

Kryptographie: Vertiefung

Hannes Federrath

<http://www-sec.uni-regensburg.de/>

⌘ Klassische und moderne Verfahren

Definition Konzellationssystem

⌘ Seien

⊗ $n, l \in \mathbb{N}$

⊗ A und B Alphabete und K eine endliche Menge

⊗ Klartext: $m \in A^n = M$

⊗ Schlüsseltext: $c \in B^l = C$

l/n : Expansionsfaktor
 $l/n=1$ längentreue Chiffre

⌘ Dann ist eine Kryptofunktion eine Abbildung

$e: A^n \times K \rightarrow B^l$

derart, dass die Abbildung

$e_k: A^n \rightarrow B^l$

definiert durch

$e_k(m) := e(m, k)$

für alle $k \in K$ injektiv ist.

- injektiv: f^{-1} ist rechtseindeutig
- rechtseindeutig (auch: partiell):
 $\forall x. \forall y. y': ((f(x)=y \wedge f(x)=y') \rightarrow (y=y'))$

⌘ Für jedes e_k existiert eine Umkehrfunktion $d_{k'}$.

⌘ Es gelten

⊗ $c=e_k(m)$ und $m=d_{k'}(c)$, d.h. $m=d_{k'}(e_k(m))$ oder

⊗ $c=e(m, k)$ und $m=d(c, k')$, d.h. $m=d(e(m, k), k')$

Klassische Chiffren: Systematik

⌘ Transpositionschiffre: (Permutationen)

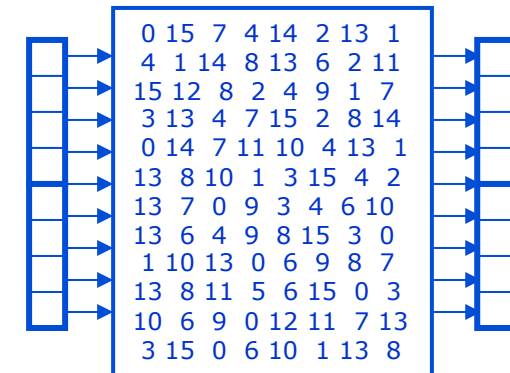
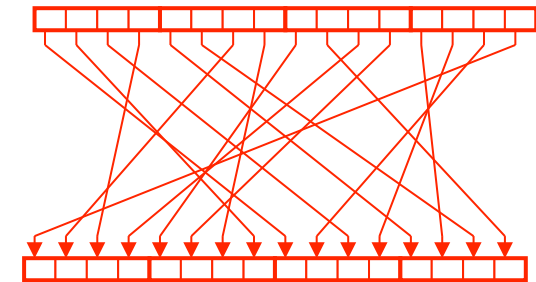
- ⊗ Veränderung der Anordnung von Schriftzeichen

⌘ Substitutionschiffre:

- ⊗ Systematische Ersetzung von Schriftzeichen

⌘ Produktchiffre:

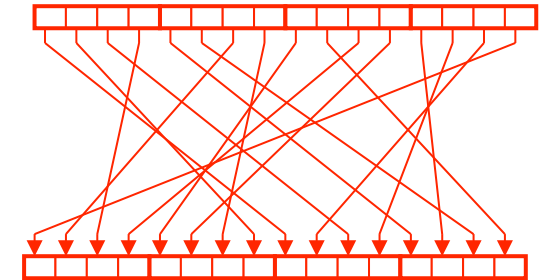
- ⊗ Kombination von Transpositionen und Substitutionen
- ⊗ Vorläufer der modernen symm. Kryptographie, bei denen Permutationen und Substitutionen (meist) iterativ angewendet werden



Klassische Chiffren: Konkrete Systeme

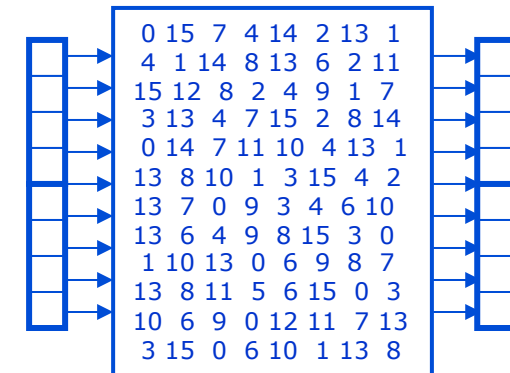
⌘ Transpositionschiffren

- ⊗ Spalten-Transpositionen
- ⊗ «freie» Permutationen



⌘ Substitutionschiffren

- ⊗ Schema von Polybios
- ⊗ Caesar-Chiffre
- ⊗ Vigenere-Chiffre
- ⊗ Vernam-Chiffre

