Master Thesis

TIA

Authentisierung und Integritätskontrolle von Thin Clients

Autor:	Viktor Dillmann
Datum:	Saarbrücken, 30. August 2012
Bearbeitungs Zeitraum:	01.03.2012 - 31.08.2012
Betreuender Professor:	Prof. Dr.-Ing. Damian Weber
Zweitkorrektor:	Prof. Dr.-Ing. Jürgen Schäfer

Hochschule:	Hochschule für Technik und Wirtschaft des Saarlandes (HTWdS)
Ort:	Goebenstraße 40, 66117, Saarbrücken
Fakultät:	Ingenieurwissenschaften (IngWi)
Studienfach:	Praktische Informatik Master (PI-M)

Abstrakt

In dieser Thesis wird die Problematik behandelt, einen Client mittels Software zu schützen. Dabei sollen, im Kontrast zu kommerziellen Lösungen, Quelloffene Komponenten benutzt werden. Genauso soll das gesamte Konzept und die Umsetzung Quelloffen sein und trotzdem den nötigen Schutz bieten können.

Es wird auf Grundlagen der Kryptologie eingegangen um auf das Thema vorzubereiten. Dabei handelt es sich um Themen wie Verschlüsselung, Hashes, Kommunikation, Trusted Systems und andere. Daneben werden auch aktuell eingesetzte Sicherheitssysteme oder Komponenten, von großen Unternehmen, aufgezeigt. Diese sind zum Teil oder komplett ausgehebelt worden.

Während der Thesis wird ein System bestehend aus zwei Komponenten entworfen. Da nicht für alle Ideen die nötige Zeit investiert werden konnte, wurden diese mindestens genannt, wenn auch nicht im Detail besprochen.

Das entworfene System wurde in Ruby und C implementiert. Ein lokaler Test zeigte, dass es grundsätzlich möglich ist, ein wie im ersten Abschnitt genanntes System zu entwickeln. Zudem wird besprochen, gegen welche Arten von Angriffen das konzipierte System nicht bestehen kann.

Inhaltsverzeichnis

1 Einleitung

Thin Clients finden zunehmend mehr Verwendung in Unternehmen, Lehreinrichtungen und im privaten Haushalt. In Unternehmen und Lehreinrichtungen werden Thin Clients überwiegend als Terminals im klassischen Sinne benutzt. Sie booten über das Netzwerk ein Betriebssystem, mit dem anschließend gearbeitet werden kann. Dies bietet dem Administrator viele Vorteile. Sicherheitsaktualisierungen und Softwareaktualisierungen können zentral gewartet werden. Jeder Benutzer hat bei jedem Start des Thin Clients automatisch den aktuellen Stand des Betriebssystems und der darauf enthaltenen Software. In Kombination mit Terminalservern können die Thin Clients so konfiguriert werden, dass kein komplettes Betriebssystem über Netzwerk geladen werden muss. Hierfür wird ein Minimalsystem erzeugt, welches die nötigsten grafischen Bibliotheken und remote desktop client Software enthält. Dieses Minimalsystem kann gepackt auf 20MB reduziert werden. Nach dem Starten dieses Minimalsystemes über Netzwerk, kann dann ein auf einem Terminalserver zur Verfügung gestelltes Betriebssystem benutzt werden.

Eine Identifizierung der Thin Client Hardware findet selten statt. Dies birgt das Risiko, dass ein fremder Thin Client unbemerkt in das interne Netzwerk gehangen werden kann. Es ist denkbar die vorhandenen als auch fremde Thin Clients in ihrer Hardware zu modifizieren um mehr Möglichkeiten zu bekommen als gewollt ist. Ohne eine Identifizierung der Hardware kann dies nicht verhindert werden. Neben Thin Clients die über das Netzwerk booten gibt es auch welche die einen integrierten Speicher besitzen. Ein Angreifer kann diesen Speicher benutzen um ein eigenes Betriebssystem einzuspielen. Es ist also notwendig, nicht nur die reine Hardware zu identifizieren, sondern auch die Software samt Betriebssystem, welche auf dem Thin Client zum Einsatz kommt. Der Angreifer kann durch ein eigenes Betriebssystem unter Anderem das Netzwerk abhören oder massiv stören, da er alle Rechte auf dem eigenen System besitzt und die entsprechende Software.

Heutzutage gibt es immer mehr Unternehmen, die ihren Mitarbeitern anbieten von zu Hause aus zu arbeiten. Dies wird oft als *Homeoffice* bezeichnet. Die Mitarbeiter müssen dann eine Möglichkeit haben sich mit dem Firmennetzwerk verbinden zu können. Hierfür wird meistens VPN eingesetzt. Leider bietet dies nur der Kommunikation über das Internet Schutz. Einem Angreifer nützt es in diesem Falle nichts, die Verbindung abzuhören. Allerdings bietet VPN kein Schutz vor Schadsoftware, die sich auf dem Rechner befindet, der zur Einwahl in das Firmennetzwerk benutzt wird. Meistens bekommen die Mitarbeiter, die von zu Hause aus arbeiten, einen Firmenlaptop. Auf diesen kann aber Software installiert werden, die nicht vom Arbeitgeber gewünscht ist. Zudem kann man sich mit dem Firmenlaptop außerhalb des VPNs im Internet aufhalten. Falls über diesen Weg Schadsoftware auf den Firmenlaptop gelangt, kann diese, sobald eine VPN Verbindung besteht, in das Firmennetzwerk gelangen. Somit ist im Bereich des Homeoffice eine Überprüfung der Hardware und Software noch kritischer, als im internen Netzwerk. Ein Angreifer kann außerhalb der Arbeitsstätte mehr kriminelle Energie walten lassen, ohne bemerkt zu werden. In komplexen Firmennetzwerken, in denen auch die Töchterunternehmen eingebunden sind, muss ebenfalls sehr sensibel mit dem Thema Sicherheit, Identifizierung und Authentifizierung umgegangen werden. Falls ein

Angreifer Zugriff zu dem Netzwerk eines Tochterunternehmens bekommt, könnte er sich als ein Mitarbeiter ausgeben und in das Hauptfirmennetzwerk gelangen.

1.1 Angenommene Umgebung

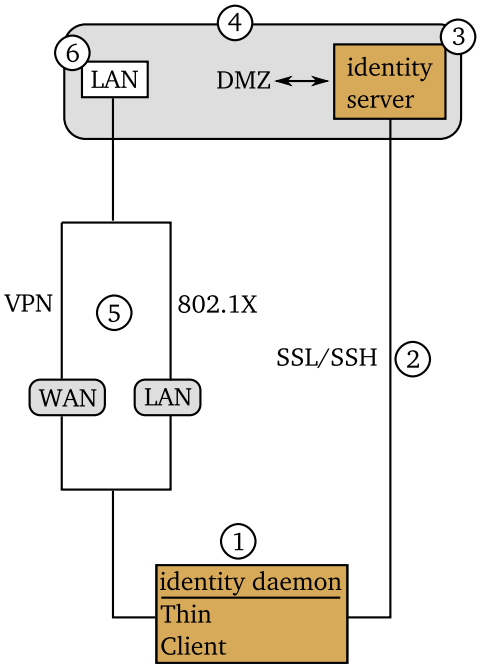

Abbildung (1) Überblick der Umgebung

1. identity daemon - *idd*
   Überprüfung der Hard-/Software

2. SSL / SSH Verbindung mit *ids*
   bevor *idd* sich nicht am *ids* identifiziert hat, darf der Thinclient nicht in das Intranet

3. identity server - *ids*
   kennt alle Thinclients und überprüft diese, danach wird erst Zugang zum Intranet gewährt

4. DMZ
   regelt Zugriff auf Intranet/Terminalserver in Abhängigkeit des *ids* welcher im Internet hängt

5. WAN / LAN Verbindung mit dem Intranet

in Abhängigkeit vom *ids* wird eine Verbindung zum Intranet mittels VPN oder 802.X aufgebaut

6. Intranet

Es wird davon ausgegangen, dass ein Client-Server Modell benutzt wird. Der Server steht im Unternehmen. Er besitzt die Funktionalitäten des DHCP, VPN, etc. Dieser Server muss in der Lage sein, den Zugriff von einzelnen IPs, Zertifikaten, usw. zu erlauben oder zu verweigern.

Der Client ist ein leistungsschwacher Rechner, bzw. Thinclient mit einem Minimalsystem. Der Thinclient hat einen Einkernprozessor, zwei Gigabyte Arbeitsspeicher und 512MB Flashspeicher. Das Minimalsystem ist gepackt auf dem Flashspeicher etwa 20MB klein und entpackt im Arbeitsspeicher etwa ein Gigabyte. Es enthält nur die nötigsten Komponenten. Unter Anderem xserver für die grafische Darstellung, SSH, Remote Desktop Client und VPN.

1.2 Zielsetzung

In dieser Thesis soll ein Konzept samt Umsetzung entstehen, womit eine Modifizierung an Software und Hardware erkannt werden kann. Damit wird die Identität eines Clients sichergestellt und kann authentifiziert werden. Dies beinhaltet unter Anderem die folgenden Punkte:

- Kommunikationsprotokoll zwischen *idd* und *ids*
- der *idd* muss gegen Manipulation und Nachbau geschützt werden
- alle Komponenten sollen opensource sein

Das Ziel ist ein Prototyp, der demonstriert was mit opensource Mitteln machbar ist. Es ist offensichtlich, dass Hardware immer stärker ist als Software. Das bedeutet, dass jede Sicherheitssoftware mit genügend Zeit und (krimineller) Energie umgangen oder geknackt werden kann, solange man hardwareseitigen Zugriff besitzt. Somit wird eine Lösung entworfen, bei der der Nutzen von krimineller Energie geringer ist, als die Kosten für diese.

1.3 Abgrenzung

Der entstehende Prototyp soll nicht mit ähnlichen Lösungen die kommerziell als Blackbox vertrieben werden konkurrieren. Das heißt, es ist nicht notwendig an die Benutzerfreundlichkeit

zu denken. Es reichen Konsolenprogramme. Eine Komponente, die komfortabel sehr viele eindeutige *identitydaemons* erzeugen und verwalten kann, wird in diesem Stadium auch nicht benötigt. Bei dem *idd* wird auf die Schnittstelle zur Verbindungsherstellung über VPN oder 802.X verzichtet. Auf die Schnittstelle zwischen *ids* und DMZ wird ebenfalls verzichtet.

Das Konzept wird ab dem Betriebssystem angesetzt. Somit wird bewusst auf folgende Komponenten vorerst verzichtet:

- Chain of trust
- alternatives BIOS wie z.B. *coreboot*
- Verwendung eines TPM Chips

1.4 Hinweise

Es wird meistens von Unternehmen gesprochen. Dies gilt allerdings genau so für Lehreinrichtungen, private Haushalte und so weiter.

Die Seitenangaben der verwandten Literatur haben das Format „S. 1“oder „S. 9/3“. Im letzteren Fall bedeutet dies, dass die Information in der elektronischen Version auf der neunten Seite zu finden ist und in der gedruckten Fassung auf der Seite mit der Seitennummer drei. Unter jedem Abschnitt stehen die verwandten Literaturstellen.

Die jeweils zu einem Codeausschnitt gehörenden Quelldateien stehen unter dem Codeausschnitt. Eine Liste der Dateien, aus denen Codeauszüge genommen wurden, steht im Quellcodeverzeichnis am Ende dieser Thesis.

Die verwandten Abkürzungen können im Abkürzungsverzeichnis, am Ende der Thesis, nachgelesen werden. Dort befinden sich auch Anhang, Abbildungsverzeichnis, Tabellenverzeichnis und Literaturverzeichnis.

Als Brotschrift oder Grundschrift wird Charter BT (Bitstream Charter) mit 11 Punkt benutzt. Für pseudo verschlüsselte Texte in Beispielen wird die Font Semafor benutzt. Das Textsatzsystem ist LaTeX.

Die folgenden Farben werden verwendet:

- #D7AA59 für Auszeichnungen

- #AA1516 für besondere Hinweise

- rgb(0.5, 0.5, 0.5) für Kommentare

Am Ende dieser Thesis befindet sich eine CD mit der elektronischen Version dieser Thesis als PDF. Des Weiteren befindet sich dort auch der C und LaTeX Quellcode. Im Anhang steht die Verzeichnisstruktur der CD.

Diese Thesis wurde selbst gebunden. Dabei wurde eine Mischform aus der gewohnten Hardcover Bindung und der alten japanischen, bzw. asiatischen Bindung benutzt. Für das Hardcover wurde Finnpappe als Innenleben genommen. Der Einband besteht aus Momigami. Als Vorsatz wurde Simili gewählt. Der Buchbindezwirn ist ein antiker Handheftzwirn aus Flachs.

Nach dieser Einleitung folgt das Kapitel *Grundlagen und Techniken*. Dort wird die fachspezifische Theorie und Begriffe erläutert. Daran schließt das Kapitel *Entwurf und Design* an. In diesem werden verschiedene Lösungsansätze besprochen und eines ausgewählt. Zudem wird hier das Kommunikationsprotokoll definiert und welche Komponente welche Funktionen bereitstellt. Danach wird im Kapitel *Implementierung* auf die Umsetzung der Kernkomponenten eingegangen. Hier finden sich auch Details zu den verwendeten Bibliotheken. Im vorletzten Kapitel *Angriffsszenarien und Härtetest* wird besprochen welche potentiellen Angriffe möglich sind und wie diese verhindert werden. Das letzte Kapitel *Fazit und Ausblick* enthält Informationen zu der Sicherheit und Einsatzfähigkeit des Prototypen. Zudem wird aufgelistet, wie der Prototyp verbessert und weiterentwickelt werden kann.

2 Grundlagen und Techniken

In diesem Kapitel werden einige grundlegende Begriffe, Verfahren und Techniken erläutert. Dazu gehört im Allgemeinen die Kryptologie. Ebenso werden Hashes und Chiffren besprochen. Anschließend wird auf eine sichere Kommunikationsverbindung und die Authentifizierung und Autorisierung eingegangen. Zudem werden die Themen Trusted System / Computing, PGP und VNC angesprochen. Zuletzt wird auf die Sicherheit von Spielekonsolen und Kopierschutz hingewiesen.

2.1 Kryptologie

Die Kryptologie fasst die Begriffe Kryptographie und Kryptoanalyse zusammen. Die Kryptographie beschäftigt sich mit dem Problem eine Nachricht sicher an einen Kommunikationspartner zu übermitteln. Sicher bedeutet in diesem Zusammenhang, dass die Nachricht nicht von einem Dritten gelesen werden kann. Hierfür wird der Klartext verschlüsselt. Dieses Problem beschäftigt die Menschheit schon seit dem alten Ägypten. Ein einfaches Schema einer verschlüsselten Nachrichtenübertragung zeigt die Abbildung 2. Inzwischen gehört zur Kryptographie allgemein die Wissenschaft der Informationssicherheit. Hierzu gehören nicht nur Verschlüsselungssysteme sondern auch Konzepte von Informationssystemen und Protokollen. Die Kryptoanalyse hingegen beschäftigt sich mit dem Gegensätzlichen. Hier wird versucht aus einer verschlüsselten Nachricht Informationen zu gewinnen. Dazu gehört aber auch die Untersuchung, wie sicher oder anfällig ein Verschlüsselungsverfahren ist.

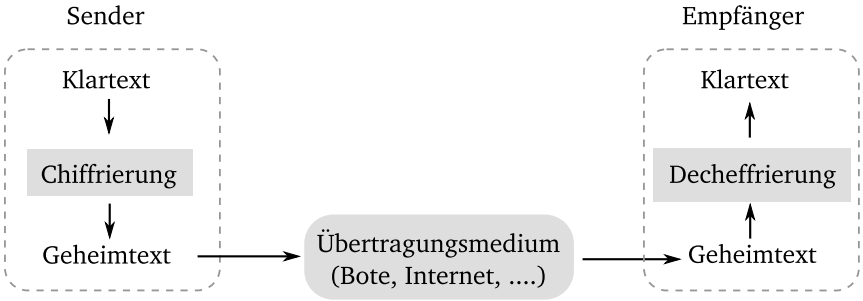

Abbildung (2) Schema einer Ver-/Entschlüsselung [28, S. 11]

2.1.1 Einmalpasswort / Sitzungsschlüssel

Ein *Einmalpasswort* wird im Gegensatz zu einem statischen Passwort und wie der Name vermuten lässt, nur ein Mal Benutzt. Es bietet einen erhöhten Schutz gegen Angriffe, die auf abhören beruhen. Es nützt dem Angreifer nichts, viele Nachrichten zu sammeln, da jede

Nachricht anders verschlüsselt ist. Ein solches Einmalpasswort muss zufällig generiert werden, damit folgende Passwörter nicht erraten werden können. Diese Sorte von Passwörtern wird in hybriden Verschlüsselungsverfahren eingesetzt. Dazu gehören zum Beispiel SSL/TLS und OpenPGP. Dabei wird bei jeder Sitzung ein zufälliger Schlüssel, für die symmetrische Verschlüsselung, generiert und über eine asymmetrische Verschlüsselung ausgetauscht. Das Einmalpasswort ist auch unter den Namen *One-Time-Password*, *Sitzungsschlüssel* und *Sesion-Key* bekannt.

2.1.2 Diffie-Hellman-Schlüsselaustausch

Damit zwei Kommunikationspartner eine verschlüsselte Verbindung aufbauen können, benötigen beide einen gemeinsamen geheimen Schlüssel. Das Problem hierbei ist diesen geheimen Schlüssel über eine unsichere Verbindung zu übertragen. Der Diffie-Hellmann-Schlüsseltausch bietet eine Lösung für das Problem und ist die Grundlage für weitere Verfahren, wie zum Beispiel die *Elgamal-Verschlüsselung*.

Alice				Bob		
Geheim	Öffentlich	Berechnet	Versendet	Berechnet	Öffentlich	Geheim
a	p, g		p, g			b
a	p, g, A	$g^a \ mod \ p = A$	$A \rightarrow$		p, g	b
a	p, g, A		$\leftarrow B$	$g^b \ mod \ p = B$	p, g, A, B	b
a, s	p, g, A, B	$B^a \ mod \ p = s$		$A^b \ mod \ p = s$	p, g, A, B	b, s
Primzahl p und Primitivwurzel $g \ mod \ p$ mit $2 \leq g \leq p-2$						
a, b Zufallszahl aus $\{1, \ldots, p-2\}$						
$s = B^a \ mod \ p = \left(g^b \ mod \ p\right)^a \ mod \ p = g^{ba} \ mod \ p =$						
$g^{ab} \ mod \ p = (g^a \ mod \ p)^b \ mod \ p = A^b \ mod \ p = s$						

Tabelle (1) Diffie-Hellman-Schlüsseltausch Schema [79]

Alice				Bob		
Geheim	Öffentlich	Berechnet	Versendet	Berechnet	Öffentlich	Geheim
5	13, 2		13, 2			7
5	13, 2, 6	$2^5 \ mod \ 13 = 6$	6 →		13, 2	7
5	13, 2, 6		← 11	$2^7 \ mod \ 13 = 11$	13, 2, 6, 11	7
5, 7	13, 2, 6, 11	$11^5 \ mod \ 13 = 7$		$6^7 \ mod \ 13 = 7$	13, 2, 6, 11	7, 7

Tabelle (2) Diffie-Hellman-Schlüsseltausch Beispiel [78]

Das Verfahren ist sicher gegenüber abhören. Allerdings scheitert es, sobald sich jemand zwischen die Verbindung schaltet und den Datenverkehr manipulieren kann. Dagegen schützen wiederum Verfahren wie das *Station-to-Station (STS) protocol* oder auch *digitale Signaturen* und *Message Authentication Code*.

[78] [79] [6, S. 129-132] [21, S. 326-332, 413-415]

2.1.3 man in the middle attack

Bei einem MITM Angriff gelingt es einem Angreifer sich zwischen zwei Kommunikationspartner zu schalten. Der Angreifer ist dann nicht nur in der Lage die Kommunikation mitzulesen, sondern auch zu manipulieren. Mit dieser Art von Angriffen wird versucht den geheimen Schlüssel von Kommunikationspartnern zu erfahren oder sich als jemand anderes auszugeben. Der Diffie-Hellman-Schlüsseltausch ist in seiner Ursprungsform gegen MITM Angriffe nicht geschützt. Das folgende Sequenzdiagramm zeigt das Schema eines MITM-Angriffes auf den D-H-Schlüsseltausch.

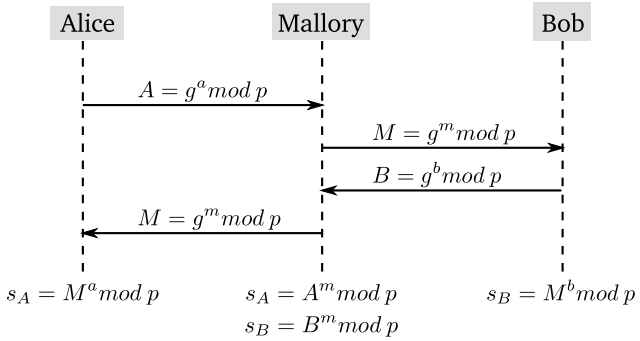

Abbildung (3) Schema eines MITM Angriffes auf D-H-Schlüsseltausch [78]

[78] [92] [93] [21, S. 558-560] [22, S. 406-417]

2.1.4 Station-to-Station (STS) protocol

Das STS Protokoll wirkt dem Problem von MITM-Attacken entgegen. Dadurch können sich beide Kommunikationspartner sicher sein, dass sie mit dem jeweils anderen kommunizieren. Die Implementierung hängt vom Anwendungsfall ab. Dabei kommen digitale Signaturen, MAC oder digitale Zertifikate zum Einsatz. Es ist auch eine Kombination von den genannten Techniken möglich und je nach Umgebung notwendig. Wenn man voraussetzt, dass die öffentlichen Schlüssel der Kommunikationsteilnehmer sicher verteilt sind kann ein *Basic STS* wie folgt aussehen.

1. Alice generiert eine Zufallszahl x und sendet $A = g^x mod\, p$ an Bob.
2. Bob generiert eine Zufallszahl y und berechnet $B = g^y mod\, p$.
3. Bob berechnet den gemeinsamen geheimen Schlüssel $s = A^y mod\, p$.
4. Bob konkateniert BA, signiert das Ergebnis mit seinem geheimen Schlüssel und verschlüsselt die Signatur mit s. Anschließend sendet er die verschlüsselte Signatur und B an Alice.
5. Alice berechnet den gemeinsamen geheimen Schlüssel $s = B^x mod\, p$.
6. Alice entschlüsselt und verifiziert Bobs Signatur.
7. Alice konkateniert AB, signiert das Ergebnis mit ihrem geheimen Schlüssel und verschlüsselt die Signatur mit s. Die verschlüsselte Signatur sendet sie an Bob.
8. Bob entschlüsselt und verifiziert Alices Signatur.[1]

Formal geschrieben lässt es sich wie in folgender Tabelle darstellen. Alle Exponenten befinden sich in der Gruppe von p.

[1] 107.

(1)	Alice → Bob	:	g^x
(2)	Alice ← Bob	:	$g^y, E_s\left(S_B\left(g^y, g^x\right)\right)$
(3)	Alice → Bob	:	$E_s\left(S_A\left(g^x, g^y\right)\right)$

Tabelle (3) Basic STS Schema [107]

Um sicherzustellen, dass die verwendeten öffentlichen Schlüssel auch dem vorgegebenen Benutzer gehören, können zusätzlich Zertifikate benutzt werden. Digitale Zertifikate bestätigen, dass ein öffentlicher Schlüssel einer bestimmten Person gehört. Dies nennt sich dann *Full STS*.

(1)	Alice → Bob	:	g^x
(2)	Alice → Bob	:	$g^y, Cert_B, E_s\left(S_B\left(g^y, g^x\right)\right)$
(3)	Alice → Bob	:	$Cert_A, E_s\left(S_A\left(g^x, g^y\right)\right)$

Tabelle (4) Full STS Schema [107]

[107] [17, S. 88]

2.1.5 zero knowledge

Als *zero-knowledge-Verfahren* bezeichnet man Protokolle bei denen das Geheimnis nicht preisgegeben wird. Ein Beweiser kann dem Verifizierenden mit einer gewissen Wahrscheinlichkeit zeigen, dass er im Besitzt eines Geheimnisses ist, ohne dem Verifizierendem das Geheimnis zu verraten. Das zero-knowledge-Verfahren kann Mathematisch bewiesen werden. Ein abstraktes Beispiel kann anhand der Abbildung 4 (eine Höhle in der rechts ein Tor ist) erläutert werden.

Alice will Bob beweisen, dass sie den Schlüssel zu dem Tor in der Höhle besitzt. Dafür geht sie zufällig in den oberen oder unteren Gang. Bob geht danach in die Höhle und sagt Alice aus welchem Gang sie wieder zurück kommen soll. Wenn Alice den unteren Gang genommen hat und Bob ebenfalls den unteren Gang wählt, dann muss Alice nicht einmal den Schlüssel benutzen. Falls Bob einen anderen Gang wählt wie Alice, dann muss Alice zwar den Schlüssel benutzen, aber sie muss ihn Bob nicht zeigen.

Abbildung (4) abstraktes zero knowledge Beispiel

[6, S. 202-204] [119] [120]

2.2 Hashes

In diesem Kapitel wird erläutert was Hashfunktionen sind und wie sie in der Kryptographie eingesetzt werden. Als Hashbeispiele werden MD5 und SHA benutzt. Was der Begriff Salt bedeutet wird ebenfalls erklärt.

2.2.1 Hashfunktion

Eine Hashfunktion bildet eine Zeichenfolge beliebiger Länge auf eine Zeichenfolge fester Länge ab. In der Kryptographie verwendet man sogenannte kryptographische Hashfunktionen oder auch Einweg-Hashfunktionen. Bei diesen Funktionen ist es sehr schwer von dem Hash auf den Klartext zu schließen. Hashes werden dazu benutzt um die Integrität von Nachrichten, dazu zählen auch Dateien, sicherzustellen. Eine Nachricht hat immer den selben Hashwert. Des weiteren benutzt man Hashes in Datenbanken. Wenn man schnell in einer bestimmten Spalte suchen möchte, dann legt man einen Index über diese Spalte. Die Datenbank erstellt dann eine Hashtabelle über die Werte in dieser Spalte. Bei einer Suchanfrage in dieser Spalte kann nun sehr schnell der Hashwert von der Anfrage berechnet werden und nachgesehen werden wo sich die entsprechende Zeile befindet. Ein wichtiges Einsatzgebiet von Hashes ist aber auch die Verschlüsselung von Passwörtern oder als Hilfsfunktion bei Signaturen. Unix Systeme speichern zum Beispiel die Passwörter der Benutzer als Hash im Dateisystem ab. Bei einem Anmeldevorgang wird das eingegebene Passwort vom Benutzer gehasht und mit dem Wert in der Passwortdatei verglichen. Wenn nun die Passwortdatei entwendet wird, kann der Angreifer die gehashten Passwörter nicht leicht entschlüsseln.

Hashes müssen kollisionsresistent sein. Dies bedeutet, dass es schwer sein muss zwei Nach-

richten m_1 und m_2 zu finden für die gilt $h(m_1) = h(m_2)$. Dabei unterscheidet man schwach und stark kollisionsresistente Hashfunktionen. „Bei schwach kollisionsresistenten Hashfunktionen ist es praktisch nicht durchführbar, zu einem gegebenen Hashwert $h(m)$ eine Nachricht m^* mit dem gleichen Hashwert $h(m^*) = h(m)$ zu finden. Bei stark kollisionsresistenten Hashfunktionen ist es praktisch nicht durchführbar, zwei Nachrichten m_1 und m_2 mit dem gleichen Hashwert $h(m_1) = h(m_2)$ zu finden.“[2] Für die beiden Definitionen gibt es jeweils unterschiedliche Angriffe, die hier nicht weiter besprochen werden.

[62, S. 95-97] [6, S. 167-169] [21, S. 471-476] [88] [91] [87]

2.2.2 MD5

Der Message-Digest-Algorithm-5 ist eine kryptographische Hashfunktion. Er wird oft bei Downloads benutzt um die heruntergeladene Datei auf Integrität zu überprüfen. Für die Signierung wird vom MD5 abgeraten. Es sind erfolgreiche Angriffe auf den Algorithmus bekannt um Kollisionen zu erzeugen. Der neueste Angriff wurde 2012 vom Flame Schädling benutzt um ein Zertifikat zu fälschen. Mit diesem Zertifikat kann der Update Mechanismus von Windows XP/Vista/7 getäuscht werden. Mit MD5 werden auch gerne Benutzerpasswörter von Webseiten verschlüsselt. Davon wird ebenfalls abgeraten wegen den bereits erfolgreichen Angriffen und rainbow tables.

MD5 bekommt eine Nachricht variabler Länge als Eingabe und erzeugt einen 128-bit *fingerprint* oder *message digest*. Bei Signaturverfahren kann eine Nachricht mit MD5 komprimiert werden bevor sie Verschlüsselt wird. Dadurch benötigt die Verschlüsselung weniger Zeit, da die Eingabe kürzer ist. Dies kommt bei public-key Verschlüsselungssystemen zum Einsatz.

In dieser Thesis wird nicht näher auf die genaue Funktionsweise des Algorithmus eingegangen und auch nicht verwendet. MD5 wurde der Vollständigkeit halber aufgeführt und da er noch trotz seiner bekannten Schwächen weit verbreitet ist. Zuletzt noch eine schematische Darstellung des Algorithmus und einige Beispiele.

[2] 62, S. 97.

