

# VSS Cheatsheet

## Inhaltsverzeichnis

<b>Schutzziele .....</b>	<b>2</b>
<b>Fehlertoleranz .....</b>	<b>2</b>
<b>RAID .....</b>	<b>3</b>
<b>Fehlererkennung .....</b>	<b>4</b>
Parität .....	4
Cyclic Redundancy Check (CRC).....	4
Hamming Codes .....	4
<b>Programme und Prozesse .....</b>	<b>6</b>
<b>Netzwerke .....</b>	<b>7</b>
<b>TCP / UDP .....</b>	<b>8</b>
<b>Subnetting .....</b>	<b>9</b>
<b>Dijkstra's Algorithmus .....</b>	<b>9</b>
<b>IT-Sicherheit.....</b>	<b>10</b>
<b>Rechnersicherheit .....</b>	<b>11</b>
<b>Kryptographie .....</b>	<b>12</b>
<b>Informationstheorie .....</b>	<b>14</b>

## Schutzziele

Informationssicherheit Freiheit v. Gefährd. f. d. <i>Daten</i> des IT-Systems	Technische Sicherheit Freiheit v. Gefährd. f. d. <i>Umgebung</i> des IT-Systems
<b>Vertraulichkeit</b> <ul style="list-style-type: none"> <li>• Abhörsicherheit</li> <li>• Sicherheit gegen unbefugten Gerätezugriff</li> <li>• Anonymität</li> <li>• Unbeobachtbarkeit</li> </ul> <b>Integrität</b> <ul style="list-style-type: none"> <li>• Übertragungsintegrität</li> <li>• Zurechenbarkeit</li> <li>• Abrechnungsintegrität</li> </ul> <b>Verfügbarkeit</b> <ul style="list-style-type: none"> <li>• Ermöglichen von Kommunikation</li> </ul>	<b>Verfügbarkeit</b> <ul style="list-style-type: none"> <li>• Funktionssicherheit</li> <li>• Technische Sicherheit</li> <li>• Schutz vor Überspannung, Überschwemmung, Temperaturschwankungen</li> <li>• Schutz vor Spannungsausfall</li> </ul> <div style="background-color: #e0e0e0; padding: 5px;"> <b>Zuverlässigkeit (reliability)</b> <ul style="list-style-type: none"> <li>• Korrektheit</li> </ul> </div> <b>Sonstige Schutzziele</b> <ul style="list-style-type: none"> <li>• Maßnahmen gegen hohe Gesundheitsbelastung (z.B. Strahlung)</li> <li>• ...</li> </ul>

**Verfügbarkeit:** Gegeben durch **Redundanz** und **Diversität**

## Fehlertoleranz

**MTTF** Erwartungswert für die Zeit (von der Inbetriebnahme) bis zum (ersten) Ausfall eines Systems

**MTTR** Erwartungswert für die Zeit zur Ersetzung des Systems und ggf. der Rekonstruktion der Daten

**MTBF** Erwartungswert für die Zeit zwischen zwei Ausfällen

**R(t)** (Zuverlässigkeit) Wahrscheinlichkeit, dass ein System Zeitintervall [0,t] überlebt, wenn es in t=0 funktioniert

**λ(t)** (Ausfallrate) Wahrscheinlichkeit, dass das System im Zeitintervall [t+dt] ausfällt

**μ(t)** (Reparaturrate) Wahrscheinlichkeit, dass das System im Zeitintervall [t+dt] wieder repariert wird

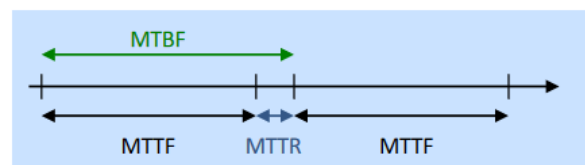
**A(t)** (Verfügbarkeit) Wahrscheinlichkeit, dass ein System zu einem vorgegebenen Zeitpunkt t bzw. in einem Zeitraum [0,t] in einem funktionsfähigen Zustand angetroffen wird

$$MTTF = \int_{t=0}^{\infty} R(t) dt = \frac{1}{\lambda}$$

$$MTBF = MTTF + MTTR$$

$$A = \frac{\mu}{\mu + \lambda} = \frac{MTTF}{MTTF + MTTR}$$

$$\lambda = \frac{1}{MTTF}, \quad \mu = \frac{1}{MTTR}$$



## RAID

keine Redundanz	RAID-0	} RAID-01, RAID-10
Datenreplikation	RAID-1	
Fehlerkorrigierende Codes	RAID-2	
Paritätsbits	RAID-3, RAID-4, RAID-5, RAID-6	

