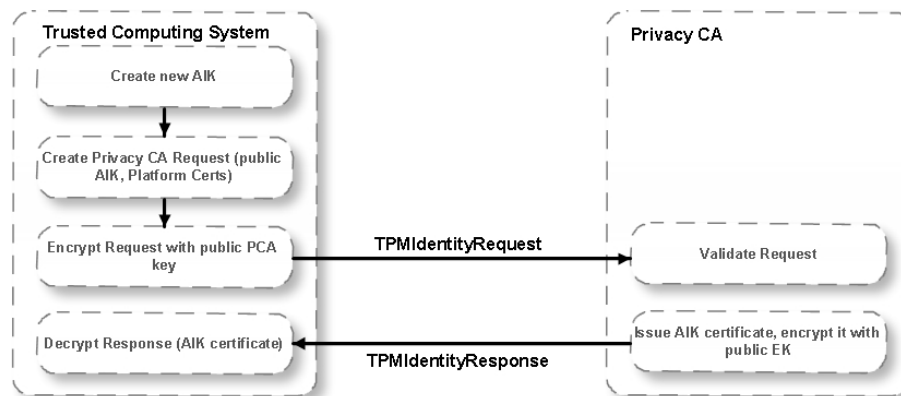


Abbildung 3.7.: AIK Credential Request [58]

erfolgreich, erzeugt die PCA ein Zertifikat für den AIK, das sie mit dem öffentlichen Teil des EK des Requestors verschlüsselt, und an diesen zurück sendet. Besitzt der Requestor den privaten Teil des EKs, kann er das AIK-Zertifikat entschlüsseln. Das von der PCA signierte Zertifikat enthält sämtliche Daten, die der Service Provider benötigt, um darauf zu vertrauen, dass der Requestor TCG-konform ist. Das naheliegendste Anwendungsszenario ist ein Content Provider, der sicher sein möchte, dass der Client entsprechende Software besitzt und diese zudem nicht modifiziert wurde. Jedoch erlaubt Remote Attestation nicht nur eine Attestierung eines Clients gegenüber eines Servers. So kann man sich auch den Fall vorstellen, dass der Online-Banking-Server in einem vertrauenswürdigen Zustand sein muss, bevor Transaktionen stattfinden.

Das oben aufgeführte Protokoll ist abhängig von einer vertrauenswürdigen Instanz, der sowohl Requestor und Provider vertrauen. Diese Instanz besitzt alle Informationen über die benutzten EKs, so dass dieser die Möglichkeit hat, einzelne Plattformen zu verfolgen. So sehr der Instanz getraut werden kann, ist die Privacy CA ein zentraler Angriffspunkt. Zudem muss für jeden AIK ein neues Zertifikat ausgestellt werden. Dieser Prozess kann längere Zeit in Anspruch nehmen, so dass die PCA einen Flaschenhals der Remote Attestation darstellt. Aufgrund dieser Probleme wurde mit Version 1.2 der TCG-Spezifikation eine Möglichkeit des *Direct Anonymous Attestation (DAA)* ermöglicht. Bei diesem Verfahren, kann auf eine PCA verzichtet werden. Beweiserbringung findet direkt zwischen Requestor und Provider statt. Eingesetzt werden dabei *Zero-Knowledge-Proofs*, bei der eine Partei einer anderen Partei den Besitz von bestimmten Informationen be-

3.2. Trusted Computing System

weisen kann, ohne die Information an sich preiszugeben. Je nach Protokoll kann nach einer bestimmten Anzahl von Wiederholungen, die Wahrscheinlichkeit, dass der Requestor die Information tatsächlich besitzt, hinreichend erhöht werden. Das von der TCG vorgeschlagene Protokoll ist aus Privacy-Sicht nicht so gut, wie die PCA-Lösung. So ermöglicht es dem Verifizierer u.U. den Requestor wiederzuerkennen. Es gibt bereits Vorschläge, wie dieses Problem behoben, und die DAA verbessert werden kann [12].

Binding, Signing und Sealing. Als kryptographische Hardware kann das TPM auch für herkömmliche Zwecke verwendet werden kann. So ist es durch *Binding* möglich, Daten mit TPM-Schlüsseln zu verschlüsseln. Da diese Schlüssel an das TPM gebunden sind, ist eine Entschlüsselung auch nur innerhalb desselben Rechners möglich. Eine besondere Art der Verschlüsselung ist das *Sealing*. Hierbei können Daten zusätzlich zu einem bestimmten Schlüssel auch noch an den Inhalt eines oder mehreren PCR-Register zu binden. Sollen die Daten zu einem späteren Zeitpunkt entschlüsselt werden, muss der korrekte Schlüssel benutzt werden, und zusätzlich muss sich das System im richtigen Zustand befinden. Neben der *Signing*-Funktion, die nichts anderes als die Erzeugung von digitalen Signaturen mit TPM-Credentials ist, gibt es auch hier eine erweiterte Version, das *Sealed-Signing*.

Trusted Network Connect. Ein weiteres Einsatzszenario für Trusted Computing ist das *Trusted Network Connect (TNC)*, welches durch eine eigene Arbeitsgruppe erarbeitet wird. TNC soll Trusted Computing in den Bereich Network Access Control bringen. Im Prinzip geht es dabei um die gegenseitige Beweiserbringung der Vertrauenswürdigkeit in Szenarien, in denen verschiedene Plattformen über Netzwerke miteinander kommunizieren. Durch TNC soll sichergestellt werden, dass sich die Kommunikationspartner in vertrauenswürdigen Zuständen befinden. TNC erweitert gängige Authentifizierungsmechanismen um TC-Funktionen. Hierbei kommen, wie auch bei Remote Attestation Integritätsmessungen ins Spiel. In den Spezifikationen wird dieser Vorgang *Platform-Authentication* genannt, also die Verifizierung der Integrität des Systemzustandes einer Plattform mit Hilfe von Trusted Computing. Solch ein Protokoll ist kompatibel mit gängigen Verfahren, wie z.B. der 802.1X-Familie [36].

3. Trusted Computing

Root of Trust for Identities. Als Architektur, die eine vertrauenswürdige Interaktion verschiedener Plattformen erlaubt, kann Trusted Computing im Bereich Identity Management (IdM) sinnvoll eingesetzt werden. So werden bspw. oft digitale Tickets bei Online-Diensten eingesetzt. Diese werden für Clients ausgestellt, um ihnen Zugriff auf bestimmte Dienste zu gewähren. Trusted Computing kann hier die Basis des Vertrauens zwischen den Beteiligten Instanzen stärken, indem ein mit einem TPM ausgestatteter *Identity Provider* eingesetzt wird. Dieser Identity Provider kann sich von einer Privacy-CA AIK Credentials erstellen lassen, die dann an die Tickets der Nutzer gehängt werden. Der Service Provider kann die Gültigkeit der Tickets nun anhand des AIK Credentials überprüfen. [24]. Somit formt das System des Identity Providers die *Root of Trust for (digital) Identities*. Es wurden bereits Konzepte vorgestellt, wie dieses Prinzip im Authentifizierungsdienst *Kerberos* eingesetzt werden kann [51].

3.3. Trusted Computing Projekte

Seit der Einführung von TC gab es bereits einige Projekte, die TC-Konzepte umsetzten. Lösungen werden sowohl von Industrie als auch von Wissenschaftseinrichtungen veröffentlicht. Zudem gibt es auch Initiativen der Europäischen Union.

Microsoft: Palladium / NGSCB. Als einer der Hauptgründer der TCG-, bzw. T CPA-Initiative war es auch Microsoft, das erste Konzepte für die Umsetzung von Trusted Computing-Szenarien veröffentlichte. Durch die ursprünglich *Palladium* genannte Technologie soll die Sicherheit von Computersystemen in Kombination von neuen Hardwarebausteinen und Anwendungen erhöht werden. Nur wenige technische Details über Palladium wurden bekannt gegeben, jedoch stieß die Palladium-Idee früh auf viel Kritik, was u.a. der Grund für eine spätere Namensänderung war. Zu dem nun *Next Generation Secure Computing Base (NGSCB)* genannten Projekt wurden auch mehr technische Details veröffentlicht. Die neue Vorstellung war eine Architektur, die ein Computersystem in vier Quadranten einteilt. Diese Einteilung sieht zunächst eine Aufteilung in eine linke Seite (*Left Hand Side*) vor, auf der ein gewöhnliches Betriebssystem läuft. In der rechten Seite (*Right Hand Side*) wird ein neues Sicherheitssystem eingeführt. Dieses baut auf einem Hardwarebaustein (TPM) auf und enthält einen neuen Sicherheitskern

3.3. Trusted Computing Projekte

Nexus, der die Aufgabe hat kritische Komponenten in einer sicheren Umgebung auszuführen und weitere sicherheitskritische Funktionalitäten wie sicheres Booten, Prozessisolierung, Schlüsselverwaltung, etc. bereitzustellen. Der Sicherheitskern wird möglichst klein gehalten, um Design- und Programmierfehler möglichst gering zu halten. Von Microsoft wurde bekannt gegeben, dass die Architektur mit der nächsten Version des Windows Betriebssystems (Codename “Longhorn”), also dem heute bereits verfügbaren Windows Vista eingeführt werden sollte. In Vista sind jedoch nur sehr wenige Konzepte umgesetzt. Dazu gehören die *TPM Base Services (TBS)*, ein Systemdienst, das den Zugriff auf das Trusted Platform Module durch verschiedene Anwendungen oder TCG Software Stacks transparent regelt. Die TBS können somit als zusätzliche Schicht zwischen den TSS Core Services (TCS) und dem TPM Device Driver (TDD) gesehen werden. Als praktische Anwendung, die vom TPM gebrauch macht, wurde die Laufwerkverschlüsselungs-Software *BitLocker* geliefert. BitLocker nutzt u.a. Sealing, um die Entschlüsselung von Laufwerken an eine bestimmte Systemkonfiguration zu binden. Von offizieller Seite, u.a. auf den Microsoft Webseiten sind kaum noch Informationen über NGSCB zu finden und die vorhandenen Daten sind teilweise veraltet. So ist eine Umsetzung von NGSCB oder einer ähnlichen Architektur auch nicht mit der aktuell im Beta-Stadium befindlichen Windows 7 zu erwarten.

EMSCB / Perseus / Turaya. Der Name der *European Multilateral Secure Computing Base (EMSCB)*-Initiative lässt bereits erahnen, dass es sich hierbei um ein zu NGSCB ähnliches Vorhaben handelt. Das Ziel des EMSCB Projektes, das u.a. vom Bundesministerium für Wirtschaft und Technologie gefördert wird, ist die Konzipierung und Realisierung von vertrauenswürdigen Computerplattformen, die die Probleme und Schwachstellen bisheriger Systeme eliminieren sollen[14]. Hierbei werden offene Standards und Trusted Computing-Technologien eingesetzt. Die Ergebnisse werden quelloffen zur Verfügung gestellt. Sehr auffällig an der Initiative und deren Öffentlichkeitsarbeit ist die – nicht explizit geäußerte – Distanzierung von der Microsoft’schen Palladium- bzw. NGSCB-Politik. Betont werden vor allem die Aspekte der *Offenheit* und des *Multilateralismus*, welches schon am Namen des Projektes zu erkennen ist. Gezielt werden bisherige Kritiken und Besorgnisse rund um Trusted Computing aufgegriffen und vermittelt, dass EMSCB nicht nur Content-Providern zu Gute kommt, sondern auch Endanwender und de-

3. Trusted Computing

ren Bedürfnisse im Bezug auf Datenschutz und Privatsphäre bei der Entwicklung von vertrauenswürdigen Systemen berücksichtigt werden. Auch aus technischer

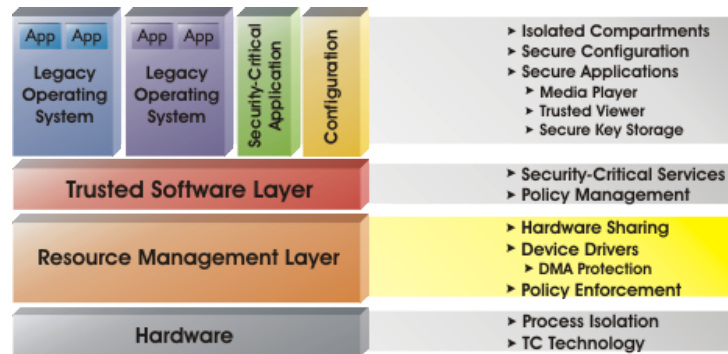


Abbildung 3.8.: Das PERSEUS-Sicherheitsframework [4]

Sicht ist die EMSCB-Architektur mit NGSCB zu vergleichen. EMSCB baut dabei auf die *Perseus*-Architektur, welche bereits seit 1999 entwickelt wird. Hauptteil von Perseus ist – wie auch in der NGSCB-Architektur – ein minimaler, einfach verwaltbarer, stabiler Sicherheitskern, der auf der Hardwareschicht aufsetzt. Abbildung 3.8 verdeutlicht die Vier-Schichten-Architektur von Perseus. Im Vergleich zum NGSCB-Modell, sind hier die Schichten übereinander angeordnet. Die unterste Schicht bildet die *Hardware-Schicht*. Diese beinhaltet neben der herkömmlichen Hardware eine Erweiterung um spezielle Sicherheitshardware. Dies kann bspw. ein Trusted Platform Module (TPM) sein. Auf der Hardwareschicht setzt die *Resource Management*-Ebene auf. Diese Schicht regelt den Zugriff auf die Hardware und erzwingt Policies für den Zugriff auf die Hardware, da dies besonders sicherheitskritisch ist. Die *Trusted Software*-Ebene beinhaltet sämtliche als sicherheitskritisch einzustufende Dienste. Dazu gehören das *Secure Booting*, Attestierungsdienste, Krypto-APIs, *Trusted Storage*, *Trusted Installer* oder eine *Trusted GUI*, das eine sichere Schnittstelle zwischen System und Benutzer bildet (bspw. sichere Eingabemasken). Diese Dienste werden im Sicherheitskern ausgeführt und sind somit von der normalen Anwendungsumgebung getrennt, wodurch eine Angreifbarkeit dieser Dienste minimiert wird. Auf der obersten Schicht, der *Anwendungsebene* können normale Betriebssysteme oder weitere nicht-sicherheitskritische Anwendungen ausgeführt werden. Hierbei wird Virtualisierung eingesetzt, wodurch bspw. mehrere Betriebssystem-Instanzen parallel in voneinander getrennten *Compartments* ausgeführt werden können. Die Architektur erfordert somit keine Anpassungen an

3.3. Trusted Computing Projekte

bisheriger Software, damit diese auf EMSCB-fähigen Plattformen lauffähig sind. Die Urversion der Palladium-Pläne hatte genau an dieser Stelle einen seiner Kritikpunkte. Anwendungen hätten für Palladium-Plattformen neu geschrieben werden müssen.

