Kryptographie – Vorlesung IT-Sicherheit

Folien von Marian Margraf

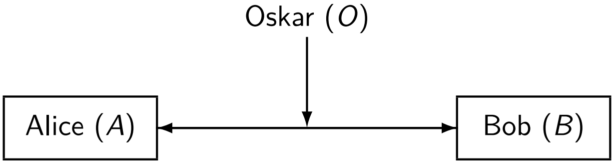
25. November 2014

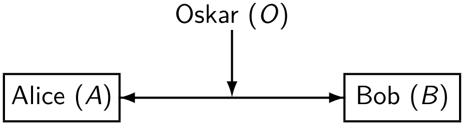
Das Wissenschaftsgebiet Kryptologie setzt sich zusammen aus den Teilgebieten:

- Kryptographie: altgriechisch „$\kappa\rho\upsilon\pi\tau o\sigma$“ (geheim) und „$\gamma\rho\alpha\varphi\epsilon\iota\nu$“ (schreiben)
  Ursprünglich also Wissenschaft der Verschlüsselung von Nachrichten
- Kryptoanalyse: altgriechisch „ $\alpha\nu\alpha\lambda\upsilon\sigma\iota\varsigma$ “ (Auflösung)
  Analyse der Verfahren, um diese zu brechen oder Sicherheit zu erhöhen

- Die klassische Kryptologie diente der Geheimhaltung von Nachrichten und wurde hauptsächlich von Militärs, Geheimdiensten und Diplomaten genutzt.
    - Schutziel Vertraulichkeit
- Die moderne Kryptographie (etwa seit 1975) beschäftigt sich mit erheblich weitergehenden Kommunikations- und Sicherheitsproblemen.
    - Schutziele Vertraulichkeit, Integrität, Authentizität, Nichtabstreitbarkeit

- Alice (A) und Bob (B) wollen sicher kommunizieren (vgl. Schutzziele)
- Oskar (O) versucht, die Schutzziele zu durchbrechen
  - Passiver Angriff: Abhören der Daten
  - Aktiver Angriff: Manipulation (z.B. Verändern, Fälschung Sender) der Daten





- *Vertraulichkeit:* Nachricht zwischen A und B kann nicht von O gelesen werden
- *Integrität:* Nachricht zwischen A und B wird nicht verändert bzw. A und B können erkennen, ob Nachrichten verändert wurden
- *Datenauthentizität:* B kann Nachricht von A zweifelsfrei A zuordnen
- *Instanzauthentizität:* B kann die Identität von A zweifelsfrei feststellen
- *Nichtabstreitbarkeit:* B kann Nachricht von A zweifelsfrei auch einer dritten Partei als Nachricht von A nachweisen

Beispiele für kryptographische Verfahren für alle Schutzziele:

- *Vertraulichkeit:* Verschlüsselung
- *Integrität:* Hashfunktionen, Message Authentication Codes (MAC), Signaturen
- *Datenauthentizität:* Message Authentication Codes (MAC), Signaturen
- *Nichtabstreitbarkeit:* Signaturen
- *Instanzauthentizität:* Challenge Response Protokolle

Auguste Kerckhoffs (ein niederländischer Linguist und Kryptograph) 1883:
(La Cryptographie militaire)

- Ist ein System nicht beweisbar sicher, so sollte es praktisch sicher sein.
- Das Design eines System sollte keine Geheimhaltung erfordern und sollte sich ohne Gefahr in den Händen des Feindes befinden können.
- Ein Kryptosystem muss einfach bedienbar sein.

Wichtigstes Kerckhoffsches Prinzip

Ein Angreifer kennt das kryptographischen Verfahren, nur die privaten oder symmetrischen Schlüssel sind geheim.

Wir unterscheiden zwei Klassen kryptographischer Verfahren

- *Symmetrische Verfahren:* A und B nutzen den selben Schlüssel
  - Der Schlüssel muss zwischen A und B sicher ausgetauscht werden
    - vertraulich: O darf den Schlüssel nicht kennen
    - authentisch: A und B müssen wissen, wem sie vertrauliche Nachrichten schicken
- *Asymmetrische Verfahren:* A und B haben jeweils ein Schlüsselpaare
  - A hat Schlüsselpaar (pk_A, sk_A) (pk_A: public key, sk_A : secret key)
  - B hat Schlüsselpaar (pk_B, sk_B) (pk_B: public key, sk_B : secret key)
  - Die öffentlichen Schlüssel müssen zwischen A und B sicher ausgetauscht wer
    - authentisch: A und B müssen wissen, wem sie vertrauliche Nachrichten schicken
    - O darf die Schlüssel pk_A, pk_B nicht kennen (sk_A, sk_B aber schon)

Schutzziel: Vertraulichkeit

- Verschlüsselungsfunktion enc (encryption):
  - Klartext m, Schlüssel k, Chiffrat $c = \mathrm{enc}(k, m)$
- Entschlüsselungsfunktion dec (decryption):
  - Chiffrat c, Schlüssel k, Klartext $m = \mathrm{dec}(k, c)$
- Zusammenhang: Für alle Klartexte m und Schlüssel k: $\mathrm{dec}(k, \mathrm{enc}(k, m)) = m$

$\boxed{A}$ Schlüssel: k		$\boxed{B}$ Schlüssel: k
compute $c := \mathrm{enc}(k, m)$		
	$\xrightarrow{c}$	
		compute $m' := \mathrm{dec}(k, c)$

Wegen $\mathrm{dec}(k, \mathrm{enc}(k, m)) = m$ gilt $m = m'$.

Beispiel CAESAR-Verfahren

Eingesetzt von Gaius Iulius Caesar (römischer Kaiser 100 - 44 v. Chr.):

- Verschlüsselung: Verschiebe alle Buchstaben um einen festen Wert k nach rechts
  Für $k = 3$: A $\to$ D, B $\to$ E, C $\to$ F, $\ldots$, Z $\to$ C
- Entschlüsselung: Verschiebe alle Buchstaben um den selben Wert nach links
  Für $k = 3$: A $\to$ X, B $\to$ Y, C $\to$ Z, $\ldots$, Z $\to$ W
- Dann gilt:
  - $\text{enc}(3, \text{CAESAR}) = \text{FDHVDU}$
  - $\text{dec}(3, \text{FDHVDU}) = \text{CAESAR}$

Um den Klartext zu erhalten, probieren wir alle Schlüssel aus:

Ciphertext:	`OLYY KLY YPUNL`
Verschiebung um 1 nach links:	`NKXX JKX XOTMK`
Verschiebung um 2 nach links:	`MJWW IJW WNSLJ`
Verschiebung um 3 nach links:	`LIVV HIV VMRKI`
Verschiebung um 4 nach links:	`KHUU GHU ULQJH`
Verschiebung um 5 nach links:	`JGTT FGT TKPIG`
Verschiebung um 6 nach links:	`IFSS EFS SJOHF`
Verschiebung um 7 nach links:	`HERR DER RINGE`

- Es gibt 26 Buchstaben im deutschen Alphabet
  - Verschiebung um 27 ist das selbe wie Verschiebung um 1, ...
- Es gibt also nur 26 verschiedene Schlüssel
  - Durchsuchen des Schlüsselraums möglich (**Schlüsselexhaustion, Brute Force**)
- Für große Nachricht ergibt sich in der Regel nur ein sinnvollen Text
- Erster kryptoanalytischer Ansatz: Schlüsselexhaustion

Erkenntnis

Für die Sicherheit eines Verschlüsselungsverfahrens muss der Schlüsselraum (Menge der möglichen Schlüssel) so groß sein, dass nicht alle Schlüssel durchprobiert werden können.

- Ersetze Buchstaben nicht mittels fester Verschiebung, sondern beliebig (Monoalphabetische Verschlüsselung)

$$\begin{array}{cccccccc} A & B & C & D & \cdots & X & Y & Z \\ \downarrow & \downarrow & \downarrow & \downarrow & \cdots & \downarrow & \downarrow & \downarrow \\ X & E & S & U & \cdots & M & H & P \end{array}$$

- Wie viele Schlüssel gibt es?
  - A kann durch 26 Buchstaben ersetzt werden
  - B kann durch 25 Buchstaben ersetzt werden
  - Insgesamt: $26 \cdot 25 \cdots 2 \cdot 1 = 26!$ verschiedene Schlüssel
- 26! Schlüssel sind zu viel, um alle durchzuprobieren

Anzahl Schlüssel: $26! \approx 2^{88} \approx 10^{27}$

- Anzahl der Atome der Erde 2^{170}
- Anzahl der Atome der Sonne 2^{190}
- Anzahl der Atome in unserer Galaxis 2^{223}
- Zeit bis die Sonne zur Nova wird 2^{30} Jahre

Angenommen, ein Computer berechnet pro Sekunde $2 \cdot 10^9$ Entschlüsselungen:

$$\frac{\text{Anzahl möglicher Schlüssel}}{\text{Entschl. je Sek. * Sek. pro Jahr}} = \frac{10^{27}}{(2 * 10^9 s^{-1}) * (3.15 * 10^7 s/\text{Jahr})}$$

$$= \frac{10^{11}}{6.3} > 10^{10} = 10.000.000.000 \text{ Jahre}$$

- Angriff über relative Häufigkeiten in sinnvollen deutschen Texten
  1. Buchstabe E: 17.4 %
  2. Buchstabe N: 09,8 %
  3. Buchstabe I: 07.6 %
  4. Buchstabe S: 07,3 %
  5. Buchstabe R: 07,0 %
  6. Buchstabe A: 06,5 %
  7. Buchstabe T: 06,2 %
  8. Buchstabe D: 05,1 %

     $\vdots$

  26. Buchstabe Q: 00,02 %
- Richtige Wahl von E und N liefert schon ca. 1/4 des Textes.
  Rest erhält man durch geschicktes Raten.

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of

sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt

yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts

sof Kdafopdtp xt Meyyxtpot uoxt Otso.
```

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e          e  e         e  e          e     e       e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
 e   e          e e   e  e  e     e       e   e          e
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e    e      e     e   e    e e      e       e   e e e
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
 e      e               e   e   E  e
```

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E							

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e          e  e  n   n  e  e   n   n  en    e       e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
 e n e          e e   e ne  e n n e       en  e          e n
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  n e      e     e   e    e en     e       e   e e e   n
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
 e      e  n   n      n en  e n En e
```

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N						

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e          e  e  n   n  e  e   n   n  en    e       e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
 e n e          e e   e ne  e n n e       en  e          e n
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  n e      e     e   e    e en     e       e   e e e   n
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
 e      e  n   n      n en  e n En e
```

Letztes Wort: Otse = Ende oder Ente

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N						

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e          e  e  n   n  e  e   nd  n  end    e  d    e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e          e e   e ne  e n nde       en  e          e n
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  nde      e     e   e    e en     e      de   e ede   nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
de      e  n   n      n en  e n Ende
```

Letztes Wort: Otse = Ende oder Ente

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N				D		

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Modifizierten CAESAR-Verfahrens: Beispiel

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e          e  e  n   n  e  e   nd  n  end   e  d    e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e          e e   e ne  e n nde       en  e          e n
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  nde      e     e   e    e en     e      de   e ede   nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
de      e  n   n      n en  e n Ende
```

Viertletztes Wort: xt = in

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N				D		

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e    i     e  e in   n  e  e   nd  n  endi  e  d    e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e          eie   eine  ein nde   i   en  e          ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  nde      e   i e   e    e en     e      de   e ede   nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
de      e  n  in     in en  ein Ende
```

Viertletztes Wort: xt = in

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I			D		

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
     e    i     e  e in   n  e  e   nd  n  endi  e  d    e
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e         eie   eine  ein nde   i   en  e          ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 e  nde      e   i e   e    e en     e      de   e ede   nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
de      e  n  in     in en  ein Ende
```

Kurze Wörter: sof,soh = der,des

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I			D		

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
  s  err  i     e  e in   n  e  e s nd  n  endi  e  d ss er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e  s    r  eier seines ein nde   i s en  e  r s   s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  r e   i e   es   e en     e    r des  eredes  nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der    re  n  in     in en  ein Ende
```

Kurze Wörter: sof,soh = der,des

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R		D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
  s  err  i     e  e in   n  e  e s nd  n  endi  e  d ss er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e  s    r  eier seines ein nde   i s en  e  r s   s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  r e   i e   es   e en     e    r des  eredes  nd
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der    re  n  in     in en  ein Ende
```

dts = und

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R		D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
  s  err  i      eu e in    n  eu e s nd  n uendi  e  d ss er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e  s   ur  eier seines einunde   i s en  e ur s   s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  r e   i es  es   e en     e    r des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der  u re un  in     in en  ein Ende
```

dts = und

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
  s  err  i     eu e in   n  eu e s nd  n uendi  e  d ss er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de n e  s   ur  eier seines einunde   i s en  e ur s   s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  r e   i es  es   e en     e    r des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der  u re un  in     in en  ein Ende
```

skhh = dass

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Modifizierten CAESAR-Verfahrens: Beispiel

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
A s  err  i     eu e in   n  eu e sand an uendi  e  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  s   ur  eier seines einunde   i s en  e ur s a s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae   i es  es   e en     e   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in     in en  ein Ende
```

skhh = dass

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
A s  err  i     eu e in   n  eu e sand an uendi  e  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  s   ur  eier seines einunde   i s en  e ur s a s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae    i es  es   e en     e   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in     in en  ein Ende
```

Kvh = Als

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
Als  err  il    eu elin   n  eu elsand an uendi  e  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  s   ur  eier seines einundel  i s en  e ur s a s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae   i es  es   e en   lle   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in     in en  ein Ende
```

Kvh = Als

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
Als  err  il    eu elin   n  eu elsand an uendi  e  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  s   ur  eier seines einundel  i s en  e ur s a s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae   i es  es    e en    lle   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un   in     in en   ein Ende
```

G = T?