**Zuverlässigkeit:**  $MTBF(\text{disk array}) = MTBF(\text{disk}) / \text{Anzahl disks}$

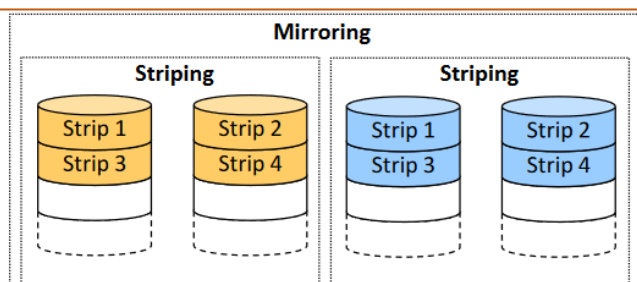
- Raid 0** Aufteilung der Daten in Blöcke, Aufteilen der Blöcke auf verschiedene Festplatten  
keine Redundanz, verbesserte Performanz
- Raid 1** Aufteilung der Daten in Blöcke, Blöcke werden mehrfach auf verschiedenen Festplatten gespeichert  
Gute Redundanz, mittlere Performanz
- Raid 2** Wie Raid 0, aber mit extra *Check Disks*, die Hamming-Codes zu den *Data Disks* enthalten  
Redundanz ok, 20%-40% Overhead, niedrige Performanz
- Raid 3** Wie Raid 0, aber mit einer extra *Check Disk*, die Parität zu den *Data Disks* enthalten  
Redundanz ok, eine Disk Overhead, leicht besser als Raid 2
- Raid 4** Effizientere Version von Raid 3, Check Disk bleibt Bottleneck
- Raid 5** Verteilung (striping) der Daten (sector interleaving) und Paritätsinformation zur Verbesserung der Performance. Paritätsinformationen werden auf alle Disks aufgeteilt.  
Gute Performanz, gute Redundanz
- Raid 6** Wie Raid-5, aber mit 2 Dimensionaler Parität

### gespiegelte Verteilung

Mirrored Strips

RAID 0+1 oder RAID-01

2 ausfallende Disks: Datenverlust mit  $p=2/3$

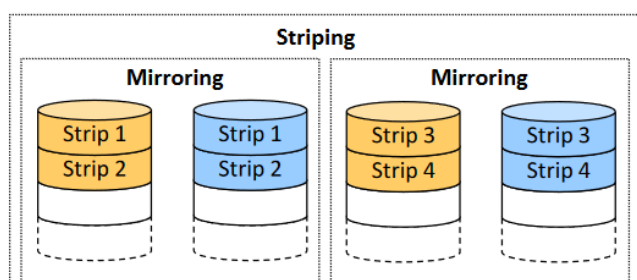


### verteilte Spiegelung

Striped Mirrors

RAID 1+0 oder RAID-10

2 ausfallende Disks: Datenverlust mit  $p=1/3$



## Fehlererkennung

### Parität

Codewörter werden um ein Bit  $P$  ergänzt.  $P$  gibt die Parität der Bits an. Die Parität einer Bitfolge gibt an, ob die Anzahl der Einsen in der Bitfolge gerade oder ungerade ist.

Anzahl Einsen gerade:  $P = 0$

Anzahl Einsen ungerade:  $P = 1$

Oder formell: Bei gegebenen Codewortbits  $x_0, \dots, x_n$

$$P = (\sum_{i=0}^n x_i) \bmod 2$$

### Cyclic Redundancy Check (CRC)

**Gegeben:** - Nachricht  $M(x)$  (Bsp.  $M(x) = 10110$ )  
 - Polynom  $G(x)$  (Bsp.  $G(x) = x^3 + x^2 + 1$ )

1. Polynom $G(x)$ in Bitfolge umwandeln	$G(x) = x^3 + x^2 + 1 \Rightarrow 1101$
2. Sei $r$ der Grad (größter Exponent) von $G(x)$ , $r$ Nullen an $M(x)$ anhängen	$r = 3$ $M(x) = 10100\ 000$
3. Polynomdivision von $M(x)$ durch $G(x)$ $\Rightarrow$ Division in Binär ist identisch zu XOR	$10100000$ $\underline{1101}$ $1110$ $\underline{1101}$ $1100$ $\underline{1101}$ $10 = \text{Divisionsrest}$
4. Divisionsrest auf Nachricht $M(x)$ addieren	$M(x) + 10 = 10100010$

Der Empfänger kann nachprüfen, ob die Nachricht korrekt übertragen wurde, indem er selbst eine CRC-Prüfung auf die Nachricht anwendet. Ist Divisionsrest = 0, wurde die Nachricht korrekt übertragen.

### Hamming Codes

Jeweils geschrieben als **(N, n)**-Hamming Code

- $n$  = Anzahl Datenbits
- $k$  = Anzahl Prüfbits
- $N = n + k$  (Codewortlänge)

Hamming Abstand  $d_{min}$ :

Wenn bezogen auf zwei Wörter: Anzahl Unterschiedlicher Zeichen zwischen zwei Wörtern

Wenn bezogen auf ganzen Code: Minimalwert aller Hamming-Abstände aller Codewörter

Es können  $d_{min} - 1$  Fehler erkannt und  $\frac{d_{min}-1}{2}$  Fehler korrigiert werden.

Paritätsbits in Hammingcodes lassen sich effizient über Generatormatrizen erstellen. Empfangene Wörter können über Prüfmatrizen auf Fehler überprüft werden.