Damit die Konzepte auch in der Praxis bestätigt werden können, wurde auch an einer Umsetzung der Technologie gearbeitet. Das bisherige Resultat ist *Turaya*. Turaya soll vor allem als Konzeptbestätigung dienen, weshalb auch für zwei Einsatzszenarien praktische Lösungen entwickelt wurden. Dies ist zum einen *Turaya.Crypt*, eine Geräteverschlüsselungssoftware, welche in einer isolierten Umgebung oberhalb des Trusted Software Layers ausgeführt wird. Kryptographische Funktionalitäten können u.a. über das TPM in Anspruch genommen werden. Grundlegende Funktionsweise ist die Freigabe der Geräte erst nach erfolgreicher Eingabe des richtigen Passwortes über eine Trusted GUI. Dies kann je nach Betriebsmodus zu Beginn des Bootvorgangs, also vor Start eines "unsicheren" Betriebssystems (pre-boot Authentifikation), oder erst bei Bedarf, z.B. bei Anhängen eines Wechselmediums (just-in-time Authentifikation) erfolgen. Die zweite Anwendung ist *Turaya.VPN*, der quasi als Gateway für das Internet dient. Sämtliche Kommunikation vom und zum Internet wird über den in einer isolierten Umgebung laufenden VPN-Client geleitet und verschlüsselt. Der zur Verschlüsselung verwendete Schlüssel befindet sich ebenfalls im Compartment des Clients und wird nur freigegeben, wenn die Integrität des Sicherheitskerns gewährleistet ist. Die EMSCB-Initiative kann zum jetzigen Zeitpunkt als relativ lebendig bezeichnet werden.

OpenTC. Ein durch die Europäische Union gefördertes Projekt ist *Open Trusted Computing (OpenTC)*, das die Entwicklung und Spezifizierung von TC-Frameworks auf Open Source-Basis anstrebt [49]. Das Projekt ist in mehrere *Work Packages (WP)* aufgeteilt, die sich jeweils mit der Entwicklung eines bestimmten Aspektes von Trusted Computing Systemen beschäftigen. Diese beinhalten bspw. die Entwicklung eines Trusted Betriebssystems (WP04), Lösungen für eingebettete Systeme (WP08), Prototypen-Entwicklung Anwendungen (WP06) oder Dienste für die Verwaltung der Infrastrukturen (bspw. Key Management, WP05). Teilweise werden auch Ergebnisse aus anderen TC-Projekten übernommen. Zum Bei-

3. Trusted Computing

spiel dient die EMSCB-Architektur als Grundlage für die Entwicklung von frühen Prototypen.

TPM Emulator. An der Eidgenössischen Technischen Hochschule wurde ein softwarebasierter TPM-Emulator entwickelt, der die TDDL für Linux beinhaltet[81]. Der Emulator ist gut für Test- und Entwicklungszwecke geeignet, wenn bspw. keine Plattform mit TPM zur Verfügung steht. Er ist als *Daemon* implementiert und kann über ein Unix-Socket angesprochen werden. Die meisten TPM-Kommandos sind bereits implementiert. Vor allem sind jedoch Kommandos rund um Schlüsselmigration noch nicht umgesetzt.

TrustedGRUB. Der von der Ruhr-Universität Bochum entwickelte *TrustedGRUB* ist eine Erweiterung für den Linux-Bootloader GNU GRUB. TrustedGRUB erweitert GRUB um Funktionalitäten, die für die Fortführung der Vertrauenskette des Trusted Bootings nötig sind. Wurde der Bootloader vom BIOS in den Trust Boundary aufgenommen, führt TrustedGRUB Messungen der Betriebssystemkomponenten durch, um die Vertrauenskette fortzuführen. Eine interessante Möglichkeit, die TrustedGRUB bietet ist die Angabe von weiteren Komponenten oder beliebigen Dateien, die bei den Integritätsmessungen berücksichtigt werden sollen. Hierzu bietet TrustedGRUB die *checkfile*-Option, über die die Dateien angegeben werden können. Die Hashwerte der gemessenen Dateien werden im Platform Configuration Register 13 abgelegt. Stellt TrustedGRUB Uneinstimmigkeiten zwischen gemessenen und Referenzwerten fest, wird der Bootprozess pausiert, und dem Nutzer die Option angeboten den Prozess zu unterbrechen, um somit das Laden von (möglicherweise) unvertrauenswürdigen Komponenten zu verhindern.

TSS Implementierungen. Es gibt bereits einige Implementierungen, die den TCG Software Stack mehr oder weniger spezifikationstreu umsetzen. *TrouSerS* ist eine C-Implementierung, die größtenteils noch auf der TSS-Spezifikation 1.1 aufbaut. Unter dem Deckmantel *Trusted Computing for the Java Platform* entwickelt die IAIK TU Graz Implementierungen rund um den TSS. Hauptsächlich sollen Implementierungen geschaffen werden, die für Java-Entwickler zugänglich sind. In diesem Projekt, das auch Teil der OpenTC-Initiative ist, entstand zunächst ein Java-Wrapper um TrouSerS, der den TSS in Java-Programmen zugänglich macht. Später entstand *jTSS*, eine vollständige Implementierung in Java. Unter anderem ist

eine Nutzung mit dem TPM Emulator möglich. Neben jTSS und dem jTSS Wrapper, entwickelt das Projekt weitere Pakete, um TPM- und TC-Funktionalitäten in Java-Projekten verfügbar zu machen. Dazu gehören bspw. die *jTPM Tools*, eine Sammlung von nützlichen Programmen, die eine Interaktion mit dem TPM ermöglichen. Grundlegende Operationen, wie z.B. TakeOwnership sind darin enthalten. Weiterhin gibt es eine Implementierung einer *PrivacyCA* und *TCcert Tool*, das die Erstellung von TCG Zertifikaten erlaubt. Weitere TC-Frameworks sind *tpm4java*, eine von der TU Darmstadt entwickelte Java-Bibliothek, die Trusted Computing in Java-Anwendungen ermöglichen soll. Auf der Homepage der TCG befindet sich eine umfangreiche Auflistung von Entwicklerwerkzeugen, die für die Umsetzung für TC-Anwendungen hilfreich sein können[32]. Des Weiteren hat das BSI eine detaillierte Analyse von verfügbaren Open Source Werkzeugen durchgeführt und die Ergebnisse veröffentlicht [75].

3.4. TPM Symbiose

Auf die unterschiedlichsten Anwendungsszenarien von TPM-basierten Systemen wurde bereits eingegangen. Eine neue Technologie muss jedoch nicht zwingend von bestehenden Systemen unabhängig umgesetzt werden. In diesem Abschnitt werden zwei in Kapitel 2 vorgestellte Produkte aufgegriffen. Hierbei soll festgestellt werden, in wie Fern diese zusammen mit einem TPM eingesetzt werden können, so dass die unterschiedlichen Technologien davon profitieren.

Smart Cards und TPM. Auch wenn die Grundmotivation für die Entwicklung beider Technologien unterschiedlicher Natur waren, ist es aufgrund der architekturellen Eigenschaften naheliegend, einen Vergleich zwischen Smart Cards und dem TPM durchzuführen. Dieser Angelegenheit haben sich bereits einige Arbeiten gewidmet. Es sollte festgestellt werden, in wie Fern sich die Technologien ausschließen, bzw. wie sie kombiniert werden können. Jedoch soll zunächst auf die grundlegenden Unterschiede eingegangen werden. Smart Cards sind portable Chipkarten, die der Benutzer mit sich führen kann. Er kann auf der Karte seine Credentials ablegen und sie somit ständig mit sich tragen. Es besteht also eine enge Verbindung zwischen der Smart Card und dem Nutzer. Im Gegensatz dazu ist ein TPM fest in ein Rechnersystem integriert und kann von diesem nicht entfernt

3. *Trusted Computing*

werden. Daten, die vom TPM geschützt werden sind somit auch an die Plattform gebunden. Als Ergebnis der Vergleiche kann festgehalten werden, dass sich durch eine Kombination der beiden Technologien sinnvolle Gesamtlösungen bilden lassen, die von den Stärken beider Systeme gebrauch machen. Als eine Schwäche von TPM-basierten Systemen wird die passwortbasierte Authentifizierung genannt. Smart Cards hingegen ermöglichen eine Zwei-Faktor-Authentifizierung, die neben dem Passwort auch den Besitz der Smart Card erfordert. Um genau zu sein, ist beim TPM der Besitz auch vorausgesetzt, jedoch ist dies gleichzusetzen mit dem Besitz der Plattform. Auf der anderen Seite wird für die Nutzung von Smart Cards eine zusätzliche Infrastruktur in Form von Lesegeräten und Software benötigt, dessen Integrität unerlässlich für eine lückenlose Sicherheit ist. Hier können Trusted Computing Systeme und deren Integritätsmessungen ihre Dienste erbringen.

Ein interessantes Szenario, in dem TC-Systeme durch den Einsatz von Smart Cards optimiert werden könnten, ist die Umgehung einer dritten Vertrauenswürdigen Instanz für die Erzeugung von AIK Credentials [29]. Vorgeschlagen wird, dass der Nutzer einmalig eine von der Zertifizierungsinstanz ausgestellte Smart Card erhält, die in Zukunft die Rolle dieser übernehmen soll. Möchte das System ein AIK Credential erzeugen, so wird der generierte AIK an die Smart Card übergeben. Diese enthält die Routinen, die für die Erstellung des Credentials notwendig sind. Die Existenz einer dritten Instanz ist, bis auf die einmalige Ausstellung der Smart Cards nicht notwendig. Weiterhin wird vorgeschlagen, die Migrierbarkeit von TPM-Schlüsseln durch den Einsatz von Smart Cards als Transportmedium zu ermöglichen. Zusammenfassend kann festgehalten werden, dass keine Technologie die andere ersetzen wird, da ihre Zwecke und Eigenschaften zu unterschiedlich sind. Der große Vorteil vom TPM ist aber die Unabhängigkeit von zusätzlichen Geräten. Ist ein TPM im System vorhanden, so können seine Funktionalitäten und Fähigkeiten direkt genutzt werden.

PKCS#11 und TPM. Der in Kapitel 2 vorgestellte Cryptoki-Standard soll Entwicklern den Zugriff auf die Funktionalitäten von kryptographischen Token vereinfachen, indem eine einfache, standardisierte Schnittstelle definiert wird. Es stellt sich also die Frage, ob der Einsatz von Cryptoki auch für das TPM – das im Grunde auch als ein kryptographisches Token gesehen werden kann – sinnvoll ist. Fakt ist, dass es bereits eine PKCS#11-Implementierung für das TPM gibt. Das

3.5. Sicherheit von Trusted Computing

OpenCryptoki-Paket[76] liefert Treiber und Bibliotheken für eine Reihe von kryptographischer Hardware (hauptsächlich IBM) und seit einiger Zeit auch für das Trusted Platform Module. Die Bibliothek `TPM STDLL` ist dafür zuständig, sämtliche `PKCS#11`-Kommandos in TSS-Kommandos zu übersetzen. Tabelle 3.1 verdeutlicht am Beispiel der Hash-Operation, dass dies in einigen Fällen geradeaus umsetzbar ist. Aus Entwicklersicht ist die Verwendung von Cryptoki durchaus

Tabelle 3.1.: Vergleich zwischen `PKCS#11`- und TPM-Kommandos

<code>PKCS#11</code>	TPM	Funktion
<code>C_DigestInit</code>	<code>TPM_SHA1Start</code>	Startet eine Hashing-Operation
<code>C_DigestUpdate</code>	<code>TPM_SHA1Update</code>	Führt die Berechnung mit weiteren Eingabedaten fort
<code>C_DigestFinal</code>	<code>TPM_SHA1Complete</code>	Beendet die Berechnung

eine Alternative zum TSS, wenn sensible Daten vom TPM gesichert werden sollen. Jedoch ist anzumerken, dass viele TPM-spezifische Eigenschaften, wie bspw. Sealing über Cryptoki nicht nutzbar sind. Diese Vorgehensweise ist also nur für ganz einfache Szenarien sinnvoll, bietet aber den Vorteil, dass die Anwendungen portabel sind und auch mit anderen Token und Plattformen ohne TPM genutzt werden können. Für komplexere, TCG-unterstützte Fälle ist eine TSS-API die bessere Wahl.

3.5. Sicherheit von Trusted Computing

Trusted Computing soll die Sicherheit von Computersystemen erhöhen. Dies kann nur dann funktionieren, wenn die vorgestellten und umgesetzten Mechanismen – ob Hard- oder Software – selbst sicher genug sind. In diesem Abschnitt sollen einige Teilaspekte von TC auf den Grad der Sicherheit untersucht werden.

TPM-Sicherheit und Angriffe

Proof of Physical Presence. Das TPM und die darin enthaltenen Schlüssel bilden einen wichtigen Teil der Vertrauensanker. Viele der TPM-Funktionen können über die entsprechende API genutzt werden. Es gibt jedoch einige Operationen, die besonders sicherheitskritisch sind. Eine Steuerung dieser Funktionen allein über Softwareschnittstellen wäre ein Risiko, da sich dann bspw. der Eigentümer der Plattform ändern ließe. Für solche Funktionen wird vorgesehen, dass die un-

3. *Trusted Computing*

mittelbare Anwesenheit des Nutzers gegeben sein muss. So lässt sich mit dem Kommando `TPM_ForceClear` das TPM ohne Eingabe eines Passwortes in den Auslieferungszustand versetzen. Wie die Beweiserbringung konkret implementiert wird, überlässt die TCG den Herstellern. Ein mögliches Szenario wäre das Drücken einer bestimmten Taste zu einem bestimmten Zeitpunkt beim Booten, wie es auch für das Starten des BIOS-Menüs weitverbreitet ist.

Angriffe. Es wurden bereits einige erfolgreiche Angriffe auf TPM-Systeme durchgeführt. Die *Reset Attacke* ermöglicht es das TPM während dem Bootprozess, also nachdem es sich bereits im *Operational*-Zustand befindet über den LPC-Bus zu resetten und die PCR mit beliebigen Daten zu füllen. Diese Daten können die vorher aufgezeichneten Messwerte einer vertrauenswürdigen Konfiguration beinhalten. Somit ist der Trusted Boot-Prozess ist ausgehebelt. Dieser Angriff wurde erfolgreich auf einem v1.1b TPM-System durchgeführt. Eine weitere Attacke basiert auf dem oben vorgestellten *time-of-measurement / time-of-attestation*-Problem. Sie ermöglicht das Manipulieren von Programmcode, nachdem er bereits ausgeführt wurde, also nachdem er möglicherweise erfolgreich auf Integrität überprüft wurde [78]. Ein Angriff, der weniger TPM-spezifisch ist, ist die *Cold Boot Attacke*, bei der geheimes Schlüsselmaterial aus dem RAM gelesen werden kann, selbst wenn das System ausgeschaltet wurde [38]. Die TCG reagierte auf diese Angriffe mit der Veröffentlichung einer Spezifikation, die Anforderungen an TPM-Systeme beschreiben, die weitere Angriffe dieser Art vorbeugen sollen. [35]. In [13] wird eine *Offline Dictionary*-Attacke auf Authentifizierungsdaten von TPM-Objekten beschrieben, die es ermöglicht Daten zu entschlüsseln oder Schlüssel zu migrieren, ohne dass die dafür notwendigen Geheimnisse bekannte sind. Wenn auch die praktische Anwendbarkeit der Angriffe nur bedingt gegeben ist, so wird dennoch klar, dass es auf ein lückenloses Design auch bei Trusted Platform Modulen kommt, wenn dieses als ein Vertrauensanker dienen soll.

Designfehler. Die Verlagerung von Kryptographie und Schutzmechanismen auf Hardwarebausteine löst zwar einige Probleme von softwarebasierten Systemen, es ergeben sich jedoch neue Hürden. Wie bereits in Abschnitt 2.2.3 erläutert, müssen Implementierungen von kryptographischen Algorithmen auf Hardwarebausteinen möglichst fehlerfrei sein, da eine nachträgliche Behebung der Mängel im Prinzip nicht möglich ist, nachdem Module bereits produziert wurden und im Einsatz sind.