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
Als  err  il    eutelin   n  eutelsand an uendi te  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  st  ur  eier seines einundel  i sten  e urtsta s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae  ti es  est  e en   lle   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in     in en  ein Ende
```

G = T?

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	T

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Modifizierten CAESAR-Verfahrens: Beispiel

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
Als  err  il    eutelin   n  eutelsand an uendi te  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae  st  ur  eier seines einundel  i sten  e urtsta s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
 es nders  rae  ti es  est  e en   lle   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in     in en  ein Ende
```

Moff = Herr, yohetsofh = besonders

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	T

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Geheimtext:

```
Kvh Moff Yxvye Yodgovxt zet Yodgovhkts ktudotsxpgo, skhh of
Als Herr Bilbo Beutelin  on Beutelsand an uendi te  dass er
sortkonmhg ldf Aoxof hoxtoh oxtdtsovalxphgot Poydfghgkph oxt
de nae hst  ur  eier seines einundel  i sten  eburtsta s ein
yohetsofh wfkonmgxpoh Aohg poyot cevvo, ckf soh Pofosoh dts
besonders  rae hti es  est  eben  olle   ar des  eredes und
sof Kdafopdtp xt Meyyxtpot uoxt Otso.
der Au re un  in Hobbin en  ein Ende
```

Moff = Herr, yohetsofh = besonders

Häufigkeiten im Text (Plätze 1 - 8):

O	T	X	F	D	S	H	G
36	18	12	12	11	11	10	9
E	N	I	R	U	D	S	T

Relative Häufigkeiten (Plätze 1 - 8):

E	N	I	S	R	A	T	D
17,4	9,8	7,6	7,3	7,0	6,5	6,2	5,1

Solche Angriffe heißen auch statistische Angriffe

Erkenntnis

- Zur Beurteilung der Sicherheit müssen alle kryptoanalytischen Methoden berücksichtigt werden
- Größe des Schlüsselaums ist nur eine notwendige (keine hinreichende) Bedingung für die Sicherheit.

Wir betrachten Nachrichten als Bitstrings:

- Eine Nachricht $m = (m_1, \ldots, m_n)$ besteht nur aus 0 und 1
- Für eine Nachricht der Länge $n \in \mathbb{N}$ schreiben wir auch kurz: $m \in \{0,1\}^n$
- Beispiel: Kodierung der Buchstaben über ASCII-Kode
  $A = 01000001$, $B = 01000010$, ...

Als Schlüssel nutzen wir ebenfalls Bitstrings:

- Für eine Nachricht der Länge $n \in \mathbb{N}$ benötigen wir
- Schlüssel $k = (k_1, \ldots, k_n) \in \{0,1\}^n$ der Länge n.

Rechenoperation: Addition modulo 2 (XOR, in Zeichen $\oplus$)

- $0 \oplus 0 = 0, 1 \oplus 0 = 1, 0 \oplus 1 = 1, 1 \oplus 1 = 0$

- Verschlüsselung: Ciphertext $c = m \oplus k$.

$$c = (c_1, \ldots, c_n) = (m_1, \ldots, m_n) \oplus (k_1, \ldots, k_n) = (m_1 \oplus k_1, \ldots, m_n \oplus k_n)$$

- Entschlüsselung: Klartext $m = c \oplus k$.

$$m = (m_1, \ldots, m_n) = (c_1, \ldots, c_n) \oplus (k_1, \ldots, k_n) = (c_1 \oplus k_1, \ldots, c_n \oplus k_n)$$

- Verschlüsselung: Ciphertext $c = m \oplus k$.

$$c = (c_1, \ldots, c_n) = (m_1, \ldots, m_n) \oplus (k_1, \ldots, k_n) = (m_1 \oplus k_1, \ldots, m_n \oplus k_n)$$

- Entschlüsselung: Klartext $m = c \oplus k$.

$$m = (m_1, \ldots, m_n) = (c_1, \ldots, c_n) \oplus (k_1, \ldots, k_n) = (c_1 \oplus k_1, \ldots, c_n \oplus k_n)$$

- Korrektheit: Wegen $0 \oplus 0 = 0$ und $1 \oplus 1 = 0$ gilt

$$k \oplus k = (k_1, \ldots, k_n) \oplus (k_1, \ldots, k_n) = (k_1 \oplus k_1, \ldots, k_n \oplus k_n) = (0, \ldots, 0)$$

- Verschlüsselung: Ciphertext $c = m \oplus k$.

$$c = (c_1, \ldots, c_n) = (m_1, \ldots, m_n) \oplus (k_1, \ldots, k_n) = (m_1 \oplus k_1, \ldots, m_n \oplus k_n)$$

- Entschlüsselung: Klartext $m = c \oplus k$.

$$m = (m_1, \ldots, m_n) = (c_1, \ldots, c_n) \oplus (k_1, \ldots, k_n) = (c_1 \oplus k_1, \ldots, c_n \oplus k_n)$$

- Korrektheit: Wegen $0 \oplus 0 = 0$ und $1 \oplus 1 = 0$ gilt

$$k \oplus k = (k_1, \ldots, k_n) \oplus (k_1, \ldots, k_n) = (k_1 \oplus k_1, \ldots, k_n \oplus k_n) = (0, \ldots, 0)$$

  - Wir erhalten

$$c \oplus k \quad = \quad (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0 = m$$

Klartext: ASCII:	G 01000111	E 01000101	H 01001000	E 01000101	I 01001001	M 01001101
	$\oplus$					
Schlüssel:	00111000	10011110	10011111	00010111	10100111	10011110
	$=$					
Ciphertext:	01111111	11011011	11010111	01010010	11101110	11010011
	$\oplus$					
Schlüssel:	00111000	10011110	10011111	00010111	10100111	10011110
	$=$					
ASCII: Klartext:	01000111 G	01000101 E	01001000 H	01000101 E	01001001 I	01001101 M

Wir betrachten folgendes Experiment: Münzwurf

- Bei einmaligem Münzwurf gibt es nur zwei Ereignisse: Kopf oder Zahl
  - Die Wahrscheinlichkeit, Kopf zu werfen, ist $\frac{1}{2}$, kurz $\Pr(\text{Kopf}) = \frac{1}{2}$
  - Die Wahrscheinlichkeit, Zahl zu werfen, ist ebenfalls $\frac{1}{2}$

Wir betrachten folgendes Experiment: Münzwurf

- Bei einmaligem Münzwurf gibt es nur zwei Ereignisse: Kopf oder Zahl
  - Die Wahrscheinlichkeit, Kopf zu werfen, ist $\frac{1}{2}$, kurz $\Pr(\text{Kopf}) = \frac{1}{2}$
  - Die Wahrscheinlichkeit, Zahl zu werfen, ist ebenfalls $\frac{1}{2}$

Identifiziere Kopf mit 0 und Zahl mit 1.

Wir betrachten folgendes Experiment: Münzwurf

- Bei einmaligem Münzwurf gibt es nur zwei Ereignisse: Kopf oder Zahl
  - Die Wahrscheinlichkeit, Kopf zu werfen, ist $\frac{1}{2}$, kurz $\Pr(\text{Kopf}) = \frac{1}{2}$
  - Die Wahrscheinlichkeit, Zahl zu werfen, ist ebenfalls $\frac{1}{2}$

Identifiziere Kopf mit 0 und Zahl mit 1.

- Bei zweimaligem Münzwurf gibt es vier Ereignisse: 00, 01, 10 und 11
  - $\Pr(00) = \Pr(01) = \Pr(10) = \Pr(11) = \frac{1}{4}$

Wir betrachten folgendes Experiment: Münzwurf

- Bei einmaligem Münzwurf gibt es nur zwei Ereignisse: Kopf oder Zahl
  - Die Wahrscheinlichkeit, Kopf zu werfen, ist $\frac{1}{2}$, kurz $\Pr(\text{Kopf}) = \frac{1}{2}$
  - Die Wahrscheinlichkeit, Zahl zu werfen, ist ebenfalls $\frac{1}{2}$

Identifiziere Kopf mit 0 und Zahl mit 1.

- Bei zweimaligem Münzwurf gibt es vier Ereignisse: 00, 01, 10 und 11
  - $\Pr(00) = \Pr(01) = \Pr(10) = \Pr(11) = \frac{1}{4}$
- Bei $n-$maligem Münzwurf gibt es 2^n Ereignisse
  - Die Wahrscheinlichkeit, das richtige Ereignis zu raten: $\frac{1}{2^n}$

Für Nachricht $m \in \{0,1\}^n$ benötigen wir Schlüssel $k \in \{0,1\}^n$

- Wir nennen k zufällig, wenn $\Pr(k) = \frac{1}{2^n}$
  (Wahrscheinlichkeit, dass Angreifer den richtigen Schlüssel errät, ist $\frac{1}{2^n}$)
- Die zufällige Wahl der eingesetzten Schlüssel ist wesentlich für die Sicherheit:
  - Sind alle Schlüssel gleichwahrscheinlich, hat der Angreifer keinen Anhaltspunkt, welcher Schlüssel eingesetzt wurde

- Welche Angriffe gibt es gegen das One-time Pad?
  (Unter der Voraussetzung, dass der Schlüssel zufällig gewählt wurde)
- In unserem Beispiel:
  - Aus Klartext $m =$ GEHEIM (48 Bit in ASCII) wurde mittels Schlüssel $k = (k_1, \ldots, k_{48})$ ein Ciphertext $c = (c_1, \ldots, c_{48})$
  - Durchprobieren aller Schlüssel auf den Ciphertext führt zu 2^{48} Klartexten
    - Nicht sinnvolle Texte: üÄY:;- AAAAAA **LAIs
    - Sinnvolle Texte: Berlin BOSTON Er ist
  - Ein Angreifer hat keine Information, welcher der sinnvollen Texte der richtige ist

$$
\begin{array}{rcl}
\text{Klartext:} & 001001110 & \longleftarrow \quad \text{Kein Zufall} \\
 & \oplus & \\
\text{Schlüssel:} & 100011100 & \longleftarrow \quad \text{Zufall} \\
 & = & \\
\text{Ciphertext:} & 101010010 & \longleftarrow \quad \text{Zufall}
\end{array}
$$

Aus Sicht des Angreifers ist der Ciphertext eine absolut zufällige Bitfolge

- Genauso zufällig wie der eingesetzte Schlüssel
- Unabhängig vom verschlüsselten Klartext

Erkenntnis

Beim One-time Pad Verfahren liefert der Ciphertext keinerlei Informationen über den Klartext (und damit auch nicht über den eingesetzte Schlüssel).

Das One-time Pad ist also sicher im Sinne der folgenden Definition.

Definition

Ein Verschlüsselungsverfahren heißt absolut sicher, wenn es auch gegen Angreifer mit unbeschränkten Ressourcen (Zeit, Rechenleistung) sicher ist.

Absolut sichere Verschlüsselungsverfahren haben einen entscheidenden Nachteil:

Claude Shannon, 1948

Ein Verschlüsselungsverfahren kann nur dann absolut sicher sein, wenn der Schlüssel genauso lang ist wie die zu verschlüsselnde Nachricht.

Zum Teil müssen aber große Datenmengen verschlüsselt werden:

- Bilder und Videos von Spionagesatelliten
- Sprach- und Videotelephonie über das Internet

Für die Verschlüsselung von 10 GB benötigen wir einen Schlüssel der Größe 10 GB.

Idee: Verwende den Schlüssel mehrfach

- Wir wollen zwei Nachrichten $m^1 = (m_1^1, \dots, m_n^1)$ und $m^2 = (m_1^2, \dots, m_n^2)$ mit **einem** Schlüssel $k = (k_1, \dots, k_n)$ verschlüsseln.
- Für die Ciphertexte $c^1 = m^1 \oplus k$ und $c^2 = m^2 \oplus k$ gilt dann

$$c^1 \oplus c^2 = (m^1 \oplus k) \oplus (m^2 \oplus k) = m^1 \oplus m^2 \oplus \underbrace{k \oplus k}_{=(0,\dots,0)} = m^1 \oplus m^2 \qquad (1)$$

Wir kennen also die Summe der Klartexte, ohne den Schlüssel kennen zu müssen.

Statistischer Angriff

- Bestimme alle sinnvollen Texte für m^1
- Überprüfe, ob die Wahl von m^1 auch einen sinnvollen Text für m^2 ergibt: Wegen $c^1 \oplus c^2 = m^1 \oplus m^2$ (Folie **??**, Formel (**??**)) gilt

$$m^2 = c^1 \oplus c^2 \oplus m^1$$

  - Wenn ja: haben wir die Nachrichten m^1, m^2 gefunden
  - Wenn nein: wählen wir einen anderen Kandidaten für m^1
- Wahrscheinlichkeit, dass es hier mehrere Möglichkeiten gibt, ist klein
- Wahrscheinlichkeit sinkt mit Anzahl der Nachrichten

Erkenntnis

Selbst auf den ersten Blick kleine Veränderungen eines kryptographsichen Verfahrens können zu einer massiven Senkung der Sicherheit führen.

- Absolute Sicherheit: Resistent gegen Angreifer mit unbeschränkten Ressourcen
- Praktische Sicherheit: Resistent gegen Angreifer mit beschränkten Ressourcen:
    - Angreifer haben nur beschränkte Rechenkapazität und Zeit
    - Kryptoverfahren sind nur sicher in Bezug auf diese Ressourcen

Definition Sicherheitsniveau

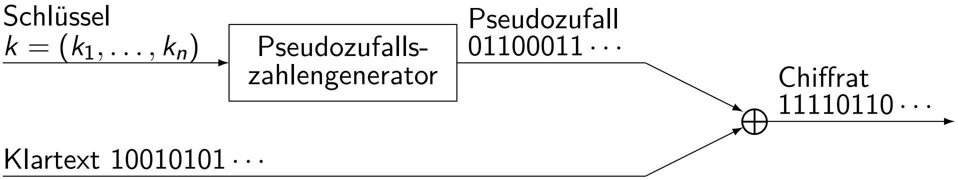
Ein Kryptoverfahren hat ein Sicherheitsniveau von $n \in \mathbb{N}$ Bit, wenn ein Angreifer 2^n Versuche benötigt, dass Verfahren zu brechen.
(Für Verschlüsselungsverfahren: den Klartext zu erhalten.)