**Bsp.: Ein (7, 4)-Hammingcode**

Aufbau eines Wortes:  $x_1x_2x_3x_4P_1P_2P_3$

$$x_1 + x_2 + x_3 + P_1 = 0$$

$$x_2 + x_3 + x_4 + P_2 = 0$$

$$x_1 + x_3 + x_4 + P_3 = 0$$

Prüfmatrix **H**:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$\underbrace{\hspace{1.5cm}}_C \quad \underbrace{\hspace{1.5cm}}_I$

Generatormatrix **G**:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$\underbrace{\hspace{1.5cm}}_I \quad \underbrace{\hspace{1.5cm}}_{C^T}$

Die H und G sind bestehen beide aus zwei Teilen, einer Identitätsmatrix **I** und den Paritätsinformationen **C**. Wichtig: C ist zwischen H und G *transponiert* (diagonal von oben links nach unten rechts gespiegelt)

Wollen wir die Paritätsbits zum Wort 1011 berechnen, gehen wir wie folgt vor:

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \cdot G^T = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \cdot \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array} \right] = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$\underbrace{\hspace{1.5cm}}_I \quad \underbrace{\hspace{1.5cm}}_C$

Ein Wort w ist gültig falls gilt:  $w \cdot H = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Falls ein Wort w nicht gültig ist, kann aus der entstandenen Matrix direkt die Fehlerposition abgelesen werden.

Wollen wir prüfen ob das Wort 1011011 gültig ist, gehen wir wie folgt vor:

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \cdot H = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \cdot \left[ \begin{array}{cccc|cccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array} \right] = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

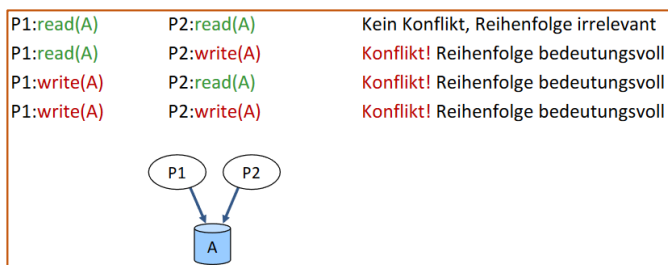
$\underbrace{\hspace{1.5cm}}_C \quad \underbrace{\hspace{1.5cm}}_I$

Zuletzt muss man die Spalte in H finden, die mit [0,1,0] übereinstimmt. Der Fehler ist hier also in Position 6. Das korrekte Wort ist demnach 1011001.

## Programme und Prozesse

<b>Programm</b>	Statische Folge von Anweisungen in einer Programmiersprache unter Nutzung von Daten. Ein Programm kann mehrere parallele Prozesse haben.
<b>Prozess</b>	Dynamische Folge von Aktionen, die durch Ausführung eines Programms auf einem Prozessor zustande kommt. Zeitliche Folge von Zustandsänderungen. Ein Prozess kann mehrere Threads haben.
<b>Thread</b>	Aktivitätsträger (sequenzieller Ausführungsfaden) mit minimalem Kontext (Stack und Register) innerhalb einer Ausführungsumgebung (Prozess).
<b>Transaktion</b>	Folge von Aktionen (Anweisungen), die ununterbrechbar ausgeführt werden soll. Begriffe: BOT - Beginn of Transaction EOT - End of Transaction / Commit ROB - Rollback, Abbruch und Zurücksetzen auf Zustand vor Transaktion

Falls mehrere Prozesse gleichzeitig auf dieselbe Ressource zugreifen wollen, kann dies zu Problemen führen.



Bspw.: Prozesse P1 und P2 wollen gleichzeitig auf Variable A zugreifen

### Sperren

#### Sperrmodus S:

Wenn eine Transaktion  $T_i$  eine S-Sperre für eine Ressource A besitzt, kann  $T_i$  A lesen. Mehrere Transaktionen können gleichzeitig eine S-Sperre für A haben. Hat A eine X-Sperre müssen Transaktionen, die eine S-Sperre für A beanspruchen wollen, warten, bis die X-Sperre freigegeben wird.

#### Sperrmodus X:

Nur eine Transaktion darf eine X-Sperre für A besitzen, alle anderen müssen auf die Freigabe warten. Transaktionen, die eine X-Sperre für A besitzen, dürfen auf A schreiben.

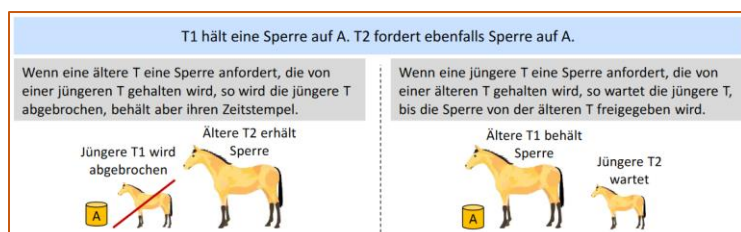
Achtung! Diese Sperren können zu einem **Deadlock** führen.

Bsp.: Transaktion  $T_1$  hat eine X-Sperre auf A,  $T_2$  hat eine X-Sperre auf B.  $T_1$  wartet auf die Freigabe von B,  $T_2$  wartet auf die Freigabe von A. Ohne Eingriff von außen sind  $T_1$  und  $T_2$  permanent gesperrt.