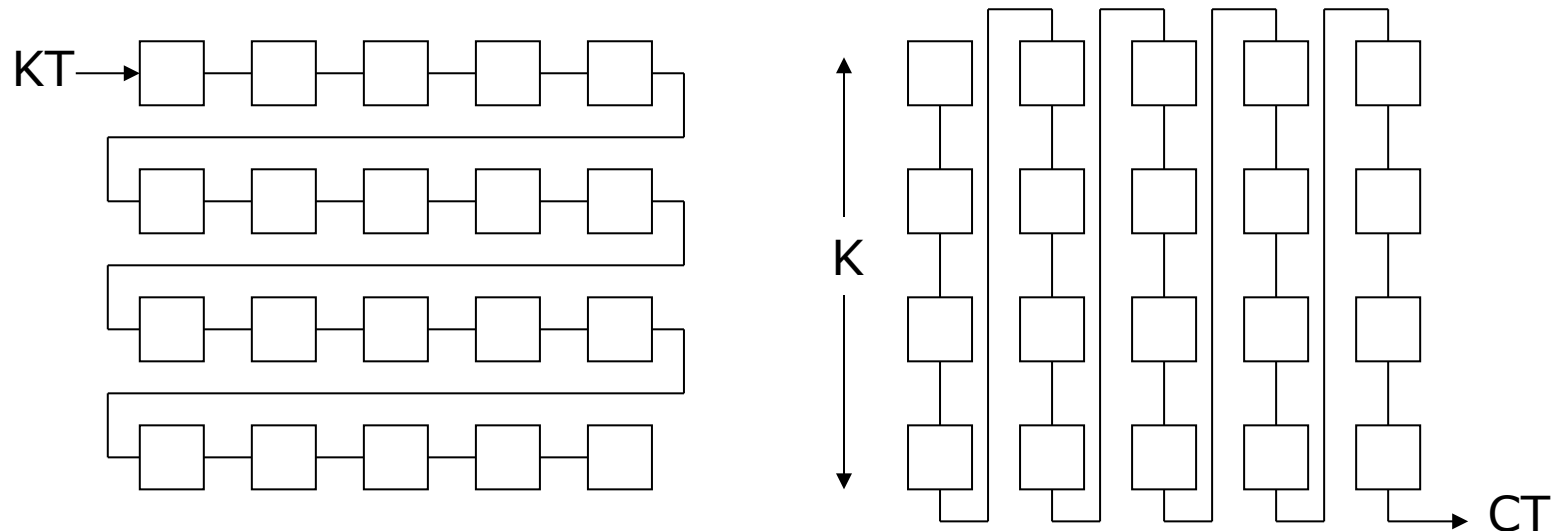


Spalten-Transposition (Skytala)

Fumy, S.40

⌘ Skytala

- ⊗ ca. 2400 Jahre alte griechische Chiffre
- ⊗ Zylinder mit gewickeltem Papierstreifen, schreiben, abwickeln
- ⊗ Empfänger hat Zylinder mit gleichem Durchmesser