3.5. Sicherheit von Trusted Computing

Dies erfordert eine ausführliche Analyse in der Design- und Implementierungsphase, um Fehler möglichst auszuschließen. Werden möglicherweise schwerwiegende Bugs in den Implementierungen der TPM erst nachträglich entdeckt, sind die Module vielleicht sogar wertlos. Manche Implementierungsdetails, die die TCG den Herstellern überlässt, führen zu unterschiedlichen Ausprägungen. Zudem können Abweichungen von TCG-Vorgaben zu kritischen Sicherheitslücken führen. Eine Strategie, um TCG-Konformität der verschiedenen Implementierungen zu überprüfen, wird in [72] diskutiert.

Probleme des Integrity Reportings. Das Vertrauen in die Integrität der Kommunikationspartner basiert in verteilten Umgebungen auf den mit AIKs signierten Berichten über bestimmte PCR-Inhalte. Abbildung 3.9 verdeutlicht ein Problem, das im Zusammenhang mit Remote Attestation existiert. Auf Anforderung von Rechner B führt Rechner A Messungen über bestimmte Komponenten, signiert die Inhalte der PCR, die um die neuen Daten erweitert wurden und sendet die Credentials zum Zeitpunkt T_0 an B . B validiert die empfangenen Daten und stellt fest, dass A sich in einem vertrauenswürdigen Zustand befindet. B sendet A eine Bestätigung, dass die Verbindung akzeptiert wird. Diese Bestätigung enthält eventuell auch geheime Daten, die an die vertrauenswürdige Plattform gerichtet sind. Diese Daten erhält A zum Zeitpunkt T_1 . Das Problem ist, dass B nicht weiß, was in der Zeit zwischen T_0 und T_1 passiert. So kann es sein, dass die entsprechenden Module und Programme bei A in dieser Zeit manipuliert wurden und das System sich dann in einem von B als nicht-vertrauenswürdig einzustufenden Zustand befindet. Dies bekommt B nicht mit und bestätigt den Verbindungswunsch, so dass die inzwischen nicht-vertrauenswürdigen Komponenten ausgeführt werden. Dasselbe Prinzip gilt auch für lokale Szenarien, in denen eine Komponente andere Module misst, um deren Vertrauenswürdigkeit zu überprüfen. Die Lücke zwischen Integritätsmessungen von Modulen und dem Zeitpunkt der entsprechenden Auswertungen stellt ein Problem dar.

3. Trusted Computing

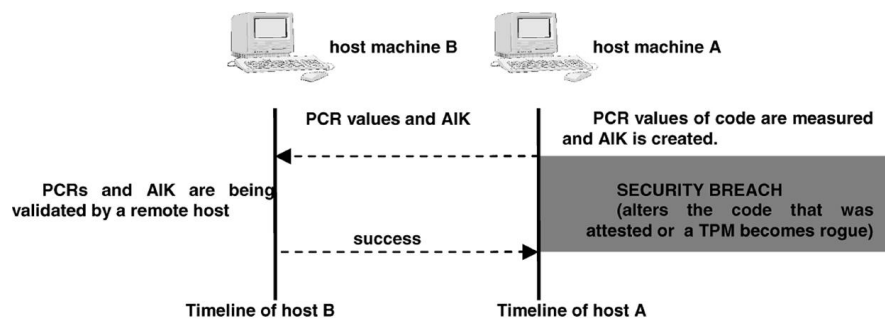


Abbildung 3.9.: *time-of-measurement - time-of-attestation* Problem [43]

3.6. Trusted Computing: Chancen, Risiken und Kritik

Die Konzepte der TCPA, bzw. TCG wurden von Anfang an von negativen Kritiken begleitet. Das hängt auch sicherlich damit zusammen, dass das Konsortium hauptsächlich aus Vertretern der Industrie besteht. Kritiker befürchteten, dass die Gestaltung von Computerplattformen in Zukunft nach deren Wünschen gehen würde, um bspw. Konzepte für Digital Rights Management durchzusetzen und zu erzwingen. Das Misstrauen ist auch mit der anfänglichen Unsicherheit über die Implementierungsdetails der TCG in Verbindung zu bringen, so wie Microsoft relativ wenige Details über die Palladium-Architektur bekannt gab. Diese Politik der Geheimhaltung führte dazu, dass Kritiken, wie von Ross Anderson in seiner vielzitierten FAQ über TCPA, die öffentliche Meinung über TCPA/TCG-Konzepte bestimmten[1]. Auch wenn nicht unbedingt weithergeholt, sind viele dieser Kritikpunkte nicht hundertprozentig vertretbar und basieren teilweise nur auf Annahmen und Vermutungen. Angebracht ist folgendes Zitat aus einer Veröffentlichung von William A. Arbaugh[2]:

“But rather than throw stones at something that might actually help improve security, let’s see if we can keep the ‘good’ and lose the ‘bad’.”

Wie in diesem Kapitel deutlich wurde, lassen sich die unterschiedlichsten Szenarien entwickeln, die tatsächlich mehrseitige Sicherheit verbessern können. So ist bspw. die Möglichkeit der Realisierung eines Trusted Storage eine ideale Basis, um effektivere und stabilere PETs zu realisieren. Daher liegt es auch in der Hand der

3.6. Trusted Computing: Chancen, Risiken und Kritik

Kritiker an der Entwicklung von offenen Plattformen teilzunehmen, um die Richtung, die Trusted Computing gehen soll, zu bestimmen. Die inzwischen zahlreichen Projekte auf offener Basis werden dazu beitragen den Ruf von TC-Technologien in Zukunft zu verbessern und sie objektiver zu beurteilen.

3. *Trusted Computing*

4. Trusted Webbrowser

Ungeachtet der Kritiken, denen Trusted Computing ausgesetzt ist, bietet diese Technologie viele neue Möglichkeiten für Umsetzung von multilateraler Sicherheit. Durch die relativ günstigen Herstellungskosten und die Partizipation von wichtigen Unternehmen der Computerindustrie könnten hardwarebasierte Sicherheitsmodule in naher Zukunft zur Standardaustattung von Computern gehören. Dadurch können bestehende Sicherheitsmechanismen um einiges verbessert werden. In diesem Kapitel wird eine Architektur für das Credential Management bei Webbrowsern vorgestellt, die durch den Einsatz von Trusted Computing und Trusted Storage die Privatsphäre erhöht.

4.1. Sicherheitsanforderungen

Charakteristisch für die jüngste Webbrowser-Generation sind die vielfältigen Erneuerungen rund um den Schutz der Privatsphäre. Features wie InPrivate-, Inkognito- und Private Browsing tun dies hauptsächlich dadurch, dass nutzerbezogene Daten erst gar nicht dauerhaft auf der Festplatte gespeichert werden. Dies ist sicherlich aus Sicht der Privatsphäre ein Gewinn, jedoch werden dadurch viele nützliche Funktionalitäten des Browsers “entschärft”. Im Unterschied zu solchen Ansätzen sind Folgende Ziele für diese Arbeit definiert:

- **TPM-basierter Schutz für User Credentials.** Jede Klasse von Browserdaten erfüllen einen bestimmten Zweck. Eine Lösung sollte daher erhöhten Schutz für die Daten bieten, ohne sie gänzlich zu entfernen.
- **Isolierte Speicherbereiche.** In Mehrbenutzer-Systemen sollte darauf geachtet werden, dass Daten aus unterschiedlichen Benutzerkonten auch sicher

4. *Trusted Webbrowser*

voneinander getrennt sind. Daten eines bestimmten Nutzers sollten nur von diesem nutzbar sein.

- **Flexible Sicherheitslevel.** Jeder Nutzer sollte die Möglichkeit haben die Schutzmaßnahmen individuell für die unterschiedlichen Credentials anzupassen.
- **Konzepte für Import/Export und Backup/Re-Storage.** Es muss gewährleistet sein, dass ein Nutzer seine Credentials bei Bedarf auf einen anderen Rechner zu migrieren. Schutz der Daten muss auch während dem Transport gewährleistet sein.

Weiterhin sollte die Lösung einfach in bestehende Systeme integrierbar sein und keine Großen Schwierigkeiten bei der Nutzung erzeugen.

4.2. Lösungsansatz

Als eine der wichtigsten Schwachstellen der Schutzmechanismen gängiger Webbrowser wurde der Einsatz von softwarebasierten Sicherheitsmodulen festgehalten. Auch wenn nicht vorgeschrieben, liegt der Vorteil von Standards wie PKCS#11 aber im vereinfachten Zugriff auf *hardwarebasierte* Token. Webbrowser sind nicht die einzigen Anwendungen, die diesen Weg gehen. Auch Programme wie VPN-Clients, E-Mail-Clients oder Office-Anwendungen setzen softwarebasierte Module ein, um kryptographische Schlüssel zu speichern. Somit weisen sie alle auch die gleichen Schwächen auf, die in Kapitel 2 vorgestellt wurden. Ausgehend von dieser Feststellung, werden die in den nächsten Abschnitten vorgestellten Ansätze dermaßen allgemein gehalten, dass auch andere Anwendungen von ihnen profitieren können. Die Lösung besteht daher im Prinzip aus zwei Teilen. Der *Trusted Key and Credential Management Service (TKCMS)* ist ein zentraler Dienst, der eine Reihe von Funktionalitäten rund um TPM-Schlüssel- und Credential-Management über eine Schnittstelle anbietet. Hauptsächlich übernimmt TKCMS die Verwaltung der TPM-Schlüssel und Credentials von Anwendungen. TKCMS ist im Prinzip der Klasse wallet-basierter Architekturen zuzuordnen. Der zweite Teil der Lösung zeigt auf, wie ein Webbrowser die Credentials von Nutzern durch Nutzung der TKCMS-Dienste schützen kann.

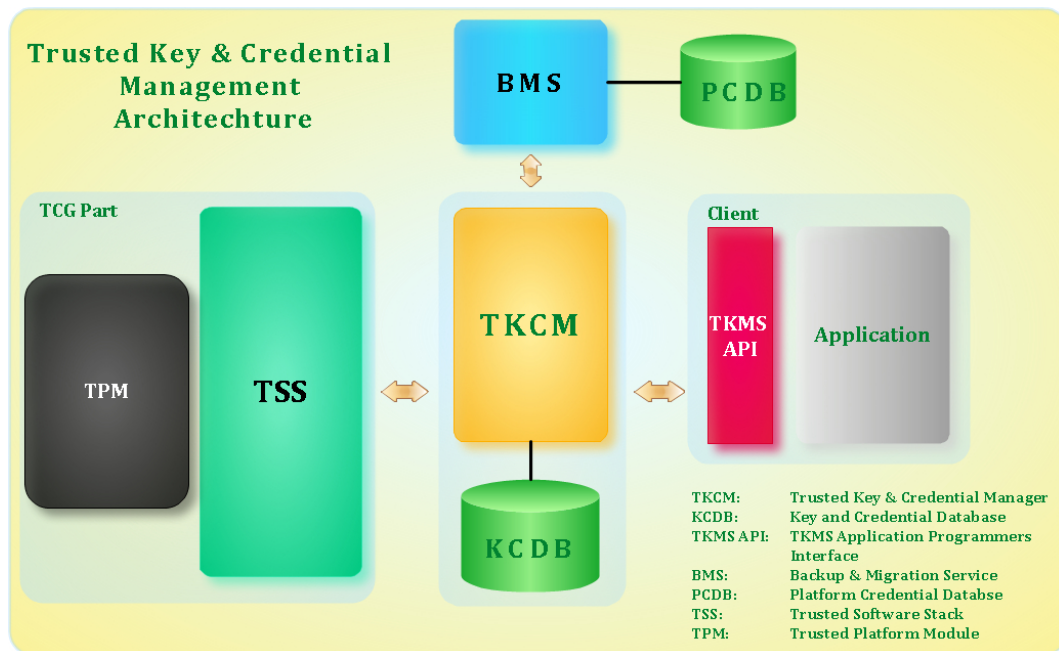


Abbildung 4.1.: TKCMS Architektur

4.3. Trusted Key and Credential Management

Der Trusted Key und Credential Management Service bildet die Schnittstelle zwischen dem TCG Software Stack und Anwendungen, die vom Trusted Storage Gebrauch machen möchten. Sämtliche Anwendungen, die den Dienst in Anspruch nehmen, bekommen einen eigenen Speicherbereich zugewiesen, in dem sämtliche zu schützende Daten und Schlüssel abgelegt werden (*E-Wallet*). Die Verwaltung dieser Speicherbereiche ist Aufgabe des Dienstes. Den Anwendungen werden als Clients die Funktionalitäten über eine API zur Verfügung gestellt. Daten, die in diesem Speicher abgelegt werden, sind mit TPM-Schlüsseln chiffriert. Somit unterscheidet sich TKCMS grundlegend von anderen wallet-basierten Systemen, da hier die Daten jederzeit im Schutz des TPM – einem Hardwarebaustein – stehen. Abbildung 4.1 skizziert die Komponenten der TKCMS-Architektur, die im folgenden Abschnitt vorgestellt werden.

4.3.1. TKCMS Komponenten

Trusted Key and Credential Manager. Der *Trusted Key and Credential Manager (TKCM)* ist die zentrale Komponente der Architektur. Sämtliche Anfragen werden hier verarbeitet und ausgeführt. Je nach Operation beruft sich TKCM auf weitere Komponenten. Seine Funktionalitäten werden über die TSS API implementiert. Anfragen von Clients werden über eine Netzwerkschnittstelle entgegen genommen. Zudem verwaltet er die Schlüssel- und Credential-Datenbank, in der das gesamte Datenmaterial abgelegt wird.

Key and Credential Database. In der *Key and Credential Database (KCDB)* wird das Schlüssel- und Datenmaterial abgelegt. Jede registrierte Anwendung bekommt einen Speicherbereich in dieser Datenbank zugewiesen. Dieser Speicherbereich ist nochmals in Schlüssel- und Datenspeicher unterteilt. In dieser Datenbank existiert für jeden Schlüssel und für sämtliche Daten ein *Handle*. Handles sind Dateien, die solche Informationen enthalten, die eine eindeutige Identifizierung der Schlüssel und Daten erlauben.

Backup and Migration Service. Sämtliche Operationen, die mit Export/Import oder Backup/Re-Storage zu tun haben, werden vom *Backup and Migration Service (BMS)* durchgeführt. Dazu gehören Aufgaben, wie z.B. das Erzeugen von *Migration Packets (MP)*, eine spezielle Datenstruktur, die zu transportierende Schlüssel und Daten beinhaltet. Dazu ist BMS neben TKCM die einzige Komponente, die auf den TSS zugreift. Diese Komponente ist als Dienstleister für TKCMS zu betrachten und ist für die Clients nicht sichtbar.

Platform Credential Database. Die Migration von TPM-Schlüsseln erfordert, dass entsprechende Informationen (bspw. öffentliche Schlüssel) der Zielplattform verfügbar sind. Diese Informationen werden vom BMS benötigt, um Migration Packets zu erzeugen. Daher verwaltet BMS die *Platform Credential Database (PCDB)*, in der solche Informationen in Form von *Platform Credentials (PC)* abgelegt werden. Über TKCM kann die PCDB jederzeit erweitert werden.

TKCMS API. Die *TKCMS API* implementiert sämtliche Funktionalitäten, die nötig sind, um mit dem Trusted Key Server zu kommunizieren. Realisiert als Bibliothek der spezifischen Programmiersprachen kann die API in Anwendungen

integriert werden. Die Nutzung einer solchen API ist optional, jedoch empfehlenswert, da sämtliche Kommunikationsprotokolle bereits implementiert sind.