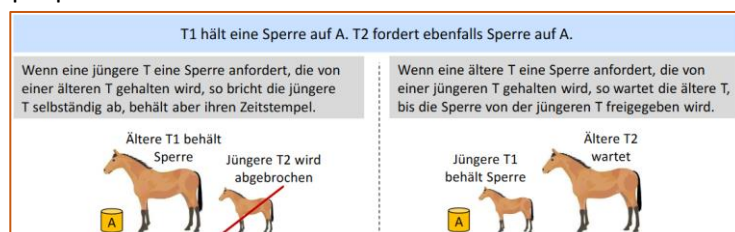
### Umgehen von Deadlocks

Methode 1: Transaktionen erst beginnen, nachdem alle für diese Transaktion erforderlichen Sperren freigegeben wurden.

Methode 2: Transaktionen über Zeitstempel priorisieren



Methode 2: „wound-wait“ Strategie



Methode 2: „wait-die“ Strategie

## Netzwerke

<b>Host</b>	Endsysteme am „Rand“ des Netzes
<b>Packet Switches</b>	Leiten Pakete zwischen Hosts weiter
<b>Communication Links</b>	Verbindungen zwischen Hosts und Packet Switches
<b>Protokoll</b>	Beschreibt erwartete Handlungen/Nachrichten zwischen Hosts. Definiert Format, Reihenfolge, vorgenommene Aktionen, ...

### Datenübertragung

1. Host teilt Nachricht in Pakete auf
2. Pakete werden über Links von Router zu Router weitergeleitet
  - ⇒ Store and Forward: Gesamtes Paket muss bei Router angekommen sein, bevor es weiterversendet wird
  - ⇒ Falls Paketankunftsrate bei Router größer ist als die Übertragungsrate:
    - Paket in Queue speichern
    - Falls Queue voll: Pakete werden gedroppt (gilt es zu vermeiden)
3. Ziel Host empfängt Pakete und fügt sie zu Nachricht zusammen

### Performance

$$\text{Paket – Übertragungsdelay} = \frac{\text{Zeit, um Daten an Link zu übergeben}}{\text{Paketlänge } L} = \frac{L}{\text{Übertragungsrate } R} = d_{trans}$$

$$\text{Paket – Übertragungsdauer} = \frac{\text{Zeit, um Daten über Link zu übertragen}}{\text{Linklänge } d} = \frac{d}{\text{Linkübertragungsrate } v} = d_{prop}$$

Zeit, die ein Paket braucht, um von einem Router/Host zum direkt nächsten übertragen zu werden:

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop} = d_{proc} + d_{queue} + \frac{L}{R} + \frac{d}{v}$$

$d_{proc}$ : Zeit, um Paket auf Bitfehler zu prüfen und Outputlink zu bestimmen

$d_{queue}$ : Zeit, die Paket in Queue verbringt

Die Übertragungsgeschwindigkeit zwischen zwei Hosts wird immer durch die langsamste Übertragungsstelle gebottleneckt.

## TCP / UDP

TCP	UDP
<ul style="list-style-type: none"> <li>- Langsam aber verlässlich</li> <li>- Jedes Paket soll ankommen</li> <li>- Abstimmung zwischen Sender/Empfänger               <ul style="list-style-type: none"> <li>o Flow Control</li> <li>o Congestion Control</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>- Schnell aber ungenau</li> <li>- Pakete gehen z.T. verloren</li> <li>- Kleiner Header</li> <li>- Keine Congestion Control</li> </ul>
Bsp. Onlinebanking	Bsp. Streaming
<b>Übertragene Daten:</b> <ul style="list-style-type: none"> <li>▪ Source Port</li> <li>▪ Dest. Port</li> <li>▪ Sequence Number</li> <li>▪ Ack. Number</li> <li>▪ Ack. Flag (Boolean)</li> <li>▪ Länge in Bytes</li> <li>▪ Flow Control</li> <li>▪ Prüfsumme</li> <li>▪ TCP Optionen</li> <li>▪ <b>Payload Data</b></li> </ul>	<b>Übertragene Daten:</b> <ul style="list-style-type: none"> <li>▪ Source Port</li> <li>▪ Dest. Port</li> <li>▪ Länge in Bytes</li> <li>▪ Prüfsumme</li> <li>▪ <b>Payload Data</b></li> </ul>