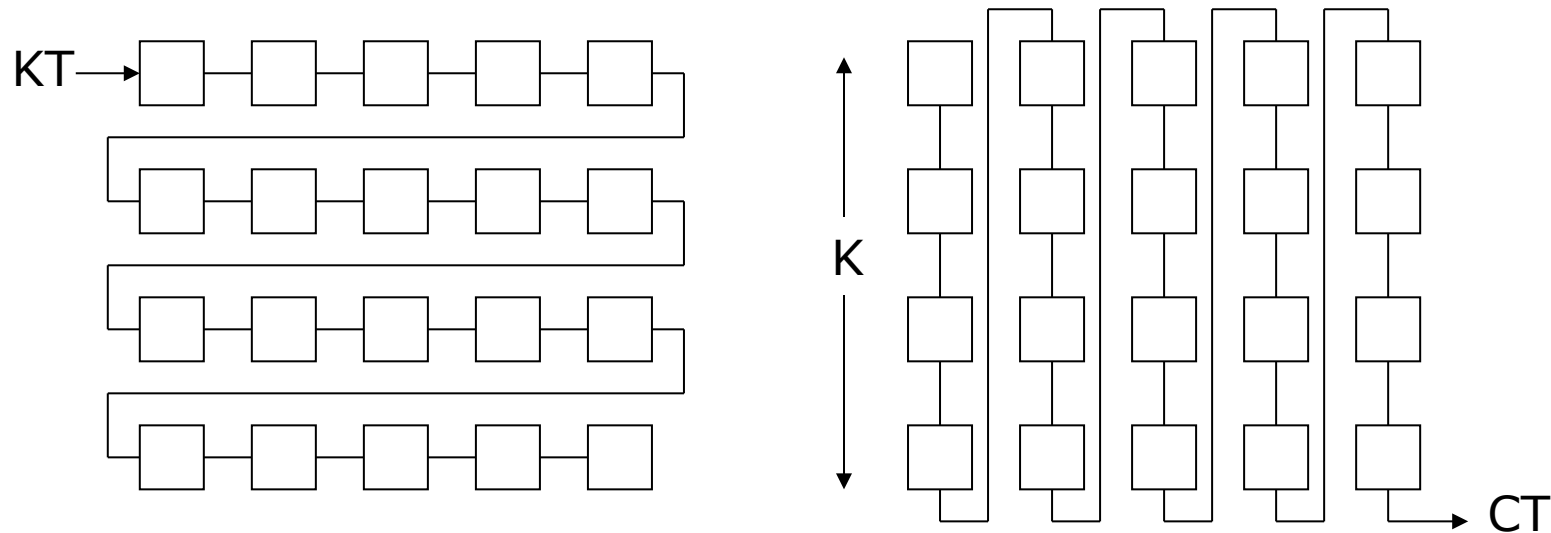
⌘ Kryptanalyse

- ⊗ Heute: Durchprobieren (sehr kleiner Schlüsselraum)
- ⊗ Statistische Analyse (Bigramme)

Spalten-Transposition (Skytala)

⌘ Variante

- ⊗ Spalten noch transponieren, d.h. Reihenfolge vertauschen



⌘ Übungsaufgabe

- ⊗ Um welchen Faktor vergrößert sich der Schlüsselraum bei s Spalten?

Skytala: Statistische Kryptanalyse:

⌘ Vorgehen

- ⊗ Suche typ. Bigramme (z.B. EN, ER, CH, ...) und ermittle die Häufigkeit der Buchstabenabstände. Beispiel:

⌘ Beispiel

- ⊗ Klartext: VERSCHLUESSELNMACHTGROSSENSPASS

- ⊗ $k=5$
VERSCHL
UESSELN
MACHTGR
OSSENSP
ASSXYZX

- ⊗ Chiffretext: VUMOAEEASSRSCSSSSHEXCETNYHLGSZLNRPX

Abstand/Häufigkeit

- ⊗ **EN**: VUMOA**EE**ASSRSCSSSS**HEXCETNYHLGSZLN**RPX: 2/1, 5/1, ...
- ⊗ **ER**: VUMOA**EE**ASSR**SCSSSSHEXCETNYHLGSZLN**RPX: 5/1, 4/1, ...
- ⊗ **EI**: VUMOA**EE**ASSRSCSSSS**HEXCETNYHLGSZLN**RPX: 0
- ⊗ **CH**: VUMOAEEASSR**SCSSSSHEXCETNYHLGSZLN**RPX: 5/2, ...