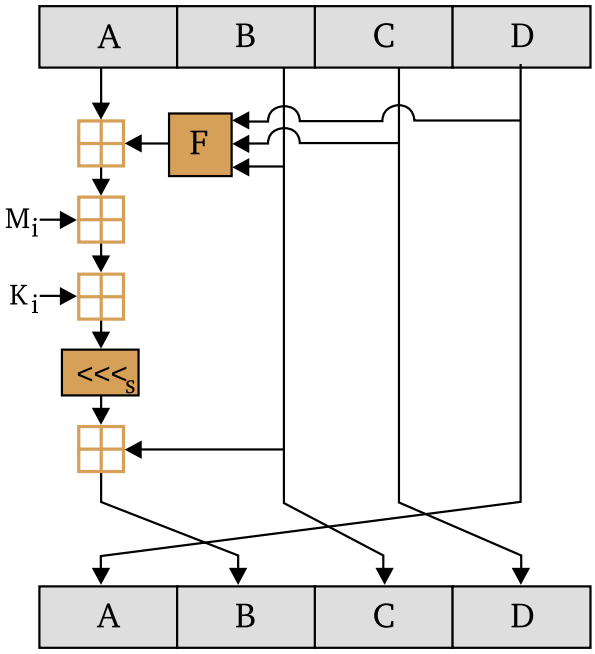

Abbildung (5) Schema einer MD5 Operation [61]

Der Algorithmus arbeitet in vier Runden, in der jeweils eine andere *F-Funktion* benutzt wird. In jeder Runde werden 16 Operationen durchgeführt was zu insgesamt 64 Operationen führt.

- $F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
- $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neq Z)$
- $H(X,Y,Z) = X \oplus Y \oplus Z$
- $I(X,Y,Z) = Y \oplus (X \vee \neq Z)$
- [21, 95]

```
1 ruby -r 'digest' -e 'puts Digest::MD5.hexdigest ""'
2 d41d8cd98f00b204e9800998ecf8427e
3
4 ruby -r 'digest' -e 'puts Digest::MD5.hexdigest "password"'
5 5f4dcc3b5aa765d61d8327deb882cf99
6
7 ruby -r 'digest' -e 'puts Digest::MD5.hexdigest "Password"'
8 dc647eb65e6711e155375218212b3964
```

Quelltext (1) Beispiel md5

[21, S. 489-495] [51] [95] [94]

2.2.3 SHA

Secure Hash Algorithm ist eine kryptographische Hashfunktion. Von dieser existieren inzwischen verschiedene Implementierungen. SHA-0 wurde kurz nach Veröffentlichung als unsicher eingestuft, worauf hin SHA-1 veröffentlicht wurde. Dieser behob die Fehler von SHA-0 und galt als sicher, bis 2005 Sicherheitsmängel gefunden wurden. Deshalb wird die SHA-2 Familie empfohlen. SHA-2 weicht in der Implementierung grundlegend der Implementierung von SHA-1 ab. Der SHA-1 ist zur Zeit noch verbreiteter als SHA-2, aber in dieser Thesis wird nur auf SHA-2 eingegangen. SHA-2 hat fünf Formen die in der Tabelle 5 entnommen werden können. Hier wird SHA-256 betrachtet.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Tabelle (5) Secure Hash Algorithm Properties [48, S. 8]

Der SHA-256 verarbeitet Nachrichten variabler Länger und gibt einen 256-Bit fingerprint aus. Die Verwendung des Hashes ist eine ähnliche wie bei MD5. Er wird für die Überprüfung der Integrität benutzt oder um Passwörter verschlüsselt abzulegen. Protokolle wie SSH, TLS, PGP, SSL und andere implementieren den SHA-2. Es gibt stand 2012 keine bekannten erfolgreichen Angriffe auf die volle Rundenzahl von 64. Bis auf die schematische Darstellung einer Runde und ein paar Beispielen wird nicht weiter auf die interne Funktionsweise des SHA-256 eingegangen.

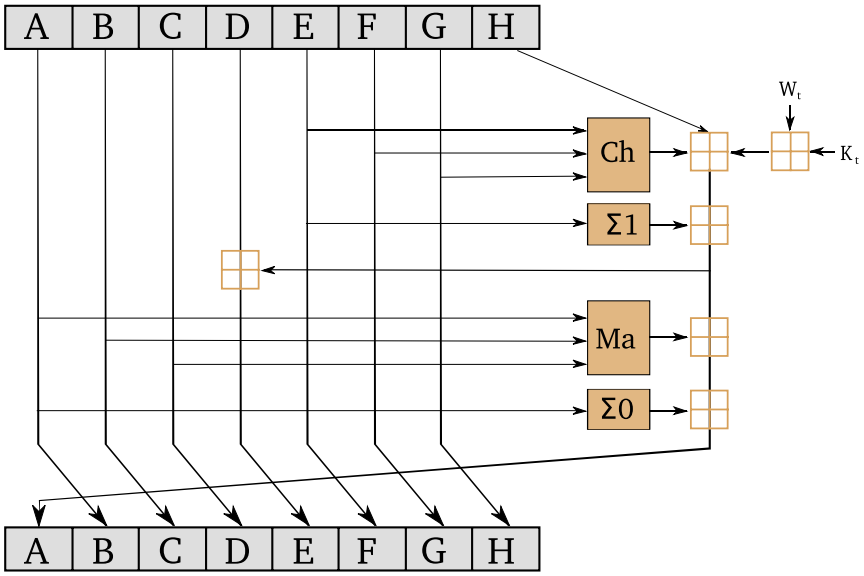

Abbildung (6) Schema einer SHA-2 Operation [39]

```
ruby -r 'digest' -e 'puts Digest::SHA2.hexdigest ""'
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

ruby -r 'digest' -e 'puts Digest::SHA2.hexdigest "password"'
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

ruby -r 'digest' -e 'puts Digest::SHA2.hexdigest "Password"'
e7cf3ef4f17c3999a94f2c6f612e8a888e5b1026878e4e19398b23bd38ec221a
```

Quelltext (2) Beispiel sha256

[21, S. 495-515] [48] [103] [106]

2.2.4 Salt

Salt bedeutet Salz und wird bekanntermaßen zum Würzen benutzt. In der Kryptographie benutzt man salts meistens um Passwörter sicher zu speichern.

Schlecht umgesetzte Webseiten sichern Passwörter im Klartext ab. Der Angreifer muss somit nur die Datenbank dazu bringen die entsprechende Tabelle auszugeben. Dies ist bereits oft über SQL Injection passiert. Einen Schutz dagegen bieten Einwegfunktionen wie zum Beispiel kryptographische Hashfunktionen. Der Angreifer muss nun nicht nur an die Passwörter herankommen sondern auch die zugehörigen Klartextpasswörter herausfinden. Allerdings ist

dies heutzutage auch kein großes Hindernis. Im Internet lassen sich sogenannte *rainbow tables* finden. Dies sind Tabellen in denen Klartext und der zugehörige Hash vor berechnet sind. Bei *Wörterbuchattacken* werden solche rainbow tables benutzt. Um sich gegen solche Angriffe zu schützen muss man die Klartextpasswörter gehörig versalzen, bevor man den Hash berechnet.

Der salt ist eine zufällige Zeichenkette. Diese wird an den Klartext angehangen und dann gehasht. Der Hash wird zusammen mit dem salt in der Datenbank hinterlegt. Solch verschlüsselte Passwörter nutzen einem Angreifer nicht mehr viel. Er müsste eine rainbow table aufbauen die zusätzlich alle Variationen des salt enthält. Solche Tabellen sind viel zu groß. Der Angreifer kann immer noch Wörterbuchattacken oder über Brutefoce versuchen den Klartext zu einem versalzenen Hash herauszufinden. Allerdings dauert dies bei ausreichend langen Passwörtern und salts zu lange als das es sich lohnen würde. Salt verhindert zudem, dass bei zwei identischen Passwörtern der selbe Hash entsteht.

```
1  ruby -r 'digest/sha2' -e \
2    'puts Digest::SHA2.hexdigest "--password--"'
3  f9a1268fb341c91494350044d887938bdc8e0291846dbfef1811e2db0b311a90
4
5  ruby -r 'digest/sha2' -e \
6    'puts Digest::SHA2.hexdigest "--salz--password--"'
7  fd7ca0bcd0639ff96006197d9d3f7eb66ce721186ad52a0c56dd5c1e1a9a6357
8
9  ruby -r 'digest/sha2' -e \
10   'puts Digest::SHA2.hexdigest "--SALZ--password--"'
11 9247c60cae31d1420b97d86efe925c73783d1132e84bfdf757a2b331672c28c0
```

Quelltext (3) Salt Beispiel sha256

[16] [22, S. 419] [23, S. 102] [102] [101]

2.3 Chiffren

Unter dem Begriff Chiffren versteht man für gewöhnlich die Verschlüsselung und Entschlüsselung von Nachrichten. Dies gilt ebenso für Texte, Dateien und so weiter. Als Chiffre versteht man auch eine verschlüsselte Nachricht. Unter der Grafik 7 werden die Block-/Stromchiffren sowie die asymmetrische und symmetrische Verschlüsselung aufgegriffen.

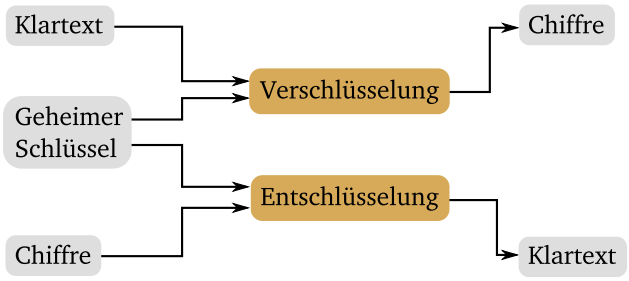

Abbildung (7) Schema einer Chiffrierung

2.3.1 block/strom Chiffren

Blockchiffren verschlüsseln wie der Name vermuten lässt blockweise. Das heißt, eine Nachricht die verschlüsselt werden soll wird zuerst in Blöcke gleicher Länge aufgeteilt. Anschließend wird jeder Block für sich verschlüsselt und die Blöcke wieder zusammen gesetzt. Dabei gibt es verschiedene Modi, die für verschiedene Einsatzzwecke geeignet sind. Bei Verwendung des falschen Modus kann ein Angreifer aus der verschlüsselten Nachricht Informationen gewinnen obwohl er die Nachricht nicht entziffern kann. Zu den bekannten Vertretern der Blockchiffren gehören DES, AES, Camellia, Serpent und Twofish.

ECB-Mode verschlüsselt jeden Block für sich. Dabei entsteht bei zwei gleichen Blöcken der gleiche verschlüsselte Block.

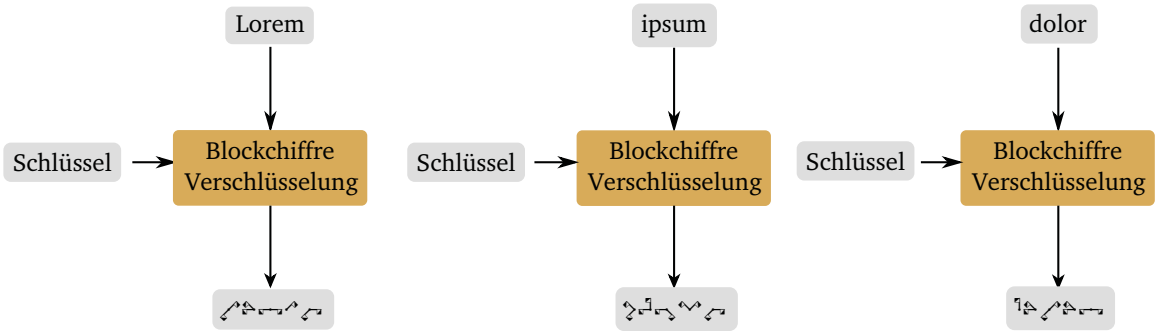

Abbildung (8) Schema des ECB Modus beim Verschlüsseln

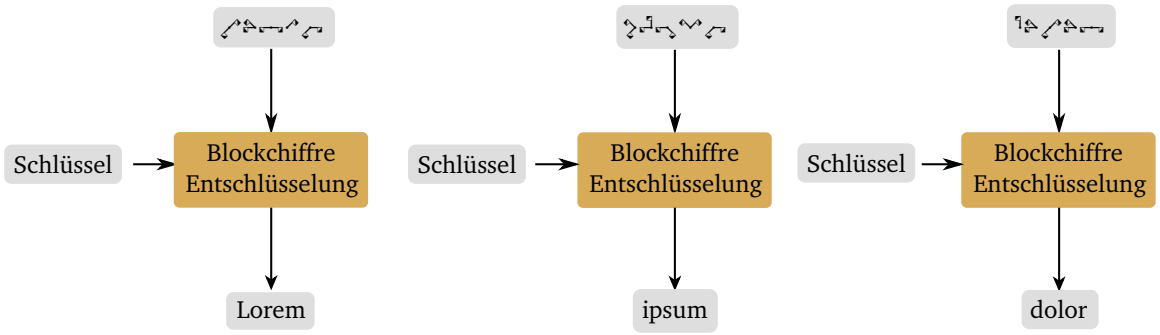

Abbildung (9) Schema des ECB Modus beim Entschlüsseln

CBC-Mode wendet vor der Verschlüsselung die XOR-Operation auf den Klartext mit dem vorhergehenden verschlüsselten Text an. Für den ersten Klartext wird ein Initialisierungsvektor benötigt. Dieser besteht aus einer zufälligen Zeichenkette. Dadurch ist jeder Block von dem vorhergehenden abhängig und kann nicht ohne weiteres von einem Angreifer ausgetauscht werden.

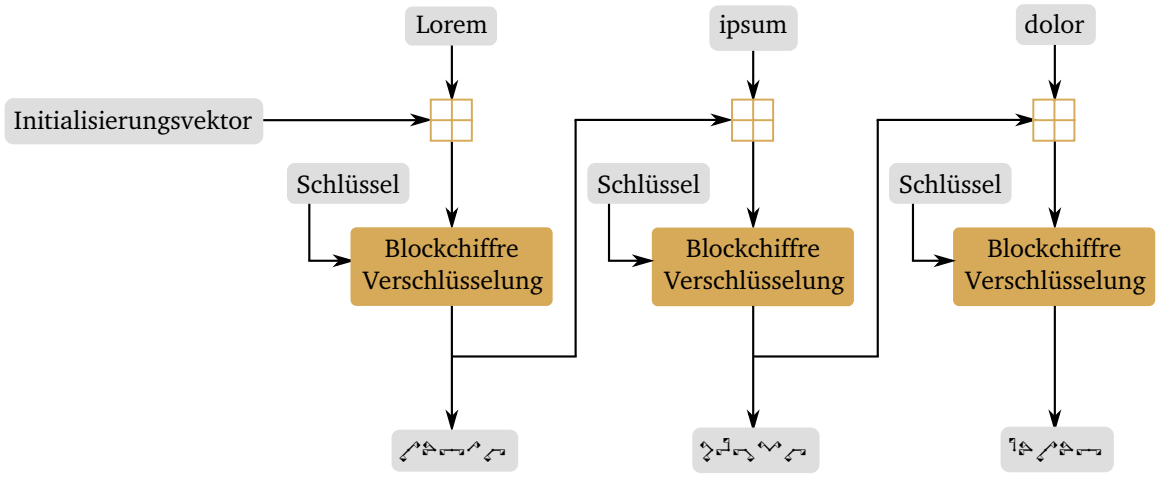

Abbildung (10) Schema des CBC Modus beim Verschlüsseln

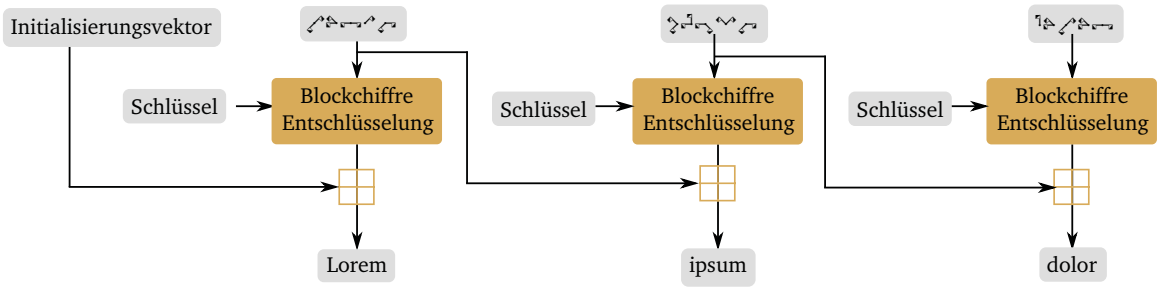

Abbildung (11) Schema des CBC Modus beim Entschlüsseln

CFB-Mode ist dem CBC-Mode ähnlich. Er hat eine Rückkopplung zum vorhergehenden Block. Allerdings wird bei diesem Verfahren der Initialisierungsvektor verschlüsselt und anschließend die XOR-Operation auf den Klartext angewandt. Bei den folgenden Blöcken wird jeweils der vorhergehende verschlüsselte Text verschlüsselt und dann XOR mit dem Klartext verbunden.

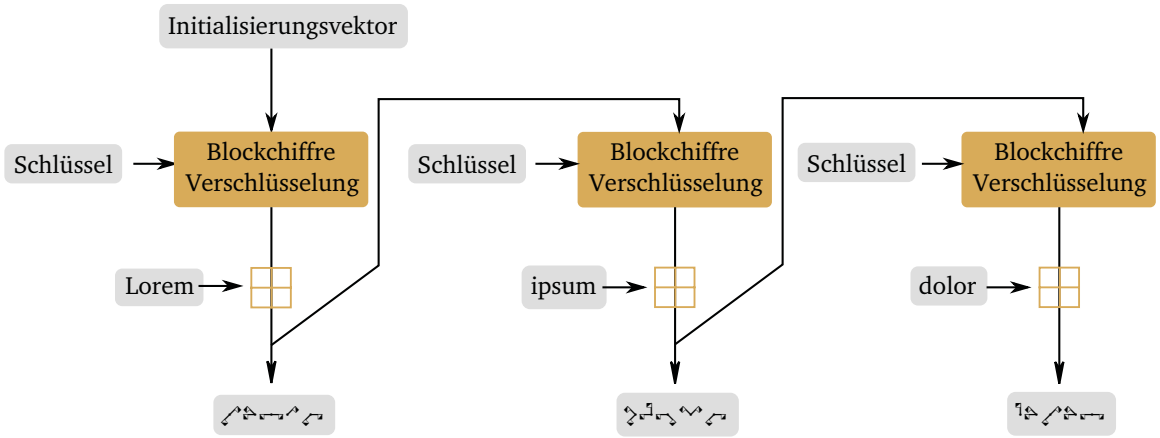

Abbildung (12) Schema des CFB Modus beim Verschlüsseln

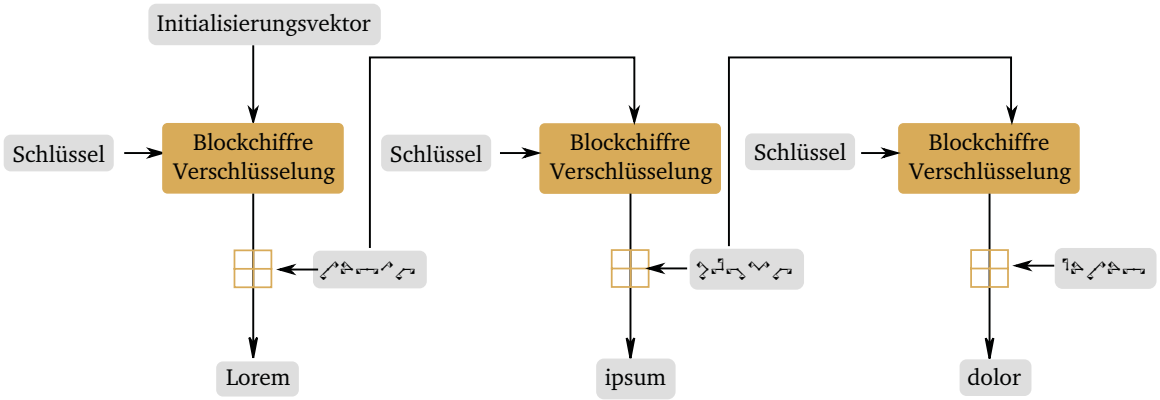

Abbildung (13) Schema des CFB Modus beim Entschlüsseln

OFB-Mode hat wie der CBC-Mode keine Rückkopplung. Er funktioniert ähnlich einem Stromchiffre indem Schlüsselströme blockweise erzeugt werden. Zuerst wird ein Initialisierungsvektor verschlüsselt. Der verschlüsselte IV wird XOR mit dem Klartext verbunden, um den verschlüsselten Text zu erhalten. Zudem wird der verschlüsselte IV als Eingabe für den nächsten Block benutzt und so wird die Ausgabe von der Verschlüsselung jeweils für den nächsten Block als Eingabe benutzt.

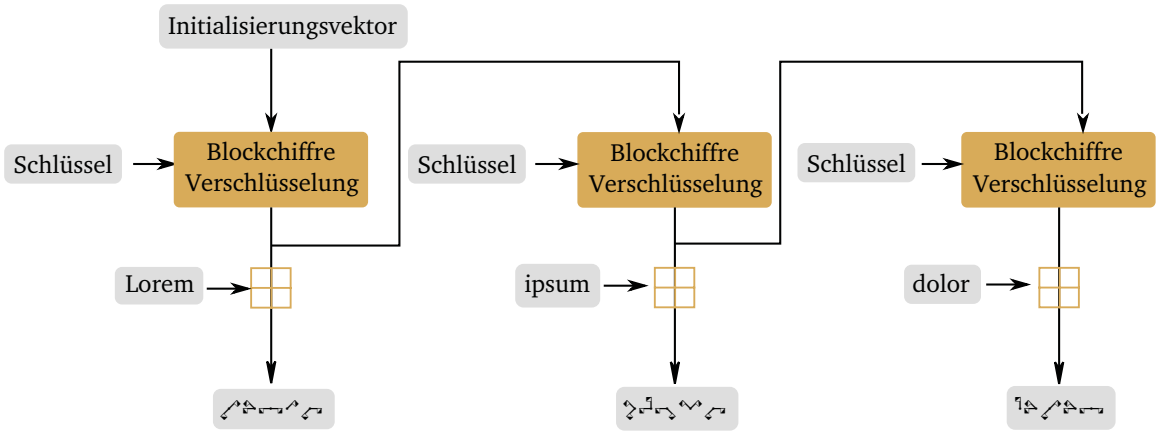

Abbildung (14) Schema des OFB Modus beim Verschlüsseln

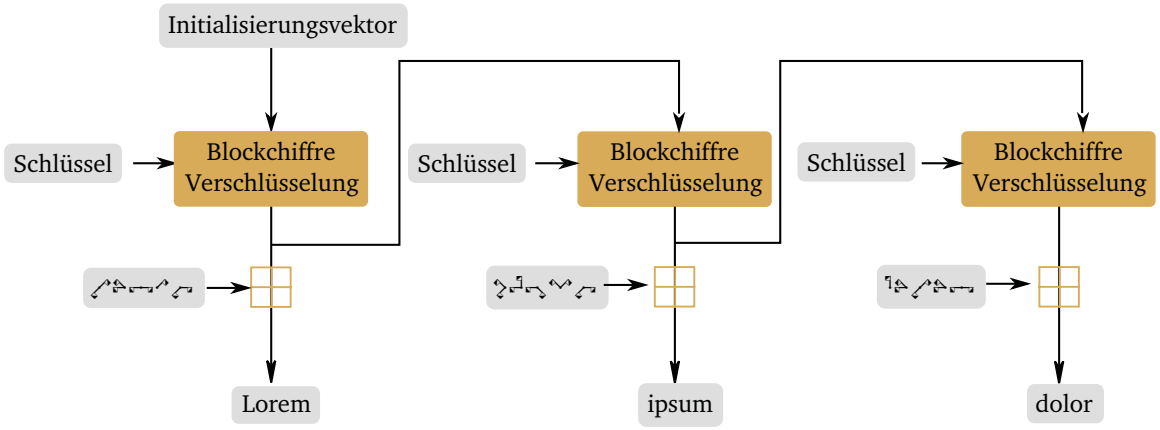

Abbildung (15) Schema des OFB Modus beim Entschlüsseln

Stromchiffren müssen im Gegensatz zu Blockchiffren nicht auf eine Eingabe bestimmter Länge (Block) warten. Sie verschlüsseln jedes Bit direkt. Dazu wird ein Schlüsselstrom benötigt. Um einen Schlüsselstrom zu erzeugen benutzt man oft Linear Feedback Shift Register und einen Schlüssel, bzw. eine zufällige Zeichenfolge. Daraus folgt eine große zufällige Zeichenfolge, die direkt mit den Eingabebits mittels XOR verbunden wird. Bei Echtzeitanwendungen bringt dies Vorteile, da nicht erst auf eine bestimmte Anzahl an Bits gewartet werden muss.

[6, S. 62-73] [21, S. 99-110, 49-52] [73] [72] [109] [108]

2.3.2 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung wird auch *public-private-key* Verfahren genannt. Wie der Name bereist vermuten lässt existiert bei diesem Verfahren ein Schlüsselpaar. Der öffentliche Schlüssel kann beliebig verteilt werden. Alle Nachrichten, die mit dem öffentlichen Schlüssel verschlüsselt werden, können nur mit dem zugehörigen privaten Schlüssel wieder entschlüsselt werden. Der private Schlüssel darf somit nie den Besitzer verlassen. Des weiteren ist es auch möglich mit dem privaten Schlüssel Nachrichten zu signieren und mit dem öffentlichen Schlüssel zu verifizieren. Der Vorteil von asymmetrischen Verfahren liegt darin, dass kein gemeinsamer geheimer Schlüssel bekannt sein muss. Der öffentliche Schlüssel darf Jedermann bekannt sein. Ein Nachteil dieses Verfahrens ist aber die geringe Geschwindigkeit.

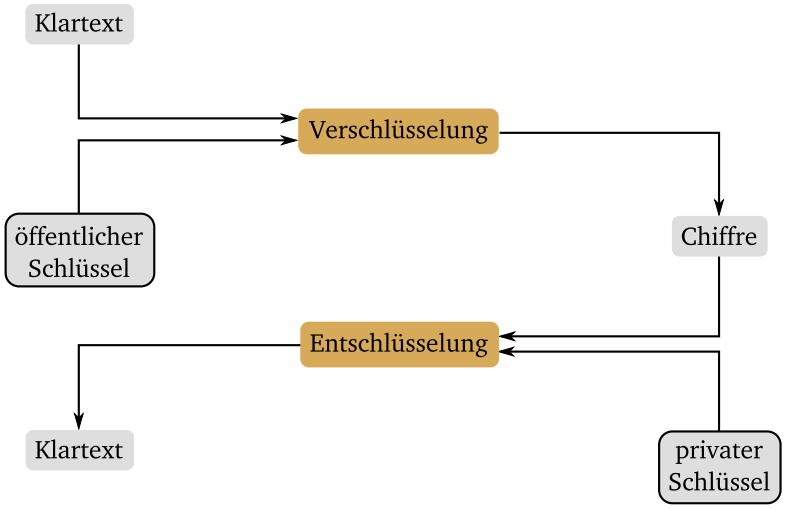

Abbildung (16) Schema einer asymmetrischen Verschlüsselung

[6, S. 56-57] [71] [100]

2.3.3 Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung gibt es nur einen Schlüssel mit dem verschlüsselt und entschlüsselt wird. Das Schema ist am Anfang des Kapitels in Abbildung 7 auf Seite 18 zu sehen. Der Nachteil bei diesem Verfahren besteht darin, dass zwei Kommunikationspartner diesen Schlüssel über eine *sichere* Verbindung austauschen müssen. Der Vorteil ist allerdings die große Geschwindigkeit. Daher wird bei Kommunikationssystemen gerne ein hybrides Verfahren eingesetzt. Zuerst wird mit einer asymmetrischen Verschlüsselung der geheime Schlüssel über eine unsichere Verbindung ausgetauscht und anschließend wird die Kommunikation symmetrisch mit dem gemeinsamen geheimen Schlüssel verschlüsselt. Zur Verschlüsselung und Entschlüsselung wird eines der bereits genannten Blockchiffren benutzt oder auch ein

Stromchiffre.

[6, S. 56f] [111] [110]

2.4 Sichere Kommunikationsverbindung

Um einen Kommunikationsweg sicher zu gestalten, müsste man diesen unter anderem gegen das Abhören und Manipulieren absichern. Beides ist sehr schwierig oder gar unmöglich im Internet. Es ist allerdings möglich einem Angreifer das Leben stark zu erschweren. Gegen das Abhören können verschlüsselte Verbindungen wie z.B. TLS oder SSH benutzt werden. Man kann zwar nicht verhindern, dass Daten manipuliert werden, aber man kann herausfinden *ob* Daten manipuliert wurden und das sie von einer bestimmten Person stammen. Hierzu werden Verfahren wie MAC, HMAC oder digitale Signaturen verwendet.

2.4.1 Message Authentication Code - MAC

MACs werden benutzt um die Integrität und Authentisierung von Nachrichten sicherzustellen. Dies bedeutet, dass zwei Kommunikationspartner verifizieren können, dass eine Nachricht nicht verändert wurde und dass diese von dem angenommenen Kommunikationspartner stammt. Hierfür gibt es zwei Verfahren. Block Cipher MAC - CMAC benutzen eine bereits vorhandene Blockverschlüsselung für die Erstellung des MAC. Das Verfahren hat Ähnlichkeiten mit dem CBC-Modus. Entsprechend wird bei Keyed Hash MAC - HMAC eine bereits vorhandene Hashfunktion für die Erstellung des MAC benutzt.