- Für die Beurteilung des Sicherheitsniveaus müssen alle kryptoanalytischen Verfahren herangezogen werden:
  - Schlüsselexhaustion (Durchprobieren aller Schlüssel)
  - statistische Angriffe (wie beim modifizierten CAESAR-Verfahren)
  - $\dots$
- Sicherheitsniveaus der bisherigen Verfahren:
  - CAESAR-Verfahren: $26 < 2^5$ Schlüssel, also Sicherheitsniveau < 5 Bit
  - modifiziertes CAESAR-Verfahren (monoalphabetische Verschlüsselung)
    - $26! \approx 2^{88}$ Schlüssel, also Sicherheitsniveau nicht größer als 88 Bit
    - statistischer Angriff: Schwer zu quantifizieren, ca. $2^6 > 50$ Versuche
    - Sicherheitsniveau: < 6 Bit
  - One-time Pad: Sicherheitsniveau ∞ Bit, unabhängig von Schlüssellänge

- Heutige Kryptoverfahren sollten ein Sicherheitsniveau $\geq$ 100 Bit haben
  - 2^{100} Versuche sind praktisch nicht durchführbar
  - Solche Verfahren gelten damit als **praktisch sicher**
- Erinnerung: Schlüsselexhaustion ist ein Angriff
  - Sicherheitsniveau $\geq$ 100 Bit bedeutet also mindestens 2^{100} Schlüssel
  - Für einen Schlüsselraum der Form $\{0, 1\}^n$ also $n \geq 100$

Nachbildung des One-time Pads

- Aus Schlüssel $k \in \{0,1\}^n$, $n \geq 100$ wird ein pseudozufälliger Schlüssel generiert (Schlüsselstrom)
- Der Schlüsselstrom wird komponentenweise mit dem Klartext addiert (XOR)



Stromchiffren

Welche Bedingungen muss der Pseudozufallszahlengenerator erfüllen?

- Erste Idee: Nutze Schlüssel k mehrmals

$$\xrightarrow{k = (k_1, \ldots, k_n)} \boxed{\begin{array}{c}\text{Pseudozufalls-}\\ \text{zahlengenerator}\end{array}} \xrightarrow{k_1, \ldots, k_n, k_1, \ldots, k_n, \ldots}$$

- Unsicher, siehe Folie **??**

Welche Bedingungen muss der Pseudozufallszahlengenerator erfüllen?

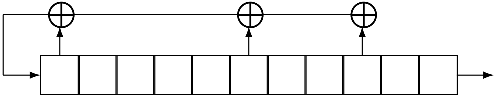
- Aus 100 Bit Zufall (Schlüssel) lässt sich nicht z.B. 200 Bit Zufall (Schlüsselstrom) berechnen
  - 200 Bit Zufall hieße: Angreifer rät Schlüsselstrom mit Wahrscheinlichkeit $1/2^{200}$
  - Er muss aber nur Schlüssel raten (Wahrscheinlichkeit $1/2^{100}$), und dann den Schlüsselstrom berechnen
- Wir benötigen nur praktische Sicherheit
  - Angreifer (beschränkte Ressourcen) sollte keinen Unterschied feststellen zwischen
    - echtem Zufall
    - Pseudozufall, der von einem Pseudozufallszahlengenerator stammt

Stromchiffren

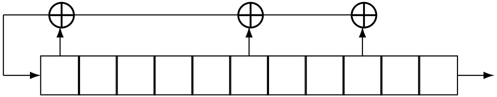
- Formale Definition von Pseudozufallszahlengenerator in den Vorlesungen
  - Kryptologie (Wahlpflicht im Bachelor)
  - Komplexitätstheorie (Master)
- Wichtig für die Sicherheit von Stromchiffren ist u.a. folgende Bedingung:
  - Aus Teilen des Schlüsselstroms dürfen keine Nachfolger bestimmt werden können
  - Aus Teilen des Schlüsselstroms dürfen keine Vorgänger bestimmt werden können

- Ansonsten statistische Angriffe wie beim modifizierten One-time Pad möglich (siehe Folie **??**)

Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)

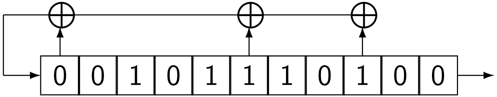


Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)



- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert

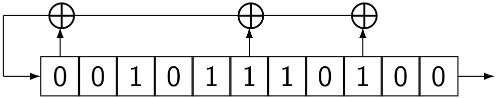
Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)



- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert

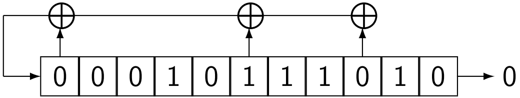
Stromchiffren: LFSR

Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)



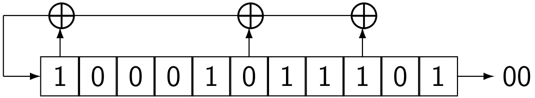
- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert
- In jedem Schritt (Takt), wird
  - ein Bit ausgegeben (jenes aus dem letzten Feld)
  - alles um eine Stelle nach rechts verschoben und
  - die ausgezeichneten Bits addiert und an die erste Stelle geschrieben

Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)



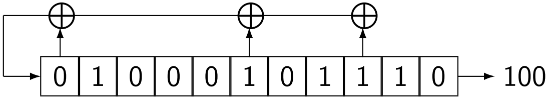
- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert
- In jedem Schritt (Takt), wird
  - ein Bit ausgegeben (jenes aus dem letzten Feld)
  - alles um eine Stelle nach rechts verschoben und
  - die ausgezeichneten Bits addiert und an die erste Stelle geschrieben

Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)

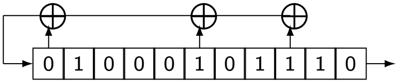


- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert
- In jedem Schritt (Takt), wird
  - ein Bit ausgegeben (jenes aus dem letzten Feld)
  - alles um eine Stelle nach rechts verschoben und
  - die ausgezeichneten Bits addiert und an die erste Stelle geschrieben

Grundbaustein vieler Pseudozufallszahlengeneratoren für Stromchiffren
Linear Feedback Shift Register (LFSR)



- Im ersten Schritt wird das Register mit dem Schlüssel initialisiert
- In jedem Schritt (Takt), wird
  - ein Bit ausgegeben (jenes aus dem letzten Feld)
  - alles um eine Stelle nach rechts verschoben und
  - die ausgezeichneten Bits addiert und an die erste Stelle geschrieben



- LFSRs allein sind nicht geeignet für sichere Pseudozufallszahlengeneratoren (werden als Grundbausteine verwendet)
- Hier lassen sich bei bekanntem Teil des Schlüsselstroms Nachfolger berechnen:
  - Inhalt des obigen Schieberegisters ist nach 11 Takten Teil des Schlüsselstroms
  - Errät ein Angreifer diesen Teil, kann er alle Nachfolger berechnen

Es gibt mehrere Möglichkeiten, LFSRs so zu kombinieren, dass die (gravierende) Schwäche verhindert wird

- Shrinking-Generator:
  - Nutzung von zwei LFSRs (R_1, R_2)
    - Wenn R_1 Bit 1 ausgibt, wird die Ausgabe von R_2 für Schlüsselstrom genutzt
    - Wenn R_1 Bit 0 ausgibt, wird die Ausgabe von R_2 verworfen
- Summations-Generator:
  - Nutzung von zwei LFSRs (R_1, R_2)
    - Die Ausgaben von R_1 und R_2 werden addiert ($\oplus$)
- Dadurch ist es einem Angreifer nicht möglich, aus dem Schlüsselstrom auf die internen Register der LFSRs zu schließen

Vorteile von Stromchiffren

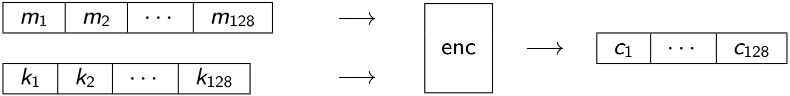
- Statistisches Verhalten von LFSRs sind mathematisch sehr gut untersucht
- LFSRs sind sehr effizient in Hardware umzusetzen
  - Benötigen lediglich die Operationen $\oplus$ und Shift
- Ver- und Entschlüsselung sind sehr effizient
- Typisches Beispiel:
  - A5/1: Sprachverschlüsselung zwischen Mobiltelefon und Funkmast
  - Eingesetzt seit den 1990ger Jahren
    Also auf Mobiltelefonen mit geringer Rechenleistung

- Blockchiffren bilden Bitstrings fester Länge auf Bitstrings der selben Länge ab:
  - Moderne Blockchiffren verarbeiten Bitstrings der Länge 128 (Klartextblock) (diese Länge heißt auch Blockgröße der Blockchiffre)
  - Genutzte Schlüssel müssen eine Bitlänge ≥ 100 haben (siehe Folie **??**) (meist werden Schlüssel der Länge 128 Bit genutzt)

Blockchiffren

- Blockchiffren bilden Bitstrings fester Länge auf Bitstrings der selben Länge ab:
  - Moderne Blockchiffren verarbeiten Bitstrings der Länge 128 (Klartextblock) (diese Länge heißt auch Blockgröße der Blockchiffre)
  - Genutzte Schlüssel müssen eine Bitlänge ≥ 100 haben (siehe Folie **??**) (meist werden Schlüssel der Länge 128 Bit genutzt)

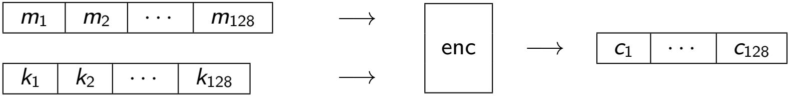
$$\mathrm{enc} : \underbrace{\{0,1\}^{128}}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^{128}}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^{128}}_{\text{Chiffrate}}$$



Blockchiffren

- Blockchiffren bilden Bitstrings fester Länge auf Bitstrings der selben Länge ab:
  - Moderne Blockchiffren verarbeiten Bitstrings der Länge 128 (Klartextblock) (diese Länge heißt auch Blockgröße der Blockchiffre)
  - Genutzte Schlüssel müssen eine Bitlänge ≥ 100 haben (siehe Folie **??**) (meist werden Schlüssel der Länge 128 Bit genutzt)

$$\mathsf{enc} : \underbrace{\{0,1\}^{128}}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^{128}}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^{128}}_{\text{Chiffrate}}$$



- Für längere Klartexte werden sogenannte Betriebsarten genutzt (siehe Folie **??**)

Wir behandeln zunächst nur die einzelne Blockchiffre

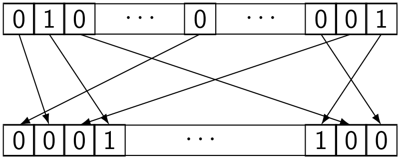
$$\text{enc} : \underbrace{\{0,1\}^{128}}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^{128}}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^{128}}_{\text{Chiffrate}}$$

Es gibt zwei etablierte Kontruktionsmethoden

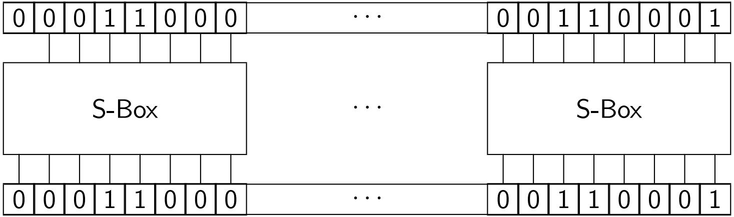
- Feistel-Chiffren:
  - Entwickelt von Horst Feistel in den 1970ger Jahren im IBM-Projekt Lucifer
  - Prominentes Beispiel: Data Encryption Standard (DES)
- Substitutions-Permutations-Netzwerk (SPN)
  - Grundidee geht auf Claude Shannon aus dem Jahr 1949 zurück
  - Prominentes Beispiel: Advanced Encryption Standard (AES), Nachfolger von DES

Beide Konstruktionsmethoden nutzen die selben Ideen:

- Diffusion:
  - Verteilung der Informationen des Klartextes über den gesamten Chiffretext
    - Jedes Bit des Chiffretext hängt von jedem Bit des Klartext ab
    - Bei Änderung eines Klartextbits ändern sich ca. 50 % der Geheimtextbits
  - Realisiert durch Permutationen (spezielle lineare Abbildungen)
- Konfusion
  - Zusammenhang zwischen Klartext und Chiffretext soll hochgradig komplex sein
    Gleiches gilt für den Zusammenhang zwischen Schlüssel und Chiffretext
  - Realisiert durch Substitutionen (nicht-lineare Abbildungen)
- Rundenbasiert: Wiederholte Anwendung von Diffusion, Konfusion und Schlüssel



- Reihenfolge der Bits werden über den gesamten Block (128 Bit) vertauscht
- Effekt der Diffusion ergibt sich erst nach mehreren Runden



- S-Boxen sind nicht-lineare Abbildungen $\{0,1\}^n \longrightarrow \{0,1\}^n$
- Werden üblicherweise nur auf Bitstrings der Länge 8 oder 16 angewandt
- Damit effizient zu implementieren über Arrays der Länge 2^8 oder 2^{16} (Arrays der Länge 2^{128} sind nicht speicherbar)

Klartextblock (128 Bit)

Klartextblock (128 Bit)

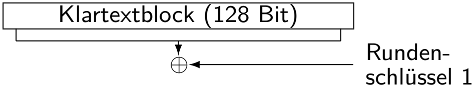
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet

Substitutions-Permutations-Netzwerk: Beispiel AES

Klartextblock (128 Bit)

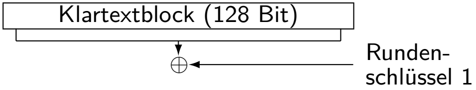
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet

2) In Runde 1 wird r_1 auf den Klartext addiert



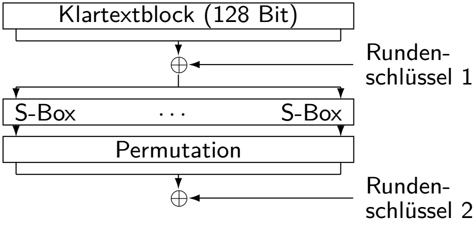
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert

Substitutions-Permutations-Netzwerk: Beispiel AES



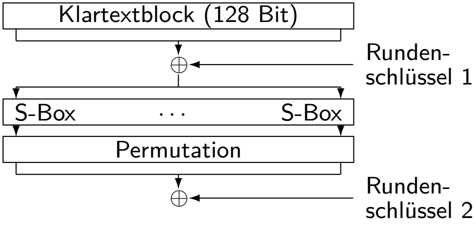
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert

Substitutions-Permutations-Netzwerk: Beispiel AES



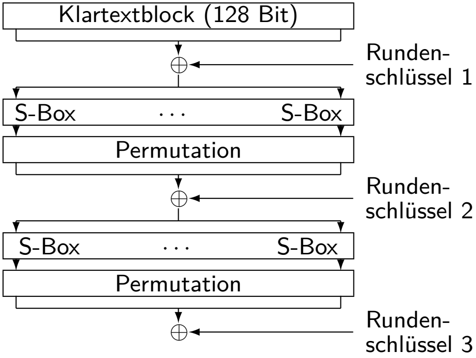
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert

Substitutions-Permutations-Netzwerk: Beispiel AES

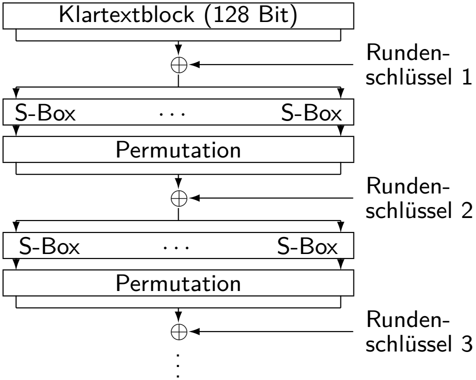


1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso

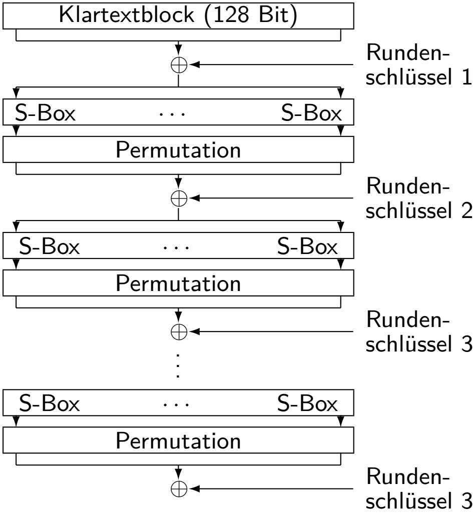
Substitutions-Permutations-Netzwerk: Beispiel AES



1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \dots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso

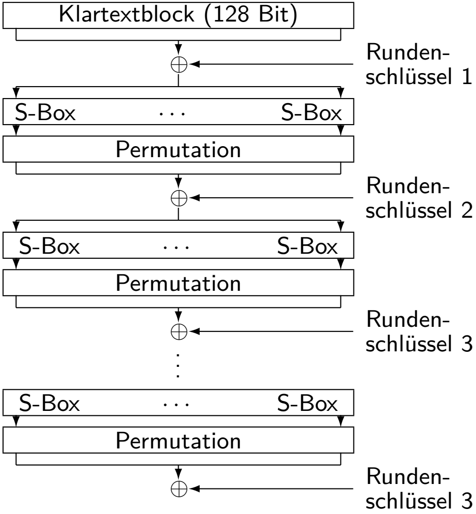


1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \dots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso



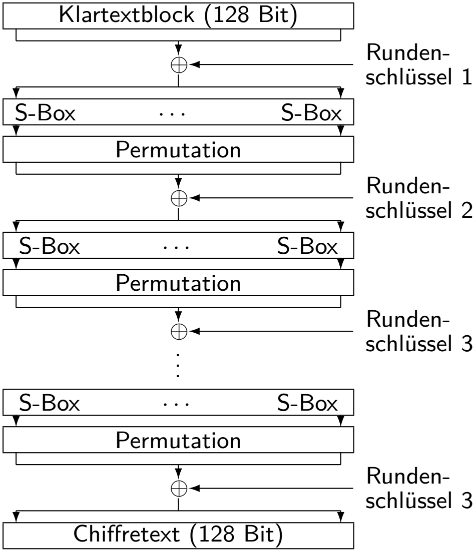
1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso

Substitutions-Permutations-Netzwerk: Beispiel AES



1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso
5) Nach Runde n erhalten wir den Chiffretext

Substitutions-Permutations-Netzwerk: Beispiel AES



1) Zunächst werden aus dem Schlüssel $k \in \{0,1\}^{128}$ n Rundenschlüssel $r_1, \ldots, r_n \in \{0,1\}^{128}$ abgeleitet
2) In Runde 1 wird r_1 auf den Klartext addiert
3) In Runde 2 werden zunächst Substitution und Permutation angewandt und danach r_2 addiert
4) Runden 3 bis n laufen genauso
5) Nach Runde n erhalten wir den Chiffretext

Beide Grundbausteine (Permutation, Substitution) werden benötigt:

- Lassen wir Substitutionen weg, dann ist die Blockchiffre eine lineaere Abbildung
    - Angreifer will aus Ciphertexten $c_1, \ldots, c_n$ Klartexte $m_1, \ldots, m_n$bestimmen
    - Es gilt

$$c_1 = \text{enc}(k, m_1), c_2 = \text{enc}(k, m_2), \ldots, c_n = \text{enc}(k, m_n) \tag{2}$$

      mit Unbekannten $k, m_1, \ldots, m_n$
    - Ist enc eine lineare Abbildung, so ist (**??**) ein lineares Gleichungssystem (Lineare Gleichungssysteme sind effizient lösbar)

Beide Grundbausteine (Permutation, Substitution) werden benötigt:

- Lassen wir Permutationen weg, dann gilt:
  - Bit 1 – 16 des Geheimtextes hängen nur von Bit 1 – 16 des Klartextblocks ab
  - Bit 17 – 32 des Geheimtextes hängen nur von Bit 17 – 32 der Klartextblocks ab
  - usw.
- Damit sind statistische Angriffe möglich
  (vgl. Sicherheitsbetrachtung monoalphabetischer Verfahren, Folie **??**)

- Blockchiffren verschlüsseln zunächst nur Klartexte einer bestimmten Blockgröße

$$\mathsf{enc} : \underbrace{\{0,1\}^l}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^n}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^n}_{\text{Chiffrate}}$$

(Schlüssellänge $l \geq 100$, Blockgröße heute meist $n = 128$)

- Blockchiffren verschlüsseln zunächst nur Klartexte einer bestimmten Blockgröße

$$\mathrm{enc} : \underbrace{\{0,1\}^l}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^n}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^n}_{\text{Chiffrate}}$$

  (Schlüssellänge $l \geq 100$, Blockgröße heute meist $n = 128$)
- Für die Verschlüsselung längerer Klartexte zerlegen wir den Klartext in Blöcke:

- Blockchiffren verschlüsseln zunächst nur Klartexte einer bestimmten Blockgröße

$$\text{enc}: \underbrace{\{0,1\}^l}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^n}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^n}_{\text{Chiffrate}}$$

  (Schlüssellänge $l \geq 100$, Blockgröße heute meist $n = 128$)

- Für die Verschlüsselung längerer Klartexte zerlegen wir den Klartext in Blöcke:

$$\underbrace{011000\cdots\cdots\cdots}_{\text{Block 1 (128 Bit)}}\ \underbrace{\cdots\cdots\cdots\cdots}_{\text{Block 2 (128 Bit)}}\ \cdots\cdots\cdots\cdots 1011110$$

- Blockchiffren verschlüsseln zunächst nur Klartexte einer bestimmten Blockgröße

$$\text{enc} : \underbrace{\{0,1\}^l}_{\text{Schlüssel}} \times \underbrace{\{0,1\}^n}_{\text{Klartexte}} \longrightarrow \underbrace{\{0,1\}^n}_{\text{Chiffrate}}$$

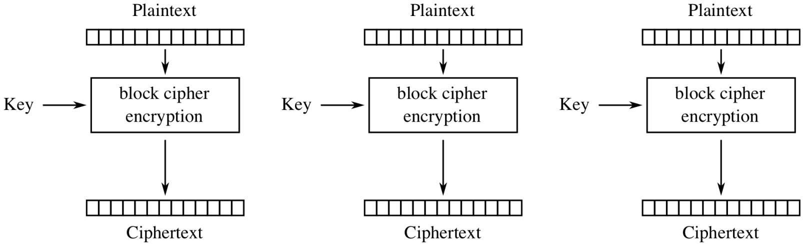
  (Schlüssellänge $l \geq 100$, Blockgröße heute meist $n = 128$)
- Für die Verschlüsselung längerer Klartexte zerlegen wir den Klartext in Blöcke:

$$\underbrace{011000\cdots\cdots\cdots}_{\text{Block 1 (128 Bit)}} \; \underbrace{\cdots\cdots\cdots\cdots}_{\text{Block 2 (128 Bit)}} \; \cdots\cdots\cdots \underbrace{1011110\text{PADDING}}_{\text{Block s (128 Bit)}}$$

- Sollte der letzte Block nicht die Länge 128 haben, füllen wir entsprechend auf

Einfachste Betriebsart

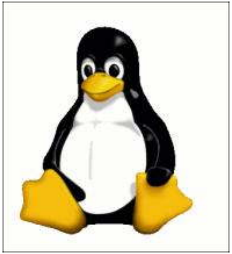
- Der Klartext wird, wie auf Folie **??** erläutert, in Blöcke der benötigten Blocklänge zerlegt und Block für Block verschlüsselt



Electronic Codebook (ECB) mode encryption

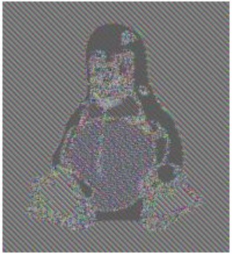
Probleme der Betriebsart ECB

- Gleiche Klartextblöcke werden zu gleichen Chiffretexten verschlüsselt
- Ein Angreifer kann so die Struktur des Klartextes erkennen



Klartext

AES im ECB Mode



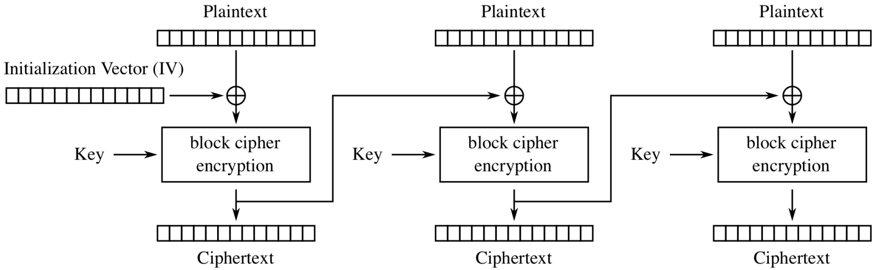
Chiffretext

ECB ist nicht sicher

- Lösung:
  - Der Chiffretext sollte nicht nur von Schlüssel und Klartext abhängen,
  - sondern von einem weiteren Parameter
- Wir lernen zwei solcher Betriebsarten näher kennen
  - Cipher Block Chaining Mode (CBC)
  - Counter Mode (Ctr)
- Weitere bekannte Betreibsarten sind:
  - Cipher Feedback Mode (CFB)
  - Output Feedback Mode (OFB)

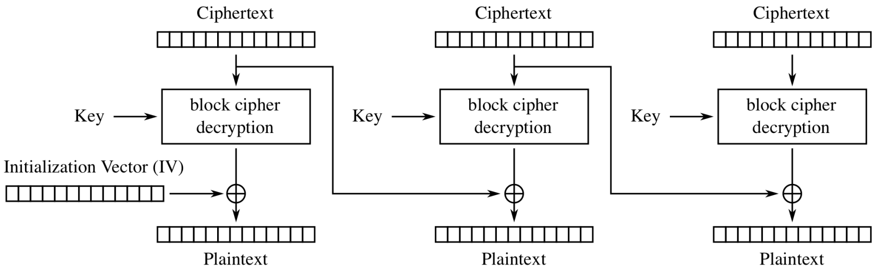
Weiterer Parameter: Vorheriger Chiffretext

- Der $i-$te Klartext (Plaintext) m_i wird mit dem vorhergehenden Ciphertext c_{i-1} addiert und anschließend verschlüsselt: $c_i = \text{enc}(k, m_i \oplus c_{i-1})$
- Für den ersten Block wird ein Initialisierungsvektor benötigt



Cipher Block Chaining (CBC) mode encryption

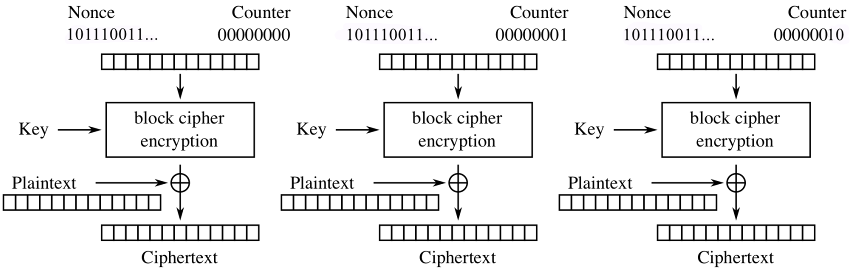
- Für die Entschlüsselung wird der $i-$te Ciphertext c_i entschlüsselt und dann mit dem $(i-1)-$ten Ciphertext c_{i-1} addiert: $m_i = \text{dec}(k, c_i) \oplus c_{i+1}$
- Wegen $c_i = \text{enc}(k, m_i \oplus c_{i-1})$ erhalten wir tatsächlich den Klartext m_i