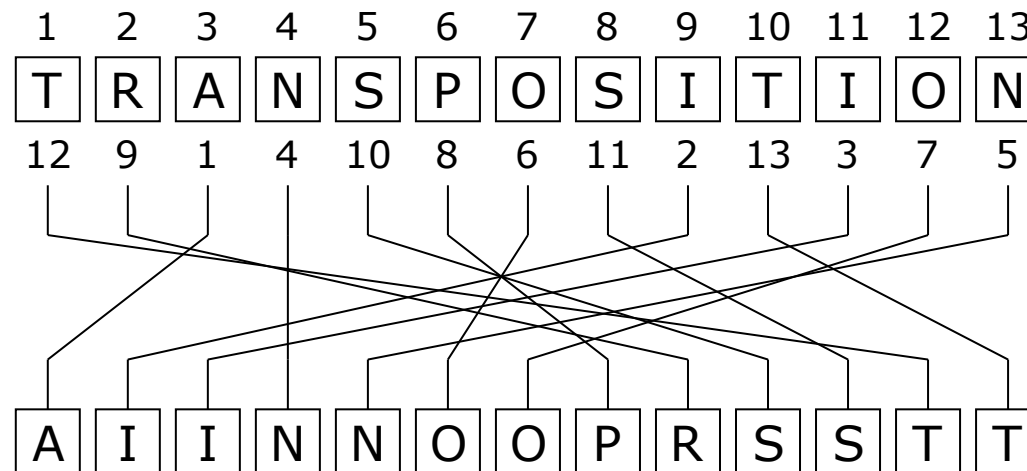
- ⊗ Abstand 5 kommt am Häufigsten vor, teste, ob Text in 5 Zeilen sinnvoll → gebrochen

«Freie» Permutationen

⌘ Idee

⊠ Zeichen werden nach einer Vorschrift vertauscht

⌘ Beispiel



Zyklenschreibweise:
(1 12 7 6 8 11 3)
(2 9) (4) (5 10 13)

⌘ Übungsaufgabe

⊠ Schreiben Sie eine Skytala mit 4 Zeilen und 3 Spalten in Zyklenschreibweise.

⌘ Bewertung

☒ Anforderung an den Schlüssel:

⊕ gutes Mischen der Klartextzeichen

⊕ Bestenfalls ist der mittlere **Abstand zwischen den Bildpositionen benachbarter Urbilder** etwa die halbe Blockgröße.

Erläuterung:

- **Abstand zwischen den Bildpositionen:** Abstand zwischen den Zeichen im Chiffrat
- **benachbarte Urbilder:** benachbare Klartextzeichen

Klassische Chiffren

⌘ Monoalphabetische Substitutionen

- ⊗ Jedem Zeichen bzw. jeder Zeichenfolge über A ist eindeutig ein Zeichen bzw. eine Zeichenfolge über B zugeordnet.

⌘ Polyalphabetische Substitutionen

- ⊗ Jedem Zeichen bzw. jeder Zeichenfolge über A ist eindeutig ein Zeichen bzw. eine Zeichenfolge über B_1, B_2, \dots, B_n zugeordnet.

⌘ Monographische Substitution

- ⊗ Es werden einzelne Zeichen ersetzt.

⌘ Polygraphische Substitution

- ⊗ Es werden Zeichenfolgen ersetzt.

Schema von Polybios

⌘ Def.

- ⊗ $A = \{A:Z\}$
- ⊗ $B = \{ (i,j) \mid i,j \in \{1:5\} \}$
- ⊗ e/d:

	\rightarrow					
	j	1	2	3	4	5
i ↓	1	A	B	C	D	E
	2	F	G	H	I	J
	3	K	L	M	N	O
	4	P	Q	R	S	T
	5	U	V	W	X/Y	Z

Beispiel:

Chiffretext: 223515452315

Klartext:

⌘ Eigenschaften

- ⊗ monoalphabetische/monographische Substitution
- ⊗ arbeitet «schlüssellos»

⌘ Ableitungen

- ⊗ Tabelle mit Zeichen (Freimaurer-Chiffre, Friedhofschiffre)

Verschiebechiffre

⌘ auch: Caesar-Chiffre

⊗ Caesar, röm. Kaiser und Feldherr (100-44 v.Chr)

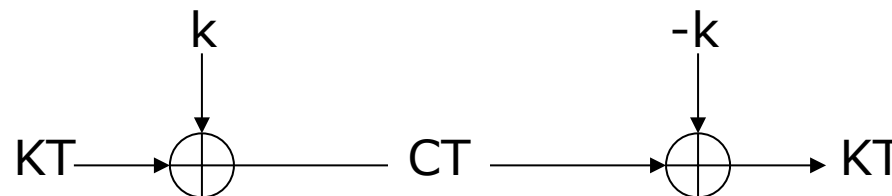
⌘ Def.