### TCP

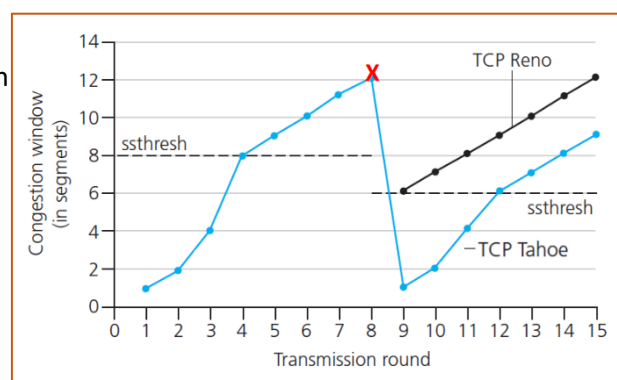
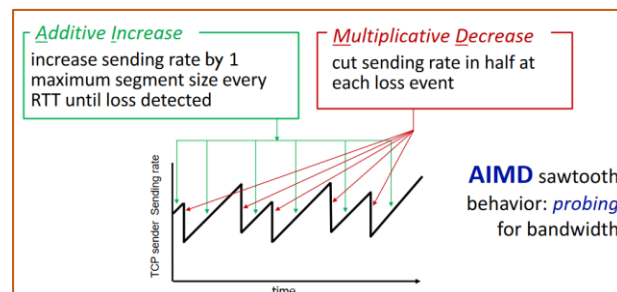
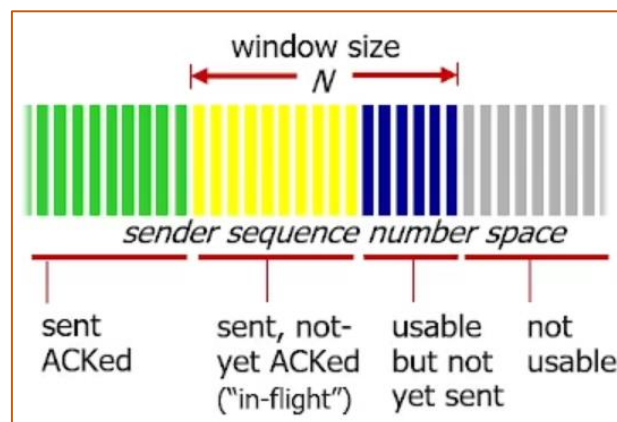
Jedes gesendete Paket hat eine **Sequence Number**. Mithilfe der *Sequence Number* weiß der Empfänger, in welcher Reihenfolge er die Pakete zusammensetzen muss.

Nachdem der Empfänger ein Paket erhält, bestätigt er dies mit einem **Acknowledgement**. Er sendet ein Paket zurück, dessen *Ack. Number* die *Sequence Number + 1* des empfangenen Paketes ist. Außerdem ist die *Ack. Flag* = 1, was ein Ack. signalisiert. Dieses Ack. wird vom Sender zurück acknowledged.

In einem Ack.-Paket kann der Empfänger über das *Flow Control* Feld angeben, wie viele unACKed Daten der Sender gleichzeitig gesendet haben darf.

Der Sender kann seine Senderate erhöhen bis Paketverlust stattfindet, danach wird seine Senderate um ein Vielfaches gedrosselt.

Bei Verbindungsaufbau oder Timeout setzt i.d.R. ein **TCP-Slow-Start** ein. Hier wird die Senderate exponentiell erhöht, bis ein Paket verloren geht.





## Subnetting

<b>IP Adresse</b>	32-bit Nummer um Host-Interface oder Router-Interface zu identifizieren
<b>Interface</b>	Verbindung zwischen Host/Router und Link
<b>Subnet</b>	Interfaces mit demselben Netzwerkteil der IP-Adresse

### IP Format

IP Adressen haben das Format **a.b.c.d/x**, wobei **x** die Anzahl Bits der Netzwerk Maske ist

$$\begin{array}{c}
 200.23.16.1/23 \\
 = \\
 \underbrace{11001000\ 00010111\ 00010000\ 0}_{\text{Netzwerk Teil}} \underbrace{00000001}_{\text{Host Teil}}
 \end{array}$$

In diesem Netzwerk lassen sich  $2^9 - 2 = 510$  Hosts unterscheiden. Die -2 kommt dadurch, da immer zwei Adressen reserviert sind, die Netzwerkadresse und die Broadcastadresse.

In der Netzwerkadresse besteht der Hostteil nur aus Nullen, In der Broadcastadresse nur aus Einsen.

<b>Netzwerkadresse</b>	$\underbrace{11001000\ 00010111\ 00010000\ 0}_{\text{Netzwerk Teil}} \underbrace{00000000}_{\text{Host Teil}}$
<b>Broadcastadresse</b>	$\underbrace{11001000\ 00010111\ 00010000\ 1}_{\text{Netzwerk Teil}} \underbrace{11111111}_{\text{Host Teil}}$

## Dijkstra's Algorithmus

Gegeben: Graph mit **Kantengewichten**  $c_{x,y}$  und **Startknoten** **u**

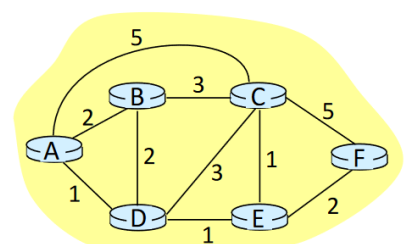
1.  $N' = \{u\}$ . Setze  $D(v_i) = c_{u,v_i}$  für alle von **u** direkt verbundenen Knoten  $v_i$ . Alle anderen  $D(v)$  sind  $\infty$ .
2. Knoten **k** ist der Knoten mit kleinstem  $D(k)$  der nicht in  $N'$  ist. Füge **k** in  $N'$  ein.
3. Update alle Distanzen zu den direkt verbundenen Knoten  $v_i$  von **k**.  $D(v_i)$  nimmt einen neuen Wert an, falls die Distanz von **k** kürzer ist als die momentane. Oder formell:  $D(v_i) = \min(D(v_i), D(k) + c_{k,v_i})$
4. Wiederhole 2. bis alle Knoten in  $N'$  sind.