^_^	°_°
$\leftarrow MAC_K() \rightarrow$	
	$x, y = MAC_K(x)$
$\leftarrow x, y$	
$y' = MAC_K(x)$	
$y == y'$?	

Tabelle (6) Schema einer Kommunikation mittels MAC

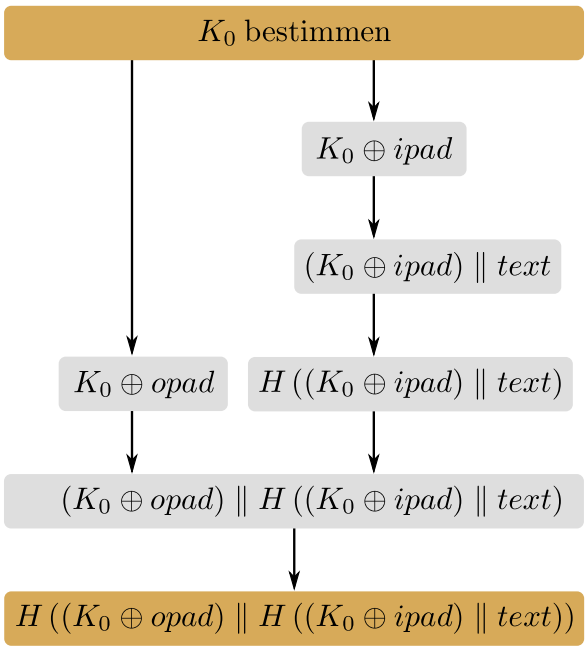

Abbildung (17) Schema eines HMAC [49, S. 11/5]

[6, S. 175f] [21, S. 528-533] [49]

2.4.2 Digitale Signatur

Digitale Signaturen dienen wie MACs der Integrität und Authentisierung von Nachrichten. Es werden public-private-key Verfahren für die Erstellung der Signatur und dessen Verifizierung benutzt. Bekannte Vertreter hierfür sind RSA, DSA oder Elgamal. Je nach Verfahren kann für die Signatur und Verifizierung die Verschlüsselungs- und Entschlüsselungsfunktion vertauscht werden. Ansonsten wird eine geeignete Funktion für die Signierung und Verifizierung konstruiert. Die Nachrichten werden vor dem Signieren gehasht. Die Signatur wird daher immer zusammen mit der Nachricht versendet.

Der Vorteil von digitalen Signaturen gegenüber von MACs liegt darin, dass kein gemeinsamer geheimer Schlüssel benötigt wird. Der Nachteil liegt in der geringen Geschwindigkeit verglichen mit MACs. Eine Nachricht kann nur der Besitzer der Nachricht und des privaten Schlüssels signieren. Die Nachricht verifizieren kann jeder, da die Verfahren und der öffentliche Schlüssel frei zugänglich sind. Der öffentliche Schlüssel wird in einem *Web of Trust* von anderen Benutzern digital signiert. Dies erleichtert die Entscheidung ob einem Schlüssel vertraut werden sollte oder nicht. Einen öffentlichen Schlüssel kann man zusätzlich mit einem digitalen Zertifikat versehen. Dieses Zertifikat wird von einer autorisierten Stelle ausgestellt und enthält Informationen über den Eigentümer des Schlüssels. Ein solches System wird auch Public-Key-Infrastructure genannt.

[21, S. 437-440] [6, S. 177-189] [47] [81] [80]

2.4.3 TLS

Transport Layer Security ist ein Verschlüsselungsprotokoll über TCP auf der Ebene 4 des OSI-Schichtenmodelles. Die bekannteste Anwendung ist HTTPS um die Kommunikation mit Webseiten abzusichern. Da TLS unter der Anwendungsschicht liegt, können beliebige Protokolle oder Anwendungen geschrieben werden, die über TLS kommunizieren. Die entsprechenden bekanntesten Bibliotheken dafür sind OpenSSL und GnuTLS. TLS benutzt ein hybrides Verschlüsselungsverfahren, Zertifikate und MACs.

Das Ziel ist wie folgt in der RFC2246 festgehalten:[3]

1. Kryptographische Sicherheit: TLS sollte dazu verwendet werden um eine sichere Verbindung zwischen zwei Kommunikationspartnern aufzubauen.

2. Interoperabilität: Unabhängige Entwickler sollten in der Lage sein Anwendungen zu entwickeln die TLS nutzen, so dass diese kryptographische Parameter austauschen können, ohne die jeweils andere Anwendung zu kennen.

3. Erweiterbarkeit: TLS versucht ein Framework bereitzustellen mit dem man TLS nachträglich um Verschlüsselungsmethoden erweitern kann. Damit werden auch zwei Unterziele abgedeckt. Die Notwendigkeit neue Protokolle und Bibliotheken zu entwickeln soll verhindert werden.

4. Relative Effektivität: Kryptographie ist sehr rechen intensiv. Daher bietet TLS einen Session Cache an und versucht die Netzwerklast gering zu halten.

[20] [112] [113]

2.4.4 SSH

Secure Shell ist der Name für ein Protokoll und eine Implementierung dieses Protokolls. Es wurde als Ersatz für Programme wie *rsh* entwickelt um eine sichere Kommunikationsverbindung zu ermöglichen. Anfangs war SSH-1 Freeware. Als später der Quelltext freigegeben wurde, wurden Sicherheitslücken entdeckt. Daraufhin wurde das SSH-2 Protokoll entwickelt. Es existieren derzeit zwei Implementierungen von SSH-2. Die erste war eine proprietäre Implementierung von dem Entwickler von SSH und einige Jahre darauf folgte eine Quell offene

[3] 20, S. 4f.

Implementierung namens OpenSSH.

SSH wird verwendet um eine sichere Verbindung mit einem entfernten Rechner aufzubauen. Die Authentifizierung kann dabei über public-key geschehen. Der öffentliche Schlüssel wird hierfür auf dem entfernten Rechner abgelegt. Statt einer Verbindung kann SSH auch einfach Dateien sicher über das Netzwerk übertragen, Ports tunneln, X11 weiter leiten und vieles mehr.

Laut RFC4251 besteht SSH aus folgenden Kernkomponenten:

- Das *Transport Layer Protocol* SSH-TRANS ist für die Authentifizierung, Vertraulichkeit und Integrität des Servers zuständig. Optional ist eine Kompression des Datenstromes damit möglich. Die Verbindung wird üblicherweise über TCP/IP geleitet, aber könnte auch über jeden anderen zuverlässigen Datenstrom geleitet werden.

- Das *User Authentication Protocol* SSH-USERAUTH authentifiziert die Client Seite dem Server gegenüber. Dieses verwendet das Transport Layer Protocol.

- Das *Connection Protocol* SSH-CONNECT teilt den verschlüsselten Tunnel in einige logische Kanäle auf. Es läuft über das User Authentication Protocol.

Die verschiedenen Kanäle werden für die verschiedenen Funktionen von SSH wie Porttunnelung oder Shell Sitzungen verwendet.

[104] [122] [105]

2.5 Authentifizierung und Autorisierung

Obwohl Authentifizierung und Autorisierung ähnlich aussehen, bedeuten sie doch etwas grundlegend anderes. Bei der Authentifizierung beweist eine Identität, sei es eine natürliche Person oder ein Server, dass diese tatsächlich die von dem Kommunikationspartner angenommene Identität und kein Betrüger ist. Mit der Autorisierung hingegen räumt man einer Identität gewisse Rechte ein oder verwehrt diese.

2.5.1 Challenge-Response

Ein challenge-response-Verfahren kann mit einem Frage-Antwort-Spiel verglichen werden. Die zu authentifizierende Identität muss eine Frage beantworten und bestätigt damit die vorgegebene Identität zu sein. Die Frage kann sehr unterschiedlich in ihrer Art sein. Das ein-

fachste Beispiel ist die Anmeldung an einem System. In dem Fall ist die Frage die Eingabe des richtigen Passwortes und die Antwort ist entsprechend das richtige Passwort. Eine kompliziertere Frage kann eine Berechnung sein. Das challenge-response-Verfahren kann auch mehrere Frage-Antwort Zyklen haben.

2.5.2 IEEE 802.1X

Der 802.1X Standard dient dem *Port-Based Network Access Control*. Es ist also eine Authentifizierung auf der Netzwerkschicht des OSI-Modelles. Der Client, Authenticator und Authentication Server sind die drei Hauptakteure bei 802.1X. Ein Client muss sich erst bei einem Authenticator authentifizieren, bevor er Zugriff auf Ressourcen im gesicherten Bereich bekommt. Dabei überprüft der Authenticator die vom Client übertragenen Zugangsdaten über einen Authentication Server. Die Kommunikation der Authentifikation läuft dabei über *EAP over LAN* und *RADIUS*.

Da in dieser Thesis 802.1X nicht direkt benutzt wird und es ein komplexes Verfahren ist, wird an dieser stelle nicht weiter auf die Details eingegangen.

[38] [89] [90] [29, S. 33ff]

2.6 Trusted System

Bei einem Trusted System geht man davon aus, dass diesem System bis zu einem gewissen Maße vertraut werden kann. Das System muss also sicher gegenüber unbefugten Zugriffen oder Veränderungen sein. Ein solches System ist oft Teil einer Sicherheitsrichtlinie. Im Folgenden werden einige Techniken angesprochen die dabei helfen können ein Trusted System aufzubauen.

[117] [118]

2.6.1 Trusted Computing

Das Trusted Computing wurde von der Trusted Computing Group entwickelt. Das System dient dazu, dass ein Rechner sich jederzeit so verhält, wie man es von ihm annimmt. Dies bedeutet, dass ein Rechner gegen unbefugten Zugriff und Veränderung geschützt wird, aber auch dass er in gewissem Maße vor dem Besitzer geschützt wird. Um dies zu erreichen wird eine Reihe von Hardware und oder Software benötigt. Zur Hardware zählt zum Beispiel das Trusted Platform Module. Das TMP ist ein Chip, welcher kryptographische Verfahren

implementiert und jeder TPM hat einen eigenen Schlüssel. Mit diesem Schlüssel kann die Festplatte verschlüsselt werden oder der Rechner identifiziert und authentifiziert werden.

[114] [116] [115]

2.6.2 chain of trust

Bei einer *chain of trust* oder Vertrauenskette wird eine Reihe von Vertrauensbeziehungen aufgebaut. Digitale Signaturen und öffentliche Schlüssel benutzen Vertrauensketten. Wenn Person A den Schlüssel von Person B signiert und Person B den Schlüssel von Person C, dann vertraut Person A automatisch Person C.

Vertrauensketten werden aber auch bei Trusted Systems benutzt. Die Hardware startet z.B. nur signierte Bootloader, diese nur signierte Kernels und die Kernels starten nur signierte Software. Dadurch kann sichergestellt werden, dass nur bestimmte Software lauffähig ist und erschwert einem Angreifer das System zu kapern. Spielekonsolen der siebten Generation wie die PlayStation3 bauen eine Vertrauenskette auf, damit keine *homebrew*, also von Spielern entwickelten Programme darauf lauffähig sind. Diese könnten nämlich das Sicherheitssystem umgehen.

[46, S. 52ff] [74]

2.6.3 coreboot

Coreboot ist eine quell offene BIOS Alternative. Es wird zusammen mit einer Payload in den ROM des proprietären BIOS geschrieben und dieses dadurch ersetzt. Dabei übernimmt Coreboot die Initialisierung der Hardware und startet dann eine Payload. Die Payload ist ein Programm, das ohne Treiber und Betriebssystem lauffähig ist. Es kann ein einfaches Programm sein um Hardwareinformationen anzuzeigen, ein kleines Spiel oder Bootloader. Im Fall eines Bootloaders kann ein vom Coreboot Team entwickelter minimalistischer Bootloader oder auch ein ausgewachsener Bootloader wie GRUB2 gestartet werden. Der Bootloader übernimmt dann den Start eines Betriebssystemes.

Der Vorteil von Coreboot gegenüber einem proprietärem BIOS ist, dass ein offenes System meistens weniger Fehler enthält als ein geschlossenes System. Zudem kann das Passwort eines BIOS leicht gelöscht werden, wenn man Hardwarezugriff hat. Ansonsten enthalten viele BIOS ein standard Passwort über das ein Angreifer in den Systemstart eingreifen könnte. Durch Coreboot erhält man frühst möglich die Kontrolle über den eigenen Rechner.

[63] [75]

2.6.4 Pre-boot authentication

Wie der Name vermuten lässt wird bei einer *pre-boot authentication* vor dem eigentlichen Bootprozess eine Authentifizierung vorausgesetzt. Dies bietet Schutz davor, dass ein Angreifer den Rechner starten kann. Die PBA kann im BIOS, Master Boot Record oder als Verweis im MBR auf ein entsprechendes Programm implementiert sein. Bei der BIOS Variante könnte der Angreifer das BIOS zurücksetzen und bei der MBR Variante die Festplatte ausbauen. Gegen das letztere hilft eine Festplattenverschlüsselung, die erst nach dem PBA entschlüsselt werden kann. Der Schlüssel kann dabei von verschiedenen Quellen bezogen werden, entweder als Passworteingabe, Zertifikat auf einer SmartCard oder auf einem Chip im Rechner fest verbaut.

Der Bootvorgang sieht bei einer PBA wie folgt aus.

1. BIOS
2. MBR
3. PBA
4. OS

[97] [98]

2.6.5 Disk-Encryption

Eine Disk-Encryption oder eher Full-Disk-Encryption ist ein Verfahren, bei dem die Festplatte transparent komplett Verschlüsselt wird. Dies kann mit Software oder Hardware realisiert werden. Bei beiden Lösungen merkt der Benutzer nichts davon, dass alle Daten direkt Verschlüsselt und entschlüsselt werden. Die FDE wird normalerweise zusammen mit der PBA eingesetzt. Falls es einem Angreifer nun gelingt die Festplatte auszubauen, kann er mit dieser nichts anfangen, da sie komplett Verschlüsselt ist. Dies wird auch bei Spielekonsolen der siebten Generation wie der PlayStation3 eingesetzt. Allerdings wird da der Schlüssel aus der Hardware ausgelesen. Dies bedeutet, dass der Besitzer die eigene Festplatte nicht auslesen kann und der Besitzer in gewisser Weiße von Sony Computer Entertainment als potentieller Angreifer angesehen wird.

[84] [85] [82]

2.6.6 Signed Kernel, Module, Software

Es gibt Bootloader, die in der Lage sind kryptographische Methoden, wie die Verifizierung von Signaturen zu verwenden. In dem Fall kann man über einen signierten Kernel verhindern, dass dieser einfach durch einen anderen ausgetauscht werden kann. Das System wird nicht starten, wenn der Kernel nicht die entsprechende Signatur enthält. Dies kann nun auch auf Module angewandt werden. Der Kernel startet dann nur Module, welche signiert sind. Dadurch kann man verhindern, dass Module manipuliert werden. Windows 7 zum Beispiel lädt keine unsignierten Treiber im normalen Modus. Es ist sogar möglich die Signierung auf normale Software auszubreiten. Das Betriebssystem würde in dem Falle nur noch Programme ausführen, welche signiert sind. Bei Desktoprechnern und Server hat ein solches Verfahren noch keine Verbreitung, allerdings benutzen Spielekonsolen der siebten Generation Softwaresignierung.

Signierung von Software oder gar Hardware hat sowohl Vorteile als auch Nachteile. Zu den Vorteilen zählt, dass ein Angreifer nicht ohne weiteres Schadsoftware auf dem Zielsystem ausführen kann. Doch dies kann auch genauso gegen den Benutzer selbst oder gegen Softwareentwickler eingesetzt werden. Es war zum Beispiel in den ersten Versionen des Windows 7 Treibers *MotionJoy* nicht möglich diesen im normalen Modus zu installieren, da er keine Signierung von Microsoft erhalten hat. Der Treiber dient dazu, dass ein Controller der PS3 oder XBox an einem Rechner benutzt werden kann, da Windows, im Gegensatz zu Apples MacOSX, selbst keine Treiber mitliefert.

2.7 Sonstiges

Am Ende des Kapitels Grundlagen und Techniken wird auf die Themen e-Mail und Spielekonsolen Sicherheit und allgemein Kopierschutz eingegangen.

2.7.1 OpenPGP

OpenPGP ist ein offener Standard basierend auf der proprietären Software PGP. Pretty Good Privacy wird hauptsächlich benutzt um e-Mails zu verschlüsseln und zu signieren. Es benutzt hierfür ein hybrides Verfahren. Die e-Mail wird mit einem symmetrischen Verschlüsselungsverfahren und einem Sessionkey verschlüsselt. Der Sessionkey wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und an den Anfang der e-Mail gehangen. Soll die e-Mail noch signiert werden, dann wird ein Hash der e-Mail erzeugt und mit dem privaten Schlüssels des Senders signiert. Die Signatur wird am ende der e-Mail angehangen, bevor beides dann verschlüsselt wird.

Mit PGP können nicht nur e-Mails sondern auch Nachrichten und Dateien verschlüsselt und

signiert werden. Eine quelloffene Implementierung von OpenPGP bietet *GNU Privacy Guard*, der auch als *GnuPG* oder *GPG* bekannt ist. GPG ist kompatibel zu PGP.

[7] [96] [99] [86]

2.7.2 Spielekonsolen

Die ersten Spielekonsolen hatten noch Kassetten in denen Platinen mit ROMs enthalten waren. Diese konnten von den wenigsten ausgelesen und somit kopiert werden. Als die ersten Spiele auf optischen Medien, angefangen mit der CD-ROM, erschienen wurden auch erste Verfahren entwickelt um diese Spiele vor dem Vervielfältigen zu schützen. Die Spielekonsolen mussten also erkennen, ob ein Spiel original war oder von einer privat Person kopiert wurde und im letzteren Fall den Start des Spieles verweigern. Bei den Spielekonsolen der siebten Generation kommen dafür starke kryptographische Systeme zum Einsatz, die mehr oder weniger sicher sind.

Alle Informationen in diesem Kapitel beruhen auf Reverse Engineering. Die Angaben sind aus Szeneforen und Präsentationen entnommen. Daher kann keine Garantie gegeben werden, dass die Angaben in diesem Kapitel vollständig oder fehlerfrei sind.

Sony PS3
SCE hat bei der PS3 eine Reihe von Sicherheitsmechanismen eingebaut. Der RAM, HDD, Startprozess und ausführbare Dateien sind verschlüsselt. Die letzten beiden zudem signiert. Zusätzlich wird ein Hypervisor, eFuses, Chips für kryptographische Funktionen und anderes eingesetzt. Das Sicherheitssystem der PS3 wurde Hackerangaben zufolge nicht attackiert, weil die PS3 mit *OtherOS* jedem ermöglichte eine Linux Distribution zu starten. Diese konnte allerdings nicht auf alle Funktionen der PS3 zugreifen, wie zum Beispiel den Grafikprozessor.

Vier Jahre nach der Veröffentlichung der PS3 hat ein Hacker einen Exploit im Hypervisor veröffentlicht. Mit diesem konnte man im OtherOS den gesamten Speicher auslesen und hat Zugriff auf den Ring0 erhalten. Dies machte SCE nervös und man hat mit der Firmware 3.21 die OtherOS Funktion entfernt. Laut Hackerangaben führte jedoch die Entfernung des OtherOS dazu, dass versucht wurde das Sicherheitssystem der PS3 zu umgehen. Einer Gruppe von Hackern gelang dies auch ein Jahr danach in der Firmware 3.41/3.55. Der *metldr* wurde ausgehebelt und man hat über eine angeblich falsche Implementierung seitens SCEs des ECDSA Algorithmus vollen Zugriff auf das System erlangt. Die Chain of Trust wurde somit gebrochen, die bis zur Firmware 3.55 wie folgt aussah.

- bootldr

- lv0
- metldr
- rvklrd, isoldr, appldr, lv2ldr, lv1ldr

Dies führte nicht nur dazu, dass Linux wieder lauffähig war. Es wurden auch Programme entwickelt die es ermöglichten Spiele von der Festplatte aus zu starten. SCE reagierte schnell darauf mit der Firmware 3.60. In dieser wurde die Chain of Trust angepasst und der metldr entfernt. Dadurch war die PS3 wieder sicher. Es gibt zum jetzigen Zeitpunkt (Abgabe der Thesis) keine Berichte darüber, dass es jemandem gelungen ist eine Firmware größer 3.55 zu knacken. Allerdings kann man mittels eines NAND Flasher die aktuelle Firmware durch die Firmware 3.55 ersetzen.

[25] [69] [5] [24] [18] [42] [43]

Nintendo Wii
Nintendo hat die günstigste Spielekonsole der siebten Generation. Trotzdem wurden weitreichende Sicherheitsmechanismen eingebaut. Die Wii besitzt neben einer PowerPC CPU von IBM eine GPU von ATI. Allerdings ist in der GPU eine ARM CPU versteckt, die Nintendo nie erwähnt hat. Hierdurch wird user mode vom kernel mode physikalisch getrennt und nicht wie bei der PS3 und XBox360 durch einen Hypervisor. Weiterhin ist ein Chip für AES Verschlüsselung und ein Chip für SHA-1 Hashes vorhanden. In dem ARM Chip ist ein OTP enthalten, der bei der Herstellung einige Schlüssel und Zertifikate einprogrammiert bekommt. Mit diesen ist auch der NAND verschlüsselt in dem der boot1, boot2, IOS und alle Dateien liegen die veränderbar sein sollen. Jegliche Software und optische Datenträger sind mittels Hashes und einer Signatur gesichert.

Der Startprozess und somit die Chain of Trust sieht wie folgt aus:

- boot0 aus ROM in CPU
- boot1 aus flash und mittels eingebrannter Hash überprüft
- boot2 aus flash
- IOS aus flash in versteckte ARM CPU
- Menü aus flash in CPU

Dabei ist boot2, IOS und das Menü signiert. Allerdings wird die Signatur nur bei der Instal-

lation überprüft, nicht bei der Ausführung. Dies ist eine Schwachstelle in der Chain of Trust, da sie über direktes Beschreiben des NAND angreifbar ist.

Nintendo hat einige grobe Fehler bei der Umsetzung des sehr guten Sicherheitskonzeptes begangen. Die AES und SHA-1 Chips haben keine direkte Anbindung an den ARM Prozessor und Kommunizieren über den DRAM. Dies geschieht fälschlicherweise im Klartext und der DRAM wird nicht gesäubert. So gelang es Team Tweezer über Hardwarezugriff an die Schlüssel der Wii zu gelangen. Damit war es nun möglich eigene Spielstände zu erstellen, welche einen Absturz des Spieles provozieren sollten. Dies gelingt tatsächlich bei fast jedem Spiel durch eine zu lange Zeichenfolge in einem Namen und es gibt einen Buffer Overflow. Durch einen entsprechenden Spielstand kann nun selbst geschriebene Software ausgeführt werden.

Viel kritischer war jedoch Nintendos Implementierung der Signaturüberprüfung. Diese kann überlistet werden, wenn man eine Signatur angibt die mit 00 beginnt. Der Fehler exisitiert in boot1. Da boot1 durch einen in Hardware programmierten Hash geprüft wird, kann Nintendo bei bereits gefertigten Wiis den Fehler nicht beheben. Nun konnte der boot2 manipuliert werden und die Chain of Trust war komplett gebrochen. Nach mehreren Anläufen gelang es Nintendo die Wii wieder abzusichern. Das Menü wurde aktualisiert und es konnten keine Spielstände mehr installiert werden, die eine seltsame Struktur aufwiesen.

In der selben Firmware wurde auch eine neue Funktion eingebaut. Diese war allerdings gegen ein Heap Overflow anfällig, wodurch die Hacker wieder selbst geschriebene Programme starten konnten. Da die Signatur weiterhin fehlerhaft überprüft wurde, war die Wii wieder offen. Nach drei Versuchen gelang es Nintendo auch diese Lücke zu schließen. Daraufhin haben die Hacker neue Exploits gefunden, die allerdings alle auf Spielen beruhen. Es ist etwas Eigenleistung gefragt, aber die Wii kann sehr schnell geknackt werden. Danach hat Nintendo keine weiteren Versuche unternommen die Wii abzusichern.

[65] [4] [5] [42] [43]

Microsoft Xbox360

Microsoft hat bei der Xbox360 wie die Konkurrenten Nintendo und Sony weitreichende Sicherheitsmechanismen eingebaut. Der Startprozess und ausführbare Dateien sind signiert und verschlüsselt. Der RAM ist gehasht und verschlüsselt. Mit einem Hypervisor wird user von kernel mode getrennt. Über IBMs eFuses wird dafür gesorgt, dass keine rückaktualisierung auf eine ältere Firmware möglich ist.

Der Startprozess und Chain of Trust sieht wie folgt aus:

- 1bl aus ROM

- CB aus NAND mit RSA Signatur und RC4 Verschlüsselung

- CD aus NAND mit Hashüberprüfung
- Kernel aus NAND

Die Kernelversionen 4532/4548 hatten eine Sicherheitslücke, über die der Hypervisor benutzt werden konnte um unsignierte Software auszuführen. Darüber konnte voller Zugriff über die Xbox360 erlangt werden. Microsoft reagierte schnell mit einer neuen Kernelversion. Dank des eFuses ist kein Kerneldowngrade möglich. Da der CD entscheidet welcher Kernel gestartet wird, muss dieser ausgehebelt werden.

Der aktuell nicht behobene *Reset Glitch Hack* sieht für die Xbox360 fat[4] wie folgt aus. Dabei ist unter einem Glitch ein kurzzeitiges Fehlverhalten der Hardware zu verstehen.

1. CPU_PLL_BYPASS Signal absetzen um die CPU stark zu verlangsamen
2. warten bis *memcmp* von erwartetem und berechnetem CD Hash beginnt und Zähler starten
3. nach erreichen eines bestimmten Zählerstandes ein CPU_RESET Signal von 100ns absetzen
4. CPU_PLL_BYPASS Signal wieder zurücksetzen
5. mit etwas Glück wird der memcmp Befehl übersprungen und der modifizierte CD wird gestartet

Da dieser Vorgang reine Glückssache ist, wird ein modifizierter System Management Controller benutzt, der den Vorgang so lange wiederholt, bis erfolgreich der modifizierte CD gestartet wurde.

Der RGH sieht für Xbox360 slim[5] aufgrund der veränderten Hardware etwas anders aus. Der Bootloader CB wurde in CB_A und CB_B aufgeteilt. Die Leiterbahn für das CPU_PLL_BYPASS Signal wurde entfernt und es ist ein HANA Chip hinzu gekommen. Dieser ist für die HDMI Ausgabe zuständig und ermöglicht den RGH.

1. i^2c Befehl an HANA senden um die CPU zu verlangsamen
2. auf memcmp des erwarteten und berechneten CB_B warten und Zähler starten
3. nach einem bestimmten Zählerstand ein 20ns Impuls auf CPU_RESET senden

[4] Erste Revision von 2005-2008.
[5] Zweite Revision ab 2008 bis Mitte 2011 mit kleinerer Bauart.

4. i^2c Befehl an HANA senden um die CPU auf ursprüngliche Taktung zurück zu setzen

5. mit etwas Glück wird der memcmp Befehl übersprungen und der modifizierte CB_B wird gestartet

Dieser Vorgang ist wie bei der fat Konsole Glückssache und deshalb wird auch hier ein modifizierter SMC benutzt. Zusätzlich muss jedoch der CPU Key herausgefunden werden um den CB_B zu verschlüsseln. Die Hacker haben den CB von der fat Konsole genommen und vermutet, dass die ersten paar Bytes mit denen von CB_B überein stimmen. Darüber ist es ihnen gelungen eine kleine Software zu verschlüsseln und den CPU Key zu dumpen.

[5] [34] [27] [26] [36] [42] [43]

2.7.3 Kopierschutz

Kopierschutz ist den meisten in Zusammenhang mit Spielen, Filmen, Musik und Software bekannt. Es wird hier kurz darauf eingegangen wie versucht wurde die entsprechenden Medien zu schützen und wie sie gescheitert sind.

Filme
Bei VHS Kassetten war es noch ungewöhnlich, wenn man keinen Film auf eine andere Kassette überspielen konnte. Bei Einführung der DVD hatte man von vornherein den Content Scramble System Kopierschutz eingebaut. Dieser konnte durch verschiedene Programme umgangen werden. Schließlich gelang es auch den CSS durch Reverse Engineering für Linux zugänglich zu machen. Auch die aktuellen Bluray Filme sind kopiergeschützt. Sie verwenden den Advanced Access Content System. Dieser ist wesentlich komplexer als sein Vorgänger CSS. Dabei ist die Blu-Ray-Disc in mehreren Stufen verschlüsselt und nur lizenzierte Geräte oder Programme können die Blu-Ray-Disc entschlüsseln. Zudem ist es möglich bereits vorhandene Schlüssel, die sich auf der Blu-Ray-Disc befinden, in neuen Blu-Ray-Discs zu verbieten. Dies ist für Benutzer von Unix Systemen, ausgenommen MacOSX, noch ärgerlicher als es bei der DVD war. Die AACS ist zwar ein offener Standard im Gegensatz zu CSS, aber die Schlüssel bekommt man nicht ohne weiteres. Es sind allerdings zur Zeit bereits die Schlüssel von vielen Blu-Ray-Filmen im Internet zu finden und es werden täglich mehr. Abgesehen davon, lassen sich von vielen oder sogar allen Filmen so genannte Rips finden. Das sind Kopien der Filme in einem anderen Format wie zum Beispiel MKV. Diese benötigen keinerlei Schlüssel und sind problemlos auf jedem Betriebssystem abspielbar. [70]

Musik
Bei Musik wird inzwischen auf Kopierschutzmechanismen verzichtet. Anfangs wurde versucht das DRM durchzusetzen, was aber gescheitert ist. Über iTunes bezogene DRM Musik

lies sich auf CD Brennen und dann wieder ohne DRM rippen, also in z.B. MP3 umwandeln. Zudem führte die Einführung von DRM auf Audio CDs zu Wiedergabeproblemen. Die Audio CDs haben nicht mehr dem Standard entsprochen und liefen teilweise nicht auf CD-Spielern. Selbst unter Windows gab es Probleme mit DRM geschützten CDs.