Cipher Block Chaining (CBC) mode decryption

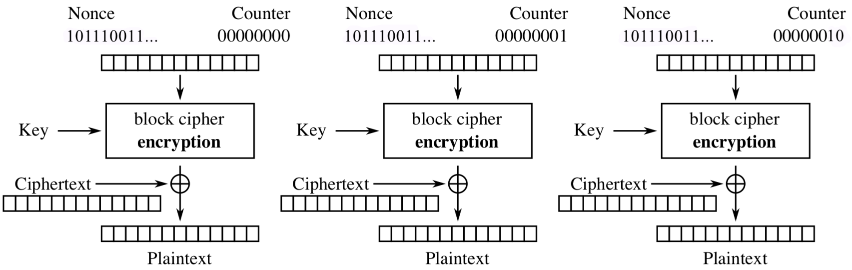
Weiterer Parameter: Zähler (Counter)

- Erzeuge Werte bestehend aus Nonce (Number Only Used Once) und Zähler
- Daraus wird ein Schlüsselstrom erzeugt und mit dem Klartext addiert
  (Die Blockchiffre wird also als Pseudozufallszahlengenerator genutzt)



Counter (CTR) mode encryption

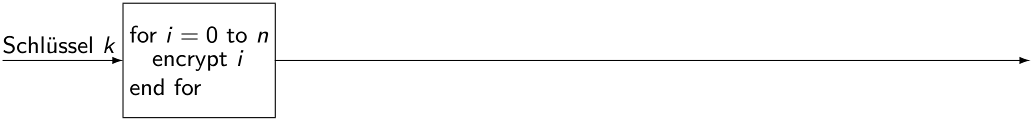
- Wie bei Stromchiffren üblich, erhalten wir Klartext zurück, indem wir den Schlüsselstrom auf den Ciphertext addieren
(Schlüsselstrom $\oplus$ Schlüsselstrom $= (0, 0, 0, \ldots))$



Counter (CTR) mode decryption

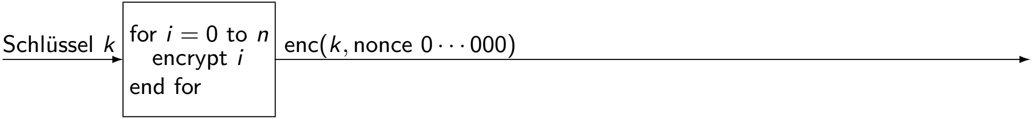
Sicherheit Counter Mode

Der Counter Mode nutzt die Blockchiffre als Pseudozufallszahlengenerator wie folgt:



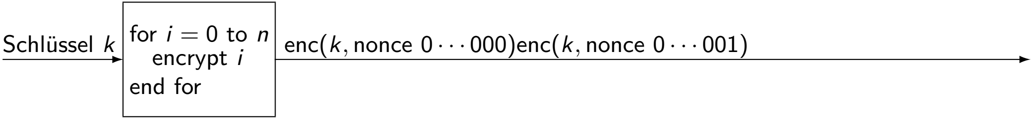
Sicherheit Counter Mode

Der Counter Mode nutzt die Blockchiffre als Pseudozufallszahlengenerator wie folgt:



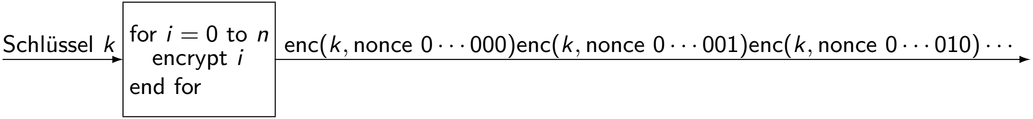
Sicherheit Counter Mode

Der Counter Mode nutzt die Blockchiffre als Pseudozufallszahlengenerator wie folgt:



Sicherheit Counter Mode

Der Counter Mode nutzt die Blockchiffre als Pseudozufallszahlengenerator wie folgt:



Sicherheit Counter Mode

Der Counter Mode nutzt die Blockchiffre als Pseudozufallszahlengenerator wie folgt:

Schlüssel k →

for $i = 0$ to n
  encrypt i
end for

→ enc(k, nonce $0\cdots000$)enc(k, nonce $0\cdots001$)enc(k, nonce $0\cdots010$)$\cdots$

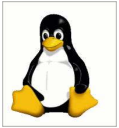
- Auf Folie **??** haben wir zwei Bedingungen für die Sicherheit formuliert:
  - Aus Teilen des Schlüsselstroms dürfen keine Nachfolger bestimmt werden können
  - Aus Teilen des Schlüsselstroms dürfen keine Vorgänger bestimmt werden können
- Wenn die Blockchiffre sicher ist (aus bekanntem Chiffetext kann der Schlüssel nicht berechnet werden), dann sind diese beiden Bedingungen erfüllt (Hausaufgabe!)

NONCE: Number Only Used Once

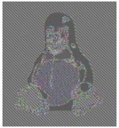
- Die Nonce darf bei gleichem Schlüssel nicht mehrfach verwendet werden
  - Ansonsten wird der selbe Schlüsselstrom generiert und der Angreifer kann erkennen, ob die selben Klartexte verschlüsselt wurden
- Die Nonce muss nicht geheim gehalten werden
  Sicherheit liegt allein an der Geheimhaltung des Schlüssels

ECB versus CBC und CTR

- ECB: Ciphertext lässt Struktur des Klartextes erkennen (Folie **??**):



ECB Mode

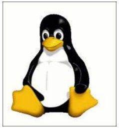


Klartext

Chiffretext

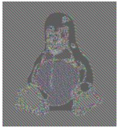
ECB versus CBC und CTR

- ECB: Ciphertext lässt Struktur des Klartextes erkennen (Folie **??**):



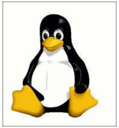
Klartext

ECB Mode $\longrightarrow$



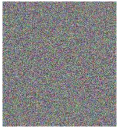
Chiffretext

- Nutzung zusätzlicher Parameter (vorheriger Ciphertext (CBC), Zähler (CTR)):



Klartext

CBC, CTR $\longrightarrow$



Chiffretext

Ziel: Integritätsschutz (erkennen, ob Daten verändert wurden)

Hashfunktionen bilden Bitstrings beliebiger Länge auf Bitstrings festen Länge ab

- Hashfunktion: $H : \{0,1\}^* \longrightarrow \{0,1\}^l$
- Menge der Bitstrings beliebiger Länge: $\{0,1\}^*$ (Definitionsbereich von H)
- Menge der Bitstrings der Länge $n \in \mathbb{N}$: $\{0,1\}^l$ (Bildbereich von H)

Wie bei Verschlüsselungsverfahren werden häufig große Datenmengen verarbeitet

- Die Berechnung muss effizient sein, d.h.
  für gegebenes $m \in \{0,1\}^*$ muss $H(m)$ schnell berechnet werden können

Hashfunktionen werden für viele kryptographische Verfahren benötigt:

- Integritätsschutz,
- Daten- und Instanzauthentisierung
- elektronische Signatur
- Pseudozufallszahlengeneratoren

Definition kryptographische Hashfunktion

Wir nennen eine Hashfunktion $H : \{0,1\}^* \longrightarrow \{0,1\}^l$ **kryptographisch stark**, wenn sie die folgenden zwei Bedingungen erfüllt:

Preimage resistance: Für gegebenes $h \in \{0,1\}^l$ ist es praktisch nicht möglich einen Wert $m \in \{0,1\}^*$ mit $H(m) = h$ zu finden.

Collision resistance: Es ist praktisch nicht möglich zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt.

Preimage resistance: Für gegebenes $h \in \{0,1\}^l$ ist es praktisch nicht möglich einen Wert $m \in \{0,1\}^*$ mit $H(m) = h$ zu finden.

Collision resistance: Es ist praktisch nicht möglich zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt.

Praktisch unmöglich:

- Bei einem geforderten Sicherheitsniveau von 100 Bit:
  Ein Angreifer benötigt ca. 2^{100} Versuche
    - einen Wert m mit $H(m) = h$ zu finden, oder
    - zwei Werte $m' \neq m$ mit $H(m) = H(m')$ zu finden

Preimage resistance: Für gegebenes $h \in \{0,1\}^l$ ist es praktisch nicht möglich einen Wert $m \in \{0,1\}^*$ mit $H(m) = h$ zu finden.

H soll effizient berechenbar sein, die Umkehrabbildung

$$(\text{finde zu } h \in \{0,1\}^l \text{ Urbild } m \in \{0,1\}^* \text{ mit } H(m) = h)$$

aber praktisch nicht berechenbar sein

Wir nennen solche Abbildungen auch Einwegfunktionen

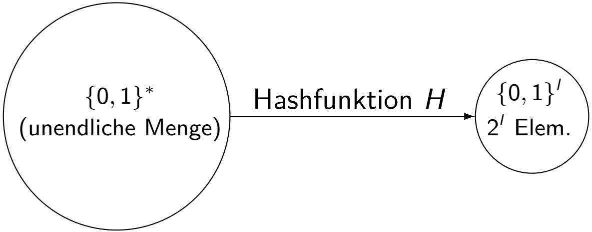
Collision resistance: Es ist praktisch nicht möglich zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt.

Zwei Werte $m, m' \in \{0,1\}^*$ mit $H(m) = H(m')$ nennen wir eine Kollision von H

Collision resistance: Es ist praktisch nicht möglich zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt.

Zwei Werte $m, m' \in \{0,1\}^*$ mit $H(m) = H(m')$ nennen wir eine Kollision von H

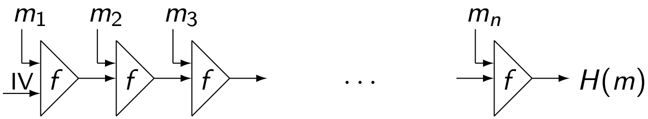
- Kollisionen lassen sich nicht vermeiden:
  - H bildet $\{0,1\}^*$ (unendliche Menge) auf $\{0,1\}^l$ (endliche Menge, 2^l Elemente) ab
  - Damit muss es (sogar unendlich viele) Kollisionen geben



Zwei Werte $m, m' \in \{0,1\}^*$ mit $H(m) = H(m')$ nennen wir eine Kollision von H

- Möglicher Angriff (Brute Force):
  - Wähle n zufällige Werte $m_1, \ldots, m_n \in \{0,1\}^*$
  - Prüfe, ob darunter zwei Werte sind, die eine Kollision von H bilden
- Praktisch nicht möglich (d.h. Sicherheitsniveau ≥ 100 Bit) bedeutet:
  - Ein Angreifer benötigt $\geq 2^{100}$ Versuche, bis er eine Kollision gefunden hat (mit hoher Wahrscheinlichkeit)
  - d.h. muss mindestens 2^{100} Werte $m_1, \ldots, m_{2^{100}}$ wählen
- Um diesen Angriff auszuschließen, muss der Bildbereich von H groß sein
  - Mindestens 2^{200} Elemente, d.h. $H : \{0,1\}^* \longrightarrow \{0,1\}^l$ mit $l \geq 200$ (Beweis in der Vorlesung Kryptologie)

Merkle-Damgard-Konstruktion:



- Nutze eine Kompressionsfunktion $f : \{0,1\}^l \times \{0,1\}^l \longrightarrow \{0,1\}^l$
  Diese muss sowohl eine Einwegfunktion, als auch kollisionsresistent sein
- Die Nachricht m wird in Blöcke $m_1, \ldots, m_n$ der Länge l aufgeteilt
- Jeder Block wird zusammen mit dem vorherigen Ergebnis mittels f verarbeitet
- Für den ersten Schritt wird ein Initialsierungsvektor (IV) benötigt

Beispiel für sichere Kompressionsfunktionen:

- Basierend auf einer Blockchiffre enc:
  - Im ersten Schritt: $f_1 := f(\mathrm{IV}, m_1) = \mathrm{enc}(IV, m_1) \oplus m_1$
  - Im zweiten Schritt: $f_2 := f(f_1, m_2) = \mathrm{enc}(f_1, m_2) \oplus m_2$
  - $\dots$
  - Im letzten Schritt: $H(m) = f_n := f(f_{n-1}, m_n) = \mathrm{enc}(f_{n-1}, m_n) \oplus m_n$

Ziel von Hashfunktionen: Integritätsschutz

- Speichern von Nachricht m und Hashwert $H(m)$ (auf verschiedenen Systemen)
- Prüfung, ob Nachricht m verändert wurde: Berechne Hashwert und vergleiche
- Hierfür wichtig: Kollisionsresistenz
  Ein Angreifer darf keine zweite Nachricht mit dem selben Hashwert erzeugen

Einsatzbeispiele:

- Download von Software, z.B. OpenOffice von der Seite
  `http://www.openoffice.org/de/downloads/index.html`
  - SHA256-Hashwert für Apache_OpenOffice_4.1.1_MacOS_x86-64_install_de.dmg:
    b905925d0d5bfb22e65910031cc1a17efce29498c4408a9deb368825ae8b236c

Es lässt sich aber nicht feststellen, wer die Nachricht erzeugt hat

- Hashfunktionen sind öffentlich
- Es wird kein kryptographischer Schlüssel verwendet

Hashfunktionen erfüllen also nicht das Schutzziel Datenauthentizität

Schutzziel: Datenauthentizität
(B kann zweifelsfrei feststellen, dass die Nachricht von A stammt)

$\boxed{A}$ Schlüssel: k		$\boxed{B}$ Schlüssel: k
compute Prüfsumme $t := \mathrm{mac}(k, m)$		
	$\xrightarrow{m,t}$	
		compute $t' := \mathrm{mac}(k, m)$ if $t' = t$ then accept else reject