### notation

- $c_{x,y}$ : direct link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- $D(v)$ : *current* estimate of cost of least-cost-path from source to destination  $v$
- $p(v)$ : predecessor node along path from source to  $v$
- $N'$ : set of nodes whose least-cost-path *definitively* known

Bsp.: Kantengewichte sind im Bild, Startknoten = A

Schritt	$N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2, A	5, A	1, A	$\infty$	$\infty$
1	AD	2, A	4, D	-	2, D	$\infty$
2	ADE	2, A	3, E	-	-	4, E
3	ADEB	-	3, E	-	-	4, E
4	ADEBC	-	-	-	-	4, E
5	ADEBCF	-	-	-	-	4, E



## IT-Sicherheit

### 3 Hauptziele:

<b>Vertraulichkeit</b>	Unbefugter Informationsgewinn
<b>Integrität</b>	Unbefugte Modifikation
<b>Verfügbarkeit</b>	Unbefugte Beeinträchtigung der Funktionalität






### Weitere Schutzziele:

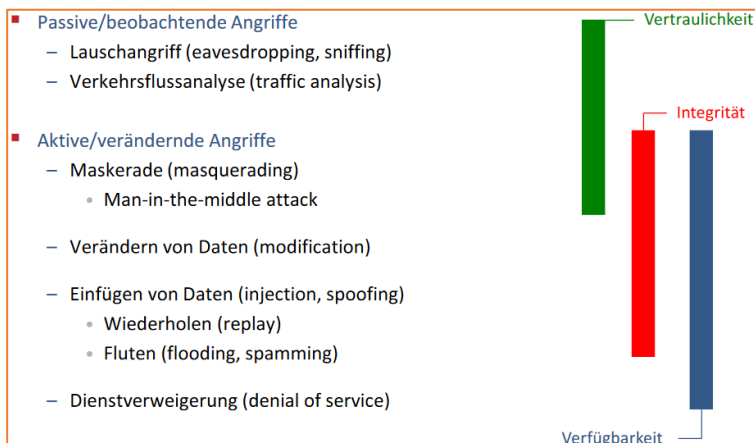
<b>Verdecktheit</b>	Versteckte Übertragung von vertraulichen Daten. Niemand außer den Kommunikationspartnern kann die Existenz eines vertraulichen Inhalts erkennen.
<b>Anonymität</b>	Nutzer können Ressourcen und Dienste benutzen, ohne ihre Identität zu offenbaren. Selbst der Kommunikationspartner erfährt nicht die Identität.
<b>Unbeobachtbarkeit</b>	Nutzer können Ressourcen und Dienste benutzen, ohne dass andere dies beobachten können. Dritte können weder das Senden noch den Erhalt von Nachrichten beobachten.
<b>Zurechenbarkeit</b>	Sendern bzw. Empfängern von Informationen kann das Senden bzw. der Empfang der Informationen bewiesen werden. Wechselwirkungen zwischen Schutzzielen
<b>Rechtsverbindlichkeit</b>	Ein Nutzer kann rechtlich belangt werden, um seine Verantwortlichkeiten innerhalb einer angemessenen Zeit zu erfüllen.
<b>Erreichbarkeit</b>	Zu einer Ressource (Nutzer oder Maschine) kann Kontakt aufgenommen werden, wenn gewünscht.

### Angreifermodell

Das Angreifermodell definiert die maximal berücksichtigte Stärke eines Angreifers, gegen den ein Schutzmechanismus gerade noch wirkt. Schutz vor einem allmächtigen Angreifer ist unmöglich.

#### Aspekte des Angreifermodells:

<b>Rollen</b>	<i>roles</i>		Außenstehender, Innentäter, auch kombiniert
<b>Verhalten</b>	<i>behavior</i>		passiv/beobachtend, aktiv/verändernd
<b>Fähigkeiten</b>	<i>abilities</i>		Wissen über das System und Verbreitung im System
<b>Absicht</b>	<i>intention</i>		versehentlich, fahrlässig, absichtlich
<b>Ressourcen</b>	<i>resources</i>		Zeit und Geld, <u>Rechenkapazität</u> unbeschränkt: informationstheoretisch beschränkt: komplexitätstheoretisch



## Rechnersicherheit

<b>Abschirmen</b>	<p>Verhindern der unbeabsichtigten Abstrahlung von Daten/Signalen, die Angreifern Einblick in das System geben. Abzuschirmende Daten könnten bspw. sein:</p> <ul style="list-style-type: none"> <li>• Elektromagnetische Abstrahlung</li> <li>• Vibration</li> <li>• Stromverbrauch</li> <li>• Tastaturgeräusche</li> </ul>
<b>Authentifizierung</b>	<p>Sichere IT-Systeme verlangen die gegenseitige Identifizierung der Menschen und der beteiligten Geräte.</p> <ul style="list-style-type: none"> <li>• Identifikation von IT-Systemen durch Menschen</li> <li>• Identifikation von IT-Systemen durch andere IT-Systeme</li> <li>• Identifikation von Menschen durch IT-Systeme <ul style="list-style-type: none"> <li>○ Biometrische Merkmale (Fingerabdruck, Retinascan)</li> <li>○ Dokumente (Schlüssel, Chipkarte)</li> <li>○ Wissen (Passwort, Sicherheitsfragen)</li> </ul> </li> </ul>
<b>Zutrittskontrolle</b>	<p>Beschreibt, wem erlaubt ist bestimmte Räumlichkeiten zu betreten. Personen ohne Zutrittskontrolle zu einem bestimmten Raum/Gebäude dürfen dieses nicht betreten.</p>
<b>Zugangskontrolle</b>	<p>Beschreibt, wem erlaubt ist IT-Systeme (i.d.R. Computer) zu benutzen. Personen ohne Zugangskontrolle zu einem IT-System dürfen dieses nicht nutzen.</p>
<b>Zugriffskontrolle</b>	<p>Beschreibt, wem erlaubt ist auf bestimmte IT-Ressourcen (bspw. Dateien/Funktionen) zuzugreifen. Personen ohne Zugriffskontrolle zu einer Funktion dürfen diese nicht ausführen.</p>