PC Spiele

Der meiste Aufwand mit Kopierschutzmechanismen wird bei PC Spielen und professioneller Software betrieben. Bei PC Spielen sind die momentan am verbreitetsten Mechanismen SecuROM und StarForce. Beide verwenden komplexe Kryptographische Verfahren. Zudem nutzen sie noch Hardwarebasierte Anomalien beim Beschreiben der Datenträger und eigene Systemtreiber. Durch die eigenen Systemtreiber wird tief in das Betriebssystem eingegriffen, was zu Instabilitäten führen kann. Solche Sicherheitsmechanismen haben auch sogenannte Blacklists in denen Programme gelistet sind die nicht auf dem System installiert sein dürfen. Davon sind unter anderem Brennprogramme wie Alcohol120% und Virtualisierungssoftware wie DaemonTools betroffen. Somit machen die Entwickler von diesen Kopierschutzmechanismen Vorgaben was Benutzer installieren dürfen und was nicht. Noch schlimmer ist die Tatsache, dass diverse Mechanismen sogar die Installation von dem legal erworbenem Spiel in virtuellen Maschinen verbieten. Somit können zum Beispiel Benutzer von MacOSX so geschützte Spiele nicht in einer Windows VM ausführen und eine MacOSX Version existiert meistens nicht. All diese Restriktionen führen dazu, dass die Sicherheitsmechanismen umgangen werden. Dafür werden unter anderem die Installationsdateien modifiziert, Systemtreiber der Sicherheitsmechanismen nachgebaut oder die entsprechenden Überprüfungen aus den Spieldaten entfernt. Bisher wurden alle bekannten und verbreiteten Spiele geknackt. Momentan weichen immer mehr Publisher auf Onlinebasierte Sicherheitsmechanismen aus. Dies bedeutet, dass sich die Spiele vor dem Start erst bei einem Server authentifizieren müssen. Dafür wird die Seriennummer des Spieles mit der Hardware des Benutzers verknüpft und auf dem Server hinterlegt. Ein großer Nachteil für den Spieler ist, dass er das Spiel nicht wieder verkaufen kann. Doch auch diese Art des Kopierschutzes lässt sich umgehen, wie man es bei Steam beobachten kann.

Software

Bei Software variieren die Kopierschutzmechanismen von einfachen Lizenzschlüsseln oder Lizenzserver über Onlineaktivierungen bis hin zu Hardware Dongles. Für die meisten Lizenzschlüssel und Onlineaktivierungen schaffen es Hacker Generatoren zu entwickeln. Es kann auch nötig sein zusätzlich noch die Startdateien oder andere Teile der Software zu modifizieren. Es gibt nur wenig Software mit solchen Mechanismen die nicht geknackt wurden. Dabei ist es dann meistens Software die nicht weit verbreitet ist. Interessanter und effektiver sind allerdings die Hardware Dongles. Dabei handelt es sich um kleine Geräte die kryptographische Schlüssel, Chips, Software und oder Lizenzen enthalten. Diese werden zwangsweise von der gesicherten Software benötigt. Je nach Implementierung der Schnittstelle zwischen Software und Dongle kann dieser emuliert werden. Allerdings ist das Reverse Engeneering von Dongles äusserst schwierig. Es gibt nur einige wenige bekannte Programme die durch einen Dongle gesichert sind und geknackt wurden. Dabei wurde der Dongle ausgelesen und ein Daemon geschrieben, der den Dongle emuliert hat.

3 Entwurf und Design

In diesem Kapitel wird der clientseitige identity daemon *IDD*, serverseitige identity server *IDS* und die Kommunikation zwischen den beiden Komponenten modelliert. Es werden auch einige Verfahren und Möglichkeiten behandelt die nicht zum Einsatz kommen.

3.1 bewusst nicht in das Konzept eingebaut

Einige Komponenten können in dieser Version nicht benutzt werden. Das liegt daran, dass die dafür benötigte Infrastruktur im Allgemeinen nicht gegeben ist und auf eine spezialisierte Implementierung an dieser Stelle verzichtet wird.

3.1.1 Kernel signed modules - KSM

Bei Kernel signed modules handelt es sich um signierte Module. Diese werden vom Kernel während des Startprozesses direkt geladen. Damit dies funktioniert muss der Kernel ebenfalls signiert sein und die Signatur der Module überprüfen können.

Dieses Verfahren spaltet allerdings die Fachleute. Es gibt durchaus sinnvolle Szenarien für die Verwendung von KSM. Hierdurch kann ein Angreifer daran gehindert werden, Schadsoftware auf einem System auszuführen. Jedoch definieren einige Unternehmen indirekt die Benutzer selbst als potentielle Angreifer. Somit können Benutzer nicht mehr beliebige Software ausführen, sondern nur die, die von den Herausgebern des Kernels signiert wurde.

Es ist aber auch denkbar mehr wie eine Signatur zu verwenden. Dadurch könnte überprüft werden wer die Software entwickelt und vertrieben hat. Der Kernel entscheidet dann anhand der Signatur ob die Software bestimmte Funktionen nutzen darf oder gibt der Software einen bestimmten Nutzer unter dem die Software laufen soll. Der Vorteil bei einem solchen Vorgehen wäre, dass es jedem ermöglicht werden könnte Software für dieses System zu schreiben, die mit minimalen Rechten lauffähig wäre. Möchte man hingegen in das System eingreifen, dann muss der Herausgeber selbst die Software signieren und kann dabei überprüfen ob die Software sicher ist.

In dieser Thesis wird auf die Signierung von Kernel, Modulen oder Software verzichtet, da dies den Rahmen sprengen würde. In einem zweiten Schritt kann und sollte ein solches Verfahren eingebaut werden um den IDD auf dem Client zu schützen. Im folgenden werden noch drei Quellen genannt mit näheren Informationen zu dem Thema.

- In dem Artikel [40] steht näheres zu Signed Kernel Modules.

- In dem Artikel [66] von IBM steht wie man Linux härtet indem man die Standard Interpreter entfernt.

- In dem Artikel [67] steht wie man einen Kernel baut, der nur signierte Binärdaten ausführt.

3.1.2 Bootloader

Ein Bootloader ist dafür zuständig ein Betriebssystem zu starten. Er wird normalerweise von dem BIOS aufgerufen und initialisiert weitere Teile des Systemes, damit Software ausgeführt werden kann. Es kann sein, dass der Bootloader zu groß für die Bootsection eines Mediums ist. In einem solchen Fall teilt man den Bootloader selbst wieder in verschiedene Teile die sich nacheinander initialisieren und starten. Ein solcher Bootloader wird als *Multistage Bootloader* bezeichnet. Verschiedene Medien auf denen ein Betriebssystem liegt wie etwa CD-ROM, Festplatte oder Netzwerk benötigen verschiedene Bootloader. Auch verschiedene Betriebssysteme benötigen verschiedene Bootloader. Das hintereinander Schalten von Bootloader wird als *Chain-Loading* bezeichnet. Damit ist es möglich auf einem System verschiedene Betriebssysteme zu installieren. Die Installation von Windows und einer Linux Distribution wird als Dual-Boot-System bezeichnet und benötigt das Chain-Loading.

Während bei Spielekonsolen und Mobilen Geräten wie Smartphones oder eBook-Readern signierte Bootloader zum Standard gehören, sind diese bei Rechner Systemen kaum verbreitet. Um einen signierten Bootloader zu starten muss das darunterliegende BIOS oder Firmware entsprechende Funktionalitäten besitzen um die Signierung zu verifizieren. Dies hat die selben Vor-/Nachteile wie signierte Software. In Abhängigkeit des BIOS und den erlaubten Signaturen könnten Hersteller von Komplettsystemen das Installieren von bestimmten Betriebssystemen unterbinden. Der Vorteil liegt darin, dass man damit eine stärkere Chain-of-trust aufbauen kann und Schadsoftware auf dieser Ebene verhindern kann. Der Nachteil liegt darin, dass der Benutzer eventuell nicht mehr frei entscheiden kann welchen Bootloader und Betriebssystem er verwendet.

Auch die Verwendung eines signierten Bootloaders entfällt in dieser Thesis. In einem dritten Schritt sollte allerdings ein solcher eingebaut werden um den IDD weiter zu schützen. Hierzu könnte der UEFI benutzt werden mit seinem *secure boot*.

3.1.3 BIOS oder Firmware

Das BIOS oder allgemeiner formuliert eine Firmware ist die erste Software die gestartet wird, sobald ein Rechner eingeschaltet wird. Bei PCs ist die Firmware das BIOS und bietet das

Minimum an Funktionen die für den weiteren Systemstart benötigt werden. Wie bei dem Bootloader ist es auch bei der Firmware und somit dem BIOS möglich dieses zu signieren. Dies wird auch bei Spielekonsolen, Smart Phones oder anderen mobilen Geräten von einigen Herstellern getan. Allerdings ist es bei PCs nicht so einfach. Die Verifizierung der Signatur muss vor der Ausführung des BIOS geschehen wobei das BIOS als erstes gestartet werden soll. Es ist also eine zusätzliche Stufe vor dem BIOS nötig. Diese Funktionalität wäre zu speziell und zu Kostspielig, als dass sie auf alle PCs eingebaut werden kann.

Ob und wie ein BIOS oder das UEFI bei einem PC Signiert und Verifiziert werden kann muss näher untersucht werden. Theoretisch möglich ist dies natürlich, allerdings muss man die damit verbundenen Kosten und Aufwand bedenken. Dies würde den Rahmen dieser Thesis leider übersteigen.

3.1.4 Data Hiding In A Binary Image[1]

Es gibt Verfahren mit denen man Informationen in einem träger Medium wie einem Bild, Video, PDF oder sonstigem verbergen kann. Dazu zählt vor allem die Steganographie. Sie stellt wie die Kryptographie einen eigenen Wissenschaftsbereich dar. Wohingegen bei Kryptographie offensichtlich ist, dass Informationen verheimlicht werden, so ist es bei der Steganographie genau umgekehrt. Ein Dritter soll an einem Trägermedium nicht erkennen können, dass dort eine weitere Nachricht verborgen ist. Dabei wird ein Trägermedium mit verborgener Nachricht als Steganogramm bezeichnet. Die so versteckte Nachricht kann nur gelesen werden, wenn der benutze Geheimschlüssel und die Methode für das Verstecken bekannt ist.

In dieser Thesis wird zwar keine Steganographie verwendet, aber man könnte damit den Nachbau des IDD erschweren. Vor dem Übersetzen des IDD könnten zufällige Bilder ausgewählt werden und zufällige Zeichenfolgen darin versteckt werden. Dabei wird jede Zeichenfolge mit einer anderen Methode und Einmalschlüssel versteckt. Die generierten Informationen müssen Serverseitig hinterlegt werden. Der IDS kann dann den IDD nach der Information in einem der Bilder befragen. Ein Angreifer kann nicht voraussehen welches Bild und welche Methode mit welchem Schlüssel angewandt werden muss.

3.1.5 obfuscation

Unter *obfuscation* versteht man die Verschleierung von Quelltext, Bytecode oder Binärcode. Es soll das Decompilieren von hauptsächlich Bytecode erschweren, denn der resultierende Quelltext ist sehr schwer für den Menschen lesbar. Entwickler die Obfuscator benutzen wollen damit ihre Programme schützen. Mit Schutz ist hier gemeint, dass die Funktionalität nicht leicht nach gebaut werden können soll. Damit erhofft man sich, dass ein Dritter nicht die eigenen Funktionen leicht stehlen kann oder ein Hacker den Sicherheitsmechanismus

umgehen kann. Allerdings wird obfuscation unter „security through obscurity“ eingestuft. Dadurch erschwert zwar obfuscation das Reverse Engineering, aber verhindert dieses nicht. Für gewöhnlich beschäftigen sich sehr gute Hacker mit Reverse Engineering und zeigen sich wenig beeindruckt durch obfuscation. Daher wird in der Theis auf obfuscation verzichtet. Bei einigen Programmiersprachen bietet obfuscation sogar Nachteile. Moderne Programmiersprachen oder auch als Hochsprachen bezeichnet profitieren stark von Introspektion. Grob formuliert bedeutet dies unter anderem, dass ein Programm sich selbst zur Laufzeit verändern kann. Durch obfuscation kann die Introspektion stark behindert werden.

3.1.6 chain-of-trust

Eine gute chain-of-trust wie sie bei Spielekonsolen der siebten Generation eingesetzt werden wird nicht umgesetzt. Dazu fehlt zum Einen die Zeit und zum Anderen die nötige Infrastruktur. Um signierte Software auszuführen muss der Kernel signiert werden. Damit die Signatur des Kernels zuverlässig geprüft werden kann wird ein signierter Bootloader benötigt. Der Bootloader muss entsprechend von der Firmware verifiziert werden. Bis zu dieser Stelle sollte in weiteren Schritten außerhalb der Thesis eine entsprechende chain-of-trust umgesetzt werden. Der letzte Schritt wird vermutlich nicht ohne weiteres machbar sein. Nämlich die Verifizierung einer signierten Firmware.

3.1.7 self-checksumming

Es ist möglich Programme so zu programmieren, dass diese sich selbst überprüfen können. Genauer gesagt, können die Programme ihre eigene Checksum berechnen und vergleichen. Es gibt allerdings nur wenig Informationen zu diesem Thema. Self-checksumming kann gegen Manipulation, aber nicht gegen Reverseengineering schützen. Falls jemand die benötigten Rechte erlangt um das Programm manipulieren zu können, dann kann er auch Reverseengineering betreiben und anhand des offenen Quelltextes die fehlenden Informationen gewinnen.

Gegen self-checksumming gibt es eine *page-replication* Attacke. Dabei werden die Speicherzugriffe für die Überprüfung der Checksumme umgeleitet. Es gibt auch ein Paper in dem ein möglicher Schutz gegen diese Attacke vorgestellt wird [30]. Weder die Attacke noch ein möglicher Schutz dagegen wurden näher untersucht, da dieses Verfahren nicht in der Thesis zum Einsatz kommt.

Die Mac OS X Software Synergy wurde im Jahr 2005 um self-checksumming erweitert. Dieses funktionierte grob wie folgt [121]:

- leeres Feld für Checksumme reservieren

- Checksumme von Programm berechnen und in das leere Feld nachträglich einfügen
- bei der Überprüfung eine Kopie des Programmes im Speicher anlegen, Feld für Checksumme leeren und Checksumme berechnen
- Checksumme von der Kopie mit der Checksumme im Programm vergleichen

Bereits im Jahr 2006 wurde dieses Verfahren wieder entfernt. Der Grund dafür war, dass MacOSX Programme zur Laufzeit verändert hat um das Gesamtsystem zu optimieren. Das hatte natürlich zur Folge, dass Synergy eine andere Checksumme hatte wie sie im Programm enthalten war. Für den Einsatz eines solches Ansatzes in der Thesis ist aber etwas anderes kritischer. Dieses Verfahren funktioniert nur, solange der Angreifer nicht weiß wie das Verfahren funktioniert. Da der Quelltext dieser Thesis offen ist, kann der Angreifer leicht das self-checksumming umgehen.

3.1.8 Schlüssel (verstreut) verstecken

Es ist eine sehr schlechte Idee einen Schlüssel in der auszuliefernden Software unter zu bringen. Bisher wurden alle Sicherheitssysteme ausgehebelt oder konnten entschlüsselt werden, bei denen der Schlüssel zusammen mit der Software ausgeliefert wurde. Dies konnte man vor allem bei Videoverschlüsselungen wie bei der DVD mit CSS und Bluray mit AACS beobachten. Bei AACS hat es zwar länger gedauert wie bei CSS, aber es ist trotzdem relativ schnell gelungen. Dabei macht es keinen großen Unterschied ob der Schlüssel kompliziert verstreut im Quelltext liegt oder an einem Stück. Sobald man hardwareseitigen Zugriff auf das System hat, kann man Angriffe auf den Speicher starten wo der Schlüssel als ganzes liegen muss. Aus diesem Grund wird in dieser Thesis auf komplexe Verfahren um den Schlüssel versteckt und verstreut unterzubringen verzichtet. Wie bereits erwähnt, beruht das Konzept darauf, dass der Benutzer des Systems keine root Rechte besitzt und den IDD nicht auslesen kann. Den Schlüssel zu verstecken würde nur erheblichen Aufwand, aber keine Sicherheit mit sich bringen. *Security through obscurity* ist wie auch *Obfuscating* nur „Augenwischerei“ und beides bietet keine Sicherheit. Es verlangsamt nur unwesentlich das Reverseengineering, denn die entsprechenden Leute haben mächtige Werkzeuge wie IDA – Interactive Disassembler zur Hand. [53]

Zu dem Thema einen Schlüssel in einer Binärdatei zu verstecken gibt es Viele Diskussionen im Internet. Diese wurden als Inspiration genutzt, werden aber nicht ausführlich in dieser Thesis Diskutiert. [19] [55] [56] [57] [58]

3.1.9 Zeitmangel

Die folgenden Punkte wurden nicht ausführlich betrachtet, da es ansonsten den Zeitrahmen dieser Thesis überstiegen hätte.

- Teile vom IDD zur Laufzeit austauschen
- Server generiert zufällig .so und schickt diese an IDD
- Binärdatei zur Laufzeit patchen
- Schlüssel im RAM verschlüsseln oder nach Benutzung entfernen
- Verschlüsselte Binärblöcke verstecken und den Schlüssel + Offset vom Server schicken lassen. Blöcke testen IDD auf Integrität (Hash?) und an welchem Port er hängt. Ähnlich self-checksumming. Hardwarerealisierung wäre sicherer, aber die Infrastruktur ist nicht gegeben.
- ELF/Binärdatei verschlüsseln [33]

3.2 identity daemon – Client

Auf der Client Seite muss der IDD laufen um den Client beim Server zu authentisieren und die Integrität zu versichern. Das gesamte Konzept funktioniert nur so lange, wie der Angreifer keine root Rechte besitzt. Als Angreifer muss leider auch der Benutzer eingestuft werden, wenn der Thin Client zum Beispiel vom Arbeitgeber zur Verfügung gestellt wird. Es muss also sicher gestellt werden, dass der Benutzer ohne root Rechte nicht in der Lage ist den IDD zu kompromittieren. Der IDD muss auch so aufgebaut werden, dass trotz Quelloffenheit es nicht möglich ist den IDD ohne Reverse Engineering nach zubauen. Im Folgenden wird behandelt wie der IDD geschützt wird und welche Funktionalitäten er für die Authentisierung und Integritätskontrolle enthält.

3.2.1 ptrace unterbinden

Der *ptrace* Syscall unter Unix Systemen wird von Debuggern, Tracern oder auch anderen Programmen verwendet. Ein Programm kann sich damit an ein anderes Programm dran hängen und es überwachen oder sogar manipulieren. Im folgenden Beispiel 4 in Zeile 11 wird der ptrace Befehl deaktiviert. Ein externes Programm kann sich danach nicht mehr an den

Prozess dran hängen. Dies gilt natürlich nicht für root.

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/prctl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
  char string[9] = "passwort\n";
  printf("%s", string);

  if (0 > prctl(PR_SET_DUMPABLE, 0))
  {
    perror("can't prctl(PR_SET_DUMPABLE)");
    return 1;
  }

  printf("gdb -q %s %d\n", argv[0], getpid());
  fflush(stdout);
  sleep(20);

  return 0;
}
```

Quelltext (4) Beispiel ptrace laufender Prozess verhindern (C)

```
> gcc -o 01_ptrace_on 01_ptrace_on.c
> ./01_ptrace_on&
[1] 5083
passwort
gdb -q ./01_ptrace_on 5083
> gdb -q ./01_ptrace_on 5083
Reading symbols from .../01_ptrace_on...done.
Attaching to program: .../01_ptrace_on, process 5083
Reading symbols from /lib64/libc.so.6...
Missing separate debuginfo for /lib64/libc.so.6
(no debugging symbols found)...done.
Loaded symbols for /lib64/libc.so.6
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/lib64/ld-2.14.1.so.debug...
done.
done.
Loaded symbols for /lib64/ld-linux-x86-64.so.2
0x00007f274e4858c0 in __nanosleep_nocancel () from
   /lib64/libc.so.6
(gdb) quit
A debugging session is active.
  Inferior 1 [process 5083] will be detached.
```

```
22 Quit anyway? (y or n) y
23 Detaching from program: .../01_ptrace_on, process 5083
24
25 > gcc -o 01_ptrace_off 01_ptrace_off.c
26 > ./01_ptrace_off&
27 [1] 5051
28 passwort
29 gdb -q ./01_ptrace_off 5051
30 > gdb -q ./01_ptrace_off 5051
31 Reading symbols from .../01_ptrace_off...done.
32 Attaching to program: .../01_ptrace_off, process 5051
33 ptrace: Die Operation ist nicht erlaubt.
34 .../5051: Datei oder Verzeichnis nicht gefunden.
35 (gdb) quit
```

Quelltext (5) Beispiel ptrace laufender Prozess verhindern (bash)

Um den `ptrace call` zu unterbinden reicht auch das entziehen des Leserechtes aus. Dies wird anhand von anderen Programmen im Folgenden gezeigt.

3.2.2 Binärdatei die Leserechte entziehen

Durch `PR_SET_DUMPABLE = 0` können sich zwar andere Prozesse nicht mehr an den IDD dran hängen, allerdings bietet dies nur geringen Schutz. Solange der Benutzer das Lese-Recht für eine Datei hat, kann er diese unter anderem auf ein externes Medium wie USB-Flash oder gar über eMail versenden. Somit kann ein Angreifer die ausführbare Datei auf einem anderem System mit spezieller Software untersuchen. Doch auch auf dem lokalen System gibt es Möglichkeiten für Reverse Engineering solange die ausführbare Datei gelesen werden kann. Dies beinhaltet Programme wie `strings`, `objdump`, `hexdump`, `readelf` oder `nm`. Um beide Probleme zu lösen genügt es das Leserecht zu entziehen. Eine Binärdatei benötigt lediglich das Ausführrecht, aber nicht das Leserecht um ausgeführt zu werden.

```
1  > l
2  -rwxrwx--x+ 1 psy-xoryves users 11649  6. Aug 10:01
     01_ptrace_off*
3  > cp 01_ptrace_off 01_test
4  > l
5  -rwxrwx--x+ 1 psy-xoryves users 11649  6. Aug 10:01
     01_ptrace_off*
6  -rwxrwx---+ 1 psy-xoryves users 11649  6. Aug 23:05 01_test*
7  > chmod a-r 01_test
8  > l
9  -rwxrwx--x+ 1 psy-xoryves users 11649  6. Aug 10:01
     01_ptrace_off*
10 --wx-wx---+ 1 psy-xoryves users 11649  6. Aug 23:05 01_test*
```

```
> cp 01_test 01_test_test
cp: "01_test" kann nicht zum Lesen geöffnet werden: Keine
   Berechtigung
> strings 01_test
strings: 01_test: Permission denied
> strings 01_ptrace_off
/lib64/ld-linux-x86-64.so.2
...
```

Quelltext (6) Beispiel Leserecht zur Ausführung bei Binärdateien

An dieser Stelle soll noch angemerkt werden, dass es nicht möglich ist einem Benutzer die Berechtigungen dermaßen zu setzen, so dass er nur bestimmte Programme ausführen kann. Solange der Benutzer das Leserecht auf eine ausführbare Datei besitzt, kann er diese auch ausführen. Selbst dann, wenn das Ausführrecht nicht gesetzt ist.

```
> l
drw-rw----+   2 psy-xoryves users   4096  6. Aug 18:07 geheim/
----------+   1 psy-xoryves users 110176  6. Aug 18:05 ls
> /lib64/ld-linux-x86-64.so.2 ./ls
./ls: error while loading shared libraries: ./ls: cannot open
   shared object file: Permission denied
> chmod +r ./ls
> /lib64/ld-linux-x86-64.so.2 ./ls
geheim  ls
> l
drw-rw----+   2 psy-xoryves users   4096  6. Aug 18:07 geheim/
-r--r--r--+   1 psy-xoryves users 110176  6. Aug 18:05 ls
> chmod g-r ./ls
> l
drw-rw----+   2 psy-xoryves users   4096  6. Aug 18:07 geheim/
-r-----r--+   1 psy-xoryves users 110176  6. Aug 18:05 ls
> chmod o-r ./ls
> l
drw-rw----+   2 psy-xoryves users   4096  6. Aug 18:07 geheim/
-r--------+   1 psy-xoryves users 110176  6. Aug 18:05 ls
> sudo chown opl ./ls
> l
drw-rw----+   2 psy-xoryves users   4096  6. Aug 18:07 geheim/
-r--------+   1 opl         users 110176  6. Aug 18:05 ls
> ./ls
bash: ./ls: Keine Berechtigung
> /lib64/ld-linux-x86-64.so.2 ./ls
./ls: error while loading shared libraries: ./ls: cannot open
   shared object file: Permission denied
> chmod a+r ./ls
chmod: Beim Setzen der Zugriffsrechte für "./ls": Die Operation
   ist nicht erlaubt
```

```
30 > sudo chmod a+r ./ls
31 > /lib64/ld-linux-x86-64.so.2 ./ls
32 geheim  ls
```

Quelltext (7) Beispiel Ausführung bei Binärdateien ohne Ausführrecht

3.2.3 Skript die Leserechte entziehen

Damit Skripte funktionieren können, müssen diese für gewöhnlich vom Benutzer gelesen werden können. Das Skript wird nämlich lediglich an den entsprechenden Interpreter weitergegeben. Damit der Interpreter das auszuführende Skript interpretieren kann, muss er dieses Lesen können. Vielmehr muss der Benutzer das Leserecht für das entsprechende Skript besitzen, damit es von dem Interpreter eingelesen werden darf.

Damit ein Benutzer ohne Leserecht ein entsprechendes Skript ausführen kann reicht es nicht dieses Skript mit dem `SUID Bit` zu versehen. Das SUID Bit wird nämlich bei Skripten vom Kernel ignoriert. Eine Lösung für dieses Problem wäre die Erzeugung eines C-Wrappers. Also eine Binärdatei, die intern das entsprechende Skript ausführt. Eine einfachere Methode ist die Benutzung von `sudo`.

In der Datei `/etc/sudoers` kann eingestellt werden, dass ein bestimmter Benutzer ein bestimmtes Programm unter einer anderen Nutzerkennung ausführen darf. Das folgende Beispiel demonstriert dieses Vorgehen.

```
1 #!/usr/bin/ruby
2 ## encoding: UTF-8
3 puts "Testausgabe..."
```

Quelltext (8) 04_ruby.rb

```
1 > sudo vi /etc/sudoers
2 Defaults targetpw   # ask for the password of the target user
     i.e. root
3 ALL     ALL=(ALL) ALL   # WARNING! Only use this together with
     'Defaults targetpw'!
4 root ALL=(ALL) ALL
5 test ALL=(psy-xoryves)
     NOPASSWD:/xoryvesData/HTWdS/PI-M/Thesis/tia/Beispiele/04_ruby.rb
```

Quelltext (9) 04_ruby_sudoers

```
1 > id
2 uid=1001(test) gid=100(users) Gruppen=100(users)
3 > l 04_ruby.rb
4 -rwx--x---+ 1 psy-xoryves users 57 12. Aug 13:52 04_ruby.rb*
5 > ./04_ruby.rb
```

```
/usr/bin/ruby: Permission denied -- ./04_ruby.rb (LoadError)
> sudo -u psy-xoryves ./04_ruby.rb
Testausgabe...
> sudo -u psy-xoryves less ./04_ruby.rb
psy-xoryves's password:
> less ./04_ruby.rb
./04_ruby.rb: Keine Berechtigung
```

Quelltext (10) 04_ruby

Falls es sich um einen Daemon handelt, kann dieser auch während der Initialisierung des Systems von root mit dem entsprechenden Benutzer gestartet werden.

3.2.4 UHID generieren

Ein *Unique Hardware Identifier* kann mittels drei Programmen erzeugt werden. Mit `dmidecode` wird das *Desktop Management Interface* über die vorhandene Hardware abgefragt. Zusätzlich müssen die angeschlossenen USB Geräte über `lsusb` erkannt werden. Zuletzt können die so gewonnen Informationen mit Hilfe eines Hashalgorithmus wie SHA512 in eine einheitliche Länge gebracht werden. Der entstandene Hash wird in dieser Thesis als UHID bezeichnet. Im folgenden Beispiel werden zur Demonstration zwei SHA256 Hashes generiert. Einmal mit und einmal ohne eingesteckten USB-Flash-Stick.