4.3.2. Schlüsselhierarchie und Trust Domains

In Kapitel 3 wurde das Prinzip der Schlüsselverwaltung von TPM-Schlüsseln vorgestellt. Das Schlüsselmanagement des Trusted Key and Credential Management Systems basiert auf der TPM-Schlüsselhierarchie. Abbildung 4.2 verdeutlicht, wie diese Hierarchie vom TKCMS erweitert wird. Auf der ersten Ebene, als unmittelbarer Nachfolger des Storage Root Keys befindet sich der *TKCMS Root Key (TRK)*. Dieser Storage Key bildet die Wurzel der gesamten TKCMS-Schlüsselhierarchie und erzeugt dadurch eine eigene Trust Domain für den Dienst. Die nächste Hierarchiestufe bilden Subdomains für jeden Nutzer. Diese werden durch die *User Storage Root Keys (USRK)* aufgespannt. Die letzte Unterteilung wird durch die *Application Storage Keys (ASK)* erzwungen. Diese Schlüssel werden jeweils für eine Anwendung erzeugt, die sich beim TKCMS registrieren. Den weiteren Aufbau der Schlüsselhierarchie unterhalb des ASK können die Anwendungen selbst gestalten. Sie können beliebig weitere Storage oder Binding Keys anfordern und sie an einer beliebigen Stelle positionieren. Diese Hierarchie erlaubt eine saubere Trennung der Schlüssel und Daten von unterschiedlichen Anwendungen und Nutzern.

4.3.3. TKCMS Funktionsumfang

Die Funktionalitäten vom TKCMS sind allgemein gehalten, so dass die genauen Umsetzungsszenarien den Anwendungen überlassen sind.

Registrierung

Voraussetzung für Inanspruchnahme der TKCMS-Dienste durch eine Anwendung ist dessen Registrierung. Dies ist notwendig, da zunächst einige Schritte durchgeführt werden müssen, um die neue Trust Domain zu initialisieren. Das beinhaltet die Erzeugung des ARK, des Speicherbereichs und des Handles für den Zugriff auf den Speicherbereich.

4. Trusted Webbrowser

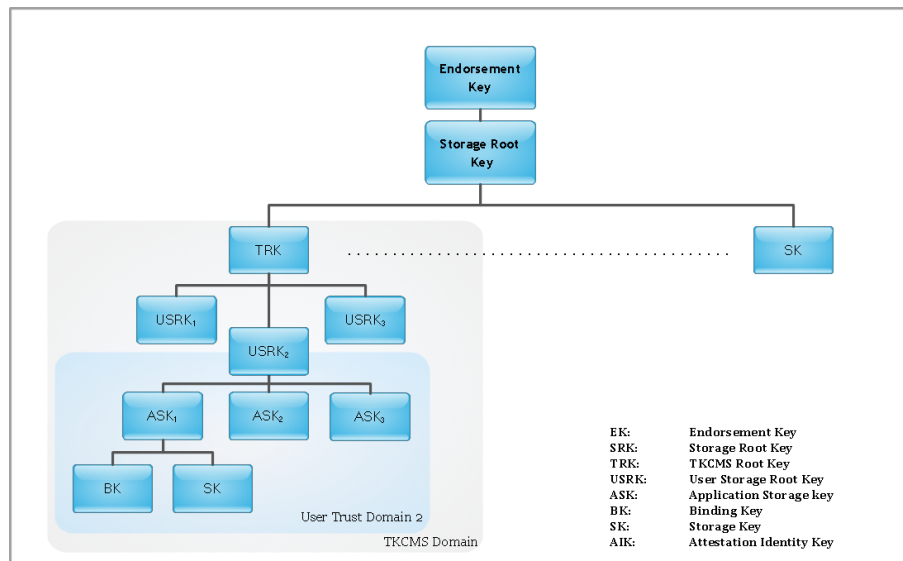


Abbildung 4.2.: Erweiterte TPM-Schlüsselhierarchie

Schlüsselerzeugung

Die Gestaltung der Schlüsselhierarchie unterhalb des ARK ist der Anwendung selbst überlassen. Der TKCMS bietet daher die Möglichkeit, beliebig viele neue Schlüssel erzeugen zu lassen. Als Schlüsseltypen werden sowohl Storage- als auch Binding Keys unterstützt. Dadurch haben Anwendungen je nach Bedürfnis die Möglichkeit die Domain beliebig weiter zu unterteilen, bspw. um unterschiedliche Speicherbereiche für Daten aus unterschiedlichen Profilen desselben Nutzers zu erzeugen. Binding Keys bilden die Blätter des Schlüsselbaumes. Sie können nur verwendet werden, um Daten zu verschlüsseln. Dies schließt jedoch nicht aus, dass mit ihnen "fremde" symmetrische Schlüssel verschlüsselt werden können. Fordert eine Anwendung einen neuen Schlüssel an, so muss sie den Schlüssel nennen, unter dem der neue Schlüssel in der Hierarchie eingetragen werden soll (wrapping key). Dabei muss es sich um einen Storage Key handeln. Nach erfolgreicher Erzeugung erhält die Anwendung ein Handle für den neuen Schlüssel. Dieser Handle dient nur zu Identifikationszwecken und nicht als Authentifizierungsmittel.

Ver- und Entschlüsselung von Daten

Hauptaufgabe des Credential Managers ist die Verschlüsselung und Speicherung von sensiblen Daten. Registrierte Clients können sich hierfür Binding Keys

4.3. Trusted Key and Credential Management

erzeugen lassen, die sie bei Verschlüsselungsanfragen referenzieren müssen. Dazu legen sie den entsprechenden Key Handle zusammen mit den Daten, die verschlüsselt werden sollen vor. Wird auch das korrekte Passwort genannt, so führt der TKCMS ein Binding durch, indem die entsprechenden Funktionen des TSS aufgerufen werden. Die verschlüsselten Daten werden im Datenspeicher des entsprechenden Clients abgelegt. Ähnlich wie für Schlüssel, wird auch ein Handle für die verschlüsselten Daten erzeugt, das dem Client nach erfolgreicher Verschlüsselung weitergereicht wird. Dieses Handle legt der Client zusammen mit dem Schlüsselhandle und dem richtigen Passwort vor, wenn die Daten wieder entschlüsselt werden sollen.

Export und Backup

An TPM gebundene Daten und Schlüssel sind nur auf der einen Plattform nutzbar. Jedoch müssen auch Szenarien unterstützt werden, in denen ein Anwender seine Daten auf anderen Rechnern nutzen möchte. Beispielsweise möchte ein Nutzer seine Credentials auf eine andere Plattform transportieren und dort verwenden. Für diese Zwecke sieht die TCG vor, dass TPM-Schlüssel als *migrierbar* definiert werden können. Diese Möglichkeit bietet auch TKCMS bei der Erzeugung von neuen Schlüsseln an. Beispielsweise kann ein Webbrowser den Binding Key B_1 , der für die Verschlüsselung von bestimmten User Credentials benutzt wird als migrierbar definieren, um die Möglichkeit zu haben, den gesamten Teilast des Schlüsselbaumes, beginnend bei B_1 als Wurzel rauszulösen. Der Teilast kann dann dem Schlüsselbaum einer anderen Plattform hinzugefügt werden. Zu den Schlüsseln können auch verschlüsselte Daten hinzugefügt werden, die der Nutzer auf der zweiten Plattform nutzen möchte. BMS erzeugt für diese Zwecke Migration Packets (MP), die sämtliche zu transportierende Daten und Schlüssel enthalten. Damit diese auch während dem Transport geschützt sind, wird ein MP mit dem öffentlichen Schlüssel¹ der Zielplattform verschlüsselt, bevor es die erste Plattform verlässt. Somit kann das Paket nur auf der Zielplattform gelesen werden. Die TCG definiert für den TSS diverse Funktionen, die eine Migration unterstützen. Ein genaues Protokoll für Migration ist von der TCG bisher nicht standardisiert worden.

¹Gemeint ist ein beliebiger Schlüssel, den die Zielplattform als sogenannte *Migration Authority (MA)* definiert hat. Solche MAs dienen als Ticket für eine Migration von Schlüsseln auf die Plattform

4. Trusted Webbrowser

Es existiert lediglich eine Richtlinie, wie solche Protokolle aussehen könnten, u.a. auch über Web Services [33]. Backups sind als Spezialfall der Migration zu betrachten. Die Zielplattform ist hierbei stets dieselbe Plattform. Ein MP wird also mit einem Schlüssel aus der eigenen Schlüsselhierarchie ummantelt.

Admin-Funktionen

Weiterhin gibt es eine Reihe von verwaltungstechnischen Funktionen. Dazu gehören das Löschen von Schlüsseln, ändern von Passwörtern für Schlüssel oder die Abmeldung vom Dienst, was zur Folge hat, dass sämtliche Schlüssel und Daten des entsprechenden Clients gelöscht, und aus dem persistenten Speicher des TSS entfernt werden.

4.4. Trusted Browser

Die Dienste des TKCMS bilden eine ideale Grundlage, um Browser Credentials TPM-basiert zu schützen. Ein Browser, der diese Dienste nutzen möchte, registriert sich beim TKCMS und lässt sich mindestens einen Binding Key erzeugen, mit dem die Credentials geschützt werden. Somit kann bspw. die softwarebasierte Cryptoki-Lösung umgangen werden. Eine Entschlüsselung der Credentials findet immer nur bei Verwendung statt.

4.4.1. Credential Verwaltung

Für eine geeignete Umsetzung muss zwischen zwei Klassen von Credentials unterschieden werden.

- *On-Demand-Credentials* sind Daten, die beim Eintreten von bestimmten Ereignissen benötigt werden. Dazu gehören bspw. Cookies, die nur dann genutzt werden, wenn die Webseite, für die der Cookie bestimmt ist, geladen werden soll. Weiterhin gehören Login Credentials zu dieser Klasse, da sie auch nur eingesetzt werden, wenn die Seite mit dem entsprechenden Login-Formular geladen wird. Was diese Klasse von Credentials ausmacht, ist dass zu jedem Zeitpunkt exakt bestimmt werden kann, welche der potenziell vielen Daten benötigt werden. Wenn z.B. Webseite *www.ineedacookie.com* auf-

gerufen wird, so reicht es aus, nur die Cookies von *ineedacookie.com* aus dem Speicher zu holen. Weiterhin können On-Demand-Credentials unmittelbar nach Verwendung wieder in den geschützten Speicher abgelegt werden.

- *On-Runtime-Credentials* müssen dagegen jederzeit verfügbar sein, sobald der Browser gestartet wird. Beispielsweise muss ein Nutzer zu jedem Zeitpunkt die Möglichkeit haben seine Bookmarks einzusehen. Der Zeitpunkt der Nutzung hängt – anders als bei On-Demand-Credentials – nicht von einem bestimmten Ereignis ab. Zu dem ist keine Unterscheidung zwischen unterschiedlichen Bookmarks bestimmbar, so dass sie als Ganzes betrachtet und geladen werden müssen.

Aufgrund ihren unterschiedlichen Eigenschaften bietet es sich an die beiden Klassen auch unterschiedlich zu behandeln. On-Demand-Cookies sollten idealerweise immer nur entschlüsselt werden, wenn sie benötigt werden. So lange dieser Fall nicht eintritt, befinden sie sich in verschlüsselter Form in der Credential Datenbank. Nach Verwendung sollten sie auch gleich wieder dort landen. Im Gegensatz dazu werden On-Runtime-Credentials gleich nach Starten des Browsers als Ganzes aus dem geschützten Bereich geholt und dem Browser zur Laufzeit verfügbar gemacht. Sobald der Browser geschlossen wird, werden sie wieder in die KCDB geladen.

4.4.2. Schlüsselverwaltung

Die Möglichkeit, die eigene Trust Domain zu gestalten sollte aus Sicherheitsgründen genutzt werden. Webbrowser sind Multi-User-Anwendungen auf Multi-User-Plattformen. Credentials von verschiedenen Nutzern sollten voneinander getrennt geschützt werden. Dazu wird beim Trusted Browser die Möglichkeit genutzt, die Trust Domain des Browsers weiter in Trust Subdomains zu unterteilen. Beispielsweise kann so für jeden einzelnen Nutzer oder für jedes Profil eines Nutzers ein neuer Storage Key erzeugt werden, der die Wurzel der Subdomain bildet. Weiterhin besteht die Option, unterschiedliche Credentials mit unterschiedlichen Binding Keys zu schützen. Sinn macht das Beispielsweise, um weniger sensible Credentials, wie die Historie oder Lesezeichen von sensibleren Daten, wie Zertifikate oder Login-Credentials zu trennen, um letztere evtl. auf ein höheres Sicher-

4. Trusted Webbrowser

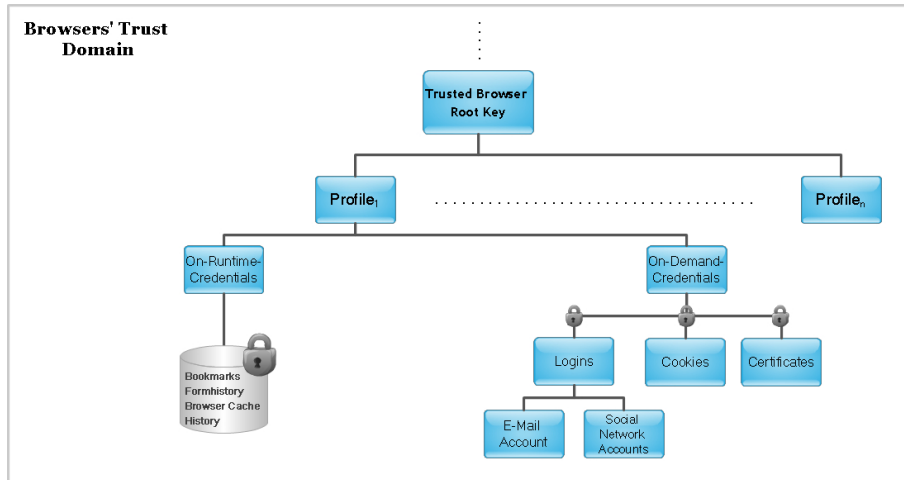


Abbildung 4.3.: Mögliche Unterteilung der Trust Domain eines Webbrowsers

heitsniveau zu stufen. Um maximale Flexibilität zu gewährleisten, ist dem Nutzer die Möglichkeit gegeben, zu entscheiden, in wie fern bestimmte Credentials mit eigenen Schlüsseln gesichert werden sollen. Dies geht so weit, dass der Nutzer die Möglichkeit hat neue Schlüssel zu erzeugen, die er für beliebige Zwecke einsetzen kann. Dies ermöglicht eine Feinjustierung des Sicherheitslevels, je nach Belieben (Abbildung 4.3).

5. Implementierung

Die vorgestellten Lösungsansätze wurden teilweise als *Proof-of-Concept* realisiert. Im Vordergrund steht dabei die praktische Machbarkeit und Nutzbarkeit der vorgestellten Architektur. Eine wichtige Herausforderung ist dabei, dass sich die Lösungen in bestehende Umgebungen integrieren lassen und die Gewohnheiten der Nutzer berücksichtigt werden. Die Implementierung besteht aus den in Kapitel 4 vorgestellten Teilen. Die Umsetzung des Trusted Key und Credential Managers erfolgt unter Nutzung des jTSS-Paketes. Der Trusted Browser wird als Erweiterung für Mozilla Firefox entwickelt.