⊗ $A=B=\{A:Z\}$

⊗ $K=\{A:Z\}$ oder allg. $K=\{0:n-1\}$ mit $n \leq \text{card}(\{A:Z\})$

⊗ e: $c=(m+k) \bmod n$

⊗ d: $m=(c-k) \bmod n$



$$(x+y) \bmod 26$$

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ABCDEFGHIJKLMNOPQRSTUVWXYZ
B BCDEFGHIJKLMNOPQRSTUVWXYZA
C CDEFGHIJKLMNOPQRSTUVWXYZAB
D DEFGHIJKLMNOPQRSTUVWXYZABC
E EFGHIJKLMNOPQRSTUVWXYZABCD
F FGHIJKLMNOPQRSTUVWXYZABCDE
G GHIJKLMNOPQRSTUVWXYZABCDEF
H HIJKLMNOPQRSTUVWXYZABCDEFG
I IJKLMNOPQRSTUVWXYZABCDEFGH
J JKLMNOPQRSTUVWXYZABCDEFGHI
K KLMNOPQRSTUVWXYZABCDEFGHIJ
L LMNOPQRSTUVWXYZABCDEFGHIJK
M MNOPQRSTUVWXYZABCDEFGHIJKL
N NOPQRSTUVWXYZABCDEFGHIJKLM
O OPQRSTUVWXYZABCDEFGHIJKLMN
P PQRSTUVWXYZABCDEFGHIJKLMNO
Q QRSTUVWXYZABCDEFGHIJKLMNO
R RSTUVWXYZABCDEFGHIJKLMNO
S STUVWXYZABCDEFGHIJKLMNO
T TUVWXYZABCDEFGHIJKLMNO
U UVWXYZABCDEFGHIJKLMNO
V VWXYZABCDEFGHIJKLMNO
W WXYZABCDEFGHIJKLMNO
X XYZABCDEFGHIJKLMNO
Y YZABCDEFGHIJKLMNO
Z ZABCDEFGHIJKLMNO

Verschiebechiffre

⌘ auch: Caesar-Chiffre

⊗ Caesar, röm. Kaiser und Feldherr (100-44 v.Chr)

⌘ Def.

⊗ $A=B=\{A:Z\}$

⊗ $K=\{A:Z\}$ oder allg. $K=\{0:n-1\}$ mit $n \leq \text{card}(\{A:Z\})$

⊗ e: $c=(m+k) \bmod n$

⊗ d: $m=(c-k) \bmod n$

⌘ Eigenschaften

⊗ monoalphabetische/monographische Substitution

⊗ additive Chiffre

⊗ Buchstabenhäufigkeiten bleiben erhalten und bilden Ansatz zur Kryptanalyse

Vigenère-Chiffre

⌘ nach: Blaise de Vigenère, 1586

⊗ französischer Kryptologe

⌘ Idee

⊗ Gleiche Klartextzeichen auf unterschiedlichen Chiffretextzeichen abbilden, um Häufigkeitsanalyse zu erschweren (polyalphabetische Substitution)

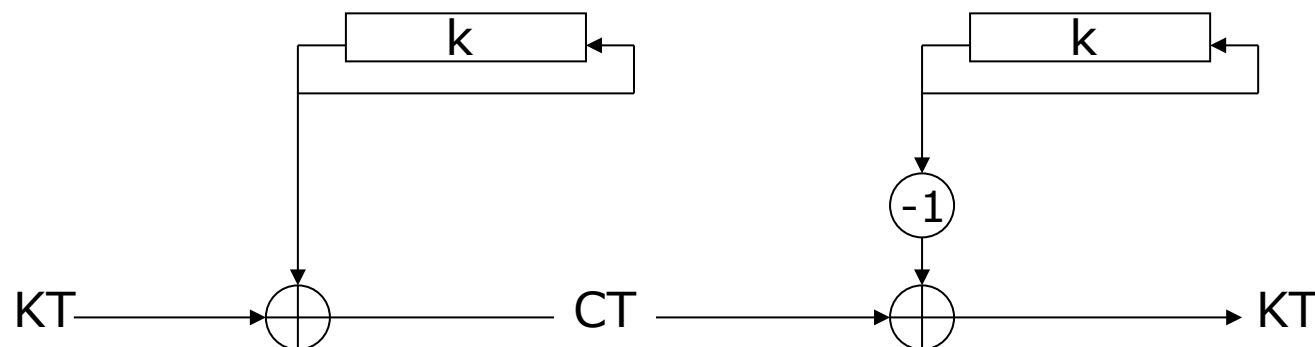
⌘ Def.