```
#!/usr/bin/ruby
## encoding: UTF-8
require "digest/sha2"

cmd_dmidecode = "sudo /usr/sbin/dmidecode"
cmd_lsusb     = "/usr/bin/lsusb"

dmidecode = `#{cmd_dmidecode}`
lsusb     = `#{cmd_lsusb}`
uhid      = Digest::SHA256.hexdigest( dmidecode + lsusb )

puts "uhid: #{uhid}"
```

Quelltext (11) 05_uhid.rb

```
> sudo grep psy /etc/sudoers
psy-xoryves ALL=(root) NOPASSWD:/usr/sbin/dmidecode
> ./05_uhid.rb > 05_uhid_256
> ./05_uhid.rb > 05_uhid_256_usb_stick
> cat 05_uhid_256 05_uhid_256_usb_stick
e1f9e9617c44656d52b568040b69e8afe4327f41436ead31364b69f2d8144ce2
37554f6f664ffccfa7c66c2bbe94cefeb77d9df9c8a21f8ed3bb2ac4befaf2ff
```

Quelltext (12) UHID generieren

3.2.5 HDD, Verzeichnisse und Partitionen Hashen

Durch das Hashen von einer kompletten Festplatte oder nur Partitionen davon, kann überprüft werden ob das Betriebssystem oder Teile davon verändert wurden. Dabei dürfen sich veränderliche Bereiche wie `/tmp` nicht beachtet werden. Unter Unix Systemen können Gerätedateien wie `/dev/sda` oder `/dev/sda1` als Dateien behandelt werden. Das heißt, diese Dateien können wie eine Textdatei eingelesen werden. Der Hash einer zwei GB großen swap Partition kann wie folgt ermittelt werden.

```
> sudo time dd if=/dev/sdd1 | sha512sum
4190208+0 Datensätze ein
4190208+0 Datensätze aus
2145386496 Bytes (2,1 GB) kopiert, 25,5603 s, 83,9 MB/s
1.83user 10.78system 0:25.56elapsed 49%CPU (0avgtext+0avgdata
   4224maxresident)k
0inputs+0outputs (0major+323minor)pagefaults 0swaps
1e9d5fbb1e8acbf97e56065bdf0b9ec5b2be53124fdd84d0f723c0f558c39d4e
d740710279d016fc62792fd2e2f515c5089b51bfbb51efe6fd49d3b208552097

> sudo ruby -ropenssl -rbenchmark -e 'puts Benchmark.measure{\
> puts OpenSSL::Digest::SHA512.hexdigest(IO.read("/dev/sdd1"))}'
1e9d5fbb1e8acbf97e56065bdf0b9ec5b2be53124fdd84d0f723c0f558c39d4e
d740710279d016fc62792fd2e2f515c5089b51bfbb51efe6fd49d3b208552097
 22.100000   5.730000  27.830000 ( 30.497467)

> sudo time dd if=/dev/sdd1 | sha256sum
4190208+0 Datensätze ein
4190208+0 Datensätze aus
2145386496 Bytes (2,1 GB) kopiert, 36,7841 s, 58,3 MB/s
1.93user 13.05system 0:36.79elapsed 40%CPU (0avgtext+0avgdata
   4192maxresident)k
2269232inputs+0outputs (4major+316minor)pagefaults 0swaps
a511f9dc194709419a19a0b77c645514daa97829fa09136476eaad6540a4c23e

> sudo ruby -ropenssl -rbenchmark -e 'puts Benchmark.measure{\
> puts OpenSSL::Digest::SHA256.hexdigest(IO.read("/dev/sdd1"))}'
a511f9dc194709419a19a0b77c645514daa97829fa09136476eaad6540a4c23e
 34.210000   6.450000  40.660000 ( 45.184068)
```

Quelltext (13) sha512(swap)

Als System wurde ein AMD E-350 mit 2*1,6GHz und 6GB RAM eingesetzt. Es ist zu erkennen, dass die Ruby Version jeweils mehr Zeit in Anspruch nimmt. Daneben ist auch gut zu erkennen, dass SHA512 auf einem 64bit System schneller arbeitetet als SHA256. Die Ruby Version benötigt zudem wesentlich mehr Speicher, da die Partition komplett in einen String eingelesen werden muss. Somit verbrauchte die Ruby Version 2GB und die Consolen Version 568MB an Hauptspeicher.

Um einzelne Verzeichnisse zu hashen, muss man zuerst Hashes von allen Dateien erzeugen und diese gemeinsam hashen. [59]

```
time sudo find /bin -type f -print0 | sort -z | xargs -0 sudo
   sha512sum | sha512sum
b162719686781cc2a251fe7ac9f19490f7553ce93479317fedae0663d123fafb
83eb7ad135064c6dbb385a7eac107e1926054ff24b878d303653afce7629fcca
real  0m2.799s
user  0m0.128s
sys 0m0.046s
```

Quelltext (14) sha512(/bin)

3.2.6 Challenge-Response

Um das Nachbauen des IDD zu erschweren, wird ein Chalenge-Response-Verfahren benutzt. Der Server stellt dem Client bei jeder Verbindung eine andere Aufgabe die gelöst werden muss. Dabei werden für jeden Client andere Aufgaben generiert.

Für jeden IDD werden zwei Schlüsselströme und eine Reihe von Methoden generiert und in den Quelltext eingebaut. Diese Methoden enthalten im Voraus zufällig generierte Berechnungen, die auf einem Schlüsselstrom arbeiten. Der IDD wird vom Server aufgefordert eine Methode auszuführen und diese mit einem Wert aus dem zweiten Schlüsselstrom zu verknüpfen.

Die Schlüsselströme werden mit Hilfe eines CSPRNG erzeugt. Ein kryptographisch sicherer Pseudozufallszahlengenerator muss im Gegensatz zu einfachen PRNGs einige zusätzliche Bedingungen erfüllen. Ein Angreifer darf mit dem Wissen von den ersten k-Bits einer Zufallssequenz nicht in polynomieller Zeit das (k+1)te-Bit mit einer Wahrscheinlichkeit von über 50% vorhersagen können. Des weiteren darf ein Angreifer mit Wissen eines Teiles einer Zufallssequenz nicht die komplette oder vorhergehende Sequenz wiederherstellen bzw berechnen können. Ein CSPRNG muss eine hohe Entropie, also Zufälligkeit aufweisen. Dies wird unter anderem durch Sammeln von unvorhersehbaren Ereignissen wie Tastatureingaben, Mausbewegungen und anderen Quellen erreicht. [50, S. 34ff] [76] [77] [83]

Das Ergebnis der Berechnung wird zum Schluss gehasht.

3.2.7 Ruby

Der IDD wird in der Programmiersprache Ruby implementiert. Es wird an dieser Stelle nicht näher darauf eingegangen warum Ruby und nicht Python oder Perl oder sonst eine Sprache.

Im Internet finden sich übermäßig viele Beiträge in Foren oder Weblogs, in denen sich die jeweiligen Anhänger gegenseitig anschwärzen.

Ruby ist eine relativ neue und moderne Programmiersprache, die in den letzten Jahren schnell gewachsen ist. Dies ist vermutlich dem Webframewrok Rails zu verdanken. In der Standardbibliothek von Ruby 1.9 sind Klassen für CSPRNG, OpenSSL, TCP Socket Server und vieles mehr bereits enthalten.

Es wird auf komplette C Implementierung verzichtet, da es wesentlich angenehmer ist mit Ruby als mit C zu arbeiten und es sich um keine zeitkritische Anwendung handelt. Im Anhang unter Kleiner Exkurs in die Welt von C ist ein Beispiel warum hier möglichst auf direkte C Benutzung verzichtet wird. Die Ruby 1.9 Version wird für jeden IDD einzeln kompiliert. Dabei werden verschiedene Daten wie generierte Schlüsselströme, öffentliche/private Schlüssel und Methoden für das Challenge-Response Verfahren zusätzlich eingebaut. Dadurch soll das Reverse Engineering erschwert werden.

3.2.8 Benutzerkennung

Neben der immer vorhandenen *root* Benutzerkennung wird es eine *idd* und eine *benutzer* Benutzerkennung geben. Der Ruby Interpreter und IDD werden idd gehören und nur dieser darf beides lesen bzw ausführen. Das Passwort von idd und root wird auf eine zufällig generierte und nicht vom Menschen eingebbare Zeichenfolge konfiguriert.

Der Benutzer mit der Kennung *benutzer* wird normale übliche Berechtigungen besitzen. Sofern man dem Benutzer erlauben möchte einen Webbrowser wie Firefox zu nutzen, sollte eine weitere Kennung angelegt werden. Diese Kennung muss starke Restriktionen bekommen und nur den Browser ausführen dürfen. Dadurch erhält man eine Art Sandbox für den Browser und Schädlinge aus dem Internet sollten es schwieriger haben sich im System einzunisten.

3.3 identity server – Server

Der IDS muss die nötige Funktionalität bereitstellen, damit sich ein Client authentifizieren kann und die Erlaubnis bekommt in das Intranet zu gelangen. Hierfür wird mit Ruby und OpenSSL ein SSL TCP Socket Server implementiert.

3.3.1 Funktionalität

Der IDS muss nach außen mit den IDDs kommunizieren können und nach innen mit Komponenten des Betriebssystems wie Firewall und Terminalservern. Von dem IDS geht auch das Challenge-Response-Verfahren aus, welches näher im Unterkapitel Challenge-Response behandelt wird.

Da der IDS weiß wo sich der IDD aufhält, also außerhalb oder innerhalb des Unternehmens, muss er dem IDD mitteilen wie er sich zu authentifizieren hat. Ebenso muss er den entsprechenden Mechanismen im Intranet mitteilen, dass die IP vom IDD dazu berechtigt ist die entsprechende Authentifizierung vorzunehmen. Die Authentifizierung wird über VPN oder 802.1X bewerkstelligt. VPN kommt zum Einsatz wenn sich der IDD außerhalb des Unternehmens befindet und 802.1X, wenn sich der IDD im Unternehmen befindet.

3.3.2 Datenbank

Um die vom IDS benötigten Informationen zu speichern wird eine Datenbank eingesetzt. Es gibt keine hohen Anforderungen an die Datenbank. Inserts finden nur statt, wenn ein IDD erzeugt wird. Deletes oder Updates finden entsprechend nur statt, wenn der IDD entfernt oder aktualisiert wird. Es werden keine Transaktionen oder komplexe Abfragen mit Joints etc benötigt. Es würde sich somit jede RDBMS wie MySQL oder PostgreSQL eignen. In diesem Falle wird allerdings die Dokumenten orientierte NoSQL Datenbank MongoDB eingesetzt. Das von MongoDB benutze Key-Value-Storage lässt sich einfacher in Ruby einbinden als Tabellen. Auch die allgemeine Benutzung von NoSQL ist einfacher und gleicht Methodenaufrufen.

3.4 Zusammenspiel von Client und Server

Nachdem das Betriebssystem hochgefahren ist, stellt der IDD eine Verbindung mit dem IDS her. Die IP des IDS ist im IDD fest einprogrammiert. Sämtliche Kommunikation findet dabei über HMAC und SSL statt. Zusätzlich wird mit den jeweils öffentlichen Schlüsseln der Gegenseite verschlüsselt. Dies verhindert, dass ein Angreifer einen Proxy IDD bauen kann oder SSL Client seitig aufbricht.

3.4.1 Challenge-Response

Sobald die Verbindung aufgebaut wurde beginnt der IDS mit dem Challenge-Response-Verfahrenn.

IDS	IDD
	Verbindung herstellen
	UUID senden
UUID überprüfen	
OK || nicht ok senden	
	UHID senden
sha(UHID) überprüfen	
OK || nicht OK senden	
	sha(HDD) senden
sha(HDD hash + salt) überprüfen	
rx.times do	
rand_meth rand_# senden	
	value rand_meth ^ arr[rand_#] senden
check	
end	
VPN || 802.1X senden	
	Verbindung schließen
	Authentifizieren

Tabelle (7) Challenge-Response-Verfahren IDS <-> IDD

4 Implementierung

In diesem Kapitel werden die konkreten Implementierungen, von den beiden Bestandteilen IDD und IDS, behandelt. Am Ende wird noch auf die Kommunikation zwischen IDD und IDS eingegangen. Zuvor werden aber noch Beispiele von Klassen vorgestellt, die für die Implementierungen benutzt werden.

4.1 in Ruby Core und Standard Library enthalten

Viele Klassen und Methoden die benötigt werden sind bereits mit dem Ruby Interpreter mitgeliefert. Hier wird die Verwendung von diesen gezeigt. Es handelt sich dabei um `ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux]`.

4.1.1 TCP Socket Server/Client

Ein TCP Socket Server und Client kann sehr einfach realisiert werden. Dabei wartet der Server auf Verbindungen von Clients und startet für jede Verbindung einen neuen Thread. Der Nachrichtenstrom ist in dem folgenden Beispiel durch ein
`n`, also ein Unix Zeilenumbruch bzw. `newline` terminiert.

```
1  #!/usr/bin/env ruby
2  # encoding: utf-8
3
4  require "socket"
5  require "thread"
6
7  @config = { host: "localhost", port: 65535 }
8
9  @server = TCPServer.new @config[:host], @config[:port]
10 while server = @server.accept
11   Thread.new server do |session|
12     begin
13       while lineIn = session.gets
14         lineIn = lineIn.chomp
15         $stdout.puts "=> #{lineIn}"
16         lineOut = "server: #{lineIn}"
17         session.puts lineOut
18       end
19     rescue Exception => e
```

```
20        $stderr.puts "Caught exception:
              #{e}\n\t#{e.backtrace.join("\n\t")}"
21      ensure
22        session.close
23      end
24    end
25  end
```

Quelltext (15) Beispiel tcp_server.rb

```
1  #!/usr/bin/env ruby
2  # encoding: utf-8
3
4  require "socket"
5  require "thread"
6
7  @config = { host: "localhost", port: 65535 }
8
9  @socket = TCPSocket.new @config[:host], @config[:port]
10
11 Thread.new do
12   while lineIn = @socket.gets
13     lineIn = lineIn.chomp
14     $stdout.puts lineIn
15   end
16 end
17
18 while lineOut = $stdin.gets
19   lineOut = lineOut.chomp
20   @socket.puts lineOut
21 end
22
23 @socket.close
```

Quelltext (16) Beispiel tcp_client.rb

Das Beispiel demonstriert einen so genannten *echo-Server*. Alles was der Client an den Server sendet, sendet der Server wieder zurück an den Client. Mit *tcpdump* und *Wireshark* wurde der Datenstrom aufgezeichnet und untersucht. Es ist leicht erkennbar, dass die Nachrichten `test` und `server: test` jeweils im Klartext zum Server bzw. zum Client gesendet wurden.

```
1  > tcpdump -w tcp.dump -i lo tcp port 65535
2  > wireshark tcp.dump
3  # File -> Export Packet Dissections -> as "Plain Text" file ...
4  > less tcp.dump.txt
5  # 14 Pakete
6  # Frame 4
7  #   Data (4 Bytes)
8  #     0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
```

```
#      ..............E.
#      0010  00 38 37 5e 40 00 40 06 05 60 7f 00 00 01 7f 00
#      .87^@.@..‘.....
#      0020  00 01 ce a9 ff ff 12 fd 52 8e 95 30 72 f6 80 18
#      ........R..0r...
#      0030  01 01 fe 2c 00 00 01 01 08 0a 00 94 71 4f 00 94
#      ...,........qO..
#      0040  65 d1 74 65 73 74
#      e.test
# Frame 8
#   Data (12 Bytes)
#      0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
#      ..............E.
#      0010  00 40 17 29 40 00 40 06 25 8d 7f 00 00 01 7f 00
#      .@.)@.@.%.......
#      0020  00 01 ff ff ce a9 95 30 72 f6 12 fd 52 93 80 18
#      .......0r...R...
#      0030  01 00 fe 34 00 00 01 01 08 0a 00 94 71 4f 00 94
#      ...4........qO..
#      0040  71 4f 73 65 72 76 65 72 3a 20 74 65 73 74
#      qOserver: test
```

Quelltext (17)]Beispiel tcp.dump[.txt]

[12] [13] [3]

4.1.2 SSL/TLS TCP Socket Server/Client

Die `OpenSSL` Klasse in Ruby ist ein einfacher Wrapper für die native OpenSSL Bibliothek. OpenSSL bietet unter anderem die folgenden Funktionen an:

- RSA Schlüssel Generierung
- X509 Certificate Generierung und Signierung
- CSPRNG
- SSL Server

Ein SSL Server wird erzeugt, indem ein vorhandener TCP Server mit dem benötigten Zertifikat und privatem Schlüssel gekapselt wird. OpenSSL bietet als Protokoll Versionen neben den SSL Versionen auch die TLS Versionen an. Das folgende Beispiel erweitert den TCP Server und Client um eine TLS Verschlüsselung.

```ruby
#!/usr/bin/env ruby
# encoding: utf-8

require "socket"
require "thread"
require "openssl"                  #für TLS

@config = { host: "localhost",
            port: 65535,
            cert: "cert.pem",      #für TLS
            key: "key.pem",        #für TLS
            ssl_version: :TLSv1 } #für TLS

@server = TCPServer.new @config[:host], @config[:port]
#für TLS{
@ssl_context = OpenSSL::SSL::SSLContext.new
@ssl_context.ssl_version = @config[:ssl_version]
@ssl_context.cert = OpenSSL::X509::Certificate.new
   File.open(@config[:cert])
@ssl_context.key = OpenSSL::PKey::RSA.new
   File.open(@config[:key])
@ssl_server = OpenSSL::SSL::SSLServer.new @server, @ssl_context
#}

while server = @ssl_server.accept
  Thread.new server do |session|
    begin
      while lineIn = session.gets
        lineIn = lineIn.chomp
        $stdout.puts "=> #{lineIn}"
        lineOut = "server: #{lineIn}"
        session.puts lineOut
      end
    rescue Exception => e
      $stderr.puts "Caught exception:
         #{e}\n\t#{e.backtrace.join("\n\t")}"
    ensure
      session.close
    end
  end
end
```

Quelltext (18) Beispiel ssl_server.rb

```ruby
#!/usr/bin/env ruby
# encoding: utf-8

require "socket"
```

```
require "thread"
require "openssl"

@config = { host: "localhost",
            port: 65535,
            cert: "root_cert.pem",#für TLS
            ssl_version: :TLSv1 } #für TLS

@socket = TCPSocket.new @config[:host], @config[:port]
#für TLS{
@ssl_context = OpenSSL::SSL::SSLContext.new
@ssl_context.ssl_version = @config[:ssl_version]
@ssl_context.ca_file = @config[:cert]
@ssl_context.verify_mode = OpenSSL::SSL::VERIFY_PEER
@ssl_socket = OpenSSL::SSL::SSLSocket.new @socket, @ssl_context
#}
@ssl_socket.connect

Thread.new do
  while lineIn = @ssl_socket.gets
    lineIn = lineIn.chomp
    $stdout.puts lineIn
  end
end

while lineOut = $stdin.gets
  lineOut = lineOut.chomp
  @ssl_socket.puts lineOut
end

@ssl_socket.close
```

Quelltext (19) Beispiel ssl_client.rb

Auch hier wurde die Datenübertragung mit tcpdump mitgeschnitten und mittels Wireshark untersucht. Durch die TLS Verschlüsselung ist die Paket Anzahl von 14 auf 19 gestiegen. Es kann keinerlei Klartext mehr erkannt werden. Das kleinste Data Fragment hat eine Länge von 37 Bytes und das größte von 2150 Bytes.

```
> tcpdump -w ssl.dump -i lo tcp port 65535
> wireshark ssl.dump
# File -> Export Packet Dissections -> as "Plain Text" file ...
> less ssl.dump.txt
# 19 Pakete
# Frame 12
#   Data (37 Bytes)
#     0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
   ..............E.
```

```
9  #     0010  00 59 0f 3c 40 00 40 06 2d 61 7f 00 00 01 7f 00
       .Y.<@.@.-a......
10 #     0020  00 01 d0 5b ff ff f0 b3 46 1c d5 2d 45 f0 80 18
       ...[....F..-E...
11 #     0030  01 83 fe 4d 00 00 01 01 08 0a 01 ac 5a 60 01 ac
       ...M........Z‘..
12 #     0040  4b f6 17 03 01 00 20 8f 08 e5 25 23 51 45 9e f9
       K.....  ...%#QE..
13 #     0050  76 cd 40 47 bc 4c 85 5e df a9 c4 7c ba 99 58 05
       v.@G.L.^...|..X.
14 #     0060  33 b8 76 f7 8d 63 a2
       3.v..c.
```

Quelltext (20)]Beispiel tcp.dump[.txt]

[9] [60]

4.1.3 X509 Certificate

Damit eine TLS Verbindung aufgebaut werden kann, muss der Server ein Zertifikat und den dazugehörigen privaten Schlüssel besitzen. Das Zertifikat wird benötigt, damit der Client verifizieren kann, dass der Server auch derjenige ist, für den er sich ausgibt. Die Verschlüsselung selbst geschieht dabei über einen von beiden Seiten vereinbarten Session Key. In dem folgenden Beispiel werden zwei RSA Schlüssel und zwei Zertifikate erstellt. Das erste Paar von Schlüssel und Zertifikat ist das so genannte *root CA*. Dieses muss mindestens selbst signiert sein und vorzugsweise von einer authentifizierten öffentlichen Stelle signiert werden. Letzteres ist nötig, damit Internet Browser und andere Anwendungen das Zertifikat verifizieren können ohne den Benutzer fragen zu müssen. Das zweite Paar wird vom Server benutzt und ist vom root CA signiert. Der Client benötigt hingegen das root CA, damit das Zertifikat vom Server verifiziert werden kann.

```
1  #!/usr/bin/env ruby
2  # encoding: utf-8
3  require "openssl"
4  require "securerandom"
5
6  root_key = OpenSSL::PKey::RSA.new 4096
7  open "root_key.pem", "w" do |io| io.write root_key.to_pem end
8  open "root_pkey.pem", "w" do |io| io.write
       root_key.public_key.to_pem end
9
10 root_ca = OpenSSL::X509::Certificate.new
11 root_ca.version = 2
12 root_ca.serial = SecureRandom.random_number 100
```

```
13 root_ca.subject = OpenSSL::X509::Name.parse "/DC=proto/CN=IDS
       CA"
14 root_ca.issuer = root_ca.subject
15 root_ca.public_key = root_key.public_key
16 root_ca.not_before = Time.now
17 root_ca.not_after = root_ca.not_before + 5 * 365 * 24 * 60 * 60
18 ef = OpenSSL::X509::ExtensionFactory.new
19 ef.subject_certificate = root_ca
20 ef.issuer_certificate = root_ca
21 root_ca.add_extension ef.create_extension("basicConstraints",
       "CA:TRUE", true)
22 root_ca.add_extension ef.create_extension("keyUsage",
       "keyCertSign, cRLSign", true)
23 root_ca.add_extension
       ef.create_extension("subjectKeyIdentifier", "hash", false)
24 root_ca.add_extension
       ef.create_extension("authorityKeyIdentifier", "keyid:always",
       false)
25 root_ca.sign root_key, OpenSSL::Digest::SHA512.new
26 open "root_cert.pem", "wb" do |io| io.print root_ca.to_pem end
27
28 key = OpenSSL::PKey::RSA.new 4096
29 open "key.pem", "w" do |io| io.write key.to_pem end
30 open "pkey.pem", "w" do |io| io.write key.public_key.to_pem end
31
32 cert = OpenSSL::X509::Certificate.new
33 cert.version = 2
34 cert.serial = SecureRandom.random_number 100
35 cert.subject = OpenSSL::X509::Name.parse "/DC=proto/CN=IDS cert"
36 cert.issuer = root_ca.subject
37 cert.public_key = key.public_key
38 cert.not_before = Time.now
39 cert.not_after = cert.not_before + 1 * 365 * 24 * 60 * 60
40 ef = OpenSSL::X509::ExtensionFactory.new
41 ef.subject_certificate = cert
42 ef.issuer_certificate = root_ca
43 cert.add_extension ef.create_extension("keyUsage",
       "digitalSignature", true)
44 cert.add_extension ef.create_extension("subjectKeyIdentifier",
       "hash", false)
45 cert.sign root_key, OpenSSL::Digest::SHA512.new
46 open "cert.pem", "wb" do |io| io.print cert.to_pem end
```

Quelltext (21) Beispiel generate_certificate.rb

[10] [11]

4.1.4 HMAC

Ruby bietet über den OpenSSL Wrapper auch HMAC an. Damit lassen sich verschlüsselte Hashes von Nachrichten erstellen. Eine Nachricht kann somit auf Integrität überprüft werden. Auch ein gewisses Maß an Authentizität kann gewährleistet werden, denn nur diejenigen, die den geheimen Schlüssel besitzen, können den entsprechenden HMAC erzeugen. HMAC ist schneller als eine RSA Signatur von Nachrichten, setzt aber einen gemeinsam bekannten geheimen Schlüssel voraus.

```
require 'openssl'
OpenSSL::Digest::SHA512.hexdigest "nachricht"
"ab15cb87be881d74b040c0b49c5a8b4e1001371f7c16993032d2517f97a6efef
21793e55ea2caf9aeb155fde2ad913e919fcfe48d50ca7fc6505bcf1864de1d3"
OpenSSL::Digest.hexdigest "sha512", "nachricht"
"ab15cb87be881d74b040c0b49c5a8b4e1001371f7c16993032d2517f97a6efef
21793e55ea2caf9aeb155fde2ad913e919fcfe48d50ca7fc6505bcf1864de1d3"
```

Quelltext (22) Beispiel SHA512

```
require 'openssl'
OpenSSL::HMAC.hexdigest "sha512", "passwort", "nachricht"
"6767f88851bd651668b9e9cb6228cdd155b574b4a0cdfdca47e97d45c36e6f98
565b9ba2fe614ae8b6fa1f4debfe00bfca8eba37cb5e9be77eed1ce82042f315"
```

Quelltext (23) Beispiel HMAC

4.1.5 CSPRNG

Über das Modul `SecureRandom` hat man einen CSPRNG zur Hand. Dieser generiert zum Beispiel den *Seed* vor der Benutzung. SecureRandom versucht zuerst OpenSSL zu benutzen und falls diese Bibliothek nicht zur Verfügung steht wird unter Unix Systemen `/dev/random` benutzt. Man kann Byte- und Zahlenströme generieren, aber auch UUIDs oder Base64, bzw. URL sichere Base64 Zufallsströme. Die folgenden Beispiele wurden in `IRB` ausgeführt.

```
require 'securerandom'
SecureRandom.random_bytes
"\xFD\xA5\xF3\xCB\tL\xF4D\xE6\xFA\xE4 a\"\xF7\x85"
SecureRandom.random_bytes 32
"q\xF4<\x19\x91,\xAC=\xEC*~H\xFA)\xA8T\xEEF\xE9\xD7\x91\x83_\x9C
\xE8\x88tvH\x8F\xD4\x8F"
```

Quelltext (24) Beispiel RandomBytes

```
require 'securerandom'
SecureRandom.random_number
0.44648710091271604
```

```
4 SecureRandom.random_number 32
5 21
```

Quelltext (25) Beispiel RandomNumber

4.2 IDD

Der IDD wird mit Hilfe von Ruby und C, angelehnt an das Beispiel ssl_client.rb, implementiert. Allerdings als Klasse und ohne Endlosschleife. Zudem werden einige Unix Programme, die bei den meisten Linux Distributionen mitgeliefert sind, für die Informationsgewinnung benutzt.

4.2.1 Ruby Implementierung

Die Hauptimplementierung ist als Ruby Klasse realisiert. Es gibt nur die öffentliche `initialize` Methode. Diese wird automatisch, durch das Erzeugen der `Idd` Klasse ausgeführt. Hierfür wird die `idd.rb` Datei gestartet. Alle restlichen Methoden sind privat deklariert. Die zwei Methoden `init` und `init_connection` dienen der Initialisierung und Konfiguration des IDD. Der Hauptablauf und die Kommunikation mit dem IDS ist in der Methode `run` umgesetzt. Die zwei Methoden `uhid` und `hdd` benutzen Unix Programme, um die benötigten Informationen zu gewinnen. Zuletzt gibt es noch eine Methode, die nur von *root* gestartet werden kann, namens `init_db`.

Während der Initialisierung wird überprüft ob der IDD durch den Benutzer root gestartet wurde. Falls ja, dann wird nur die DB um ein paar Werte ergänzt und wieder beendet. Ansonsten wird die Verbindung zum IDS hergestellt und der Hauptablauf gestartet.

```
1  def initialize
2    init
3
4    if Process.uid == 0
5      init_db
6      puts "db initialized"
7      exit 0
8    end
9
10   init_connection
11   run
12 end
```

Quelltext (26) idd.rb initialize

In der init Methode werden die Variablen für die weitere Verwendung konfiguriert. Durch die zentrale Verwaltung der benutzten Parameterwerte und Ausführungsbefehle kann die nachträgliche Erweiterung oder Anpassung leichter vorgenommen werden.

```
1  def init
2    @config = { host: "localhost",
3                port: 65535,
4                cert: "root_cert.pem",
5                ssl_version: :TLSv1,
6                uuid: "2e473c76-355e-440d-9748-c03996737234",
7                hash_funk: "sha512" }
8
9    @commands = { dmidecode: "sudo /usr/sbin/dmidecode",
10                 lsusb: "/usr/bin/lsusb",
11                 cpuinfo: "/proc/cpuinfo",
12                 diskids: "/dev/disk/by-id/*",
13                 hdparam: 'sudo /sbin/hdparm -I /dev/sd?|grep -E
                      "Number|/dev"',
14                 hdd_part: "sudo dd if=%s | sha512sum",
15                 hdd_fold: "sudo find %s -type f -print0 | sort
                      -z | xargs -0 sudo sha512sum" }
16
17   @hdd_parts = []
18   @hdd_folds = ["/etc", "/bin"]
19 end
```

Quelltext (27) idd.rb init

Die Initialisierung der TLS Verbindung mit dem Server geschieht wie in dem Beispiel 19. Die Überprüfung des Server Zertifikates wird erzwungen, damit ein Proxy entdeckt und unterbunden werden kann.