## Kryptographie

### Hashfunktion

$$h: X \rightarrow Y, \{0,1\}^* \rightarrow \{0,1\}^l$$

- Effizient berechenbare Einwegfunktion
- Die Umkehrfunktion ist nicht, bzw. nur sehr schwierig berechenbar
- Hashfunktionen haben eine feste Outputlänge  $l$

**Kollision** Zwei Inputs  $X_1, X_2$  bilden auf denselben Hashwert  $Y$  ab

### Schwache Kollisionsresistenz

Eine Einwegfunktion wird als schwach (im Sinne von „leichter zu erreichen“) kollisionsresistent bezeichnet, wenn kein Angreifer in der Lage ist, zu einem gegebenen Input  $X_1$  ein zweites  $X_2$  zu finden, das auf denselben Wert  $Y$  abgebildet wird. Inputs sind frei vom Angreifer wählbar.

Durchschnittlicher Aufwand:  $2^{\frac{l}{2}}$

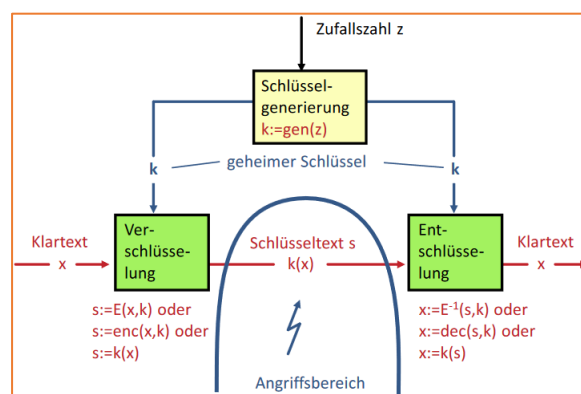
### Starke Kollisionsresistenz

Finde eine Kollision  $h(X_1)$  zu einem vorgegebenen  $h(X_2)$ . Es ist in der Regel nicht mit vertretbarem Aufwand möglich, eine Kollision gezielt herbeizuführen.

Durchschnittlicher Aufwand:  $2^l$

### Symmetrische Verschlüsselung

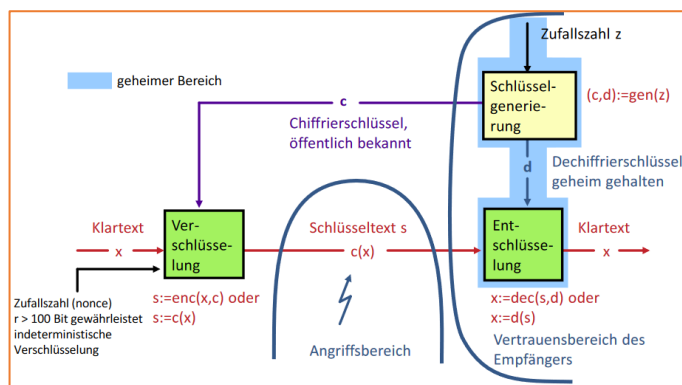
Sender und Empfänger ver- und entschlüsseln ihre Daten mit demselben Schlüssel  $k$ .



### Asymmetrische Verschlüsselung

Der Empfänger generiert ein Schlüsselpaar  $c, d$ .  $c$  wird zum Verschlüsseln,  $d$  zum Entschlüsseln genutzt.

$c$  kann frei an den Sender übertragen werden. Der Sender verschlüsselt die Daten mit  $c$  und überträgt sie an den Empfänger. Der Empfänger kann nun die Daten mit  $d$  entschlüsseln.



## Maskerade-Angriff

Ein Angreifer greift auf die Leitung zwischen dem Empfänger und Sender ein. Der Angreifer blockiert die Übertragung von  $c_{\text{Empfänger}}$  vom Empfänger an den Sender und schickt ihm stattdessen seinen eigenen Chiffrierschlüssel  $c_{\text{Angreifer}}$ . Nachdem der Sender seine Daten mit  $c_{\text{Angreifer}}$  verschlüsselt hat und an den Empfänger überträgt, kann der Angreifer diese entschlüsseln und auslesen.