Schutzziel: Datenauthentizität

- Nur die Inhaber des Schlüssels k können korrekte Prüfsummen berechnen
- Erfüllen also das Schutzziel Datenauthentizität (wenn nur A und B den Schlüssel kennen)
- Erfüllen nicht das Schutzziel Nichtabstreitbarkeit
  - Beide kennen Schlüssel k, also können auch beide Prüfsumme berechnen
  - Gegenüber Dritten (z.B. einem Richter) kann B also nicht beweisen, dass nicht er, sondern A die Prüfsumme berechnet hat
  - Hierfür benötigen wir Signaturen (asymmetrische Verfahren), siehe Folie **??**

Wie schon Verschlüsselungsverfahren und Hashfunktionen, müssen auch MAC-Verfahren zum Teil sehr große Datenmengen effizient verarbeiten können.
Heutige Verfahren basieren auf

- Blockchiffren (z.B. CBC-MAC, XOR-MAC)
- Kryptographisch starken Hashfunktionen (z.B. HMAC)

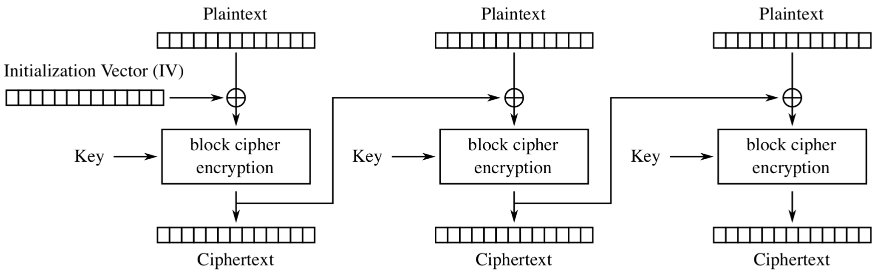
Cipher Block Chaining Message Authentication Code (CBC-MAC)

- Basiert auf einer sicheren Blockchiffre
- Nachricht m wird zunächst in Blöcke $m_1, \ldots, m_n$ zerlegt

Cipher Block Chaining Message Authentication Code (CBC-MAC)

- Basiert auf einer sicheren Blockchiffre
- Nachricht m wird zunächst in Blöcke $m_1, \ldots, m_n$ zerlegt

Erinnerung Betriebsmode Cipher Block Chaining (CBC):

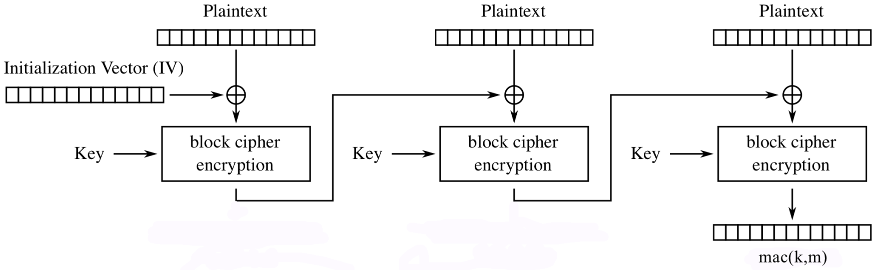


Cipher Block Chaining (CBC) mode encryption

Cipher Block Chaining Message Authentication Code (CBC-MAC)

- Basiert auf einer sicheren Blockchiffre
- Nachricht m wird zunächst in Blöcke $m_1, \ldots, m_n$ zerlegt

Prüfsumme ist der letzte Ciphertext



Cipher Block Chaining Message Authentication Code (CBC-MAC)

MAC-Verfahren basierend auf Hashfunktionen $H : \{0,1\}^* \longrightarrow \{0,1\}^l$

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

Dabei sind

- $opad := 0x5C \cdots 0x5C$
- $ipad := 0x36 \cdots 0x36$

Konstanten (die Sicherheit hängt nicht von der konkreten Wahl der Konstanten ab)

Ziel: Sicherer Kanal zwischen Alice und Bob (vertraulich und authentisch)

- Mehrere Umsetzungen denkbar:
  1) Verschlüsseln des Klartextes und MAC über den Geheimtext (z.B. bei IPsec)
  2) MAC über den Klartext und danach Verschlüsseln des Klartextes (z.B. bei SSH)
  3) MAC über den Klartext und Verschlüsseln von Klartext und MAC (z.B. bei SSL)
- Nur Umsetzung 1) ist sicher

Erst Klartext verschlüsseln, dann MAC über den Geheimtext

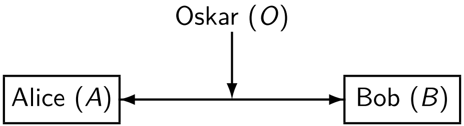
- Vertraulichkeit ist ein anderes Schutzziel als Datenauthentizität: MAC-Verfahren müssen nicht die Vertraulichkeit garantieren
- Weiterverarbeitung von Daten nur, wenn Sender bekannt ist: Verhindert z.B. Entschlüsselung von Schadsoftware

Weitere Bedingung:

- Für Verschlüsselung und MAC sollten verschiedene Schlüssel genutzt werden
  - Wesentliches kryptographisches Grundprinzip: Trenne wo du trennen kannst
  - Wird der Schlüssel für ein Verfahren kompromittiert, ist nicht automatisch das zweite Verfahren betroffen

Instanzauthentisierung

Schutzziel Instanzauthentisierung: B kann die Identität von A zweifelsfrei feststellen



Verfahren zur Instanzauthentisierung sind:

- Nachweis eines Geheimnisses (z.B. Benutzername/Passwort)
- Nachweis von Eigenschaften (z.B. biometrische Merkmale wie Fingerabdruck)
- Nachweis eines Geheimnisses, ohne dieses offen legen zu müssen (Challenge-Response Verfahren, siehe nächste Folie)

Wir widmen diesem Aspekt ein eigenes Kapitel (ab Folie **??**)

Ziel: Nachweis eines Geheimnisses (hier Schlüssel k), ohne dieses offen zu legen
Als Basis dienen Message Authentication Codes:

$\boxed{A}$ Schlüssel: k		$\boxed{B}$ Schlüssel: k
choose random c (challenge)	$\xrightarrow{c}$	
	$\xleftarrow{r}$	compute $r := \mathrm{mac}(k, c)$ (response)
compute $r' := \mathrm{mac}(k, c)$ if $r' = r$ then accept else reject		

Ziel: Nachweis eines Geheimnisses (hier Schlüsssel k), ohne dieses offen zu legen

- Nur die Inhaber des Schlüssels k können
  - aus der Challenge c
  - eine korrekte Response r

  berechnen
- Erfüllen also das Schutzziel Instanzauthentizität
  (wenn nur A und B den Schlüssel kennen)

Bei symmetrischen Verfahren wird immer der selbe Schlüssel genutzt

- Schlüssel zum Ver- und Entschlüsseln sind gleich
- Schlüssel für Berechnung und Verifikation von Prüfsummen sind gleich
- Schlüssel zur Berechnung und Verifikation der Response sind gleich

Bei asymmetrischen Verfahren gibt es ein Schlüsselpaar (pk, sk)
pk : Public Key (ist öffentlich), sk : Secret Key (ist geheim)

- Verschlüsselung mit pk: jeder kann verschlüsseln
  Entschlüsselung mit sk: nur Inhaber des geheimen Schlüssels kann entschlüsseln
- Berechnung der Prüfsumme mit sk: nur Inhaber von sk kann berechnen
  Verifikation der Prüfsumme mit pk: jeder kann prüfen

Grundidee asymmetrischer Verfahren: Einwegfunktionen mit Falltür

- Einwegfunktion f (Begriff im Abschnitt Hashfunktionen kennen gelernt):
  - f ist effizient berechenbar
  - Umkehrfunktion von f ist praktisch nicht berechenbar
- Falltür:
  - Mit zusätzlichem Wissen lässt sich auch die Umkehrfunktion effizient berechnen

Kandidaten sind:

- Faktorisierungsproblem großer natürlicher Zahlen
  Multiplikation ist deutlich einfacher als Faktorisieren
- Das diskrete Logarithmusproblem
  Potenzieren ist deutlich einfacher als Logarithmus

RSA-Verfahren

- publiziert 1977
- benannt nach Rivest, Shamir und Adleman
- gilt als das erste bekannte asymmetrische Verfahren

Sicherheit beruht auf der vermuteten Schwierigkeit des Faktorisierens großer Zahlen

Faktorisierungsproblem:
Gegeben: Zusammengesetzte natürliche Zahl $n = p \cdot q \in \mathbb{N}$, p, q Primzahlen.
Lösung: Finde die beiden Primfaktoren p und q.

Für das RSA-Verfahren benötigen wir einige mathematische Grundlagen

- Die Menge $\mathbb{Z}_n$ besteht aus allen natürlichen Zahlen $< n$
  $\mathbb{Z}_n := \{0, 1, 2, \ldots, n-1\}$
- Wir rechnen in dieser Menge wie folgt:
  - Für eine natürliche Zahl $t \in \mathbb{N}$ sei $t \bmod n$ diejenige eindeutige Zahl aus $\mathbb{Z}_n$, die aus t durch (ev. mehrmaliges) Subtrahieren von n entsteht
  - Beispiele: $6 \bmod 4 = 2$ $(6 - 4 = 2)$, $10 \bmod 4 = 2$ $(10 - 2 \cdot 4 = 2)$
  - Für Addition und Multiplikation in $\mathbb{Z}_n$ gilt nun:

$$\begin{aligned} a + b &:= a + b \bmod n \\ a \cdot b &:= a \cdot b \bmod n \end{aligned}$$

  - Beispiel in $\mathbb{Z}_4 = \{0, 1, 2, 3\} : 2 + 3 \bmod 4 = 5 \bmod 4 = 1$

Weitere Eigenschaften der Menge $\mathbb{Z}_n$:
Inverse Elemente bzgl. Addition und Multiplikation in $\mathbb{Z}_n$

- $a \in \mathbb{Z}_n$ heißt additiv invers (modulo n) zu $b \in \mathbb{Z}_n$, wenn

$$a + b = 0 \bmod n$$

- $a \in \mathbb{Z}_n$ heißt multiplikativ invers (modulo n) zu $b \in \mathbb{Z}_n$, wenn

$$a \cdot b = 1 \bmod n$$

Weitere Eigenschaften der Menge $\mathbb{Z}_n$:
Inverse Elemente bzgl. Addition und Multiplikation in $\mathbb{Z}_n$

- $a \in \mathbb{Z}_n$ heißt additiv invers (modulo n) zu $b \in \mathbb{Z}_n$, wenn

$$a + b = 0 \bmod n$$

- $a \in \mathbb{Z}_n$ heißt multiplikativ invers (modulo n) zu $b \in \mathbb{Z}_n$, wenn

$$a \cdot b = 1 \bmod n$$

- Beispiele in $\mathbb{Z}_4$:
  - $3 + 1 = 4 = 0 \bmod 4$ (1 ist additiv invers zu 3)
  - $3 \cdot 3 = 9 = 1 \bmod 4$ (3 ist multiplikativ invers zu sich selbst)

Welche Elemente in $\mathbb{Z}_n$ haben Inverse?

- Bzgl. Addition alle Elemente: für alle $a \in \mathbb{Z}_n$ gilt mit $b = n - a$:

$$a + b = a + (n - a) = n = 0 \bmod n$$

Welche Elemente in $\mathbb{Z}_n$ haben Inverse?

- Bzgl. Addition alle Elemente: für alle $a \in \mathbb{Z}_n$ gilt mit $b = n - a$:

$$a + b = a + (n - a) = n = 0 \bmod n$$

- Bzgl. Multiplikation nicht alle Elemente: z.B. in $\mathbb{Z}_4$
  - 1 ist zu sich selbst invers ($1 \cdot 1 = 1$)
  - 3 ist zu sich selbst invers (hatten wir schon)
  - zu 2 gibt es kein inverses Element
    ($2 \cdot 1 = 2 \neq 1$, $2 \cdot 2 = 4 = 0 \bmod 4 \neq 1$, $2 \cdot 3 = 6 = 2 \bmod 4 \neq 1$)

Welche Elemente in $\mathbb{Z}_n$ haben multiplikativ Inverse?

- Für $a \in \mathbb{N}$ heißt $t \in \mathbb{N}$ *Teiler* von a (in Zeichen $t|a$), wenn $\frac{a}{t} \in \mathbb{N}$
  $1, 2, 3, 4, 6$ sind Teiler von 12
- $t \in \mathbb{N}$ heißt *gemeinsamer Teiler* von $a, b \in \mathbb{N}$, wenn $t|a$ und $t|b$
  $1, 2, 4$ sind gemeinsame Teiler von 12 und 20
- $a, b \in \mathbb{N}$ heißen *teilerfremd*, wenn 1 einziger gemeinsamer Teiler von a und b ist
  8 und 15 sind teilerfrend

Welche Elemente in $\mathbb{Z}_n$ haben multiplikativ Inverse?

- Für $a \in \mathbb{N}$ heißt $t \in \mathbb{N}$ *Teiler* von a (in Zeichen $t|a$), wenn $\frac{a}{t} \in \mathbb{N}$
  $1, 2, 3, 4, 6$ sind Teiler von 12
- $t \in \mathbb{N}$ heißt *gemeinsamer Teiler* von $a, b \in \mathbb{N}$, wenn $t|a$ und $t|b$
  $1, 2, 4$ sind gemeinsame Teiler von 12 und 20
- $a, b \in \mathbb{N}$ heißen *teilerfremd*, wenn 1 einziger gemeinsamer Teiler von a und b ist
  8 und 15 sind teilerfrend

Ein Element $a \in \mathbb{Z}_n$ besitzt genau dann ein multiplikativ Inverses modulo n, wenn a und n teilerfremd sind

Welche Elemente in $\mathbb{Z}_n$ haben multiplikativ Inverse?