```
1  def init_connection
2    @socket = TCPSocket.new @config[:host], @config[:port]
3    @ssl_context = OpenSSL::SSL::SSLContext.new
4    @ssl_context.ssl_version = @config[:ssl_version]
5    @ssl_context.ca_file = @config[:cert]
6    @ssl_context.verify_mode = OpenSSL::SSL::VERIFY_PEER
7    @ids = OpenSSL::SSL::SSLSocket.new @socket, @ssl_context
8    @ids.connect
9  end
```

Quelltext (28) idd.rb init_connection

Der Hauptablauf des IDD geschieht in der run Methode. Die Arbeitsweise wurde bereits in der Tabelle Challenge-Response-Verfahren IDS <-> IDD skizziert. Während des challenge-response Schrittes wird über das `IddM` Modul auf die C Extension zugegriffen. Bevor die berechneten Werte an den IDS geschickt werden, werden diese vom Bytestring zu einem

Hexstring umgewandelt. Dadurch kann der Newline Charakter weiter als Trennzeichen für Nachrichten fungieren und das Leerzeichen als Trennzeichen für die beiden Rückgabewerte. Zum Schluss wird der Name der Authentifizierungsmethode ausgegeben und der IDD beendet. Da dies ein Prototyp ist, wird auf die Implementierung des Interfaces, zu anderen Komponenten an dieser Stelle, verzichtet.

```
1  def run
2    @ids.puts @config[:uuid]
3    ret = @ids.gets.chomp
4
5    unless ret == "OK"
6      puts "server: " + ret
7      exit 1
8    end
9
10   @ids.puts uhid
11   ret = @ids.gets.chomp
12
13   unless ret == "OK"
14     puts "server: " + ret
15     exit 2
16   end
17
18   @ids.puts hdd
19   ret = @ids.gets.chomp
20
21   cond = ["wrong response, closing session", "VPN", "802.1X"]
22   if cond.include? ret
23     puts "server: " + ret
24     exit 3
25   end
26
27   while not cond.include? ret
28     rm, rx = ret.split
29     puts "server: #{rm}, #{rx}"
30     value, xor = IddM.xor(rm, rx.to_i).unpack("A16"*2)
31
32     @ids.puts "#{value.unpack("H*").first}
         #{xor.unpack("H*").first}"
33
34     ret = @ids.gets.chomp
35   end
36
37   if ret == cond.first
38     puts "server: " + ret
39     exit 4
40   end
41
```

```
42   puts "server: " + ret
43   puts "Authentication complete."
44
45   exit 0
46 end
```

Quelltext (29) idd.rb run

Um die UHID zu generieren werden eine Reihe von Unix Kommandos benutzt. Durch `/procs/cpuinfo` werden die Informationen über den Prozessor abgerufen. Das Kommando `dmidecode` liefert weitreichende Informationen über die verbaute Hardware. Die angeschlossenen USB-Geräte werden mittels `lsusb` ermittelt. IDs von Festplatten können aus dem Unix Verzeichnisbaum unter dem Pfad `/dev/disk/by-id/*` abgelesen werden. Dort sind auch die Partitionen erkennbar. Der Befehl `hdparam` liefert unter anderem die Modellnummer und Seriennummer getrennt, im Gegensatz zu der Auflistung im Verzeichnisbaum. Alle gewonnenen Informationen werden in einen String verpackt und anschließend mit SHA512 gehasht.

```
1 def uhid
2   cpuinfo    = File.read @commands[:cpuinfo]
3   dmidecode = `#{@commands[:dmidecode]}`
4   lsusb      = `#{@commands[:lsusb]}`
5   diskids    = Dir[@commands[:diskids]].join("\n")
6   hdparam    = `#{@commands[:hdparam]}`
7   uhid       = cpuinfo + dmidecode + lsusb + diskids + hdparam
8   uhid_hash = OpenSSL::Digest.hexdigest(@config[:hash_funk],
      uhid)
9 end
```

Quelltext (30) idd.rb uhid

Um eine Veränderung an Festplatten oder Partitionen zu erkennen wird das Kommando `dd` benutzt. Für Verzeichnisse samt Unterverzeichnissen werden mehrere Kommandos, wie in dem Beispiel sha512(/bin) gezeigt, zusammen geschachtelt. Dabei wird immer ein SHA512 Hash generiert. Diese Hashes werden zu einer Zeichenfolge verknüpft und wieder mit SHA512 gehasht.

```
1 def hdd
2   hdd_hashes = ""
3   @hdd_parts.each do |partition|
4     hdd_hashes += `#{@commands[:hdd_part] % partition}`
5   end
6   @hdd_folds.each do |folder|
7     hdd_hashes += `#{@commands[:hdd_fold] % folder}`
8   end
9   hdd_hash = OpenSSL::Digest.hexdigest @config[:hash_funk],
      hdd_hashes
10 end
```

Quelltext (31) idd.rb hdd

Da einige Werte für die DB erst zur Laufzeit erzeugt werden können, kann der IDD unterscheiden ob er normal oder von dem Benutzer root gestartet wurde. Im letzteren Fall wird eine Verbindung mit der DB hergestellt und die Werte für UHID und HDD Hash nachgetragen. Danach kann der IDD normal gestartet werden.

```
1  def init_db
2    require "mongo"
3    require "securerandom"
4    config = { db:"TIA", collection:"clients",
        hash_funk:"sha512", salt_size:128 }
5    db = Mongo::Connection.new.db config[:db]
6    collection = db[config[:collection]]
7    idd_infos = collection.find_one(uuid:@config[:uuid])
8    id = idd_infos["_id"]
9
10   uhid_hash = OpenSSL::Digest.hexdigest config[:hash_funk], uhid
11   collection.update({_id:id}, {"$set" => {"uhid" => uhid_hash}})
12
13   #pub_key not in prototype
14
15   salt = SecureRandom.hex(config[:salt_size]/2)
16   hdd_hash = OpenSSL::Digest.hexdigest(config[:hash_funk],
        hdd+salt) + " " + salt
17   collection.update({_id:id}, {"$set" => {"hdd_hash" =>
        hdd_hash}})
18
19   collection.update({_id:id}, {"$set" => {"authorization" =>
        "802.1X"}})
20 end
```

Quelltext (32) idd.rb init_db

4.2.2 Ruby C Extension

Ein Teil der Funktionalität des IDD wird in C implementiert. Dies wird nicht getan um an Geschwindigkeit zu gewinnen. Es wird viel mehr Versucht, den Nachbau des IDD zu erschweren. Hierfür wird der Quelltext in der fast 8000 Zeilen langen `string.c` Datei untergebracht. Dies geschieht durch zwei `include` Befehle, die zum Einen die eignetliche Funktionalität und zum Anderen die Initialisierungsaufrufe einbindet.

```
1  //...
2  #include "idd_core"
3
4  void
5  Init_String(void)
6  {
```

```
7   //...
8   #include "idd_init"
9 }
```

Quelltext (33) string.c

Am Ende der Stringinitialisierung wird die C Datei `idd_init` eingebunden. Diese enthält die Methodenaufrufe um ein Modul zu erzeugen und diesem zwei Methoden zuzuweisen. Die erste Methode `idd` ist irrelevant für die Funktionalität in dieser Thesis. Die zweite Methode `xor` erwartet zwei Parameter und wird vom IDD während des Challenge-Response verwendet.

```
1 rb_mIdd = rb_define_module("IddM");
2 rb_define_module_function(rb_mIdd, "idd", rb_idd, 0);
3 rb_define_module_function(rb_mIdd, "xor", idd_xor, 2);
```

Quelltext (34) idd_init

Die `idd_code` Datei bindet die generierten Arrays und Methoden ein. Daneben ist hier die xor Operation für zwei Arrays implementiert.

```
1  #include "idd_arrays"
2
3  static VALUE rb_mIdd;
4  static VALUE
5  rb_idd(VALUE self)
6  {
7    return rb_str_new2("Yey!");
8  }
9
10 void
11 ary_xor_ary(unsigned char* xor,
12             int size,
13             const unsigned char* ary1,
14             const unsigned char* ary2)
15 {
16   int i;
17   for(i = 0; i < size; i++)
18   {
19     xor[i] = ary1[i] ^ ary2[i];
20   }
21 }
22
23 #include "idd_methods"
```

Quelltext (35) idd_code

Die zwei Arrays werden von dem Skript generate_c_arrays.rb generiert. Dabei handelt es sich um jeweils 1024 Einträge. Diese bestehen aus jeweils 16 zufälligen Bytes, die als eine

Zeichenkette behandelt werden. Das erste Arrays wird intern von den generierten Methoden benutzt. Das zweite Array wird benutzt, um einen vom IDS zufällig gewählten Indexeintrag, mit der ebenfalls zufällig gewählten Methode, mit xor zu verknüpfen.

```
1  unsigned char idd_ary0[1024][16] = {
2    {0x19, 0xf8, 0xb0, 0x4b, 0x59, 0x27, 0x77, 0x9d, 0x5a, 0xd1,
        0x98, 0x9f, 0x74, 0x45, 0x75, 0xa2},
3    {0xbf, 0xfe, 0x18, 0xd6, 0xc9, 0x2e, 0x2a, 0x25, 0x38, 0x2a,
        0x59, 0x78, 0x82, 0x4d, 0x3d, 0xc2},
4    ...
5    {0x12, 0xe0, 0x5d, 0x5a, 0x47, 0xfd, 0x4d, 0x14, 0x5e, 0xf1,
        0xd9, 0x51, 0x06, 0x8c, 0xdc, 0xcf},
6    {0x3c, 0x78, 0xf4, 0x3e, 0x90, 0x36, 0x49, 0x7a, 0x00, 0xe2,
        0x51, 0xa3, 0x8f, 0x7b, 0x04, 0x3f}
7    };
8  unsigned char idd_ary1[1024][16] = {
9    {0x82, 0x3e, 0xb3, 0x79, 0x16, 0x71, 0x8d, 0xa4, 0x71, 0x53,
        0x36, 0x02, 0x0a, 0xd9, 0x4e, 0xdd},
10   {0x38, 0x38, 0xe4, 0x2d, 0x5b, 0x4a, 0xfa, 0xdd, 0x98, 0x2b,
        0x39, 0x33, 0x1d, 0xa1, 0xf8, 0x84},
11   ...
12   {0x6b, 0xd2, 0xee, 0x9e, 0x71, 0xda, 0xf0, 0x2b, 0xcd, 0xb3,
        0x7e, 0x49, 0x5b, 0xfb, 0xc3, 0x53},
13   {0x70, 0xc0, 0xa2, 0xb9, 0x4a, 0x53, 0x9d, 0xa1, 0xb4, 0xe0,
        0x83, 0x33, 0x74, 0x10, 0x84, 0xc7}
14   };
```

Quelltext (36) idd_arrays

Die Funktionalität wird von dem Skript generate_c_methods.rb generiert. Dabei handelt es sich um eine Reihe von Methoden, die elf zufällige Einträge aus dem ersten Array mit xor verknüpfen. Die Anzahl der Methoden ist in dem Generatorskript definiert. In der Datei `idd_methods` ist auch die Schnittstelle zwischen C und Ruby implementiert. Auf diese Methode, `idd_xor`, wird in der Initialisierung verwiesen und wird somit vom Modul angeboten.

```
1  void
2  meth000(unsigned char* xor, int size)
3  {
4  ary_xor_ary(xor, size, idd_ary0[561], idd_ary0[870]);
5  ary_xor_ary(xor, size, xor, idd_ary0[143]);
6  ary_xor_ary(xor, size, xor, idd_ary0[679]);
7  ary_xor_ary(xor, size, xor, idd_ary0[239]);
8  ary_xor_ary(xor, size, xor, idd_ary0[676]);
9  ary_xor_ary(xor, size, xor, idd_ary0[402]);
10 ary_xor_ary(xor, size, xor, idd_ary0[43]);
11 ary_xor_ary(xor, size, xor, idd_ary0[581]);
12 ary_xor_ary(xor, size, xor, idd_ary0[774]);
13 ary_xor_ary(xor, size, xor, idd_ary0[959]);
```

```
14 }
15 ...
16 //Wenn "VALUE self" weg gelassen wird, dann wird "VALUE index"
       als String behandelt, egal was übergeben wird!
17 static VALUE
18 idd_xor(VALUE self, VALUE method, VALUE index)
19 {
20   char* method_s = RSTRING_PTR(method);
21   int   method_i = RSTRING_LEN(method);
22   int ary1_index = NUM2INT(index);
23   //value of method
24   unsigned char xor1[16];
25   //value of method^ary1[x]
26   unsigned char xor2[16];
27   //xor1xor2
28   unsigned char xor[32];
29
30   if(strncmp(method_s, "meth000", method_i) == 0)
31   {
32     meth000(xor1, 16);
33   }
34   else if(strncmp(method_s, "meth001", method_i) == 0)
35   {
36     meth001(xor1, 16);
37   }
38   ...
39   else
40   {
41     return Qnil;
42   }
43
44   ary_xor_ary(xor2, 16, xor1, idd_ary1[ary1_index]);
45
46   memcpy(xor, xor1, 16);
47   memcpy(&xor[16], xor2, 16);
48
49   return rb_str_new(xor, 32);
50 }
```

Quelltext (37) idd_methods

[37] [2] [64] [15] [41] [32] [31] [52]

4.2.3 Generatoren für C Extension

Die für den IDD benötigten zwei Arrays und zugehörigen Methoden sind sich alle ähnlich. Bei den Arrays unterscheiden sich die Werte und bei den Methoden die Indices des Arrays. Somit kann die Implementierung dahingehend vereinfacht werden, dass sowohl Arrays als auch Methoden zufällig generiert werden können. Somit ist zwar öffentlich bekannt welche Struktur gegeben ist, allerdings nicht wie die genaue Implementierung aussieht, da sie generiert wird.

Die Array Generierung wird mit Hilfe von Rubys `SecureRandom.random_bytes` Methode implementiert. Dabei wird für jeden der 1024 Einträge eines Arrays eine zufällige Bytefolge der Länge 16 generiert. Für C wird jede Bytefolge in die 16 Einzelbytes zerlegt, diese in einen 8-Bit unsigned Char umgewandelt und als Hex Darstellung der Form `"0x%02x"` in die entsprechende C Datei geschrieben. Ein Ausschnitt der so entstandenen C Datei ist in idd_arrays zu sehen. Zusätzlich werden die Bytefolgen zu einer Zeichenkette zusammengefügt, mit AES verschlüsselt und in der DB hinterlegt. Damit der IDD die verschlüsselte Bytekette wieder in ein Array transferieren kann, muss diese zuerst entschlüsselt werden und dann mit der Methode `.unpack("A16"*1024)` wieder in die 1024 Einzelbyteketten zerlegt werden.

```
1  #!/usr/bin/env ruby
2  # encoding: UTF-8
3
4  require "securerandom"
5  require "mongo"
6
7  @config = { size:1024, bytes:16, times:2 }
8  @config_mongo = { db:"TIA", collection:"clients" }
9  @config_ids = { key_path:"../ids/Beispiele/key.pem",
       cipher:"aes-256-cbc" }
10 @config_idd = { uuid:"2e473c76-355e-440d-9748-c03996737234" }
11
12 @key = OpenSSL::PKey::RSA.new File.read(@config_ids[:key_path])
13 @cipher = OpenSSL::Cipher.new @config_ids[:cipher]
14 @cipher.encrypt
15 enc_key = @cipher.random_key
16 enc_key_bin = BSON::Binary.new @key.public_encrypt(enc_key)
17 @db = Mongo::Connection.new.db @config_mongo[:db]
18 @collection = @db[@config_mongo[:collection]]
19
20 @id = @collection.insert({uuid:@config_idd[:uuid],
       enc_key:enc_key_bin})
21
22 file = File.open "idd_arrays", "wb"
23
24 @config[:times].times do |t|
25   bytes = []
26   bytes_hex_array = []
```

```
27   file.puts "  unsigned char
        idd_ary#{t}[#{@config[:size]}][#{@config[:bytes]}] = {"
28   @config[:size].times do
29     bytes_hex_string = SecureRandom.random_bytes @config[:bytes]
30     bytes_hex_array << bytes_hex_string
31     bytes_bin_string =
          bytes_hex_string.unpack("C*").collect{|b| "0x%02x" %
          b}.join(", ")
32     bytes_bin_string = "    {" + bytes_bin_string + "}"
33     bytes << bytes_bin_string
34   end
35   file.puts bytes.join(",\n")
36   file.puts "  };"
37
38   iv = @cipher.random_iv
39   iv_enc = @key.public_encrypt iv
40   iv_enc_bin = BSON::Binary.new iv_enc
41   bytes_enc = @cipher.update(bytes_hex_array.join("")) +
        @cipher.final
42   bytes_enc_bin = BSON::Binary.new bytes_enc
43   @collection.update({_id:@id}, {"$set" => {"array#{t}_iv" =>
        iv_enc_bin, "array#{t}" => bytes_enc_bin}})
44 end
45
46 file.close
```

Quelltext (38) generate_c_arrays.rb

Die Methoden werden wie in idd_methods zu sehen generiert. Der Großteil wird in einer Schleife durch den selben String generiert. Dieser enthält lediglich einen Platzhalter für die Array Indices, der durch eine Zufallszahl ersetzt und dann in die C Datei geschrieben wird. Gleichzeitig wird das Ergebnis der Methoden im Voraus berechnet und in die Datenbank geschrieben. Für die Vorberechnung wird jede Bytefolge in einen Hexstring umgewandelt und dieser zu einem Integer konvertiert. Das Ergebnis wird direkt im Anschluss wieder in eine Bytefolge transferiert.

```
1 #!/usr/bin/env ruby
2 # encoding: UTF-8
3
4 require "securerandom"
5 require "mongo"
6
7 @config = { size:1024, bytes:16, min:50, max:256 }
8 @config_mongo = { db:"TIA", collection:"clients" }
9 @config_ids = { key_path:"../ids/Beispiele/key.pem",
     cipher:"aes-256-cbc" }
10 @config_idd = { uuid:"2e473c76-355e-440d-9748-c03996737234",
     salt_size:128, hash_funk:"sha512" }
```

```
11
12 @db = Mongo::Connection.new.db @config_mongo[:db]
13 @collection = @db[@config_mongo[:collection]]
14 @idd_infos = @collection.find_one("uuid" => @config_idd[:uuid])
15 @id = @idd_infos["_id"]
16
17 @key = OpenSSL::PKey::RSA.new File.read(@config_ids[:key_path])
18 enc_key = @key.private_decrypt(@idd_infos["enc_key"].to_s)
19 iv = @key.private_decrypt(@idd_infos["array0_iv"].to_s)
20 cipher = OpenSSL::Cipher.new @config_ids[:cipher]
21 cipher.decrypt
22 cipher.key = enc_key
23 cipher.iv = iv
24
25 array0 = cipher.update(@idd_infos["array0"].to_s) + cipher.final
26 array0 = array0.unpack "A16"*1024
27
28 t_outer = 0
29 while t_outer < @config[:min]
30   t_outer = SecureRandom.random_number @config[:max]
31 end
32 t_inner = 9
33
34 file = File.open "idd_methods", "w"
35
36 methods = {}
37 meth_names = []
38 t_outer.times do |t|
39   meth_name = "meth%03i" % t
40   meth_names << meth_name
41
42   file.puts "void"
43   file.puts meth_name + "(unsigned char* xor, int size)"
44   file.puts "{"
45   rx0 = SecureRandom.random_number @config[:size]
46   rx1 = SecureRandom.random_number @config[:size]
47   file.puts "ary_xor_ary(xor, size, idd_ary0[#{rx0}],
       idd_ary0[#{rx1}]);"
48   xor1 = array0[rx0].unpack("H*").first.to_i(16)
49   xor2 = array0[rx1].unpack("H*").first.to_i(16)
50   xor  = Array.new(1){"%032x" % (xor1 ^ xor2)}.pack("H*")
51   t_inner.times do
52     rx = SecureRandom.random_number @config[:size]
53     file.puts "ary_xor_ary(xor, size, xor, idd_ary0[#{rx}]);"
54     xor1 = xor.unpack("H*").first.to_i(16)
55     xor2 = array0[rx].unpack("H*").first.to_i(16)
56     xor  = Array.new(1){"%032x" % (xor1 ^ xor2)}.pack("H*")
```

```
57    end
58    file.puts "}"
59    #SecureRandom.hex(S) returns random hex of size S*2
60    salt = SecureRandom.hex(@config_idd[:salt_size]/2)
61    xor_hash = OpenSSL::Digest.hexdigest(@config_idd[:hash_funk],
          xor+salt)
62    methods[meth_name] = xor_hash + " " + salt
63  end
64
65  @collection.update({_id:@id}, {"$set" => {"methods" =>
        methods}})
66
67  file.puts "static VALUE"
68  file.puts "idd_xor(VALUE self, VALUE method, VALUE index)"
69  file.puts "{"
70  file.puts "  char* method_s = RSTRING_PTR(method);"
71  file.puts "  int   method_i = RSTRING_LEN(method);"
72  file.puts "  int ary1_index = NUM2INT(index);"
73  file.puts "  //value of method"
74  file.puts "  unsigned char xor1[16];"
75  file.puts "  //value of method^ary1[x]"
76  file.puts "  unsigned char xor2[16];"
77  file.puts "  //xor1xor2"
78  file.puts "  unsigned char xor[32];"
79  file.puts ""
80  meth_name = meth_names.shift
81  file.puts "  if(strncmp(method_s, \"#{meth_name}\", method_i)
        == 0)"
82  file.puts "  {"
83  file.puts "    #{meth_name}(xor1, 16);"
84  file.puts "  }"
85  meth_names.each do |name|
86    file.puts "  else if(strncmp(method_s, \"#{name}\", method_i)
          == 0)"
87    file.puts "  {"
88    file.puts "    #{name}(xor1, 16);"
89    file.puts "  }"
90  end
91  file.puts "  else"
92  file.puts "  {"
93  file.puts "    return Qnil;"
94  file.puts "  }"
95  file.puts ""
96  file.puts "  ary_xor_ary(xor2, 16, xor1, idd_ary1[ary1_index]);"
97  file.puts ""
98  file.puts "  memcpy(xor, xor1, 16);"
99  file.puts "  memcpy(&xor[16], xor2, 16);"
```

```
100 file.puts ""
101 file.puts "  return rb_str_new(xor, 32);"
102 file.puts "}"
103
104 file.close
```

Quelltext (39) generate_c_methods.rb

Der Weg von einem Binärstring zu seiner Integerdarstellung ist recht einfach. Der umgekehrte Weg ist allerdings etwas tückisch. Das folgende Beispiel demonstriert den falschen und richtigen Weg. Dabei wird auch der Unterschied verdeutlicht.

```
1  # falscher Weg
2  81558196332667903337895086958571069 92.to_s(16)
3  "622c0f423b5a21683930fc344791030"
4  # die führende Null fehlt, somit kommt eine falsche Bytefolge
      zustande
5  ["622c0f423b5a21683930fc344791030"].pack("H*")
6  => "b,\x0FB;Z!h90\xFC4G\x91\x03\x00"
7
8  # richtiger Weg
9  "%032x" % 81558196332667903337895086958571069 92
10 "0622c0f423b5a21683930fc344791030"
11 # die führende Null ist wichtig
12 ["0622c0f423b5a21683930fc344791030"].pack("H*")
13 => "\x06\"\xC0\xF4#\xB5\xA2\x16\x83\x93\x0F\xC3Dy\x100"
14
15 # wenn man nun diese Bytefolgen in ein int umwandeln möchte,
      kommen zwingend zwei verschiedene Werte raus
16 "b,\x0FB;Z!h90\xFC4G\x91\x03\x00".unpack("H*").first
17 => "622c0f423b5a21683930fc3447910300"
18 ["622c0f423b5a21683930fc344791030"].pack("H*").unpack("H*").first.to_i(16)
19 => 1304931141322686453406321391331371 3872
20
21 "\x06\"\xC0\xF4#\xB5\xA2\x16\x83\x93\x0F\xC3Dy\x100".unpack("H*").first
22 => "0622c0f423b5a21683930fc344791030"
23 ["0622c0f423b5a21683930fc344791030"].pack("H*").unpack("H*").first.to_i(16)
24 => 81558196332667903337895086958571069 92
25
26 # wenn die richtigen Werkzeuge benutzt werden,
27 # dann kommen auch die richtigen Ergebnisse bei raus
```

Quelltext (40) Problem Binärstring<->Integer

4.3 IDS

Die Implementierung des IDS wird hier im Detail besprochen. Der IDS besteht aus zwei Kernkomponenten. Der Basic Teil kümmert sich um die Initialisierung des Servers. Er lädt das Zertifikat, den privaten Schlüssel und lauscht an einem Port. Sobald sich ein Client verbindet, wird eine TLS Session gestartet, um den Core erweitert und an einen Thread übergeben. Der Core überprüft die Identität des Clients und entscheidet ob sich der Client mit dem Intranet verbinden darf oder nicht.

4.3.1 IDS Basic

Der Server ist von der Grundstruktur wie in Beispiel ssl_server.rb zu sehen aufgebaut. Der einzige Unterschied ist der, dass in der Schleife, die Session um den IDS Core erweitert wird. Somit muss die Grundstruktur des Servers bei der weiteren Implementierung nicht mehr verändert werden.

```
1  # ...
2  require_relative 'ids_core'
3  # ...
4  while server = @ssl_server.accept
5    server.extend IDSCore
6    Thread.new server do |session|
7      begin
8        session.run
9      rescue Exception => e
10 # ...
```

Quelltext (41) IDS Basic Unterschied zu Beispiel Server

4.3.2 IDS Core

Der Core besteht aus einer öffentlichen Methode und fünf privaten Methoden. Die öffentliche Methode `run` wird in jeweils einem neuen Thread gestartet. Falls an irgend einer Stelle eine Überprüfung fehl schlägt, dann wird die Verbindung zum Client geschlossen und die `run` Methode verlassen.

Die fünf privaten Methoden lauten `init`, `check_uuid`, `check_uhid`, `check_hdd` und `challenge_response`. Die erste Methode initialisiert den privaten RSA Schlüssel und die Mongo DB Verbindung samt Auswahl der Collection. Anschließend werden die drei `check*` Methoden ausgeführt. Damit wird überprüft ob der Client die richtigen Werte für UUID, UHID und HDD Hash liefert. Zuletzt wird das Challenge-Response-Verfahren durchgeführt.

Die `challenge_response` Methode entschlüsselt zuerst das zweite Array, das in dem IDD enthalten ist. Dieses ist mit dem öffentlichen Schlüssel des IDS in der Mongo DB hinterlegt. Danach wird eine Zufallszahl ungleich null generiert. Diese bestimmt die Anzahl an Wiederholungen der Challenge-Response. In jeder Wiederholung wird zufällig eine Methode und eine Zufallszahl gewählt. Die möglichen Methodennamen sind in der Mongo DB, bei dem zugehörigen IDD Eintrag, hinterlegt. Aus der Größe des entschlüsselten Arrays ergibt sich die obere Grenze für die Zufallszahl. Der IDS sendet den Methodennamen und die Zufallszahl an den IDD. Der IDD sendet das Ergebnis der Methode und das Ergebnis xor dem Wert, der in dem zweiten Array an der Stelle der Zufallszahl steht. Der IDS vergleicht den gehashten Methodenwert mit Salt, mit dem Wert in der Mongo DB. Danach wird der zurückgelieferte Wert xor dem Wert in dem entschlüsselten Array an der Stelle der Zufallszahl berechnet und mit dem zurückgesendetem Wert verglichen. Falls einer der Werte nicht dem Wert in der Mongo DB entspricht, dann wird die Verbindung geschlossen und die gesamte Methode abgebrochen.

Sofern alle Methoden erfolgreich ausgeführt werden konnten, wird die Autorisierungsmethode an den IDD gesendet und die Verbindung geschlossen. An dieser Stelle muss die Schnittstelle zu externen Anwendungen wie Firewall noch implementiert werden. Darauf wird in dieser Thesis verzichtet.