5.1. Entwicklungsumgebung

Virtualisierte Linux TC-Umgebung. Neben dieser Arbeit wurden am Fraunhofer SIT eine Reihe weiterer Diplomarbeiten rund um Anwendungsszenarien von Trusted Computing durchgeführt. Der TPM-Emulator eignet sich besonders für Test- und Implementierungszwecke. Um trotz emuliertem TPM eine möglichst authentische Trusted Computing-Umgebung zu schaffen, entwickelten A. Brett und A. Leicher eine virtualisierte TC-Plattform als eine gemeinsame Entwicklungsumgebung. Diese Lösung besteht aus einer Linux-Distribution mit TPM Emulator und ein darauf aufbauendes virtuelles, minimales Linux, das mit der für TPM-Unterstützung gepatchten Virtualisierungssoftware QEMU[5] gestartet wird. Der Kern des minimalen Linux wurde zudem noch für IMA-Unterstützung neu kompiliert. Dieses virtuelle Linux diente während der ganzen Entwicklungszeit als Ersatz für eine vollwertige TC-Plattform. Zudem wurde ein Framework auf Basis von jTSS entwickelt, das die Nutzung von grundlegenden Funktionalitäten vereinfachen soll [11]. Diese Plattform diente als Grundlage für die Entwicklung des Trusted Key and Credential Management Systems.

5.2. Trusted Key and Credential Manager

Im Folgenden werden die Implementierungsdetails des Key Managers vorgestellt. Dies beinhaltet die implementierten Funktionalitäten, die Datenbankstruktur und auch das eingesetzte Kommunikationsprotokoll.

5.2.1. Implementierungsziele

Ziel der Implementierung ist die Umsetzung des Trusted Key und Credential Management Services in Java durch Nutzung der jTSS-Bibliotheken. Hierbei sollen Funktionen wie Registrierung von Clients, Schlüsselerzeugung, Verschlüsselung, etc. umgesetzt, und über Socket-basierte Kommunikation zur Verfügung gestellt werden. Der Dienst soll sowohl vom lokalen, als auch von entfernten Rechnern in Anspruch genommen werden können. Weiterhin soll ein einfaches Kommunikationsprotokoll definiert werden, das Clients die Inanspruchnahme der Dienste ermöglicht. Hierzu wird ein Satz von Anfragen anhand eines simplen XML Schemas definiert. Die Kommunikation ist XML-basiert, was eine einfache Bearbeitung der Anfragen und Antworten ermöglicht. Die Lösung soll nahtlos in bestehende Systeme integrierbar sein und Benutzergewohnheiten berücksichtigen. Die Benutzungsschnittstelle zur Konfiguration soll einfach gehalten werden. Die Integrität der Serverdaten und der Credential Datenbank muss gewährleistet sein. Nur autorisierte Clients dürfen Zugriff auf die Credentials haben. Trust-Domains sollen sicher voneinander abgekapselt werden. Wo es möglich ist, sollen TPM-Funktionalitäten eingesetzt werden.

5.2.2. TKCM und Credential Database

Den Hauptteil der Implementierung bildet der Trusted Key und Credential Manager. Dieser Dienst implementiert sämtliche Funktionalitäten und bietet diese über eine Socket-Schnittstelle an. Er verwaltet außerdem die KCDB. Wird der Dienst zum ersten Mal gestartet, werden zunächst einige Initialisierungsschritte durchgeführt. Dazu gehören die Überprüfung der TPM-Konnektivität, die Erzeugung der KCDB und die Generierung des TRK. Wurden sämtliche Schritte erfolgreich durchgeführt, ist der TKCMS betriebsbereit und das Serversocket wird

geöffnet. Das Handle wird zusammen mit weiteren Konfigurationsdaten im Server-Verzeichnis abgelegt.

5.2.3. Key und Datahandles

Die Arbeit mit *Security Objects (SO)*, (TPM-Schlüssel oder Daten) erfolgt anhand von Handles. Sie enthalten nicht die eigentlichen Daten (z.B. Key Blobs), sondern nur die Informationen, die nötig sind, um das entsprechende Objekt zu identifizieren. Ob Schlüssel oder Daten, Security Objects besitzen gemeinsame Eigenschaften und zusätzlich objektspezifische Attribute. Die Handles werden immer nur vom TKCMS erzeugt und an den Client weitergereicht. Der Besitz des Handles alleine reicht dabei nicht aus. Jedes SO ist mit einem Passwort verbunden, das vom Client vorgewiesen werden muss, um auf dem Objekt operieren zu können. Die Handles sind also keineswegs als Authentifizierungstoken zu sehen. Realisiert werden sie in Form von XML Dokumenten. XMLs lassen sich leicht parsen, modifizieren und sind in den gängigsten Bibliotheken vorhanden. Die Organisation des Speichers entspricht dem Aufbau der Trust Domains. Die Verzeichnisstruktur sieht zunächst eine Unterteilung in jeweils ein Verzeichnis pro Client vor. Diese Clientverzeichnisse sind weiter in *keys*-Verzeichnis und *data*-Verzeichnis unterteilt. Diese beiden Verzeichnisse enthalten die entsprechenden Handles der Clients. Die Wahl dieser Verzeichnisstruktur hat organisatorische Gründe. Der Server kann die entsprechenden Handles leicht identifizieren und laden. Die Informationen, die er dafür benötigt sind lediglich die jeweiligen UUIDs. Listing 5.1 zeigt ein Beispiel für ein SO-Handle. Die Bedeutung der einzelnen Tags sind im folgenden erläutert.

Listing 5.1 Ein Key Handle

```
<SecurityObject>
  <Uuid>23-234-2342352345-342523</Uuid>
  <FriendlyName>TestKey</FriendlyName>
  <ownerId>1234-567-891011-34567</ownerId>
  <status>Active</status>
  <timestamp>1240073027</timestamp>
  <policy>
    <ttl>0</ttl>
    <migratable>1</migratable>
  </policy>
</SecurityObject>
```

5. Implementierung

- **Uuid.** Dies ist der *Universally Unique Identifier*, der das Security Object eindeutig identifiziert. Bei Schlüsseln ist es die UUID, mit der der Schlüssel im persistenten Speicher des TPM registriert wurde. Ohne Kenntnis dieser ID können SO nicht identifiziert werden.
- **ownerId.** Handelt es sich um ein Key- oder Datahandle, bezeichnet die ownerId die UUID des Clients, dem das Objekt gehört.
- **FriendlyName.** Dies ist ein Bezeichner für das Objekt. Er ist nicht eindeutig und wird nur für Client-interne Zwecke verwendet.
- **Status.** Beschreibt den Status, in dem sich das Objekt befindet. Beispielsweise kann ein Schlüssel aktiv oder inaktiv sein, womit feststellbar ist, ob das Objekt benutzt werden darf oder vom Besitzer gesperrt wurde.
- **timestamp.** Ein Zeitstempel, der während der Erzeugung des Objektes erstellt wird. Dient bei Schlüsseln bspw. dazu, den Lebenszyklus des Schlüssels zu verwalten. So kann bspw. über eine Policy bestimmt werden, dass ein Schlüssel nur über einen bestimmten Zeitraum benutzt werden darf.
- **Policy.** Jedem Security Object ist eine Policy zugewiesen. In der Policy können bestimmte Nutzungsrestriktionen festgehalten werden. Beispielsweise kann angegeben werden, dass ein Schlüssel nur zu bestimmten Zwecken oder für eine bestimmte Anzahl an Operationen benutzt werden darf. Zusätzlich ist für Schlüssel festgehalten, ob ein Schlüssel migrierbar ist.

5.2.4. Kommunikationsprotokoll

Der Server stellt seine Funktionalitäten über eine Socketschnittstelle zur Verfügung. Dies hat den Vorteil, dass der Server auch von entfernten Plattformen genutzt werden kann. Dies ermöglicht bspw. ein Szenario, bei dem TKCMS im Firmennetzwerk als zentrale Credentialverwaltung dient. Clients können sich über Netzwerkprotokolle mit dem Server verbinden und die Dienste in Anspruch nehmen. Realisiert ist die Schnittstelle über ein einfaches, XML-basiertes Protokollschema. Sowohl Anfragen, als auch Serverantworten sind somit einfach zu parsen. Die XML-Anfragen und -Antworten enthalten jeweils eine ID, über die die Anfrage identifiziert werden kann und sämtliche Parameter. Jede vom Server bereitgestellte

Funktion besitzt eine eindeutige Kennziffer, die in Anfragen als Parameter genannt werden muss. Im Wesentlichen besteht eine Anfrage aus dem `TkmsRequest`-Tag, das die Wurzel jeder Anfrage darstellt. Dieser Tag besitzt das Attribut `id`-Attribut, das die Anfrage identifiziert. Als Nachfolgerknoten muss ein `Parameters`-Tag existieren. Unter diesem Tag sind alle Anfrageparameter in `Parameter`-Tags enthalten.

5.2.5. Kommandos

TkmsRegister. Dies ist die Anfrage, mit der sich ein Client beim Server registrieren kann. Listing 5.2 zeigt die Struktur einer Registrierungs-Anfrage. Der Client muss einen Namen angeben, der als Grundlage für die Benennung des Client-Verzeichnisses dient. Der zweite Parameter ist ein Passwort für den Storage Key, der für die Anwendung erzeugt wird.

Listing 5.2 Ein Registration Request

```
<TkmsRequest id='1'>
  <Parameters>
    <Parameter>Mozilla Firefox</Parameter>
    <Parameter>password</Parameter>
  </Parameters>
</TkmsRequest>
```

TkmsNewKey. Mit dieser Anfrage kann ein neuer Schlüssel angefordert werden. Dabei kann es sich um einen neuen Storage oder Binding Key handeln. Angegeben wird das durch einen Parameter. Weiterhin muss als Parameter angegeben werden, welcher Schlüssel als Wrapping Key benutzt werden soll. Weitere Parameter sind das Passwort für den Wrapping Key und ein Passwort für den neuen Schlüssel.

TkmsBind. Dies ist die Verschlüsselungs-Anfrage. Wichtige Parameter hierfür sind der Binding Key (Handle), das Passwort für den Binding Key und die Daten, die verschlüsselt werden sollen. Textbasierte Daten können direkt in die Anfrage integriert werden. Binäre Daten können dem Server als zusätzlicher Datenstrom nach der Anfrage-XML geschickt werden. Um welchen Typ von Daten es sich handelt, kann als weiterer Parameter in der Anfrage genannt werden. Den zu verschlüsselnden Daten kann ein Name zugewiesen werden. Dies kann für den Client hilfreich sein, um verschiedene Daten (bspw. verschiedene Credentials) voneinan-

5. Implementierung

der zu unterscheiden zu können. Nach erfolgreicher Verschlüsselung erzeugt der Server ein Handle, das unter anderem eine UUID für die Daten enthält.

TkmsUnbind. Das Gegenstück zum Binding. Ein zuvor erhaltenes Datenhandle muss der Client bei einer Unbind-Anfrage zusammen mit dem Binding Key und dem Passwort an den Server senden. Sind alle Daten korrekt, führt der Server das Unbinding durch und sendet die Daten an den Client. Wichtig hierbei ist, dass sämtliche Schlüssel auf dem Pfad vom SRK bis zum benutzten Binding Key geladen werden müssen. Der Key Manager geht hierzu alle Schlüssel ab dem Binding Key durch und lädt den jeweiligen Wrapping Key, bis der Storage Root Key erreicht wird.

TkmsUnregister. Mit der *TkmsUnregister*-Funktion kann sich ein Client vom TKCMS abmelden. Hierfür muss er das Passwort für den Storage Key der Anwendung nennen. Eine Abmeldung hat zur Folge, dass der Speicherbereich des Clients samt Inhalte gelöscht wird. Der Client kann optional in der Anfrage angeben, ob in der Datenbank evtl. noch vorhandene Daten an ihn geschickt werden sollen. Alle Schlüssel des Clients werden auch aus dem persistenten Speicher des TSS gelöscht.

TkmsListOfFunctions. Als Antwort auf diese Anfrage liefert der Server eine Liste der unterstützten Funktionen. Zu jeder unterstützten Anfrage ist die ID angegeben, mit der die sie angefordert werden kann.

TkmsChangeSecret. Clients können sämtliche Passwörter von Security Objects ändern. Dazu muss in der Anfrage das alte und das neue Passwort angegeben werden.

TkmsDestroyKey / TkmsDestroyData. Mit diesen beiden Operationen kann ein Client das löschen eines Schlüssels oder Daten anfordern. Der Server entfernt die Daten und unregistriert alle Schlüssel, die entfernt werden sollen.

5.3. TrustFox: Trusted Firefox Extension

In diesem Abschnitt wird erläutert, wie das im letzten Abschnitt eingeführte Trusted Browser-Modell realisiert wird. Es werden die eingesetzten Technologien und auch die Arbeitsweise der Lösung vorgestellt.

5.3.1. Implementierungsziele

TrustFox ist die Umsetzung des Trusted Browsers als Extension für Firefox. Die Möglichkeiten des Trusted Storages und der Trust-Domänen wird von TrustFox genutzt, um Browser Credentials sicher abzuspeichern. Weiterhin bietet TrustFox eine Konfigurationsoberfläche, über die Nutzer TC-Schutzmechanismen verwalten können.

5.3.2. Eingesetzte Techniken

Firefox Extension Architektur. Im Kapitel über Webbrowser wurde der Wandel von diesen verdeutlicht. Dabei wurde klar, dass sie sich zu einer Allzwecksoftware entwickelt haben, deren Funktionalitäten im Prinzip nicht begrenzt sind. Moderne Webbrowser sind daher modular aufgebaut, so dass die Basisfunktionalitäten um zahlreiche weitere erweitert werden können. Beispielsweise sind zahlreiche Add-Ons für Firefox verfügbar, die den Browser in Aspekten wie es Sicherheit, Verwaltung oder sonstiges erweitert. Die Architektur von Firefox sieht vor, dass die Extensions auf alle grundlegenden Funktionalitäten der Browserbase zugreifen können. So kann z.B. auf die interne Credentialverwaltung oder ähnliches zugegriffen werden. Sicherheitstechnische Restriktionen gibt es kaum, da Extensions implizit vom Nutzer installiert werden. Daher ist die Installation von Extensions kritisch [86]. Extensions sind als gebündeltes Zip-Paket verfügbar. In diesem Paket sind alle Skripte und Oberflächenelemente enthalten, die für die Extension notwendig sind. Viele Features aus einer Standardinstallation sind bereits als Extension realisiert. Die Oberflächenelemente werden mit der XML-basierten Beschreibungssprache *XML User Interface Language (XUL)* erzeugt. Um die Oberflächenelemente mit Logik zu erweitern, können wie auch bei HTML Skripte in XUL-Files eingebettet werden. JavaScript kann ohne Einschränkun-

5. Implementierung

```
helloworld/  
  chrome.manifest  
  install.rdf  
  content/  
    overlay.js  
    overlay.xul  
    hello.xul  
  locale/  
    en-US/  
      overlay.dtd  
      hello.dtd  
  skin/  
    overlay.css
```

Abbildung 5.1.: Struktur eines Firefox Extension-Pakets

gen eingesetzt werden. Über JavaScript kann innerhalb einer Extension zusätzlich auf Mozilla XPCOM-Komponenten zugegriffen werden. Diese ermöglichen den Zugriff auf Funktionalitäten der Browserbase, auf die allein mit JavaScript aus Sicherheitsgründen nicht zugegriffen werden kann. Abbildung 5.1 zeigt die Verzeichnisstruktur einer Firefox Extension. Die einzelnen Komponenten werden im folgenden Abschnitt erläutert.