Möglichkeit dies zu verhindern: **Schlüsselzertifizierung**

## Schlüsselzertifizierung

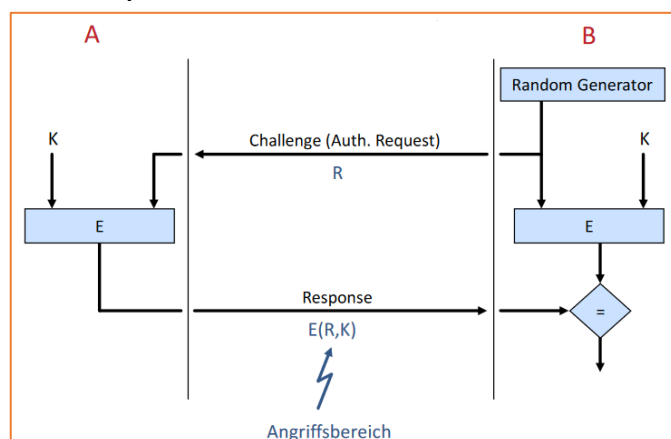
1. Empfänger generiert Schlüssel  $c$  und  $d$
2. Empfänger lässt Schlüssel von Zertifizierungsstelle **CA** eintragen und erhält ein Schlüsselzertifikat zurück
3. Sender besorgt sich  $c$  vom Empfänger
4. Sender prüft das Schlüsselzertifikat bei **CA** und fragt ggf. ob der Schlüssel noch gültig ist
5. Sender verschlüsselt die Daten mit  $c$  und überträgt sie an den Empfänger

## Challenge-Response-Authentifikation (Frage-Antwort-Verfahren)

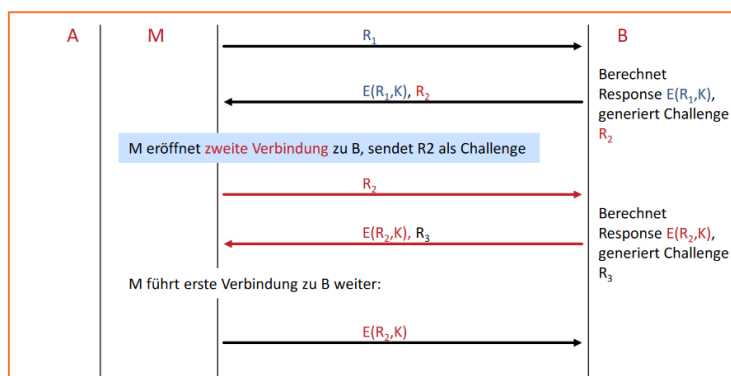
- Basiert meist auf symmetrischen Systemen
- A soll sich vor B authentisieren

B überträgt Zufallszahl  $R$  an A. A verschlüsselt  $R$  mit dem symmetrischen Schlüssel  $K$  und sendet das Resultat an B. B tut dasselbe und prüft, ob das Ergebnis identisch zu dem von A ist.

Dies kann auch zur gegenseitigen Authentifizierung genutzt werden. A schickt eine Challenge an B und B eine an A.



Das Verfahren hat eine Schwachstelle. Sei M ein Angreifer der A imitieren möchte. M sendet eine Challenge an B, B eine an M. Um die Challenge von B zu lösen, startet M eine neue Verbindung zu B und überträgt ihm die Challenge, die B zuvor an M gesendet hatte. Mit der Antwort von B löst M die zuvor gestellte Challenge von B.



## Informationstheorie

### One-Time-Pad

- Informationstheoretisches sicheres Verfahren
- Klartext Nachricht XOR Schlüssel = Schlüsseltext
- Aber: Schlüssellänge = Nachrichtenlänge
- Schlüssel darf nur 1x benutzt werden

### Informationstheorie

- Ein Verfahren ist perfekt (informationstheoretisch) sicher, wenn drei Bedingungen gelten:
  - Für Nachrichtenraum  $M$ , Schlüsselraum  $K$ , Schlüsseltextraum  $C$

$$|M| = |K| = |C|$$

- Schlüssel werden gleichverteilt zufällig gewählt  $p(k) = \frac{1}{|K|}$
- Für jedes  $m \in M, c \in C$  gibt es ein eindeutiges  $k \in K$  mit  $Enc_k(m) = c$

**(Möglicher) Informationsgehalt  $H_0$**  einer Quelle mit  $n$  Elementen:

$$H_0 = \log_2(n) \frac{\text{bit}}{\text{Element}}$$

**Informationsgehalt  $I(x_i)$**  eines einzelnen Zeichens  $x_i$  basierend auf seiner Wahrscheinlichkeit:

$$I(x_i) = \log_2\left(\frac{1}{p(x_i)}\right) = -\log_2(p(x_i)) \text{ bit}$$

**Entropie  $H(X)$**  (Ungewissheit) eines Codes  $X$  mit  $n$  Zeichen, liegt zwischen 0 und 1:

$$H(X) = \sum_{i=1}^n (p(x_i) \log_2\left(\frac{1}{p(x_i)}\right)) = - \sum_{i=1}^n (p(x_i) \log_2(p(x_i)))$$

**Redundanz  $R$**  eines Codes mit Informationsgehalt  $H_0$  und Entropie  $H$ :

$$R = H_0 - H$$

- Höhere Redundanz bedeutet bessere Komprimierbarkeit

Geschrieben von David Rath

*david.rath@studium.uni-hamburg.de*