```
1  def init
2    @mongo_config = { db:"TIA", collection:"clients" }
3    @config = { key_path:"Beispiele/key.pem", session_key_size:64,
4                hash_funk:"sha512", rx:11, cipher:"aes-256-cbc" }
5
6    @key = OpenSSL::PKey::RSA.new File.read(@config[:key_path])
7
8    @db = Mongo::Connection.new.db @mongo_config[:db]
9    @collection = @db[@mongo_config[:collection]]
10 end
```

Quelltext (42) ids_core init

```
1  def check_uuid
2    uuid = gets.chomp
3    @idd_infos = @collection.find_one(uuid:uuid)
4    if @idd_infos.nil?
5      puts "unknown uuid, closing session"
6      return 1
7    else
8      puts "OK"
9    end
10 end
```

Quelltext (43) ids_core check_uuid

```
1  def check_uhid
2    uhid = gets.chomp
3    uhid_sha = OpenSSL::Digest.hexdigest @config[:hash_funk], uhid
```

```
4   uhid_hash = @idd_infos["uhid"]
5   if uhid_hash.nil? || uhid_sha != uhid_hash
6     puts "unknown uhid, closing session"
7     return 2
8   else
9     puts "OK"
10  end
11 end
```

Quelltext (44) ids_core check_uhid

```
1 def check_hdd
2   hdd = gets.chomp
3   hdd_hash, salt = @idd_infos["hdd_hash"].split
4   hdd_sha = OpenSSL::Digest.hexdigest(@config[:hash_funk],
       hdd+salt)
5   if hdd_sha != hdd_hash
6     puts "unknown hdd_hash, closing session"
7     return 3
8   end
9 end
```

Quelltext (45) ids_core check_hdd

```
1   def challenge_response
2     enc_key = @key.private_decrypt(@idd_infos["enc_key"].to_s)
3     iv = @key.private_decrypt(@idd_infos["array1_iv"].to_s)
4     cipher = OpenSSL::Cipher.new @config[:cipher]
5     cipher.decrypt
6     cipher.key = enc_key
7     cipher.iv = iv
8
9     array1 = cipher.update(@idd_infos["array1"].to_s) +
         cipher.final
10    array1 = array1.unpack "A16"*1024
11
12    rx = 0
13    while rx == 0
14      rx = SecureRandom.random_number @config[:rx]
15    end
16    idd_methods       = @idd_infos["methods"]
17    idd_methods_names = idd_methods.keys
18    idd_methods_size  = idd_methods_names.size
19    rx.times do
20      rand_meth = idd_methods_names[
21                    SecureRandom.random_number(idd_methods_size)]
22      rand_num  = SecureRandom.random_number array1.size
23      puts "#{rand_meth} #{rand_num}"
24
```

```
25        value, xor = gets.chomp.split
26        value = Array.new(1){value}.pack("H*")
27        xor = Array.new(1){xor}.pack("H*")
28
29        value_hash, salt = idd_methods[rand_meth].split
30        value_sha =
             OpenSSL::Digest.hexdigest(@config[:hash_funk],
             value+salt)
31        xor1 = value.unpack("H*").first.to_i(16)
32        xor2 = array1[rand_num].unpack("H*").first.to_i(16)
33        value_xor = xor1 ^ xor2
34
35        if value_sha != value_hash ||
             xor.unpack("H*").first.to_i(16) != value_xor
36          puts "wrong response, closing session"
37          return 4
38        end
39      end
40    end
```

Quelltext (46) ids_core challenge_response

4.3.3 Mongo DB Struktur

Die Einträge einer Mongo DB Collection sehen wie Dokumente und nicht wie Tabellen bei RDBMS aus. Diese Dokumente sind an JSON oder auch Ruby Hashes angelehnt. Dies ermöglicht ein sehr einfaches und bequemes Arbeiten mit den Inhalten der Datenbank. Der folgende Quellcodeausschnitt zeigt Beispielhaft wie ein solches Dokument für einen IDD aussieht.

```
1  {
2    "uuid" => "uuid",
3    "uhid" => "sha512(uhid)",
4    "public_key" => "public key in pem format",
5    "hdd_hash" => "sha512(hdd+SALT) SALT",
6    "methods" = {
7      "method0" => "sha512(value0+SALT) SALT",
8      "methodX" => "sha512(valueX+SALT) SALT",
9      "method9" => "sha512(value9+SALT) SALT"
10   },
11   "enc_key" = "pub_ids_enc(random_key)",
12   "array1_iv" = "pub_ids_enc(random_iv)",
13   "array1" = "enc(byte_array1.join(""))",
14   "array2_iv" = "pub_ids_enc(random_iv)",
15   "array2" = "enc(byte_array2.join(""))",
16   "hdd_img_path" => "path to image",
```

```
17   "authorization" => "vpn || 802.1X"
18 }
```

Quelltext (47) Mongo DB Struktur

Um die Einträge der Datenbank zu schützen, falls einem Angreifer gelingt die Datenbank zu kompromittieren, werden diese bis auf wenige Ausnahmen nicht im Klartext abgelegt. Die folgende Auflistung zeigt, wie die einzelnen Einträge zusammengesetzt sind.

- Die UUID wird im Klartext abgelegt. Dieser Wert ist mit einem Benutzernamen zu vergleichen. Die Datenbank wird also nach diesem Wert durchsucht.

- Die UHID wird als SHA512 Hash abgelegt. Auf einen Salt wird hier verzichtet, da die UHID wie die UUID eindeutig sein muss.

- Der öffentliche Schlüssel des IDD wird im Klartext abgelegt, da er ohnehin für die Öffentlichkeit gedacht ist. In dieser Thesis wird allerdings auf die Verwendung des öffentlichen IDD Schlüssels verzichtet.

- Der HDD Hash wird zuerst mit einem Salt versehen und dann mittels SHA512 gehasht. Der SHA512 Hash wird zusammen mit dem Salt abgelegt. Als Trennzeichen dient ein Leerzeichen. Der Salt ist nötig, da es mehrere IDDs mit dem selben Festplatteninhalt geben könnte.

- Die generierten Methodennamen und der zugehörige Wert werden als Hashzuordnung abgelegt. Dabei ist der Name der Schlüssel und der Rückgabewert der Methode der Wert. Der Wert wird wie der HDD Hash mit einem Salt versehen als SHA512 abgelegt.

- Es wird ein AES Schlüssel benötigt um die generierten Arrays zu verschlüsseln und entschlüsseln. Dabei wird dieser Schlüssel mit dem öffentlichen Schlüssel des IDS verschlüsselt in der Datenbank abgelegt.

- Die beiden generierten Arrays sind relativ groß und müssen deshalb symmetrisch verschlüsselt werden. Als Verschlüsselung wurde AES-256-CBC eingesetzt und der zugehörge IV wurde jeweils wie der AES Schlüssel verschlüsselt.

- Der Pfad zu dem HDD Image kann im Klartext abgelegt werden. Allerdings wird dies in der Thesis nicht verwendet.

- Die Autorisierungsmethode wird als Klartext abgelegt. In der Thesis wird lediglich dieser Wert an den IDD zurück geschickt. Auf die weitere Implementierung wird an dieser Stelle verzichtet.

4.4 Zusammenspiel von Client und Server

Das Zusammenspiel von IDS und IDD funktioniert etwas launisch. Beim Testen wurde festgestellt, dass die Überprüfung vom IDS nicht immer funktioniert. Je nach dem, welche generierte Methode, mit welchem Wert aus dem zweiten Array verknüpft wird, scheitert die Überprüfung von dem XOR. Die Berechnung, die auf dem ersten Array basiert wird immer korrekt ausgeführt. Die Fehlerursache konnte nicht gefunden werden. Es besteht der Verdacht, dass es an den generierten Arrays liegt. Da diese auf zufälligen Bytesströmen basieren, könnte es sein, dass C und Ruby einige Werte anders interpretieren oder behandeln. Dies muss genauer untersucht werden. Höchstwahrscheinlich wurde auch einfach nur die falsche Methode benutzt um die Bytefolgen umzuwandeln.

Sofern die generierten Werte fehlerfrei sind, sieht eine mögliche Session wie in dem folgenden Beispiel aus. Dabei ist der Ablauf wie in Tabelle Challenge-Response-Verfahren IDS <-> IDD geschildert. Es wurde nur lokal getestet.

```
1  > ruby ./ids_basic.rb
2  uuid: 2e473c76-355e-440d-9748-c03996737234
3  uhid:
4  ef8fdd8c4903cce3120a2e7d2800748635d2e3f1e49ef4e511202455d9aece3e
5  aa913c8c840b3dca2788d014e8905e39906aa67ca220ff5c5cd0e4b455554625
6  hdd:
7  28e310687060eb8aa4f7b2c90944373dbbe2dc6a67b93db459f045e450e43ab6
8  d49d161825dd18d1c8ba3d5582e020868e61c6d2cf19d25a7a1775f4fd000851
9  val: "\xDA\xD3\xEF\x1A\xC2\x14Q\xA3\xA2\r\xFDW\x19;r‘"
10 xor: "\xDD\xCB\x99\x86.\x98\xB1\x91\xA6\xD3\xCC-\xC8\xEBvU"
11 hash:
12 "02bf97ef6e85e572d0db3738762798139ab4e0a381595bcd91c021c61f39a2b
13 910f5d98c53c58bc9bbee0d3ae6987855b4f705e9df37ca032263b2c49d53d9e7"
14 sha :
15 "02bf97ef6e85e572d0db3738762798139ab4e0a381595bcd91c021c61f39a2b
16 910f5d98c53c58bc9bbee0d3ae6987855b4f705e9df37ca032263b2c49d53d9e7"
17 ary: 29481653717041301925935898427950546 6965
18 xor: 29481653717041301925935898427950546 6965
19 sha!=hash? false
20 -------------------------
21 ##########################################################################
22 > ruby/bin/ruby ./idd.rb
23 server: meth097, 485
24 "\xDA\xD3\xEF\x1A\xC2\x14Q\xA3\xA2\r\xFDW\x19;r‘"
25 "\xDD\xCB\x99\x86.\x98\xB1\x91\xA6\xD3\xCC-\xC8\xEBvU"
26 -------------------------
27 server: 802.1X
28 Authentication complete.
```

Quelltext (48) Beispiel Prototyp IDS<->IDD

5 Angriffsszenarien und Härtetest

In diesem Kapitel werden verschiedene Angriffsszenarien besprochen. Da kein Sicherheitsexperte, Reverse Engineer oder Hacker diese Thesis geschrieben hat, gehen die Erläuterungen nicht so tief ins Detail, wie es wünschenswert wäre.

5.1 root Rechte

Wie schon mehrmals erwähnt, scheitert das gesamte Konzept, sobald ein Angreifer über root Rechte verfügt. Man kann zwar einem Angreifer das Aushebeln von Sicherheitsmechanismen erschweren, aber nicht unterbinden. Zumindest nicht mit Software. Dies zeigen genug Beispiele aus der Musik, Film und Spiele Industrie. Es ist lediglich eine Frage der Zeit. Dabei spielt nicht unbedingt der Kosten-Nutzen-Faktor eine Rolle. Genügend Hacker brechen Sicherheitssysteme aus reinem Vergnügen und Interesse, oft ohne einen kriminellen Hintergedanken. Es macht ihnen einfach Spaß.

Mit root Rechten kann ein Angreifer den IDD dazu bringen alle relevanten Werte aus der Ruby C Extension auszugeben. Dafür muss nicht einmal die Binärdatei untersucht werden. Durch das `strings` Kommando können unter anderem alle Methodennamen aus der `libruby-static.a` ausgelesen werden. Nun müssen allerdings 71393 Zeilen nach den relevanten Methodennamen durchsucht werden. Sofern man die entsprechenden Namen gefunden hat, kann `IddM.xor` in einer Schleife mit den Methodennamen gefüttert werden. Hinzu kommen allerdings noch alle möglichen Werte für den zweiten Parameter, nämlich der Array Index. Nichtsdestotrotz ist dieser brutforce Ansatz recht schnell. Die Methodennamen sind das einzige Problem, wobei dies auch automatisiert werden kann. Es werden alle Namen durchprobiert und gemerkt ob das IddM Modul eine passende Ausgabe geliefert hat.

5.2 strings, objdump und hexdump

Der kompilierte C Teil wird in die `libruby-static.a` gelinkt. Es wurde mit den Programmen `strings`, `objdump` und `hexdump` jeweils von dem originalen und von dem selbst übersetzen libruby-static.a eine Ausgabe gespeichert. Anschließend wurden mit `diff` die Unterschiedene in jeweils weitere Dateien geschrieben.

- strings
  - original: 70.612 Zeilen

- C Extension: 781 Zeilen mehr
- diff: 55.425 Zeilen

- objdump
  - original: 2.132.698 Zeilen
  - C Extension: 449.096 Zeilen mehr
  - diff: 4.647.179 Zeilen

- hexdump
  - original: 824.795 Zeilen
  - C Extension: 375.398 Zeilen mehr
  - diff: 2.024.984 Zeilen

An den großen Zahlen ist erkennbar, dass eine Analyse von Hand unmöglich ist. Es ist also auch mit einem diff des originalen Ruby Interpreters und dem selbst gebauten nicht so leicht einen Nachbau zu erstellen. Es ist zudem möglich, das der `gcc` Teile von Arrays oder Methoden weg optimiert hat. Dies würde das Suchen nach bekannten Mustern erschweren.

5.3 Hardwarezugriff

Sobald man Hardwarezugriff hat, also zum Beispiel die Festplatte ausbauen kann, kann man jedes Sicherheitssystem umgehen. Dies gilt natürlich nur für solche, die allein mit Software umgesetzt sind. Der XBox Hack 2.7.2 zeigt sehr deutlich, wie stark man das Gesamtsystem mit Hardwarezugriff manipulieren kann.

Sobald die Festplatte im eigenen Rechner eingebaut ist, können alle Dateien verändert werden. So auch der IDD. Dafür werden dann nicht einmal root Rechte benötigt.

5.4 Server/Client nachbauen

Den IDS nachzubauen würde keinen Sinn ergeben. Um den IDD nachzubauen benötigt man root Rechte. Ohne diese können einige Programme nicht ausgeführt werden, die zum Beispiel für die Generierung der UHID benötigt werden. Zudem kann der erweiterte Ruby Interpreter nicht ausgeführt werden, in dem die benötigten, generierten Arrays und Methoden liegen. Sobald man allerdings root Rechte besitzt, macht es keinen Sinn den IDD komplett nachzubauen. Es ist einfacher die entsprechenden Stellen direkt abzuändern.

5.5 nutzlos

Die folgenden Punkte wurden näher betrachtet, aber nicht ausführlich beschrieben. Es sind Ideen, für die bereits genannte Angriffe erfolgreich sein müssen. Allerdings sind die hier genannten Punkte dann bereits nutzlos, da es andere Möglichkeiten gibt, die durch die vorherigen Angriffe ermöglicht wurden.

- Proxy auf dem System vom IDD setzen: (IDD -> Proxy) -> (IDS)
  Benötigt root Rechte und dann kann der IDD auch direkt manipuliert werden.

- reicht HMAC oder wird AES benötigt?
  Sowohl HMAC als auch AES benötigt einen geheimen Schlüssel. Die TLS Kommunikation benutzt bereits HMAC. Sollte TLS umgangen werden, dann kann der geheime Schlüssel mit gelesen werden und ein eigenes HMAC oder AES ist wirkungslos. Dies gilt natürlich nur, falls der geheime Schlüssel als OTK implementiert ist. Ansonsten siehe nächsten Punkt.

- SSL aufbrechen / ersetzen -> mit session key / public key verschlüsseln?
  Dafür muss mindestens ein Schlüssel im IDD versteckt werden. Sobald man allerdings root Rechte besitzt, kann man die Binärdatei nach diesem Schlüssel durchsuchen. Einfacher dürfte aber das durchsuchen des Hauptspeichers sein.

5.6 nicht näher untersucht

Es konnten nicht alle Ideen und Szenarien behandelt werden. Diese werden aber der Vollständigkeit halber hier aufgezählt.

- IDD authentifiziert sich erfolgreich, Verbindung besteht, Angreifer hängt sich direkt hinter den IDD Client an das LAN Kabel dran und schickt modifizierte Pakete

- Client sha(value) -> Server; Server sha(sha value) in DB; bringt keine Sicherheit für "value"vor Client seitigem MITM

- Cold Boot Attacks on Encryption Keys [35]

- Hauptspeicher durchsuchen nach Schlüsseln

- Schlüssel zufällig in Client verstreuen -> zweiten Client nehmen und diff um Schlüsselteile zu finden

6 Fazit und Ausblick

Wenn man Software schützen muss, dann muss man dies mittels Hardwareunterstützung bewerkstelligen. Man kann kein System schützen, wie es bei Kopierschutzverfahren angewendet wird. Zudem ohne Erfolg.

Es wurde gezeigt, dass mit vorhandenen Quelloffenen Mitteln sehr viel erreicht werden kann. Allerdings geht dies nur so lange, wie der Benutzer nicht die volle Gewalt über einen Rechner hat. Dies gilt für alle Sicherheitslösungen. Sobald jemand die volle Kontrolle über einen Rechner besitzt ist jede Software machtlos. Man kann zwar mit sehr viel Arbeit eine Software so gestalten, dass sie möglichst schwer zu knacken ist, allerdings ist es nur eine Frage der Zeit, bis sie doch geknackt ist.

Ein anderer wichtiger Aspekt der gerne untergraben wird ist der, dass nur diejenigen ein System schützen können, die auch wissen wie man es brechen kann. Somit müssen Sicherheitssysteme von erfahrenen Hackern und Reverse Engineers ausgiebig getestet werden. Dies wird aber im Allgemeinen nicht getan. Vor allem bei Spielekonsolen hätten viele gravierende Fehler im Vorhinein erkannt werden können.

Eine effektive und sichere Anwendung von Sicherheitssystemen kann mit Hardwareunterstützung realisiert werden. Dabei kann auf die Verschleierung des Quelltextes verzichtet werden wie bei Verschlüsselungssystemen. Trotzdem wäre das System geschützt. Die benötigte Hardware ist aber leider nicht in breiter Verwendung.

Einige Entwickler versuchen das Problem des Softwareschutzes zu umgehen, indem sie auf Cloudentwicklung setzen. Dies funktioniert solange, wie tatsächlich die gesamte Software in der eigenen Cloud ist. Der Anwender bekommt lediglich ein Interface, meist durch den Internet Browser, zur Verfügung gestellt. Sobald allerdings eine Mischform gewählt wird, so dass die Software bei dem Benutzer liegt und die Authentifizierung über den eigenen Server geregelt wird, ist der Schutz wieder hinfällig. Bei der Spieleplatform Steam wird so ein Ansatz benutzt. Zur Zeit sind einige Emulatoren und gefälschte Server im Umlauf. Dies hat zur Folge, dass die entsprechenden Spiele, die über Steam Authentifiziert werden, illegal benutzt werden können.

7 Anhang

7.1 CD Inhalt

```
..
|-- .hg
|-- .hgignore
|-- Beispiele
|   |-- 01_ptrace_off
|   |-- 01_ptrace_off.c
|   |-- 01_ptrace_on
|   |-- 01_ptrace_on.c
|   |-- 01_test
|   |-- 02_exit
|   |-- 02_exit.c
|   |-- 03_password1
|   |-- 03_password1.c
|   |-- 03_password2
|   |-- 03_password2.c
|   |-- 03_password3
|   |-- 03_password3.c
|   |-- 04_ruby.rb
|   |-- 05_uhid.rb
|   |-- 05_uhid_256
|   |-- 05_uhid_256_usb_stick
|   |-- 05_uhid_512
|   |-- 05_uhid_512_usb_stick
|   |-- strings_03_1
|   |-- strings_03_2
|   `-- strings_03_3
|-- Dokumentation
|   |-- 01_Hauptdokument.fdb_latexmk
|   |-- 01_Hauptdokument.log
|   |-- 01_Hauptdokument.pdf
|   |-- 01_Hauptdokument.tex
|   |-- 02_Titelseite.tex
|   |-- 03_Abstrakt.tex
|   |-- 04_Inhaltsverzeichnis.tex
|   |-- 05_01_Einleitung.tex
|   |-- 05_02_Grundlagen_und_Techniken.tex
|   |-- 05_03_Entwurf_und_Design.tex
|   |-- 05_04_Implementierung.tex
|   |-- 05_05_Angriffsszenarien_und_Haertetest.tex
|   |-- 05_06_Fazit_und_Ausblick.tex
```

```
|   |-- 05_Inhalt.tex
|   |-- 06_Anhang.tex
|   |-- 07_Abkuerzungsverzeichnis.tex
|   |-- 08_Abbildungsverzeichnis.tex
|   |-- 09_Tabellenverzeichnis.tex
|   |-- 10_Listings.tex
|   |-- 11_Literaturverzeichnis.tex
|   |-- 12_EidesstattlicheErklaerung.tex
|   |-- Grafiken
|   |   |-- 01_01_Ueberblick.pdf
|   |   |-- 01_01_Ueberblick.svg
|   |   |-- 02_01_VerEntschluesselung.pdf
|   |   |-- 02_01_VerEntschluesselung.svg
|   |   |-- 02_02_MITM.pdf
|   |   |-- 02_02_MITM.svg
|   |   |-- 02_03_zero_knowledge.pdf
|   |   |-- 02_03_zero_knowledge.svg
|   |   |-- 02_04_MD5.pdf
|   |   |-- 02_04_MD5.svg
|   |   |-- 02_05_SHA-2.pdf
|   |   |-- 02_05_SHA-2.svg
|   |   |-- 02_06_Chiffre.pdf
|   |   |-- 02_06_Chiffre.svg
|   |   |-- 02_07_ecb1.pdf
|   |   |-- 02_07_ecb1.svg
|   |   |-- 02_07_ecb2.pdf
|   |   |-- 02_07_ecb2.svg
|   |   |-- 02_08_cbc1.pdf
|   |   |-- 02_08_cbc1.svg
|   |   |-- 02_08_cbc2.pdf
|   |   |-- 02_08_cbc2.svg
|   |   |-- 02_09_cfb1.pdf
|   |   |-- 02_09_cfb1.svg
|   |   |-- 02_09_cfb2.pdf
|   |   |-- 02_09_cfb2.svg
|   |   |-- 02_10_ofb1.pdf
|   |   |-- 02_10_ofb1.svg
|   |   |-- 02_10_ofb2.pdf
|   |   |-- 02_10_ofb2.svg
|   |   |-- 02_11_ppk.pdf
|   |   |-- 02_11_ppk.svg
|   |   |-- 02_12_hmac.pdf
|   |   |-- 02_12_hmac.svg
|   |   |-- HTW_Logo.pdf
|   |   |-- HTW_Logo.svg
|   |   `-- svg2pdf.sh
|   |-- LaTeX
```

```
|   |   `-- scrartcl_psy-xoryves.cls
|   |-- Literaturverzeichnis.bib
|   `-- setzen.sh
|-- Literatur
|   |-- 1780_27c3_console_hacking_2010.pdf
|   |-- 802.1X-2004.pdf
|   |-- 9783642041006-c1.pdf
|   |-- Anonymous Hacker - Xbox 360 Hypervisor Privilege Escalation
|        Vulnerability.pdf
|   |-- DemonicMember - Mathieulh breaks the PS3 security chain of trust
|        - Metldr Pwn Goodness.pdf
|   |-- EurAsiaWiki - PS3 Glitch Hack.pdf
|   |-- FIPS-198-1_final.pdf
|   |-- GCK05-acsac.pdf
|   |-- LinuxSupport_LinuxHardening.pdf
|   |-- M1k1 - Security in different consoles xbox360 wii psp ect.pdf
|   |-- Master Thesis TIA - Grundlagen und Techniken - WikiDe.pdf
|   |-- Master Thesis TIA - Grundlagen und Techniken - WikiEn.pdf
|   |-- PS3 Dev Wiki - Boot Order.pdf
|   |-- PS3 Glitch Hack EurAsiaWiki.pdf
|   |-- Richard Leadbetter - Hackers leave PS3 security in tatters.pdf
|   |-- RichtigReferenzieren.pdf
|   |-- Salted Password Hashing - Doing it Right.pdf
|   |-- Team Tweezer - Console Hacking 2008 Wii Fail - Is implementation
|        the enemy of design.pdf
|   |-- The potential of Trusted Computing for Strengthening Security in
|        Massivley Multiplayer Online Games Master Thesis.pdf
|   |-- Trusted_Boot_Loader.pdf
|   |-- Wincent - A farewell to self-checksumming.pdf
|   |-- coldboot.pdf
|   |-- coreboot.pdf
|   |-- fips-180-4.pdf
|   |-- fips180-2.pdf
|   |-- fips_186-3.pdf
|   |-- free60 - Reset Glitch Hack.pdf
|   |-- free60 - SMC Hack.pdf
|   |-- grugq und scut - Armouring the ELF: Binary encryption on the UNIX
|        platform.pdf
|   |-- l-lockdown1-pdf.pdf
|   |-- l-lockdown2-pdf.pdf
|   |-- lecture23-keys.pdf
|   |-- mcms2003.pdf
|   |-- rfc1321.pdf
|   |-- rfc2246.pdf
|   |-- rfc2440.pdf
|   |-- rfc4251.pdf
|   |-- state-of-the-hack.pdf
```

```
|   |-- the tpm and security in online games - take1.pdf
|   |-- wii_sec.pdf
|   `-- wiki.archlinux - BluRay.pdf
|-- Literatur.7z
|-- Literatur.tar.bz2
|-- Thesis_10pt.pdf
|-- Thesis_12pt.pdf
|-- src
|   |-- idd
|   |   |-- Beispiele
|   |   |   |-- a.out
|   |   |   |-- idd_arrays
|   |   |   `-- xor.c
|   |   |-- generate_c_arrays.rb
|   |   |-- generate_c_methods.rb
|   |   |-- idd.rb
|   |   |-- idd_arrays
|   |   |-- idd_methods
|   |   |-- libruby-static.a.hexdump
|   |   |-- libruby-static.a.hexdump.diff
|   |   |-- libruby-static.a.hexdump.orig
|   |   |-- libruby-static.a.objdump
|   |   |-- libruby-static.a.objdump.diff
|   |   |-- libruby-static.a.objdump.orig
|   |   |-- libruby.strings
|   |   |-- libruby.strings.diff
|   |   |-- libruby.strings.orig
|   |   |-- root_cert.pem
|   |   |-- ruby
|   |   |   |-- bin
|   |   |   |-- include
|   |   |   |-- lib64
|   |   |   `-- share
|   |   |-- ruby-1.9.3-p194
|   |   |   |-- idd_arrays
|   |   |   |-- idd_code
|   |   |   |-- idd_init
|   |   |   |-- idd_methods
|   |   |   |-- string.c
|   |   |   `-- win32
|   |   `-- ruby-1.9.3-p194.tar.bz2
|   `-- ids
|       |-- Beispiele
|       |   |-- cert.pem
|       |   |-- key.pem
|       |   |-- pkey.pem
|       |   |-- root_cert.pem
```

```
|       |   |-- root_key.pem
|       |   |-- root_pkey.pem
|       |   |-- ssl.dump
|       |   |-- ssl.dump.txt
|       |   |-- ssl_client.rb
|       |   |-- ssl_server.rb
|       |   |-- tcp.dump
|       |   |-- tcp.dump.txt
|       |   |-- tcp_client.rb
|       |   `-- tcp_server.rb
|       |-- generate_certificate.rb
|       |-- ids_basic.rb
|       `-- ids_core.rb
`-- test
    |-- Farben.pdf
    |-- Farben.pdf_tex
    |-- Farben.svg
    |-- Master_Thesis-Viktor_Dillmann-2012_07_27.pdf
    |-- Master_Thesis-Viktor_Dillmann-2012_08_05.pdf
    |-- Master_Thesis-Viktor_Dillmann-2012_08_20.pdf
    |-- Master_Thesis-Viktor_Dillmann-2012_08_29.pdf
    |-- farben1.gpa
    |-- farben2.gpa
    |-- print_dir.rb
    |-- verzeichnisstruktur.txt
    `-- verzeichnisstruktur_.txt
```

7.2 Farben

- #D39250
- #CE8F4C
- #D2965A
- #D3A14C
- #AD742D
- #B37D37

- #D7AA59
- #D9A039
- #735D38
- #AC7324
- #D69E49
- #AA1516
- rgb 0.5, 0.5, 0.5
- #2E2C2F

7.3 Kleiner Exkurs in die Welt von C

Die Programmiersprache C zählt zu den wichtigsten Sprachen. Eine ebene Tiefer ist Assembler und eine höher C++. Es gibt Leute die C lieben und welche die C versuchen zu meiden. In eingebetteten Systemen, Hardware nahen oder Zeitkritischen Anwendungen gibt es allerdings meistens keinen Weg an C vorbei. C kann sehr tückisch sein was die Fehlersuche betrifft. In diesem kleinen oder winzigen Exkurs wird so ein „Fehler“gezeigt.