- **install.rdf.** Dies ist das Installationsskript, das grundlegende Informationen über die Extension enthält. Beispielsweise enthält sie die eindeutige Extension-ID, die nötig ist, um die Extension zu identifizieren. Weitere Daten sind der Anzeigename der Extension, Name des Authors, die unterstützten Firefox Versionen, Homepage der Extension und Weiteres. Diese Datei muss im Paket enthalten sein.
- **chrome.manifest.** In dieser Datei wird angegeben, welches XUL-File die Hauptdatei der Extension ist und an welcher Stelle sie dem Browser hinzugefügt werden soll. Dabei kann ein beliebiges, bereits vorhandenes XUL-Element (*Overlay*) angegeben werden. Jedes der verfügbaren Overlays kann anhand dessen ID referenziert werden. So ist es bspw. möglich das Präferenzen-Menü um einen weiteren Reiter zu erweitern. Weiterhin wird in dieser Datei angegeben, wo sich Content-Elemente befinden (weitere XUL-Files, Skripte, Bilder, Sprachversionen der angezeigten Texte, Skins, etc.).
- **chrome.** In diesem Verzeichnis sind alle Content-Elemente abgelegt. Das Verzeichnis enthält drei Unterverzeichnisse:

5.3. *TrustFox: Trusted Firefox Extension*

- `content` enthält alle XUL-Files, Skripte, Stylesheets, Bilder und sonstiges,
- `locale` enthält die unterschiedliche Sprachdateien (optional),
- `skin` enthält verschiedene Skins (optional)

- **components.** Hier können eigene XPCOM-Komponenten abgelegt werden

Weiterhin können beliebige weitere Verzeichnisse erstellt und genutzt werden. Wird eine Extension installiert, befindet sie sich im *extensions*-Verzeichnis des jeweiligen Firefox-Profilordners.

XML User Interface Language. Die *XML User Interface Language (XUL)* ist eine von Mozilla entwickelte XML-basierende Beschreibungssprache für grafische Oberflächen. Sie wird in den meisten Mozilla-Projekten eingesetzt. Wie auch in HTML, kann in XUL-Dokumenten JavaScript eingebettet werden. Zur Nutzung in Mozilla Extensions, können drei verschiedene Arten von XUL-Files definiert werden. *Overlays* können definiert werden, um XUL-Elemente anderen Elementen anzufügen (*glueing*). Sie werden dem angegebenen Element als Kind-Element hinzugefügt. *Windows* sind eigenständige Fenster, die unabhängig vom Browserfenster existieren können. Diese besitzen einen vom Hauptfenster unabhängigen Lebenszyklus (bleibt bspw. geöffnet, wenn Hauptfenster geschlossen wird). *Dialoge* sind im Gegensatz zu Fenstern an das Hauptfenster gebunden und werden geschlossen, sobald auch das Browserfenster geschlossen wird. Listing 5.3 zeigt ein einfaches Beispiel für ein XUL-Dokument.

XPCOM Dienste. Die XPCOM-Bibliothek¹ stellt eine Reihe von Diensten zur Verfügung, die innerhalb von XUL-Dokumenten über JavaScript genutzt werden können. So stehen den Extensions Funktionen zur Arbeit mit dem Filesystem, Netzwerkbibliotheken und sonstiges zur Verfügung. Über XPCOM können HTTP-Requests modifiziert, Konfigurationen bearbeitet, Sockets geöffnet werden und vieles mehr. XPCOM-Objekte können auch selbst definiert, und den Extensions als Dienste zur Verfügung gestellt werden. Solche Objekte können mit JavaScript oder höheren Programmiersprachen, wie C++ oder Python erstellt werden. Somit sind den Funktionalitäten von Extensions keine Grenzen gesetzt. Sämtliche regis-

¹Cross Platform Component Object Model

5. Implementierung

trierte XPCOM-Bibliotheken können innerhalb von XUL-Files über JavaScript angesprochen werden. Im folgenden einige Dienste aus der Standard XPCOM-Bibliothek:

- `nsFileInputStream` erlaubt es Daten aus Dateien einzulesen,
- `nsIServerSocket` realisiert serverseitige Sockets,
- `nsICookieManager` bietet eine Möglichkeit zum Manipulieren der Browser-Cookies,
- `nsIHttpChannel` erlaubt u.a. die Modifikation von HTTP-Anfragen und das Parsen von HTTP-Antworten

Listing 5.3 Beispiel eines XUL-Dokuments

```
<window onload="fillData()">
<script src="chrome://TrustFox/content/subXuls/lib/serverdata.js"/>
<vbox>
  <label value="XUL-Beispiel"/>
  <label value=""/>
  <hbox>
    <label value="Server-IP:"/>
    <textbox id="serverip"/>
    <label value="Port:"/>
    <textbox width="60" id="serverport"/>
  </hbox>
  <label value=""/>
  <hbox>
    <button label="Speichern" oncommand="setServerInfo();window.close()"/>
    <button label="Abbrechen" oncommand="window.close()"/>
  </hbox>
</vbox>
</window>
```

5.3.3. TrustFox Arbeitsweise

Der entwickelte Prototyp von TrustFox erweitert das Firefox Präferenzen-Panel um einen zusätzlichen Reiter. Dort kann der Nutzer sämtliche Konfigurationen vornehmen. Über diese Oberfläche kann angegeben werden, welche Credentials von TrustFox geschützt werden sollen. Wird TrustFox zum ersten mal aus einem bestimmten Profil heraus aktiviert, registriert es sich zunächst beim TKCMS, um das Handle auf seinen Storage Key zu erhalten. Nach erfolgreicher Registrierung fordert TrustFox auch gleich einen ersten Binding Key an, der als Standardschlüs-

sel für den Schutz der Credentials verwendet wird. Daraufhin übernimmt TrustFox die Verwaltung der Credentials, für die er zuständig ist. Ist TrustFox deaktiviert, greift es in keiner Weise in das Firefox Credential Management ein.

Firefox Preferences System

Firefox besitzt ein Präferenzen-System, auf die über die XPCOM-Schnittstelle `nsIPrefService` zugegriffen werden kann. Durch diesen Dienst können alle Präferenzen bearbeitet werden oder auch neue Präferenzen hinzugefügt werden. Dies betrifft sämtliche Präferenzen, die über die Pseudo-URL `about:config` in Firefox aufgelistet werden. TrustFox verwendet das Präferenzen-System von Firefox, um sämtliche Einstellungen zu verwalten. Präferenzen sind in so genannten *Branches* organisiert, was etwa mit Namensräumen vergleichbar ist. TrustFox Präferenzen werden im Branch *TrustFox* organisiert. In Tabelle 5.1 werden einige TrustFox-Präferenzen aufgelistet. Listing 5.4 zeigt, wie der entsprechende XPCOM-Dienst genutzt werden kann.

Listing 5.4 Zugriff auf Präferenzensystem über XPCOM

```
/* Instanziiere Dienst, um Zugang zu Präferenzen zu erhalten */
this.prefManager = Components.Classes["@mozilla.org/preferences-service;1"]
                    .getService(Components.interfaces.nsIPrefBranch);

/* Den aktuellen Betriebszustand von TrustFox auslesen */
this.enabled = this.prefManager.getBoolPref("TrustFox.enabled");

/* Starte Credentialmanagement, falls TrustFox aktiviert wurde */
if(this.enabled) {
    this.startCredentialManagement();
}
```

Kommunikation mit TKCMS

JavaScript bietet von sich aus keine Client-Sockets. Eine mögliche Lösung hierfür wäre die Integration von Bibliotheken von höheren Programmiersprachen als XPCOM-Komponenten, um Sockets nutzen zu können. Die Extension SIMILE demonstriert, wie über die *LiveConnect*-Schnittstelle eigene Java-Bibliotheken

5. Implementierung

Präferenz	Typ	Bedeutung	Standardwert
TrustFox.enabled	Boolean	Status von TrustFox	False
TrustFox.id	String	Die ID, die der Extension vom Server zugewiesen wurde	0
TrustFox.protection.cookies	Boolean	Gibt an, ob Cookies von TrustFox geschützt werden	false
TrustFox.protection.logins	Boolean	Gibt an, ob Login-Credentials von TrustFox geschützt werden	false
TrustFox.protection.certificates	Boolean	Gibt an, ob Zertifikate von TrustFox geschützt werden	false
TrustFox.keys.default	String	Die UUID des Standard Binding Keys	0
TrustFox.keys.cookies	String	Die UUID des Binding Keys für Cookies	0
TrustFox.server.host	String	IP-Adresse des TKCM Servers	0

Tabelle 5.1.: TrustFox Präferenzen

über JavaScript in Firefox Extensions genutzt werden können[41]. Jedoch ist die LiveConnect Methode veraltet und wird von zukünftigen Firefox Versionen nicht unterstützt werden. Daher wird bei TrustFox die XPCOM-Komponente `nsISocketTransportService` eingesetzt. Dieser Dienst bietet einfache Client-Sockets.

Credential Management

Cookies. Der TrustFox-Prototyp realisiert TKCMS-basierten Schutz für Cookies, da dies besonders feine Mechanismen voraussetzt. Anhand von Cookies kann sehr gut gezeigt werden, wie der Schutz für On-Demand-Credentials implementiert werden sollte. Wurde der Schutz für Cookies aktiviert, werden zunächst sämtliche Cookies, die sich in der Cookie-Datenbank von Firefox befinden ausgelesen und zur Verschlüsselung an den Credential Manager gesendet. Für jedes verschlüsselte Cookie erhält TrustFox ein Handle, das nach dem Cookie und dem Host benannt ist. Dadurch kann TrustFox sämtliche Cookies für einen bestimmten Host identifizieren. TrustFox registriert einen *Observer* für HTTP-Verbindungen, der die Routinen für die Cookie-Bearbeitung erhält. Setzt ein HTTP-Server den *Set-Cookie*-Flag in einer Antwort, liest TrustFox diesen Wert ein, schickt es dem TKCMS zur Verschlüsselung und verhindert, dass der Cookie in der Firefox-Datenbank abgespeichert wird. Wird eine HTTP-Anfrage vorbereitet, greift TrustFox unmittelbar vor dem Versenden der Anfrage ein und überprüft, ob ein Cookie für den Zielhost

5.3. TrustFox: Trusted Firefox Extension

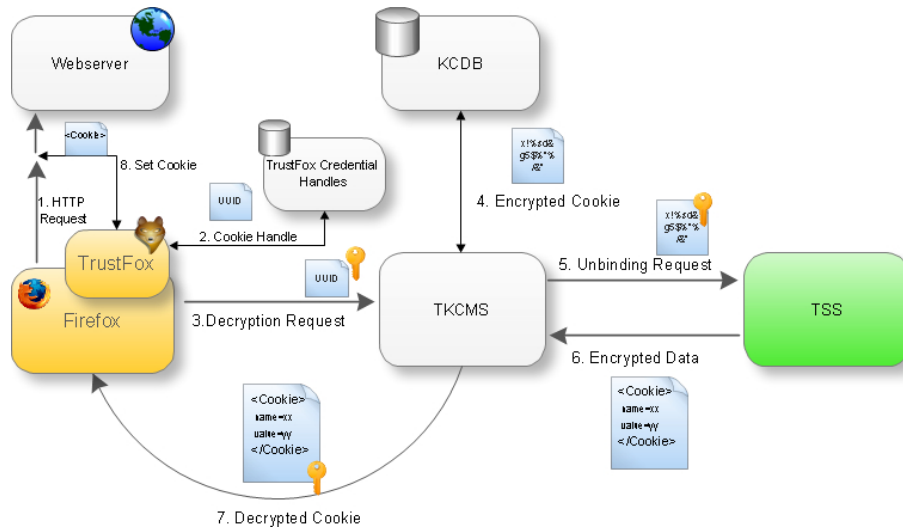


Abbildung 5.2.: TrustFox Cookie-Mechanismus

vorhanden ist. Hierzu überprüft TrustFox, ob ein Cookie-Handle mit dem entsprechenden Host als Label existiert. Ist dies der Fall, lässt TrustFox den Cookie vom TKCMS entschlüsseln und setzt daraufhin den *Cookie*-Flag in den Header der Anfrage. Dieser Ablauf wird in Abbildung 5.2 verdeutlicht. So lange TrustFox-Schutz für Cookies aktiviert ist, befinden sich niemals Cookies unverschlüsselt in der Firefox Cookie-Datenbank.

Weitere Credentials. Für andere Credentials kann eine ähnliche Vorgehensweise implementiert werden. Beispielsweise kann der oben erläuterte Prozess genauso auf den Schutz von Login Credentials angewandt werden. Das Ereignis, das in diesem Fall ein Entschlüsseln auslöst, ist das Laden einer Webseite, für die Login-Daten in der Credential Datenbank zur Verfügung stehen. On-runtime-Credentials müssen bereits nach dem Laden des Browsers aus dem Speicher geholt werden, sofern diese dem Nutzer zur Verfügung stehen sollen. Wird der Browser geschlossen, muss TrustFox eingreifen, um die Credentials wieder an den Credential Manager zu senden. Hierfür können die Ereignisse `app-startup` und `quit-application` oder Varianten davon beobachtet werden.

5. Implementierung

6. Diskussion

Die in dieser Arbeit vorgestellte Architektur soll die Sicherheit der Schutzmechanismen für User Credentials von Web Browsern durch den Einsatz von Trusted Computing erhöhen. Es ist darauf zu achten, dass dabei keine neuen Sicherheitsrisiken entstehen. Weiterhin ist bei der Entwicklung einer Lösung wichtig, die Gewohnheiten und Erfahrungen der Nutzer zu berücksichtigen. Wenn die Benutzbarkeit eines Systems nicht gegeben ist, wird es zu keiner Akzeptanz kommen, daher muss stets eine Balance zwischen erhöhten Schutzmaßnahmen und Benutzbarkeit gefunden werden. Dies wurde bei der Entwicklung der TrustFox-Architektur berücksichtigt. In diesem Abschnitt wird analysiert, in wie Fern dieses Ziel erreicht wird und an welchen Stellen die Maßnahmen noch verbessert werden können. Im Anschluss werden einige Arbeiten vorgestellt, die ähnliche Lösungen liefern.

6.1. Sicherheit

Es werden nun einige Aspekte der Architektur auf potenzielle Schwachstellen analysiert.

Trusted Wallet

Credentials werden zentral beim TKCMS gespeichert. Eine mögliche Gefahr ist der Zugriff auf die Daten durch Unbefugte. Es ist zu gewährleisten, dass der alleinige Zugriff auf die Datenbank keinen Verlust darstellt. Die vorgestellten Mechanismen sehen vor, dass sämtliche Daten, die in der Datenbank abgelegt werden, verschlüsselt werden. Hierbei kommen relativ starke Algorithmen wie RSA mit Schlüssellängen ab 2048 Bit zum Einsatz. Wie in Kapitel 3 erläutert, stehen sämtliche Schlüssel dabei im Schutz des Root of Trust for Storage. *Offline-Attacken* auf

6. Diskussion

2048 Bit RSA sind nach bisherigem Stand nur ineffizient durchzuführen, weshalb dies im Prinzip ausgeschlossen werden kann.