- Für $a \in \mathbb{N}$ heißt $t \in \mathbb{N}$ *Teiler* von a (in Zeichen $t|a$), wenn $\frac{a}{t} \in \mathbb{N}$
  $1, 2, 3, 4, 6$ sind Teiler von 12
- $t \in \mathbb{N}$ heißt *gemeinsamer Teiler* von $a, b \in \mathbb{N}$, wenn $t|a$ und $t|b$
  $1, 2, 4$ sind gemeinsame Teiler von 12 und 20
- $a, b \in \mathbb{N}$ heißen *teilerfremd*, wenn 1 einziger gemeinsamer Teiler von a und b ist
  8 und 15 sind teilerfrend

Ein Element $a \in \mathbb{Z}_n$ besitzt genau dann ein multiplikativ Inverses modulo n, wenn a und n teilerfremd sind

Vgl. auch Untersuchung von $\mathbb{Z}_4$ auf Folie **??**:

- 1 und 3 teilerfremd zu 4
- 2 ist nicht teilerfremd zu 4

Schlüsselgenerierung:

- Wähle zwei Primzahlen: p, q. Wir nennen $n = p \cdot q$ das *Modul*
- $\phi(n) = (p-1) \cdot (q-1)$ heißt auch *Euler-Zahl* von n
- Verschlüsselungsschlüssel (pk): Wähle e teilerfremd zu $\phi(n)$
- Entschlüsselungsschlüssel (sk): Wähle d so, dass $e \cdot d = 1 \bmod \phi(n)$
  (Solch ein d existiert, da e teilerfremd zu $\phi(n)$ ist)

Verschlüsselung einer Nachricht $m \in \mathbb{Z}_n$

- Berechne $c = m^e \bmod n$

Entschlüsselung des Ciphertextes c

- Berechne $c^d \bmod n$

Verschlüsselung: $c = m^e \bmod n$, Entschlüsselung: $c^d \bmod n$

- Warum gilt $m = c^d \bmod n$
  - Hängt mit der Wahl von e und d zusammen: $e \cdot d = 1 \bmod \phi(n)$
  - Daher gilt (etwas unformal):

$$
\begin{array}{rcl}
c^d & = & (m^e \bmod n)^d \\
 & = & (m^e)^d \bmod n \\
 & = & m^{e \cdot d} \bmod n \\
 & (=) & m^{1 \bmod \phi(n)} \bmod n \\
 & (=) & m^1 \bmod n
\end{array}
$$

  - Gleichungen 4 und 5 benötigen einen formalen Beweis (siehe Vorlesung Kryptologie)

Schlüsselgenerierung:

- Primzahlen: $p = 3, q = 5$
- Modul: $n = p \cdot q = 15$, Eulerzahl: $\phi(n) = (p-1) \cdot (q-1) = 8$
- Verschlüsselungsschlüssel: $e = 3$ (ist teilerfremd zu $\phi(n)$)
- Entschlüsselungsschlüssel: $d = 3$ (es gilt $e \cdot d = 9 = 1 \bmod 8$)

Verschlüsselung der Nachricht $m = 7$:

- $c = m^e \bmod n = 7^3 \bmod 15 = 343 \bmod 15 = 13$

Entschlüsselung des Ciphertextes $c = 13$

- $c^d \bmod n = 13^3 \bmod 15 = 2197 \bmod 15 = 7.$

- Angreifer kennt
  - den öffentlichen Verschlüsselungsschlüssel e und
  - das Modul n (also die Menge $\mathbb{Z}_n$, in der gerechnet wird)
- Ziel des Angreifers: Bestimme den Klartext m zum Ciphertext $c = m^e \bmod n$
- Einziger Angriff heute: Bestimme den geheimen Entschlüsselungsschlüssel d
  - Zusammenhang zwischen e und d: es gilt $e \cdot d = 1 \bmod \phi(n)$
    Erinnerung: Für $n = p \cdot q$, p, q Primzahlen, gilt $\phi(n) = (p-1) \cdot (q-1)$
- Aber: Der Angreifer kennt $\phi(n)$ nicht, kann also d nicht aus e berechnen
- Um $\phi(n)$ zu bestimmen, kann er versuchen, p, q zu berechnen (n faktorisieren)

Faktorisierungsproblem:
Gegeben: Zusammengesetzte natürliche Zahl $n = p \cdot q \in \mathbb{N}$, p, q Primzahlen.
Lösung: Finde die beiden Primfaktoren p und q.

- Erste Idee: Probedivision:

Faktorisierungsproblem:
Gegeben: Zusammengesetzte natürliche Zahl $n = p \cdot q \in \mathbb{N}$, p, q Primzahlen.
Lösung: Finde die beiden Primfaktoren p und q.

- Erste Idee: Probedivision:
  - for $i = 1$ to $\sqrt{n}$
  - if $n/i \in \mathbb{N}$ stopp
  - return $i, n/i$

Faktorisierungsproblem:
Gegeben: Zusammengesetzte natürliche Zahl $n = p \cdot q \in \mathbb{N}$, p, q Primzahlen.
Lösung: Finde die beiden Primfaktoren p und q.

- Erste Idee: Probedivision:
  - for $i = 1$ to $\sqrt{n}$
  - if $n/i \in \mathbb{N}$ stopp
  - return $i, n/i$
- Aber: Es gibt bessere Algorithmen zum Faktorisieren:
  - Für Sicherheitsniveau 100 Bit werden Primzahlen der Größe ca. 2^{1000} benötigt
  - Nachweis in der Vorlesung Kryptologie

Weitere bekannte Verfahren

- Rabin-Verfahren (basiert auch auf Schwierigkeit des Faktorisierungsproblems)
- ElGamal-Verfahren (basiert auf Schwierigkeit des Logarithmusproblems)
- Merkle-Hellman und McEliece (basieren auf Schwierigkeit anderer Probleme)

Nachteil von asymmetrischen Verschlüsselungsverfahren:

- Sind deutlich ineffizienter als symmetrische Verschlüsselungsverfahren
- Verwendung daher in der Praxis nur für den Austausch symmetrischer Schlüssel

Kombination von asymmetrischen (für Schlüsselaustausch) und symmetrischen (für die eigentliche Verschlüsselung) Verschlüsselungsverfahren

- A will B eine vertrauliche Nachricht m übermitteln
- B besitzt ein Schlüsselpaar (pk, sk) (z.B. für das RSA-Verfahren)
- A vertraut dem öffentlichen Schlüssel pk (weiß, dass dieser B gehört)

$\boxed{A}$ öffentlicher Schlüssel: $pk = e$		$\boxed{B}$ geheimer Schlüssel: $sk = d$
choose random key k compute $k' = k^e \bmod n$ and $c = \mathrm{enc}(k, m)$	$\xrightarrow{k', c}$	compute $k = k'^d \bmod n$ and $m = \mathrm{dec}(k, m)$

Schutzziel: Nichtabstreitbarkeit
(B kann auch gegenüber Dritten nachweisen, dass die Nachricht von A stammt)

$\boxed{A}$ geheimer Schlüssel: sk		$\boxed{B}$ öffentlicher Schlüssel: pk
compute Signatur $s := \mathrm{sig}(sk, m)$	$\xrightarrow{m,s}$	
		compute $b := \mathrm{verify}(pk, m, s)$ if $b =$ true then accept else reject

Wir benötigen Verfahren:

- sig: Signieren von Nachrichten m mit einem geheimen Schlüssel sk
- verify: Prüfen von Signaturen mit dem zugehörigen öffentlichen Schlüssel pk

Schlüsselgenerierung: (ähnlich dem RSA-Verschlüsselungsverfahren)

- Wähle zwei Primzahlen: p, q, setze $n = p \cdot q$ und $\phi(n) = (p-1) \cdot (q-1)$
- Verifikationsschlüssel (pk): Wähle e teilerfremd zu $\phi(n)$
- Signaturschlüssel (sk): Wähle d so, dass $e \cdot d = 1 \bmod \phi(n)$
- Zusätzlich benötigen wir eine Hashfunktion H

Schlüsselgenerierung: (ähnlich dem RSA-Verschlüsselungsverfahren)

- Wähle zwei Primzahlen: p, q, setze $n = p \cdot q$ und $\phi(n) = (p-1) \cdot (q-1)$
- Verifikationsschlüssel (pk): Wähle e teilerfremd zu $\phi(n)$
- Signaturschlüssel (sk): Wähle d so, dass $e \cdot d = 1 \bmod \phi(n)$
- Zusätzlich benötigen wir eine Hashfunktion H

Signatur einer Nachricht $m \in \mathbb{Z}_n$ (Signaturfunktion sig)

- Berechne $h = H(m)$
- Berechne $s = h^d \bmod n$ (Verschl. von $h = H(m)$ mit dem geheimen Schlüssel)

Schlüsselgenerierung: (ähnlich dem RSA-Verschlüsselungsverfahren)

- Wähle zwei Primzahlen: p, q, setze $n = p \cdot q$ und $\phi(n) = (p-1) \cdot (q-1)$
- Verifikationsschlüssel (pk): Wähle e teilerfremd zu $\phi(n)$
- Signaturschlüssel (sk): Wähle d so, dass $e \cdot d = 1 \bmod \phi(n)$
- Zusätzlich benötigen wir eine Hashfunktion H

Signatur einer Nachricht $m \in \mathbb{Z}_n$ (Signaturfunktion sig)

- Berechne $h = H(m)$
- Berechne $s = h^d \bmod n$ (Verschl. von $h = H(m)$ mit dem geheimen Schlüssel)

Signaturverifikation (Verifikationsfunktion verify)

- Berechne $h = H(m)$
- Berechne $v = s^e \bmod n$ (Entschl. von s mit dem öffentlichen Schlüssel)
- Wenn $h = v$, akzeptiere die Signatur, sonst nicht

Signaturverfahren erfüllen auch das Schutzziel Nichtabstreitbarkeit

- Die Signatur wird mit dem geheimen Schlüssel erzeugt
- Also kann nur der Inhaber des geheimen Schlüssels die Signatur berechnet haben

Weitere Signaturverfahren

- Digital Signature Algorithm (DSA), El-Gamal, Schnorr-Signatur: basieren auf dem diskreten Logarithmusproblem
- Merkle-Signatur: basiert auf Hashbäume
- McEliece-Niederreiter-Signatur: basiert auf lineare Codes

Ziel: Nachweis eines Geheimnisses (hier Schlüssel sk), ohne dieses offen zu legen
Als Basis können auch Signaturverfahren dienen
(ein symmetrisches Verfahren haben wir bereits auf Folie **??** gesehen)

$\boxed{A}$ Bs öff. Schlüssel: pk		$\boxed{B}$ Schlüsselpaar: (pk, sk)
choose random c		
(challenge)	$\xrightarrow{c}$	
		compute $r := \mathrm{sig}(sk, c)$
	$\xleftarrow{r}$	(response)
compute $b := \mathrm{verify}(pk, c, r)$		
if $b =$ true then accept		
else reject		

Zusammenfassung RSA

Ein RSA-Schlüsselpaar kann genutzt werden für

- Verschlüsselung
- Datenauthentisierung (Signaturverfahren)
- Instanzauthentisierung (Challenge-Response-Verfahren)

Für alle drei Verfahren müssen verschiedene Schlüssel eingesetzt werden

- Wichtiges kryptographisches Grundprinzip: Trenne wo du trennen kannst
- Es gib auch Angriffe, wenn die selben Schlüssel genutzt werden (nächste Folie)

Schlüssel für Daten- und Instanzauthentisierung müssen verschieden sein

Andernfalls ist folgender Angriff möglich:

- Angreifer erzeugt Nachricht m und bildet $H(m) = c$ (challenge)
- Angreifer fordert B zur Authentisierung auf und sendet c
- B berechnet Prüfsumme r
  (B sieht nicht, ob c Zufall oder der Hashwert einer Nachricht ist)
- Dem Angreifer liegt eine von B korrekt signierte Nachricht vor

Problem: Zuordnung eines Schlüssels zum Schlüsselinhaber
(wem gehört der Schlüssel)

- Beispiel 1: Verschlüsselung mit öffentlichem Schlüssel, Entschlüsselung mit geheimem Schlüssel:
    - Bei Nutzung des öffentlichen Schlüssels einer falschen Person kann diese die Nachricht entschlüsseln (Verlust der Vertraulichkeit)
- Beispiel 2: Signaturerzeugung mit geheimem Schlüssel, Verifikation mit öffentlichem Schlüssel:
    - Bei falscher Zuordnung des öffentlichen Schlüssels zu einer Person vertraue ich fälschlicherweise auf die Authentizität der Daten (Verlust der Datenauthentizität)

Wir unterschieden drei Vertrauensmodelle

- Direct Trust: Nutzer erhält den öff. Schlüssel direkt vom Schlüsselinhaber
    - kleine Virtual Private Networks
- Web of Trust: Nutzer signieren gegenseitig ihre öff. Schlüssel
    - PGP, GNU-PG
- Hierarchical Trust: Öff. Schüssel werden von einer zentralen Instanz verwaltet
    - Qualifizierte elektronische Zertifikate nach Signaturgesetz
    - Public Key Infrastruktur des Deutschen Forschungsnetzwerkes
    - Server-Authentisierung via https

Direct Trust ist nur für kleine Gruppen anwendbar
(Paarweiser Schlüsselaustausch notwendig)

- 2 Personen: ein Austausch
- 3 Personen: 3
- 4 Personen: 6
- n Personen: $n(n-1)/2$
- Kommt neuer Partner hinzu: Austausch mit n Personen

Idee: Nutzer signieren öffentliche Schlüssel andere Nutzer (und garantieren so für die Authentizität des Schlüssels).