```
1  > rvm --version
2  rvm 1.15.1 (latest) by Wayne E. Seguin
       <wayneeseguin@gmail.com>, Michal Papis <mpapis@gmail.com>
       [https://rvm.io/]
3  > ruby --version
4  ruby 1.9.3p194 (2012-04-20 revision 35410) [x86_64-linux]
5  > hg clone http://hg.subforge.org/subtle
6  # der folgende Aufruf konnte natürlich erst erfolgen, nachdem
       der Fehler gefunden und behoben wurde
7  > subtle --version
8  subtle 0.11.3231 - Copyright (c) 2005-2012 Christoph Kappel
9  > gcc --version
10 gcc (SUSE Linux) 4.5.1 20101208 [gcc-4_5-branch revision 167585]
11
12 > cd subtle
13 > rake
14 CC build/subtle/shared.o
```

```
15 src/shared/shared.c: In function 'subSharedRegexNew':
16 src/shared/shared.c:91:7: error: call to function
     'onig_error_code_to_str' without a real prototype
     [-Werror=unprototyped-calls]
17 /usr/include/ruby-1.9.1/ruby/oniguruma.h:692:5: note:
     'onig_error_code_to_str' was declared here
18 cc1: all warnings being treated as errors
19 > grep onig_error_code_to_str
     /usr/include/ruby-1.9.1/ruby/oniguruma.h
20 int onig_error_code_to_str PV_((OnigUChar* s, int err_code,
     ...));
21 # Alles da, die Methode hat einen Prototypen. Nur was bedeutet
     das PV_? Ganz ehrlich, keine Ahnung. Auch Google konnte nicht
     weiter helfen.
22 > rake --trace
23 # größten Teil der Ausgabe weggelassen
24 gcc -o build/subtle/shared.o -c -Wall -Werror -Wpointer-arith
     -Wstrict-prototypes -Wunused -Wshadow -std=gnu99 -DNDEBUG
     -fPIC -DSUBTLE -I. -Ibuild -Isrc -Isrc/shared -Isrc/subtle
     -idirafter/usr/include/ruby-1.9.1
     -idirafter/usr/include/ruby-1.9.1/x86_64-linux
     -I/usr/include/freetype2 src/shared/shared.c
25 # jemand der sich mit C und GCC gut auskennt weiß, dass mit -E
     der Compiler nach der PreprocessingStage anhält
26 > gcc -E -o build/subtle/shared.o -c -Wall -Werror
     -Wpointer-arith -Wstrict-prototypes -Wunused -Wshadow
     -std=gnu99 -DNDEBUG -fPIC -DSUBTLE -I. -Ibuild -Isrc
     -Isrc/shared -Isrc/subtle -idirafter/usr/include/ruby-1.9.1
     -idirafter/usr/include/ruby-1.9.1/x86_64-linux
     -I/usr/include/freetype2 src/shared/shared.c
27 > grep onig_error build/subtle/shared.o
28 int onig_error_code_to_str ();
29 onig_error_code_to_str((OnigUChar*)ebuf, ecode, &einfo);
30 # Seltsam, aus dem Prototypen ist eine Deklaration geworden und
     deshalb die berechtigte Fehlermeldung. Aber, warum?
31 > grep -C1 PV_ /usr/include/ruby-1.9.1/ruby/oniguruma.h
32 #ifndef PV_
33 #ifdef HAVE_STDARG_PROTOTYPES
34 # define PV_(args) args
35 #else
36 # define PV_(args) ()
37 #endif
38 # nun macht das PV_ (evtl.) Sinn, ee nachdem ob
     HAVE_STDARG_PROTOTYPES gesetzt ist oder nicht werden die
     Argumente weggelassen
39 > grep -C2 HAVE_STDARG_PROTOTYPES
     /usr/include/ruby-1.9.1/ruby/oniguruma.h
```

```
40 #ifdef HAVE_STDARG_H
41 # ifndef  HAVE_STDARG_PROTOTYPES
42 #  define HAVE_STDARG_PROTOTYPES 1
43 # endif
44 #endif
45 # damit also HAVE_STDARG_PROTOTYPES gesetzt wird, muss zuvor
     HAVE_STDARG_H gesetzt sein
46 > grep -R stdarg.h .
47 ./src/shared/shared.h:#include <stdarg.h>
48 # stdarg.h wird eingebunden
49 > rpm -ql gcc46 | grep stdarg
50 /usr/lib64/gcc/x86_64-suse-linux/4.6/include/stdarg.h
51 > grep HAVE_STDARG_H
     /usr/lib64/gcc/x86_64-suse-linux/4.6/include/stdarg.h
52 # nichts...
53 > grep -R HAVE_STDARG_H .
54 ./config.h:#define HAVE_STDARG_H 1
55 # interessant, in einer von rake erstellten Datei befindet sich
     die benötigte Definition
56 > grep -R -C1 config.h .
57 ./src/shared/shared.h-#include <ruby/oniguruma.h>
58 ./src/shared/shared.h:#include "config.h"
59 # da ist auch der Fehler, die config.h muss VOR dem oniguruma.h
     eingebunden werden
```

Quelltext (49) Beispiel Fehlersuche in C

7.4 Weiterführende Begriffe

An dieser Stelle werden einige Begriffe und Verweise genannt, die zwar interessant sind, für die allerdings keine Zeit gefunden werden konnte.

- strncmp vs memcmp
- heap/stack/buffer overflow (gcc -fstack-protector)
- Direct memory access (DMA)
  http://rdist.root.org/2007/09/28/protecting-memory-from-dma/
- Address space layout randomization (ASLR)
  http://tuxwave.net/coding/aslr.txt
  http://xorl.wordpress.com/2011/01/16/linux-kernel-aslr-implementation/

- Data Execution Prevention (DEP)
- http://en.wikipedia.org/wiki/NX_bit
- http://www.linux-web.de/thread/5332/nx-stack-und-heap-unausf%C3%BChrbar.html
- Seconding Hex-rays, Boomerang, interactive disassembler

7.5 Ruby Interpreter als lokale Installation

Der Ruby Interpreter kann nicht aus dem `src` Verzeichnis heraus gestartet werden. Er muss zwingend installiert werden. Dabei ist es irrelevant in welches Verzeichnis der Interpreter installiert wird.

```
1 ./configure --prefix=~/HTWdS/PI-M/Thesis/tia/src/idd/ruby
2 make
3 make install
```

Quelltext (50) Ruby Interpreter bauen

Abkürzungsverzeichnis

AACS ..... Advanced Access Content System

AES ..... Advanced Encryption Standard

API ..... Application Programming Interface

BIOS ..... BasicInputOutputSystem

Bit ..... Eine binär Zahl aus {0,1}

BT ..... Block Transfer

Byte ..... Eine Gruppe von acht Bit.

CIM ..... Common Information Management

CMAC ..... Block Cipher Message Authentication Code

CSPRNG ..... Cryptographically secure pseudorandom number generator

CSS ..... Content Scramble System

DLO ..... Dynamically Loadable Object

DMI ..... Desktop Management Interface

DRM ..... Digital Rights Managemet

DSA ..... Digital Signature Algorithm

ELF ..... Executeable and Linking Format

FDE ..... Full-Disk-Encryption

FIPS ..... Federal Information Processing Standard

HMAC ..... Keyed Hash Message Authentication Code

HTWdS ..... Hochshcule für Technik und Wirtschaft des Saarlandes

IDA ..... Interactive Disassembler

IP ..... Internet Protocol

IPC ..... Interprocesscommunication

IV ..... Initialisierungsvektor

KCS ..... Keyboard Controller Style

KSM ..... Kernel signed modules

MAC ..... Message Authentication Code

MBR ..... Master Boot Record

MC ..... Memory Controller

NIST ..... National Institute of Standards and Technology

OEM ..... Original Equipment Manufacturer

OS ..... Operating System

PBA ..... Pre-Boot Authentication

PCI ..... Peripheral Component Interconnect

PI-M ..... Praktische Informatik Master

RDBMS ..... Relational Database Management System

RGH ..... Reset Glitch Hack

RPM ..... RPM Package Manager

RSA ..... Rivest, Shamir und Adleman Kryptosystem

SCE ..... Sony Computer Entertainment

SCSI ..... Small Computer System Interface

SHA ..... Secure Hash Algorithm

SM ..... System Management

SMC ..... System Management Controller

SMS ..... Short Message Service

SMS ..... System Management Software

SOL ..... Serial over LAN

SSH ..... Secure Shell

TCP ..... Transmission Control Protocol

UHID ..... Unique Hardware Identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Literaturverzeichnis

[1] Ahmed Al-Jaber und Khair Eddin Sabri. *Data Hiding In A Binary Image*. Techn. Ber. http://www.cas.mcmaster.ca/~sabrike/wp-content/uploads/2008/02/mcms2003.pdf. Department of Computer Science - King Abdullah II School for Information Technology (KASIT) - University of Jordan,

[2] bgeddy und linuxquestions.org. *How to define hex character array in C?* [Online; Stand 29. August 2012]. Aug. 2008. URL: http://www.linuxquestions.org/questions/programming-9/how-to-define-hex-character-array-in-c-668213/#post3273729.

[3] blade.nagaokaut.ac.jp und Brian Candler. *Re: TCP Socket read and write*. [Online; Stand 21. August 2012]. Sep. 2004. URL: http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/113171.

[4] Pierre Bourdon. *Nintendo Wii Security Model*. [Online; Stand 23. Juli 2012]. 2011. URL: http://lse.epita.fr/data/2011-lse-summer-week/wii_sec.pdf.

[5] Pierre Bourdon. *State of the Hack*. [Online; Stand 24. Juli 2012]. 2012. URL: http://delroth.net/state-of-the-hack.pdf.

[6] Professor Dr. Johannes Buchmann. *Einführung in die Kryptographie*. Springer-Lehrbuch. Springer, 2001. ISBN: 9783540412830. URL: http://books.google.de/books?id=X6mKAAAACAAJ.

[7] J. Callas u. a. *OpenPGP Message Format (IETF RFC 2440)*. http://www.ietf.org/rfc/rfc2440.txt. Network Associates, IN-Root-CA Individual Network e.V., EIS Corporation. Nov. 1998.

[8] Ruby community. *Ruby-Doc – Help and documentation for the Ruby programming language.* [Online; Stand 21. August 2012]. URL: http://www.ruby-doc.org/.

[9] Ruby community. *Ruby-Doc – OpenSSL*. [Online; Stand 21. August 2012]. URL: http://www.ruby-doc.org/stdlib-1.9.3/libdoc/openssl/rdoc/OpenSSL.html.

[10] Ruby community. *Ruby-Doc – OpenSSL::X509::Certificate*. [Online; Stand 21. August 2012]. URL: http://www.ruby-doc.org/stdlib-1.9.3/libdoc/openssl/rdoc/OpenSSL/X509/Certificate.html.

[11] Ruby community. *Ruby-Doc – SecureRandom*. [Online; Stand 21. August 2012]. URL: http://ruby-doc.org/stdlib-1.9.3/libdoc/securerandom/rdoc/SecureRandom.html.

[12] Ruby community. *Ruby-Doc – TCPServer*. [Online; Stand 21. August 2012]. URL: http://www.ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/TCPServer.html.

[13] Ruby community. *Ruby-Doc – TCPSocket*. [Online; Stand 21. August 2012]. URL: http://www.ruby-doc.org/stdlib-1.9.3/libdoc/socket/rdoc/TCPSocket.html.

[14] Ruby Community und Yukihiro „ matz“ Matsumoto. *Ruby – A Programmer's Best Friend*. [Online; Stand 28. August 2012]. 1995. URL: http://www.ruby-lang.org.

[15] John Croisant. *Ruby C Extension Cheat Sheet*. [Online; Stand 29. August 2012]. URL: http://blog.jacius.info/ruby-c-extension-cheat-sheet/.

[16] Defuse Cyber-Security. *Salted Password Hashing - Doing it Right*. [Online; Stand 16. August 2012]. URL: http://crackstation.net/hashing-security.htm.

[17] H. Delfs und H. Knebl. *Introduction to Cryptography: Principles and Applications*. Information Security and Cryptography. Springer, 2007. ISBN: 9783540492436. URL: http://books.google.de/books?id=Nnvhz_VqAS4C.

[18] DemonicMember. *Mathieulh breaks the PS3 security chain of trust – Metldr Pwn Goodness!* [Online; Stand 23. Juli 2012]. 2011. URL: http://gamingghouls.com/2011/11/09/mathieulh-breaks-the-ps3-security-chain-of-trust-metldr-pwn-goodness/.

[19] devshed.com. *Best approach to hide a private key inside an application*. [Online; Stand 12. August 2012]. März 2010. URL: http://forums.devshed.com/security-and-cryptography-17/best-approach-to-hide-a-private-key-inside-an-application-683627.html.

[20] T. Dierks und C. Allen. *The TLS Protocol Version 1.0 (IETF RFC 2246)*. http://www.ietf.org/rfc/rfc2246.txt. Certicom. Jan. 1999.

[21] Adam J. Elbirt. *Understanding and Applying Cryptography and Data Security*. CRC Press, 2009. ISBN: 9781420061604. URL: http://books.google.de/books?id=ZGYvPQAACAAJ.

[22] Jon Erickson. *Hacking: The Art of Exploitation*. 2nd. No Starch Press Series. United States of America: No Starch Press, 2008. ISBN: 9781593271442. URL: http://books.google.de/books?id=0FW3DMNhl1EC.

[23] Wolfgang Ertel. *Angewandte Kryptographie*. Hanser, 2007. ISBN: 9783446411951. URL: http://books.google.de/books?id=hyCqjODGntUC.

[24] EurAsiaWiki. *PS3 Glitch Hack*. [Online; Stand 25. Juli 2012]. 2010. URL: http://www.eurasia.nu/wiki/index.php?title=PS3_Glitch_Hack&oldid=955.

[25] fail0verflow. *PS3 Epic Fail*. [Online; Stand 23. Juli 2012]. 2010. URL: http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf.

[26] free60. *Reset Glitch Hack*. [Online; Stand 25. Juli 2012]. 2012. URL: http://www.free60.org/index.php?title=Reset_Glitch_Hack&oldid=2488.

[27] free60. *SMC Hack*. [Online; Stand 25. Juli 2012]. 2012. URL: http://www.free60.org/index.php?title=SMC_Hack&oldid=2305.

[28] K. Freiermuth u. a. *Einführung in die Kryptologie: Lehrbuch für Unterricht und Selbststudium*. Vieweg+Teubner Verlag, 2010. ISBN: 9783834810052. URL: http://books.google.de/books?id=zz9ZBtDsriwC.

[29] J. Geier und J.T. Geier. *Implementing 802.1X Security Solutions for Wired and Wireless Networks*. Wiley, 2008. ISBN: 9780470168608. URL: http://books.google.de/books?id=p9TcjHo4GwYC.

[30] Jonathon T. Giffin, Mihai Christodorescu und Louis Kruger. *Strengthening Software Self-Checksumming via Self-Modifying Code*∗. Techn. Ber. http://pages.cs.wisc.edu/~giffin/papers/acsac05/GCK05-acsac.pdf. Computer Sciences Department - University of Wisconsin,

[31] eqqon GmbH. *Ruby C Extension API Documentation (Ruby 1.8)*. [Online; Stand 29. August 2012]. URL: http://www.eqqon.com/index.php/Ruby_C_Extension_API_Documentation_%28Ruby_1.8%29#Method.2Fsingleton_method_definition.

[32] eqqon GmbH. *Ruby V1.9 C Extension*. [Online; Stand 29. August 2012]. URL: http://www.eqqon.com/index.php/Ruby/Ruby_V1.9_C_Extension.

[33] grugq und scut. „Armouring the ELF: Binary encryption on the UNIX platform“. In: *Phrack Magazine* 10.58 (Dez. 2001). http://www.phrack.com/issues.html?issue=58&id=5.

[34] Anonymous Hacker. *Xbox 360 Hypervisor Privilege Escalation Vulnerability*. [Online; Stand 24. Juli 2012]. 2007. URL: http://www.securityfocus.com/archive/1/461489.

[35] J. Alex Halderman u. a. „Lest We Remember: Cold Boot Attacks on Encryption Keys“. In: *Proc. 17th USENIX Security Symposium (Sec '08)* (Juli 2008). http://citp.princeton.edu/pub/coldboot.pdf.

[36] A. Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press Series. No Starch Press, 2003. ISBN: 9781593270292. URL: http://books.google.de/books?id=FdPNE6beKcMC.

[37] Shailesh N. Humbad. *Binary Logic Bit Operations In C and C++*. [Online; Stand 29. August 2012]. Jan. 2003. URL: http://www.somacon.com/p125.php.

[38] IEEE. *802.1X, IEEE Standard for Local and metropolitan area networks, Port-Based Network Access Control*. Institute of Electrical und Electronics Engineers, Inc. Dez. 2004", note = "http://standards.ieee.org/getieee802/download/802.1X-2004.pdf".

[39] Wikipedia Kockmeyer, Wikipedia Slashme und Wikipedia Tobyvoss. *A schematic that shows the SHA-2 algorithm — Wikipedia, The Free Encyclopedia*. [Online; Stand 8. Juli 2012]. 2008. URL: http://en.wikipedia.org/wiki/File:SHA-2.svg.

[40] Greg Kroah-Hartman. „Signed Kernel Modules“. In: *Linux Journal* 117 (Jan. 2004). http://www.linuxjournal.com/article/7130.

[41] Chris Lalancette. *Writing Ruby Extensions in C - Part 4, Types and Return Values*. [Online; Stand 29. August 2012]. Jan. 2011. URL: http://clalance.blogspot.de/2011/01/writing-ruby-extensions-in-c-part-4.html.

[42] Richard Leadbetter. *Hackers leave PS3 security in tatters*. [Online; Stand 23. Juli 2012]. 2011. URL: http://www.eurogamer.net/articles/digitalfoundry-ps3-security-in-tatters.

[43] M1k1. *Security in different consoles xbox360/wii/psp ect*. [Online; Stand 23. Juli 2012]. 2011. URL: http://www.se7ensins.com/forums/threads/security-in-different-consoles-xbox360-wii-psp-ect.453777/.

[44] MongoDB und 10gen. *MongoDB (from „humongous“) is a scalable, high-performance, open source NoSQL database.* [Online; Stand 28. August 2012]. 2009. URL: http://www.mongodb.org/.

[45] MongoDB und 10gen. *MongoDB Ruby Driver Tutorial*. [Online; Stand 28. August 2012]. Aug. 2012. URL: http://api.mongodb.org/ruby/current/file.TUTORIAL.html.

[46] T. Müller und T. Caspers. *Trusted Computing Systeme: Konzepte Und Anforderungen*. Springer, 2008. ISBN: 9783540764090. URL: http://books.google.de/books?id=PPj3BshL61cC.

[47] NIST. *Digital Signature Standard (DSS) FIPS PUB 186-3*. http://csrc.nist.gov/publications/drafts/fips186-3/fips_186-3.pdf. National Institute of Standards und Technology. Juni 2009.

[48] NIST. *Secure Hash Standard (SHS) FIPS PUP 180-4*. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. National Institute of Standards und Technology. März 2012.

[49] NIST. *The Keyed-Hash Message Authentication Code (HMAC) FIPS PUB 198-1*. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf. National Institute of Standards und Technology. Juli 2008.

[50] Christof Paar und Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer, 2010. ISBN: 9783642041006. URL: http://www.springer.com/978-3-642-04100-6.

[51] Ronald L. Rivest. *The MD5 Message-Digest Algorithm (IETF RFC 1321)*. http://www.ietf.org/rfc/rfc1321.txt. Massachusetts Institute of Technology. Apr. 1992.

[52] Wim Vander Schelden. *Writing your very own Ruby extension with C*. [Online; Stand 29. August 2012]. Jan. 2007. URL: http://fixnum.org/blog/2007/ruby_c/.

[53] Adi Shamir, Nicko Van Someren und Daniel Komaromy. *Playing hide and seek with stored keys*. Techn. Ber. http://www.cc.gatech.edu/~traynor/f08/slides/lecture23-keys.pdf. College of Computing - Georgia Institute of Technology,

[54] Øyvind Skaar. „The potential of Trusted Computing for Strengthening Security in Massively Multiplayer Online Games“. Master Thesis. Norway: University of Oslo, Department of Informatics, Mai 2010. URL: http://folk.uio.no/oyvs/thesis/thesis-final.pdf.

[55] stackoverflow. *How can I obfuscate a string into a C++ binary?* [Online; Stand 12. August 2012]. Nov. 2011. URL: http://stackoverflow.com/questions/8281427/how-can-i-obfuscate-a-string-into-a-c-binary.

[56] stackoverflow. *How to hide a string in binary code?* [Online; Stand 12. August 2012]. Aug. 2009. URL: http://stackoverflow.com/questions/1356896/how-to-hide-a-string-in-binary-code.

[57] stackoverflow. *How to hide strings in a exe or a dll?* [Online; Stand 12. August 2012]. Mai 2009. URL: http://stackoverflow.com/questions/926172/how-to-hide-strings-in-a-exe-or-a-dll.

[58] stackoverflow. *How to store a secret API key in an application's binary?* [Online; Stand 12. August 2012]. Apr. 2011. URL: http://stackoverflow.com/questions/5525305/how-to-store-a-secret-api-key-in-an-applications-binary.

[59] stackoverflow. *Linux: compute a single hash for a given folder contents?* [Online; Stand 29. August 2012]. Feb. 2009. URL: http://stackoverflow.com/questions/545387/linux-compute-a-single-hash-for-a-given-folder-contents.

[60] stackoverflow. *Trying to create a simple Ruby server over SSL*. [Online; Stand 21. August 2012]. Mai 2011. URL: http://stackoverflow.com/questions/5872843/trying-to-create-a-simple-ruby-server-over-ssl.

[61] Wikipedia Surachit. *One MD5 operation SVG — Wikipedia, The Free Encyclopedia*. [Online; Stand 6. Juli 2012]. 2007. URL: http://en.wikipedia.org/wiki/File:MD5.svg.

[62] J. Swoboda, S. Spitz und M. Pramateftakis. *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen- eine Einführung*. Vieweg+Teubner Verlag, 2008. ISBN: 9783834802484. URL: http://books.google.de/books?id=aQMgp7Wq9K0C.

[63] coreboot Team. *coreboot*. [Online; Stand 20. Juli 2012]. 2012. URL: http://www.coreboot.org/Welcome_to_coreboot.

[64] David Thomas und Andrew Hunt. *Programming Ruby - The Pragmatic Programmer's Guide: Extending Ruby*. [Online; Stand 29. August 2012]. URL: http://ruby-doc.org/docs/ProgrammingRuby/html/ext_ruby.html.

[65] Team Tweezer. *Console Hacking 2008: Wii Fail. Is implementation the enemy of design?* [Online; Stand 23. Juli 2012]. 2008. URL: http://marcansoft.com/uploads/25c3_console_hacking/.

[66] Gary V. Vaughan. *Industrial-strength Linux lockdown, Part 1: Removing the shell*. [Online; Stand 05. August 2012]. Mai 2007. URL: http://www.ibm.com/developerworks/linux/tutorials/l-lockdown1/l-lockdown1-pdf.pdf.

[67] Gary V. Vaughan. *Industrial-strength Linux lockdown, Part 2: Executing only signed binaries*. [Online; Stand 05. August 2012]. Juli 2007. URL: http://www.ibm.com/developerworks/linux/tutorials/l-lockdown2/l-lockdown2-pdf.pdf.

[68] Prof. Dr.-Ing. Damian Weber. *Sicherheit und Kryptographie*. Vorlesung. PI-M HTWdS. Okt. 2011.

[69] PS3 Dev Wiki. *Boot Order*. [Online; Stand 23. Juli 2012]. 2012. URL: http://www.ps3devwiki.com/index.php?title=Boot_Order&oldid=12616.

[70] wiki.archlinux. *BluRay*. [Online; Stand 26. Juli 2012]. 2012. URL: https://wiki.archlinux.org/index.php?title=BluRay&oldid=208623.

[71] Wikipedia. *Asymmetrisches Kryptosystem — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 9. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Asymmetrisches_Kryptosystem&oldid=104296427.

[72] Wikipedia. *Block cipher modes of operation — Wikipedia, The Free Encyclopedia*. [Online; Stand 9. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Block_cipher_modes_of_operation&oldid=498325862.

[73] Wikipedia. *Blockverschlüsselung — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 9. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Blockverschl%C3%BCsselung&oldid=104241324.

[74] Wikipedia. *Chain of trust — Wikipedia, The Free Encyclopedia*. [Online; Stand 19. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Chain_of_trust&oldid=485749797.

[75] Wikipedia. *Coreboot — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 20. Juli 2012]. 2011. URL: http://de.wikipedia.org/w/index.php?title=Coreboot&oldid=94589224.

[76] Wikipedia. *Cryptographically secure pseudorandom number generator — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. August 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Cryptographically_secure_pseudorandom_number_generator&oldid=499495179.

[77] Wikipedia. */dev/random — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. August 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=/dev/random&oldid=507093115.

[78] Wikipedia. *Diffie-Hellman-Schlüsselaustausch — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 4. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Diffie-Hellman-Schl%C3%BCsselaustausch&oldid=104323317.

[79] Wikipedia. *Diffie–Hellman key exchange — Wikipedia, The Free Encyclopedia*. [Online; Stand 4. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Diffie%E2%80%93Hellman_key_exchange&oldid=497293835.

[80] Wikipedia. *Digital signature — Wikipedia, The Free Encyclopedia*. [Online; Stand 11. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Digital_signature&oldid=500694481.

[81] Wikipedia. *Digitale Signatur — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 11. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Digitale_Signatur&oldid=103832492.

[82] Wikipedia. *Disk encryption — Wikipedia, The Free Encyclopedia*. [Online; Stand 19. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Disk_encryption&oldid=498173237.

[83] Wikipedia. *Entropy (computing) — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. August 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Entropy_(computing)&oldid=505270593.

[84] Wikipedia. *Festplattenverschlüsselung — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 19. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Festplattenverschl%C3%BCsselung&oldid=104425712.

[85] Wikipedia. *Full Disk Encryption — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 19. Juli 2012]. 2011. URL: http://de.wikipedia.org/w/index.php?title=Full_Disk_Encryption&oldid=94965702.

[86] Wikipedia. *GNU Privacy Guard — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 22. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=GNU_Privacy_Guard&oldid=105567633.

[87] Wikipedia. *Hash function — Wikipedia, The Free Encyclopedia*. [Online; Stand 6. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Hash_function&oldid=500352152.

[88] Wikipedia. *Hashfunktion — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 6. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Hashfunktion&oldid=104774198.

[89] Wikipedia. *IEEE 802.1X — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 16. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=IEEE_802.1X&oldid=101279315.

[90] Wikipedia. *IEEE 802.1X — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=IEEE_802.1X&oldid=497060496.

[91] Wikipedia. *Kryptologische Hashfunktion — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 6. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Kryptologische_Hashfunktion&oldid=105164988.

[92] Wikipedia. *Man-in-the-middle-Angriff — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 4. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Man-in-the-middle-Angriff&oldid=103844302.

[93] Wikipedia. *Man-in-the-middle attack — Wikipedia, The Free Encyclopedia*. [Online; Stand 4. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Man-in-the-middle_attack&oldid=497733131.

[94] Wikipedia. *MD5 — Wikipedia, The Free Encyclopedia*. [Online; Stand 6. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=MD5&oldid=499582012.

[95] Wikipedia. *Message-Digest Algorithm 5 — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 6. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Message-Digest_Algorithm_5&oldid=105173079.

[96] Wikipedia. *OpenPGP — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 22. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=OpenPGP&oldid=101403863.

[97] Wikipedia. *Pre-Boot Authentication — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 19. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Pre-Boot_Authentication&oldid=102367667.

[98] Wikipedia. *Pre-boot authentication — Wikipedia, The Free Encyclopedia*. [Online; Stand 19. Juli 2012]. 2011. URL: http://en.wikipedia.org/w/index.php?title=Pre-boot_authentication&oldid=467107067.

[99] Wikipedia. *Pretty Good Privacy — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 22. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Pretty_Good_Privacy&oldid=105827018.

[100] Wikipedia. *Public-key cryptography — Wikipedia, The Free Encyclopedia*. [Online; Stand 9. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Public-key_cryptography&oldid=500929012.

[101] Wikipedia. *Salt (cryptography) — Wikipedia, The Free Encyclopedia*. [Online; Stand 3. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Salt_(cryptography)&oldid=498661445.

[102] Wikipedia. *Salt (Kryptologie) — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 3. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Salt_(Kryptologie)&oldid=104206617.

[103] Wikipedia. *Secure Hash Algorithm — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 8. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Secure_Hash_Algorithm&oldid=105185019.

[104] Wikipedia. *Secure Shell — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 15. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Secure_Shell&oldid=105519564.

[105] Wikipedia. *Secure Shell — Wikipedia, The Free Encyclopedia*. [Online; Stand 15. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Secure_Shell&oldid=499448306.

[106] Wikipedia. *SHA-2 — Wikipedia, The Free Encyclopedia*. [Online; Stand 8. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=SHA-2&oldid=497381954.

[107] Wikipedia. *Station-to-Station protocol — Wikipedia, The Free Encyclopedia*. [Online; Stand 4. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Station-to-Station_protocol&oldid=499530765.

[108] Wikipedia. *Stream cipher — Wikipedia, The Free Encyclopedia*. [Online; Stand 9. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Stream_cipher&oldid=493843900.

[109] Wikipedia. *Stromverschlüsselung — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 9. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Stromverschl%C3%BCsselung&oldid=103541860.

[110] Wikipedia. *Symmetric-key algorithm — Wikipedia, The Free Encyclopedia*. [Online; Stand 9. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=495492829.

[111] Wikipedia. *Symmetrisches Kryptosystem — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 9. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Symmetrisches_Kryptosystem&oldid=104192040.

[112] Wikipedia. *Transport Layer Security — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 11. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=105246655.

[113] Wikipedia. *Transport Layer Security — Wikipedia, The Free Encyclopedia*. [Online; Stand 11. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=500970926.

[114] Wikipedia. *Trusted Computing — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 16. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Trusted_Computing&oldid=105520059.

[115] Wikipedia. *Trusted Computing — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Trusted_Computing&oldid=501348347.

[116] Wikipedia. *Trusted Platform Module — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 16. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Trusted_Platform_Module&oldid=104351141.

[117] Wikipedia. *Trusted System — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 16. Juli 2012]. 2008. URL: http://de.wikipedia.org/w/index.php?title=Trusted_System&oldid=43597304.

[118] Wikipedia. *Trusted system — Wikipedia, The Free Encyclopedia*. [Online; Stand 16. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Trusted_system&oldid=483368986.

[119] Wikipedia. *Zero Knowledge — Wikipedia, Die freie Enzyklopädie*. [Online; Stand 4. Juli 2012]. 2012. URL: http://de.wikipedia.org/w/index.php?title=Zero_Knowledge&oldid=103888094.

[120] Wikipedia. *Zero-knowledge proof — Wikipedia, The Free Encyclopedia*. [Online; Stand 4. Juli 2012]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Zero-knowledge_proof&oldid=495432169.

[121] Wincent. *A farewell to self-checksumming*. [Online; Stand 07. August 2012]. Apr. 2006. URL: http://www.wincent.com/a/about/wincent/weblog/archives/2006/04/a_farewell_to_s.php.

[122] T. Ylonen und C. Lonvick. *The Secure Shell (SSH) Protocol Architecture (IETF RFC 4251)*. http://www.ietf.org/rfc/rfc4251.txt. SSH Communications Security Corp, Cisco Systems, Inc. Jan. 2006.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Saarbrücken, 30. August 2012
Viktor Dillmann