Authentifizierungsmechanismen: Identity Theft

Es ist zu gewährleisten, dass nur Befugte Zugriff auf entsprechende Daten haben. Hierzu wurde das Prinzip der Trust Domains eingeführt. Dies soll gewährleisten, dass Daten von unterschiedlichen Subjekten in voneinander abgeschotteten virtuellen Tresoren abgelegt werden. Ausgehend von der grundlegenden TPM-Schlüsselhierarchie, wurde eine Sub-Hierarchie vom TKCMS eingeführt, in der unterschiedliche Abzweigungen zu unterschiedlichen Trust Domains führen. Die sukzessive Unterteilung in weitere Subdomains durch Anwendungen ist durch die Erzeugung von neuen Schlüsseln möglich. Grundsätzlich besitzt jede Anwendung einen eigenen Schlüssel und erhält somit einen von anderen Anwendungen getrennten Bereich. Weiterhin haben Anwendungen die Möglichkeit, jedem Nutzerprofil eine eigene Subdomain bieten. Ist dies gewährleistet, sind sämtliche Daten von einem Nutzer mit einem Schlüssel aus seiner Subdomain verschlüsselt. Für die Nutzung der Schlüssel muss stets das richtige Passwort genannt werden.

Integrität von TKCMS

Abgesehen vom Schutz vor Unbefugten, ist zu gewährleisten, dass die gespeicherten Daten vor Manipulation geschützt sind. Sämtliche Informationen über Credentials und Schlüssel sind in den Handles gespeichert. So könnten Informationen, wie UUIDs oder Policies prinzipiell manipuliert werden. Um solche Manipulationen zu erkennen, könnte zusätzlich ein Mechanismus für Integritätsprüfungen eingeführt werden. Bspw. kann der TKCMS sämtliche Handles digital signieren. Listing 6.1 zeigt, wie solch ein *Signed Handle* aussehen könnte. Vor Benutzung der Handles hätte er somit die Möglichkeit zu erkennen, ob sich die Daten im Originalzustand befinden. Für diese Zwecke könnte sich der Trusted Key and Credential Manager einen Signing Key generieren. Zudem ist er als *Trusted Third Party* eine zentrale Verwaltung für sämtliche sensiblen Daten von Nutzern und Anwendungen und somit eine sehr kritische Komponente. Daher ist auch die Integrität des TKCMS zu sichern. Es ist zu empfehlen, dass der Dienst bei Integritätsmessungen eines Trusted Operating Systems berücksichtigt wird. Sobald eine Abweichung von Referenzwerten festgestellt wird, darf der Dienst nicht ausgeführt werden.

Listing 6.1 Ein Signed Handle

```
<SignedHandle>
  <SecurityObject>
    <ClientName>Mozilla Firefox</ClientName>
    <ClientSecret>pwdhash</ClientSecret>
    <Uuid>123-4567-89123-24454</Uuid>
    <Policy>
      <Ttl>0</Ttl>
      <Status>1</Status>
    </Policy>
  </SecurityObject>
  <TrustedSignature>
    0123456789abcdef987654321fedcbabcdfe21635
    761dca683aa690fecsf23561ac4e7a5c46e875cf
  </TrustedSignature>
</SignedHandle>
```

Protokollsicherheit

Einen weiteren potenziellen Angriffspunkt bildet das Kommunikationsprotokoll. Hierüber werden sensible Daten, unter anderem auch die Passwörter versendet. Es gehört nicht zum Ziel dieser Arbeit, Mechanismen für den Schutz auf Kommunikationsebene zu entwickeln. Hierzu existieren etablierte Standards, wie zum Beispiel Transport Layer Security, die neben einer Verschlüsselung noch die Integrität der Kommunikation wahrt. Diese Standards können dabei helfen, die Vertraulichkeit der Kommunikation zwischen Client und Server zu sichern. Weiterhin könnte das in Kapitel 3 Trusted Network Connect (TNC) hilfreich sein. Durch den Einsatz dieser Technologie kann eine vertrauenswürdige Konnektivität der Kommunikationspartner gewährleistet werden.

Trusted Computing Sicherheit

Die gesamte Architektur baut auf der Sicherheit der TC-Mechanismen auf. Der hardwarebasierte Schutz und Trusted Storage funktionieren nur, wenn Trusted Computing lückenlos umgesetzt wird. Integritätsmessungen von kritischen Diensten müssen u.U. wiederholt zur Laufzeit eines Systems durchgeführt werden. Dies erfordert die Existenz eines vertrauenswürdigen Betriebssystems, das die Vertrauenskette dynamisch fortführt. Ein solches Betriebssystem ist noch nicht im Produktiveinsatz, jedoch sind Ansätze wie EMSCB unterstützenswert.

TrustFox Sicherheit

TrustFox eliminiert die Pflicht des Browsers, sich um die sichere Speicherung von Credentials zu kümmern. Dabei nutzt TrustFox die Möglichkeit der Erzeugung von Trust Domains für unterschiedliche Nutzer und Profile aus und sorgt dafür, dass die Credentials durch den TKCMS geschützt werden. Die Schwächen der softwarebasierten Module werden dadurch eliminiert. Jedoch ist auch dies nur gewährleistet, so lange die Extension und die Browserbase vertrauenswürdig sind. Insbesondere sind Extensions sehr mächtig, da sie Zugriff auf die meisten Funktionalitäten des Browsers haben. Mögliche Gefahren werden in [86] beschrieben. Daher ist auch dafür zu sorgen, dass diese Komponenten von einem vertrauenswürdigen Betriebssystem überwacht werden. Jeder Nutzer muss sich mit einem Passwort authentifizieren, um an seine Credentials zu kommen. Die Eingabe des Passwortes muss auch eine vertrauenswürdige Schnittstelle erfolgen. Es gibt bereits einige Ansätze, wie solche *Trusted GUIs* realisiert werden können [19; 23].

6.2. Benutzbarkeit

Gute Sicherheitslösungen werden nicht akzeptiert, wenn Benutzer nicht mit ihnen umgehen können. Ein gutes Beispiel hierfür sind Protokolle für sicheres E-Mailing [91]. Trusted Computing bringt jedoch relativ viele Erneuerungen. Daher ist die Entwicklung von TC-Anwendungen besonderen Herausforderungen gestellt. Beim Entwurf des TKCMS und TrustFox wurde darauf Wert gelegt, möglichst einfach in bestehende Systeme integrierbar zu sein. TrustFox ist als Erweiterung für Firefox realisiert. Es ist zu erwarten, dass den meisten Nutzern der Umgang mit solchen Extensions vertraut ist. Sämtliche Konfigurationsmöglichkeiten wurden in das vorhandene Präferenzen-Panel von Firefox integriert. Nach einer einfachen Installation bietet TrustFox bereits mit den Standardeinstellungen erhöhten Schutz, ohne dass der Nutzer zusätzliche Konfigurationen vornehmen muss.

6.3. Ähnliche Arbeiten und Architekturen

Der Trusted Key und Credential Manager ist einer der beiden Hauptteile der vorgestellten Architektur. Die Lösung berücksichtigt bei weitem nicht alle Aspek-

6.3. Ähnliche Arbeiten und Architekturen

te, die ein Schlüsselmanagement beachten sollte. Es gibt zahlreiche Standards und Veröffentlichungen, die sich ausgiebiger mit dem Thema Schlüsselmanagement beschäftigen. Einer dieser Standards ist der *XML Key Management Standard (XKMS)*[16]. XKMS besteht aus den beiden Teilen *XML Key Information Service Specification (XKISS)* und *XML Key Registration Service Specification (XKRSS)*. Zweck des Standards ist die Schaffung eines Protokolls, das die Verteilung und Registrierung von asymmetrischen Schlüsselpaaren vereinheitlichen soll. Eine von der IETF erarbeitete Spezifikation ist RFC 2367, das eine socketbasierte API für Schlüsselmanagement beschreibt. Ein weiterer interessanter Ansatz wird von der *Security in Storage Working Group* der *IEEE* verfolgt (*IEEE 1619 SIG-WG*)[42]. Die Aufgabe dieser Arbeitsgruppe ist die Erarbeitung von Standards rund um Verschlüsselung von Datenspeichern, wobei dies auch das Key Management beinhaltet. Interessant ist im Bezug zu TKCMS das Projekt *1619.3*, in dem die Spezifikationen für das Key Management erarbeitet werden. Schlüssel als Objekte werden dabei sehr ausführlich beschrieben und auch deren Lebenszyklus wird detailliert modelliert. Dazu gehören noch Aspekte wie Policies und Austausch von Schlüsselobjekten über Web Services. Im Bezug auf Trusted Computing existiert eine Untergruppe der *Trust in Storage Workgroup*, die sich dem Thema Key Management widmet. Die *Trusted Storage Key Management Services Subgroup (KMSS)* hat bis zum jetzigen Zeitpunkt (April 2009) noch keine Spezifikationen veröffentlicht, jedoch werden Lösungen für folgende Problemstellungen als Ziele definiert:

- Schlüsselgenerierung
- Nutzung von Schlüsseln
- Empfang, Bearbeitung und Speicherung von Schlüsseln
- Suchen nach Schlüsseln
- Zugriffsrechte für Schlüssel
- Deaktivierung und Vernichtung von Schlüsseln

Insgesamt wird klar, dass sich im Bereich Key Management noch einiges bewegen wird. Bei der Erarbeitung des TKCMS dienen Teilaspekte mancher Standards,

6. Diskussion

wie z.B. Aspekte der Spezifikationen von IEEE P1619.3 als Inspiration und gingen auch an manchen Stellen in die Arbeit ein.

Neben den in Kapitel 2 vorgestellten Lösungen wird in [71] ein wallet-basiertes Credentialmanagement vorgestellt, das auf dem Perseus-Framework aufbaut. Hier wird ein Browser um ein sogenanntes *Wallet-Proxy* erweitert, das sich zwischen Webserver und Webbrowser befindet. Der Wallet-Proxy nutzt TC-Funktionalitäten wie das Trusted Storage, um User Credentials sicher abzuspeichern. Bei Webanfragen mit Authentifizierung übernimmt er das Senden der Credentials, so dass der Nutzer die sensiblen Daten nur gegenüber dem vertrauenswürdigen Wallet-Proxy preisgeben muss. Dadurch wird klar, dass die Lösung vor allem als Maßnahme gegen Phishing gedacht ist. Auch Microsoft wird mit der nächsten Version des Windows Betriebssystems einen zentralen Credential Manager anbieten. In der aktuellen Beta-Version von Windows 7 ist *Windows Vault* integriert. Dies ist ein zentraler Credential Manager für Authentifizierungsdaten von Webseiten, Servern oder sonstigen Ressourcen.

6.4. Fazit und Ausblick

In dieser Arbeit wurde eine Architektur vorgestellt, die die Schwachstellen gängiger Webbrowser im Bezug auf Credential Management eliminiert. Diese basieren im Grunde auf der Tatsache, dass kritische Daten und kryptographische Schlüssel von softwarebasierten Token geschützt werden. Die aktuellen Trends im Browser-Markt zeigen deutlich, dass Sicherheit und vor allem Privatsphäre diejenigen Aspekte sind, die bei der Entwicklung von neuen Browser-Versionen im Vordergrund stehen. Mit Features wie *InPrivate-Browsing* soll der Nutzer die Möglichkeit haben, die Ansammlung von Credentials zu reduzieren, bzw. zu verhindern. Dies mag ein anonymes Surfen im Web ermöglichen, jedoch werden mit Credential wie Cookies erst viele Funktionen im Web erst möglich. Eine vollständige Entfernung der Daten kann also nicht in jedem Fall eine Lösung sein. Es gibt jedoch kaum neue Ansätze, was die technischen Schutzmaßnahmen der Credentials verbessert. Im Gegensatz zu diesen Entwicklungen basiert der in dieser Arbeit vorgestellte Lösung auf den Möglichkeiten eines Trusted Platform Modules und des Trusted Storages. Sämtliche Daten werden somit hardwarebasiert geschützt,

wodurch das Risiko einer Entwendung von kritischem Datenmaterial verringert wird. Es ist auch eine Lösung vorstellbar, die bestehende Systeme, wie z.B. die Key Storage Provider in Windows erweitert, indem ein Credential Manager, wie in dieser Arbeit vorgestellt, als Third-Party-Provider integriert wird. Ein Trusted Key Manager würde auch die Sicherheit von Schlüssel aus Anwendungen wie PGP o.ä. erhöhen. Wenn sich Trusted Computing durchsetzt, werden zahlreiche neue Wege ermöglicht, um Richtlinien rund um Datenschutz und Privatsphäre umzusetzen. Anwendungen wie TrustFox und Initiativen wie OpenTC, EMSCB u.ä. sind notwendig, um eine erhöhte Akzeptanz für Technologien wie TC zu schaffen. Denn in solchen Anwendungen liegen nicht nur die Interessen von Content- oder Service-Providern im Vordergrund.

6. *Diskussion*

Abbildungsverzeichnis

2.1	Webbrowser Marktanteile	7
2.2	Referenzarchitektur von Webbrowsern	9
2.3	Lebenszyklus von kryptographischen Schlüsseln	18
2.4	Ende-zu-Ende-Authentifizierung mit Kreditkarte	21
2.5	Sicheren Plattform von IBM	22
2.6	Übersicht des Common Data Security Architecture-Frameworks	24
2.7	Cryptoki Funktionsweise	26
2.8	Kryptographie Modul-Manager in Firefox	30
2.9	Arbeitsweise von DPAPI	32
3.1	Innerer Aufbau eines Trusted Platform Moduls	43
3.2	TPM-Kommunikationsprotokolle	47
3.3	Das Prinzip des transitive Trust	48
3.4	TPM Schlüsselhierarchie und Trusted Storage	50
3.5	Interne Struktur eines TPM-Schlüssels	52
3.6	Die TCG Software Stack Hierarchie	54
3.7	AIK Credential Request	56
3.8	Das PERSEUS-Sicherheitsframework	60
3.9	time-of-measurement – time-of-attestation Problem	68
4.1	TKCMS Architektur	73
4.2	Erweiterte TPM-Schlüsselhierarchie	76
4.3	Mögliche Unterteilung der Trust Domain eines Webbrowsers	80
5.1	Struktur eines Firefox Extension-Pakets	88
5.2	TrustFox Cookie-Mechanismus	93

Abbildungsverzeichnis

Tabellenverzeichnis

2.1	Wichtige Cryptoki-Funktionen	27
3.1	Vergleich zwischen PKCS#11- und TPM-Kommandos	65
5.1	TrustFox Präferenzen	92

Tabellenverzeichnis

Quellcodeverzeichnis

5.1	Ein Key Handle	83
5.2	Ein Registration Request	85
5.3	Beispiel eines XUL-Dokuments	90
5.4	Zugriff auf Präferenzensystem über XPCOM	91
6.1	Ein Signed Handle	97

Quellcodeverzeichnis

Abkürzungsverzeichnis

AACP	Asymmetric Authorization Change Protocol
ADCP	Authorization Data Change Protocol
ADIP	Authorization Data Insertion Protocol
AIK	Attestation Identity Key
ASK	Application Storage Key
BMS	Backup and Migration Service
CA	Certification Authority
CDSA	Common Data Security Architecture
CRTM	Core Root of Trust for Measurement
DAA	Direct Anonymous Attestation
DPAPI	Data Protection API
EK	Endorsement Key
FIPS	Federal Information Processing Standard
KCDB	Key and Credential Database
MP	Migration Packet
NSS	Network Security Services
OIAP	Object-Independent Authorization Protocol
OSAP	Object-Specific Authorization Protocol
PC	Platform Credential
PCDB	Platform Credential Database
PCR	Platform Configuration Register
PKCS#11	Public Key Cryptography Standards #11

Quellcodeverzeichnis

PStore	Protected Storage
RA	Remote Attestation
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SO	Security Object
SRK	Storage Root Key
TC	Trusted Computing
TCG	Trusted Computing Group
TCP	Trusted Computing Platform
TKCM	Trusted Key and Credential Manager
TKCMS	Trusted Key and Credential Management Service
TNC	Trusted Network Connect
TOS	Trusted Operating System
TPM	Trusted Platform Module
TRK	TKCMS Root Key
TSS	TCG Software Stack
USRK	User Storage Root Key
XPCOM	Cross Platform Component Object Model
XUL	XML User Interface Language

Literaturverzeichnis

- [1] Ross Anderson. Trusted computing frequently asked questions. <http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html>, August 2003. zugegriffen am 27. April 2009.
- [2] William A. Arbaugh. The tcpa; what's wrong; what's right and what to do about. <http://www.cs.umd.edu/waa/TCPA/TCPA-goodnbad.html>, July 2002. zugegriffen am 27. April 2009.
- [3] Jean-Daniel Auessel. Smart cards and digital security. In *Computer Network Security*, volume 1, chapter 1, pages 42–56. Springer Berlin Heidelberg, 2007.
- [4] European Multilaterally Secure Computing Base. General architecture. <http://www.emscb.com/content/pages/49373.htm>.
- [5] Fabrice Bellard. Qemu generic and open source machine emulator and virtualizer. <http://www.nongnu.org/qemu/about.html>. zugegriffen am 27. April 2009.
- [6] Tim Berners-Lee and Mark Fischetti. *Weaving the Web - The original design and ultimate destiny of the World Wide Web*. HarperBusiness, 1999.
- [7] Daniel J. Bernstein. Cache-timing attacks on AES, 2004.
- [8] R.M. Best. Preventing software piracy with crypto-microprocessors. In *Proceedings of IEEE Spring Comcon '80*, pages 466 – 469, 1980.
- [9] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 513–525, London, UK, 1997. Springer-Verlag.