- Beispiel (Alice, Bob und Carl):
  - Alice signiert Bobs öffentlichen Schlüssel
  - Bob signiert Carls öffentlichen Schlüssel
  - Alice vertraut Carls öffentlichem Schlüssel
- Warum vertraut Alice Carls Schlüssel?
  - Alice prüft mit Bobs öff. Schlüssel Bobs Signatur von Carls öff. Schlüssel (Ist aber nur dann sicher, wenn Alice Bob vertraut, nur authentische Schlüssel zu signieren, also vor der Signierung die Bindung zwischen Schlüssel und Schlüsselinhaber zu prüfen (z.B. über Direct Trust))

Jeder Nutzer hat einen Schlüsselbund (key ring) mit öff. Schlüsseln anderer Nutzer

- Jedem öffentlichen Schlüssel sind zugeordnet
  - Name des Schlüsselinhabers
  - Owner Trust (5 Stufen, Grad des Vertrauens in die Prüffähigkeit des Schlüsselinhabers)
  - Key Legitimacy (3 Stufen, Grad des Vertrauens in den öffentlichen Schlüssel) leitet sich ab aus den Owner Trusts der Signierer und der Anzahl der Signaturen
  - Signaturen des öffentlichen Schlüssels
- Vorgehen in RFC 2440 (OpenPGP Message Format) beschrieben

Web of Trust: Owner Trust

- Wert für Owner Trust legt jeder Nutzer selbst fest.
- Gibt an, wie der Schlüsselinhaber öffentliche Schlüssel anderer Nutzer prüft
- Fünf Level:
  - unbekannt (unknown) für Nutzer, über die man keine weiteren Informationen hat
  - kein Vertrauen (not trusted) für Nutzer, denen nicht vertraut wird
  - geringes Vertrauen (marginal) für Nutzer, denen nicht voll vertraut wird
  - volles Vertrauen (complete) für Nutzer, denen voll vertraut wird
  - absolutes Vertrauen (ultimate) für Nutzer, deren privater Schlüssel sich im privaten Schlüsselbund befindet (üblicherweise nur die eigenen privaten Schlüssel)

Nachteile:

- Einstufung (Owner Trust) erfordert hohes Wissen der Nutzer
- Die Signaturen sind juristisch nicht bindend (vgl. Signaturgesetz)
- Zurückziehen von Zertifikaten nicht einfach umsetzbar

Vorteile:

- Gegenüber Direct Trust eine deutliche Verbesserung
- Umgesetzt in PGP (Pretty Good Privacy) und GnuPG (Open Source)
- Es gibt zahlreiche Schlüsselserver, auf denen öffentliche Schlüssel und zugehörige Signaturen hochgeladen werden können
- Einige Institutionen bieten einen Certification Service für PGP- und GPG-Schlüssel an (z.B. c't)

Ziele:

- Zuordnung eines öffentlichen Schlüssels zum Schlüsselinhaber
- Festlegung der Schlüsselnutzung (Verschlüsselung, Authentisierung, Signatur)
- Etablierung einer gemeinsamen Sicherheitsinfrastruktur
  - Wie sicher sind die Schlüssel gelagert, erzeugt usw.
  - Wie stark ist die Bindung zwischen Schlüssel und Schlüsselinhaber (z.B. wie wird geprüft)

Hierarchical Trust

Public Key Infrastrukturen

- Öffentliche Schlüssel werden von einer vertrauenswürdigen Instanz verwaltet
- Nutzer erhalten Zertifikate C_i für ihren öffentlichen Schlüssel pk_i
- Vertrauensanker bildet der öffentliche Schlüssel pk_{CA} der CA

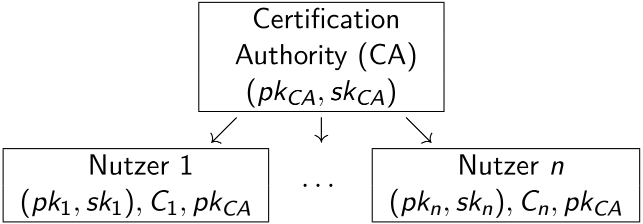


Abbildung : Schematischer Aufbau einer Public Key Infrastruktur

$C_i =$

Angaben zum Nutzer Name, Organisation, usw.
öffentlicher Schlüssel pk_i
Verwendeter Algorithmus z.B. RSA, DSA
Gültigkeitszeitraum
Angaben zur Certification Authority Name, Kontaktdaten usw.
Schlüsselnutzung z.B. Signatur, Authentisierung, Verschlüsselung
Signatur der CA (mit sk_{CA})

Abbildung : Zertifikat für Nutzer i (ausgestellt von der Certification Authority)

Public Key Infrastrukturen: Beispiel Signatur

$$
\begin{array}{ccc}
 & \boxed{\begin{array}{c}\text{Certification}\\ \text{Authority (CA)}\\ (pk_{CA}, sk_{CA})\end{array}} & \\
\swarrow & \downarrow & \searrow \\
\boxed{\begin{array}{c}\text{Nutzer } 1\\ (pk_1, sk_1), C_1, pk_{CA}\end{array}} & \dots & \boxed{\begin{array}{c}\text{Nutzer } n\\ (pk_n, sk_n), C_n, pk_{CA}\end{array}}
\end{array}
$$

Nutzer 1 signiert Dokument m, Nutzer 2 prüft Signatur

$\boxed{\text{Nutzer 1}}$ Schlüssel: $(pk_1, sk_1), C_1$		$\boxed{\text{Nutzer 2}}$ Schlüssel: pk_{CA}
compute $s := \mathrm{sig}(sk_1, m)$	$\xrightarrow{s, C_1, m}$	$\mathrm{verify}(pk_{CA}, C_1)$
		$\mathrm{verify}(pk_1, m, s)$

Häufig weitere Aufteilung:

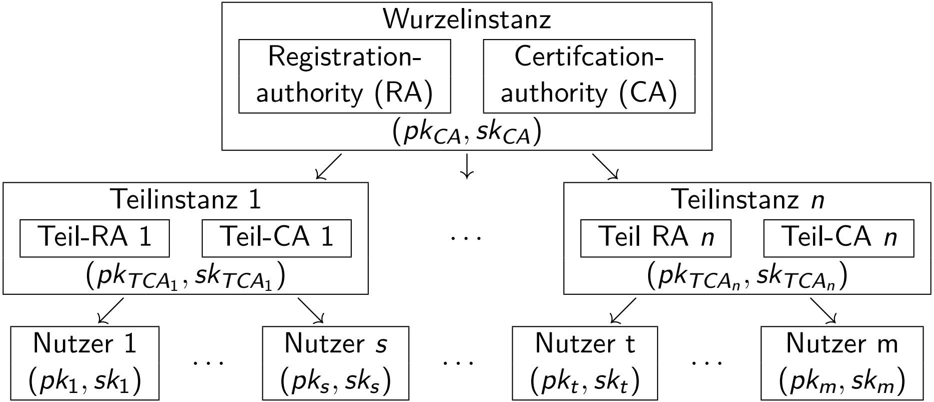


Abbildung : Schematische Darstellung einer 3-stufigen Public Key Infrastruktur

Aufgabenverteilung:

- Registration Authority
    - Legt Sicherheitsrichtlinien fest
    - Prüft Zertifizierungsanträge
    - Legt Inhalte der Zertifikate fest
    - Leitet Anträge an Certification Authority weiter
- Certification Authority:
    - Stellt die eigentlichen Zertifikate aus

Zurückziehen von Zertifikaten:

- Bei Sicherheitsvorfällen (z.B. Schlüsselkompromittierung, Alg. werden unsicher)
- Eine Instanz hält sich nicht an Sicherheitsvorgaben
- Hierzu: Führen von Certificate Revokation Lists (CRLs)
- Zusätzlich: Prüfung, ob Zertifikate in CRL enthalten ist

All Instanzen einer PKI (Root-CA, Teil-CAs, Endnutzer) müssen ein definiertes Maß an Sicherheitsstandards einhalten.
Sicherheitsvorgaben werden in zwei Dokumenten festgelegt:

- Certificate Policy: welche Sicherheitsvorgaben müssen eingehalten werden
- Certificate Practise Statement: wie werden die Sicherheitsvorgaben umgesetzt
- RFC 3647: Internet X.509 Public Key Infrastructure Certificate Policy an Certificate Practise Framework
  (beschreibt wird Aufbau und Inhalt beider Dokumente im Detail)

- Allgemeines
  - Teilnehmer der PKI
  - Zertifikatsnutzung
  - Genutzte Algorithmen
  - Schlüssellängen
  - Datenfelder in den Zertifikaten
- Initialisierung
  - Registrierung der Teilnehmer
  - Generierung der Schlüsselpaare
  - Generierung der Zertifikate
  - Schlüssel- und Zertifikatsverteilung
  - Schlüssel-Backup
- Nutzung
  - Zertifikatsneuausstellung
  - Zertifikatsvalidierung
  - Schlüsselupdate
  - Schlüssel-Recovery
- Aufhebung
  - Ablauf des Zertifikates
  - Zurückziehen des Zertifikates
  - Archivierung

Zertifikate müssen:

- Inhaber eindeutig identifizieren,
- Schlüsselnutzung festlegen,
- Ausstellende CA identifizieren
  (hieraus folgen u.a. die Sicherheitsvorgaben für die PKI)

Es existieren zwei Standards für Zertifikate:

- X509: Eingesetzt zur sicheren Kommunikation im Internet (https)
- Card Verifiable Certificates (cvc): Eingesetzt zur sicheren Kommunikation zwischen/zu Smartcards

X509-Zertifikate: Kodiert nach ASN.1 (Abstract Syntax Notation Nr. 1)

```
Certificate::= SEQUENCE {
   tbsCertificate       TBSCertificate,       TBS: To Be Signed
   signatureAlgorithm   AlgorithmIdentifier,  Signaturalgorithmus
   signatureValue       BITSTRING }           Signatur "uber TBSCert.
```

```
TBSCertificate::= SEQUENCE {
   version                     EXPLICIT Version DEFAULT v1,
   serialNumber                CertificateSerialNumber,
   signature                   AlgorithmIdentifier,
   issuer                      Name,
   validity                    Validity,
   subject                     Name,
   subjectPublicKeyInfo        SubjectPublicKeyInfo,
   extensions                  EXPLICIT Extensions OPTIONAL
                               --If present, version MUST be v3 }
```

```
Version::= INTEGER { v1(0), v2(1), v3(2) }
```

Aktuell v3, v1, v2 aber weiterhin nutzbar

```
CertificateSerialNumber::= INTEGER
```

Unterschiedlich für alle Zertifikate, die von einer CA (issuer) ausgestellt werden:
Meist Zähler oder Hashwert über Public Key.

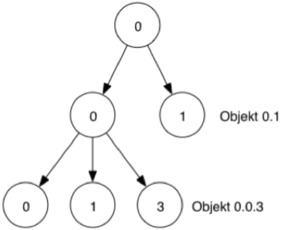
```
AlgorithmIdentifier::= SEQUENCE {
   algorithm        OBJECT IDENTIFIER,
   parameters       ANY DEFINED BY algorithm OPTIONAL }
```

Legt Signaturalgorithmus zum Signieren des Zertifikats über eine OID fest

Selber Wert wie unter `signatureAlgorithm`

Object Identifier (OID) bezeichnen Informationsobjekte (weltweit eindeutig)

- Eine OID ist ein Knoten in einem hierarchisch zugewiesenen Namensraum
  Folge von Nummern, die seine Position, beginnend an der Wurzel, angibt
- Drei Wurzeln: ITU-T (0), ISO (1), joint-iso-itu-t (2)
  - ITU-T: International Telecommunication Union-Telecommunication Standardization Sector)
  - ISO: International Organization for Standardization
- ISO/IEC 9834, DIN 66334: Regeln für Vergabe und Registrierung von OIDs



Beispiel:

- Signaturalgorithmus `sha1-with-rsa-signature`: 1.2.840.113549.1.1.5

  iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)

- Auflösen der OIDs in OID-Respositories:
  - Z.B.: `http://www.oid-info.com`

`issuer`, `subject` beschreiben Aussteller, Inhaber eindeutig über `Name`
Enthält Anzahl von Attributen, z.B.

- country: DE
- organization: HDA
- organizational unit: FBI
- common name: John Sixpack
- serial number: 1,2,...

```
Validity::= SEQUENCE {
   notBefore   Time,
   notAfter    Time }
```

Zeitraum, in dem der geheime Schlüssel genutzt werden kann

```
SubjectPublicKeyInfo::= SEQUENCE {
   algorithm            AlgorithmIdentifier,
   subjectPublicKey     BIT STRING }
```

Enthält den öffentlichen Schlüssel

Ab Version v3 sind `Extensions` erlaubt, z.B. für

- Informationen über Schlüsselnutzung
- Informationen über Sicherheitsrichtlinien (wo findet man CP und CPS)
- Erweiterte Attribute für issuer und subject:
  mögliche Kontaktdaten für Rückfragen (E-Mailadresse, Fax, Telefon)
- Einschränkungen des Zertifikatspfades, z.B. maximale Pfadkette, d.h. wie viele Zertifikate müssen maximal bis zum Root-Zertifikat geprüft werden

```
Extensions::= SEQUENCE SIZE (1..MAX) OF Extension

Extension::= SEQUENCE {
   extnID      OBJECT IDENTIFIER,
   critical    BOOLEAN DEFAULT FALSE,
   extnValue   OCTET STRING }
```

`extnID`: Identifier der Extension
`critical`: gibt an, ob eine bestimmte Angabe geprüft werden muss oder das Zertifikat auch ohne die Überprüfung dieser Angabe als gültig akzeptiert werden kann
`extnValue`: Inhalt der Extension

Extension Schlüsselnutzung (OID: 2.5.29.15)

joint-iso-itu-t(2) ds(5) certificateExtension(29) keyUsage(15)

```
KeyUsage::= BIT STRING {
   digitalSignature  (0),
   nonRepudiation    (1),
   keyEncipherment   (2),
   dataEncipherment  (3),
   keyAgreement      (4),
   keyCertSign       (5),
   cRLSign           (6),
   encipherOnly      (7),
   decipherOnly      (8) }
```