- [10] Bert Boer and Antoon Bosselaers. Collisions for the compression function of md5. In *In Advances in Cryptology, Proceedings of EUROCRYPT '93*, pages 293–304, 1994.
- [11] Andreas Brett and Andreas Leicher. Ethemba trusted host environment—mainly based on attestation. *CoRR*, abs/0901.4496, 2009.
- [12] Jan Camenisch. Better privacy for trusted computing platforms. In *Proceedings of the 9th European Symposium on Research in Computer Security*, 2004.
- [13] Liqun Chen and Mark Ryan. Offline dictionary attack on tcg tpm weak authorisation data, and solution. In *Proceedings of Future of Trust in Computing 2008, Berlin, Germany*, 2008.
- [14] EMSCB Consortium. Towards trustworthy systems with open standards and trusted computing. <http://www.emscb.com/content/pages/49241.htm>. zugegriffen am 27. April 2009.
- [15] World Wide Web Consortium. Web storage editor’s draft 26 april 2009. <http://dev.w3.org/html5/webstorage>. zugegriffen am 27. April 2009.
- [16] World Wide Web Consortium. Xml key management specification (xkms). <http://www.w3.org/TR/xkms>. zugegriffen am 27. April 2009.
- [17] D. E. Denning. Protecting public keys and signature keys. *Computer*, 16(2):27–35, 1983.
- [18] Bundesministerium der Justiz. Sozialgesetzbuch (sbg) fünftes buch (v) - gesetzliche krankensversicherung, §291a (3).
- [19] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, New York, NY, USA, 2005. ACM.
- [20] J. Dyer, R. Perez, S. Smith, and M. Lindermann. Application support architecture for a high-performance, programmable secure coprocessor. In *Proceedings of the 22nd National Information Systems Security Conference*, October 1999.

- [21] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg, 2007.
- [22] Mikhael Felker. Password management concerns with ie and firefox. <http://www.securityfocus.com/infocus/1882>, 12 2006.
- [23] Norman Feske and Christian Helmuth. A nitpicker's guide to a minimal-complexity secure gui. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 85–94, Washington, DC, USA, 2005. IEEE Computer Society.
- [24] Barbara Fichtinger, Ekehard Herrmann, Nicolai Kuntze, and Andreas U. Schmidt. Trusted infrastructures for identities. In Rüdiger Grimm and Berthold Hass, editors, *Virtual Goods: Technology, Economy, and Legal Aspects. Proceedings of the 5th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods, Koblenz, October 11-13, 2007*, Hauppauge, New York, 2008. Nova Publishers.
- [25] G P Fick. Implementation issues for master key distribution and protected keyload procedures. In *Proceedings of the 2nd IFIP international conference on Computer security: a global challenge*, pages 571–580, Amsterdam, The Netherlands, The Netherlands, 1984. North-Holland Publishing Co.
- [26] International Organisation for Standardization. Iso/iec 7816: Integrated circuit(s) cards with contacts.
- [27] Internet Engineering Task Force. Rfc 2965: Http state management mechanism. <http://www.ietf.org/rfc/rfc2965.txt>.
- [28] Sandro Gauci. Surf jacking - https will not save you. Technical report, ENABLESECURITY, 2008.
- [29] Florian Gawlas, Gisela Meister, Axel Heider, and Sebastian Wallner. *Trusted Computing*, chapter Interaktionen TPM und Smart Card, pages 110–121. Vieweg+Teubner, 2008.
- [30] Alan Grosskurth and Michael W. Godfrey. A reference architecture for web browsers. In *ICSM '05: Proceedings of the 21st IEEE International Confe-*

Literaturverzeichnis

- rence on *Software Maintenance*, pages 661–664, Washington, DC, USA, 2005. IEEE Computer Society.
- [31] Open Group. Common security: Cdsa and cssm, version 2. <http://www.opengroup.org/online-pubs?DOC=9690989599&FORM=PDF>, May 2000.
- [32] Trusted Computing Group. Developer resources. <http://www.trustedcomputinggroup.org/developers/>. zugegriffen am 27. April 2009.
- [33] Trusted Computing Group. Interoperability specification for backup and migration services. Technical report, Trusted Computing Group, Incorporated, 2005.
- [34] Trusted Computing Group. Tcg specification architecture overview. Technical report, Trusted Computing Group, Incorporated, 2007.
- [35] Trusted Computing Group. Tcg platform reset attack mitigation specification. Technical report, Trusted Computing Group, 2008.
- [36] Trusted Computing Group. Tcg trusted network connect: Architecture for interoperability. Technical report, Trusted Computing Group, Incorporated, 2008.
- [37] Peter Gutmann. An open-source cryptographic coprocessor. In *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, 2000.
- [38] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, , and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th USENIX Security Symposium (Sec '08), San Jose, CA, July 2008*, pages 45 – 60, 2008.
- [39] J. Alex Halderman, Brent Waters, and Edward W. Felten. A convenient method for securely managing passwords. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 471–479, New York, NY, USA, 2005. ACM.

- [40] heise.de. Schwache krypto-schlüssel unter debian, ubuntu und co. <http://www.heise.de/security/Schwache-Krypto-Schluessel-unter-Debian-Ubuntu-und-Co-/news/meldung/107808>, May 2008.
- [41] David Huynh. Java firefox extension. http://simile.mit.edu/wiki/Java_Firefox_Extension. zugegriffen am 27. April 2009.
- [42] IEEE. P1619 security in storage working group (siswg). <https://siswg.net/>. zugegriffen am 27. April 2009.
- [43] Julian Jang, Surya Nepal, and John Zic. Establishing a trust relationship in cooperative information systems. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 426–443. Springer Berlin / Heidelberg, 2006.
- [44] C. J. Jansen. On the key storage requirements for secure terminals. *Comput. Secur.*, 5(2):145–149, 1986.
- [45] keepass.info. Keepass password safe. <http://keepass.info/>. zugegriffen am 27. April 2009.
- [46] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *ESORICS '98: Proceedings of the 5th European Symposium on Research in Computer Security*, pages 97–110, London, UK, 1998. Springer-Verlag.
- [47] P. Kocher, J. Jaffe, and B. Yun. Introduction to differential power analysis and related attacks. <http://www.cryptography.com/dpa/technical/index.html>, December 1998.
- [48] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.
- [49] Dirk Kuhlmann, Rainer Landfermann, Harigovind Ramasamy, Matthias Schunter, Gianluca Ramunno, and Davide Vernizzi. An open trusted com-

Literaturverzeichnis

- puting architecture - secure virtual machines enabling user-defined policy enforcement, 2006.
- [50] RSA Laboratories. Pkcs 11 v2.20: Cryptographic token interface standard.
- [51] Andreas Leicher, Nicolai Kuntze, and Andreas U. Schmidt. Implementation of a trusted ticket system. In *Proceedings of the IFIP sec2009, Pafos, Cyprus*, 2009.
- [52] Daniel Lin and Michael C. Loui. Taking the byte out of cookies: privacy, consent, and the web. In *ACM POLICY '98: Proceedings of the ethics and social impact component on Shaping policy in the information age*, pages 39–51, New York, NY, USA, 1998. ACM.
- [53] live.gnome.org. Gnome keyring. <http://live.gnome.org/GnomeKeyring>. zugegriffen am 27. April 2009.
- [54] John P. McGregor and Ruby B. Lee. Protecting cryptographic keys and computations via virtual secure coprocessing. *SIGARCH Comput. Archit. News*, 33(1):16–26, 2005.
- [55] John P. McGregor, Yiqun L. Yin, and Ruby B. Lee. Virtual secure coprocessing on general-purpose processors. Technical Report CE-L2002-003, Princeton University Department of Electrical Engineering, November 2002.
- [56] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.
- [57] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.
- [58] Thomas Müller. *Trusted Computing Systeme: Konzepte und Anforderungen*. Xpert.Press, 2008.
- [59] Mozilla. Network security services (nss). <http://www.mozilla.org/projects/security/pki/nss/>.

- [60] mozillaZine. Key3.db. <http://kb.mozillazine.org/Key3.db>.
- [61] Microsoft Developer Network (MSDN). Cng key storage providers. [http://msdn.microsoft.com/en-us/library/bb931355\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb931355(VS.85).aspx).
- [62] Microsoft Developer Network (MSDN). Cryptography reference. <http://msdn.microsoft.com/en-us/library/aa380256.aspx>.
- [63] Inc. NAI Labs, Network Associates. Windows data protection. <http://msdn.microsoft.com/en-us/library/ms995355.aspx>, October 2001.
- [64] National Institute of Standards and Technology. Security requirements for cryptographic modules. <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>.
- [65] National Institute of Standards and Technology. Cryptographic hash algorithm competition. <http://csrc.nist.gov/groups/ST/hash/sha-3/index.html>, August 2006.
- [66] Tim O'Reilly. What is web 2.0 - design patterns and business models for the next generation of software. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>, September 2005.
- [67] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006*, pages 1–20. Springer-Verlag, 2006.
- [68] passwordsafe.sourceforge.net. Password safe. <http://passwordsafe.sourceforge.net/>. zugegriffen am 27. April 2009.
- [69] W. Price. Key management for data encipherment. In *Proceedings of IFIP/SEC '83*. Elsevier Science Publishers, 1983.
- [70] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

- [71] Ahmad-Reza Sadeghi, Sebastian Gajek, Christian Stübke, and Marcel Winandy. Compartmented security for browsers - or how to thwart a phisher with trusted computing. In *Proceedings of the The Second International Conference on Availability, Reliability and Security (ARES 2007)*, pages 120–127. IEEE Computer Society, 2007.
- [72] Ahmad-Reza Sadeghi, Marcel Selhorst, Christian Stübke, Christian Wachsmann, and Marcel Winandy. Tcg inside: A note on tpm specification compliance. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 47–56, New York, NY, USA, 2006. ACM.
- [73] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996.
- [74] Daniel Schröder and Pascal Poschenrieder. Browserkrieg 2.0: Revolutioniert google chrome das surfen? http://www.cnet.de/praxis/specials/39195682/browserkrieg+2_0+revolutioniert+google+chrome+das+surfen.htm, September 2008.
- [75] Marcel Selhorst, Christian Stübke, and Felix Teerkorn. Tss studie - einföhrung und analyse des oss tcg software stack trousers und werkzeuge in dessen umfeld. Technical report, Bundesamt für Sicherheit in der Informationstechnik, 2008.
- [76] sourceforge.net. opencryptoki. <http://sourceforge.net/projects/opencryptoki>. zugegriffen am 27. April 2009.
- [77] sourceforge.net. opencryptoki pkcs#11 implementation for linux. <http://sourceforge.net/projects/opencryptoki>.
- [78] Evan R. Sparks. A security assessment of trusted platform modules. Technical report, Department of Computer Science Dartmouth College, 2007.
- [79] George Staikos. Kwallet - the kde wallet system.
- [80] Douglas Stinson. *Cryptography: Theory and Practice*. Taylor & Francis, 2002.
- [81] Mario Strasser. Software-based tpm emulator for unix. <http://tpm-emulator.berlios.de/>. zugegriffen am 27. April 2009.

- [82] Joachim Swoboda, Stephan Spitz, and Michael Pramateftakis. *Chipkarten und Sicherheitsmodule*, chapter 7, pages 199–240. Vieweg+Teubner, 2008.
- [83] tagesthemen.de. Der nächste skandal, dieselbe firma. <http://www.tagesthemen.de/inland/telekom236.html>, December 2008.
- [84] Nagareshwar Talekar. Firemaster - the firefox master password recovery tool. <http://www.securityxploded.com/firemaster.php>.
- [85] Ruth Taylor. A comparison of cdsa to cryptoki. In *Proceedings of the 22nd National Information Systems Security Conference*. National Institute of Standards and Technology (NIST), 1999.
- [86] Mike Ter Louw, Jin Soon Lim, and V. N. Venkatakrisnan. Extensible web browser security. In *DIMVA '07: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 1–19, Berlin, Heidelberg, 2007. Springer-Verlag.
- [87] Matthias Thieme. Skandal bei der lbb: Gigantisches datenleck. http://www.fr-online.de/top_news/1645133_Gigantisches-Datenleck.html, 12 2008.
- [88] J.D. Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. In *Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment*, 1991.
- [89] w3schools. Web statistics and trends - browser statistics month by month. http://www.w3schools.com/browsers/browsers_stats.asp, April 2009. zugegriffen am 27. April 2009.
- [90] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005.
- [91] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

Literaturverzeichnis

- [92] Andreas Wilkens. Fernsehmagazin: Datenpanne bei einwohnermeldeämtern. <http://www.heise.de/newsticker/Fernsehmagazin-Datenpanne-bei-Einwohnermeldeamtern-Update-/meldung/109835>, June 2008.
- [93] Min Wu, Robert C. Miller, and Greg Little. Web wallet: preventing phishing attacks by revealing user intentions. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 102–113, New York, NY, USA, 2006. ACM.

A. Inhalt der CD

Die mitgelieferte CD-ROM enthält folgende Dateien und Verzeichnisse:

Pfad	Inhalt
TrustedBrowser.pdf	Die vorliegende Diplomarbeit in elektronischer Form.
AusschreibungDA.pdf	Ausschreibung für diese Diplomarbeit
tkcms	Enthält den Sourcecode für den Trusted Key and Credential Manager. Hier sind auch die jTSS-Bibliotheken enthalten
trustfox	Enthält den Sourcecode für die Firefox Extension TrustFox.
docs	Enthält sowohl die JavaDocs als auch eine Benutzerdokumentation.