⊗ $A=B=\{A:Z\}$

⊗ $K=\{(k_1, k_2, k_3, \dots, k_r) \mid k_i \in \{A:Z\}, r \in \{1, 2, \dots\}\}$

⊗ r : Periodenlänge des Schlüssels

⊗ e, d : für jedes k_i analog Caesar-Chiffre



⌘ Vigenère-Chiffre: Beispiel

⊕ VIGENERECHIFFRE = Klartext

⊕ HUGOHUGOHUGOHUG = Schlüssel mit $r=4$

⊕ CCM..... = Chiffretext

⊗ Kryptanalyse:

⊕ Periodenlänge ermitteln, dann weiter wie Caesar-Chiffre

⌘ Variante: Autokey-Verfahren

⊗ Klartext als Teil des Schlüssels verwenden:

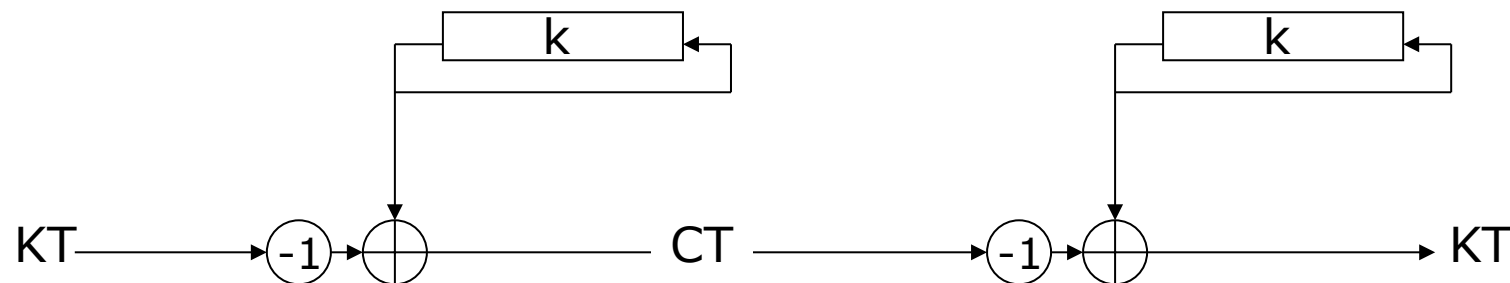
⊕ VIGENERECHIFFRE = Klartext

⊕ HUGOVIGENERECHI = Schlüssel

Beaufort-Chiffre

⌘ nach: Francis Beaufort (1857)

- ⊗ jedoch bereits 1710 von Giovanni Sesti vorgeschlagen
- ⊗ Variante der Vigenère-Chiffre
- ⊗ involutorische Chiffre ($e=d$)



⌘ Beispiel für die $e=d$ -Eigenschaft

- ⊗ $c = (-1 \cdot m + k) \bmod n$ und $m = (-1 \cdot c + k) \bmod n$
- ⊗ $FED = m, k = 3$
- ⊗ $FGA = C$
- ⊗ $FED = m$

$i=0$	1	2	3	4	5	6	($n=7$)
A	B	C	D	E	F	G	

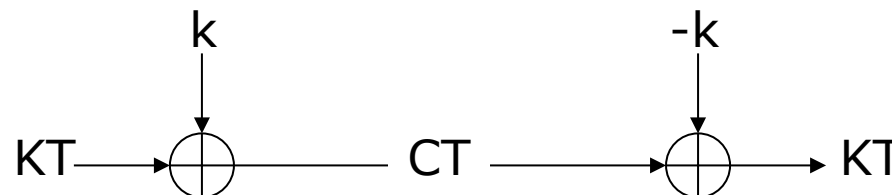
Vernam-Chiffre (One-Time-Pad)

⌘ Def.

- ⊗ $A=B=\{A:Z\}$
- ⊗ $K=\{(k_1, k_2, k_3, \dots, k_l) \mid k_i \in \{A:Z\}, i=[1, l]\}$
- ⊗ Die k_i werden unabhängig und zufällig erzeugt
- ⊗ Der Schlüssel $k=(k_1, k_2, k_3, \dots, k_l)$ hat die gleiche Länge wie der Klartext $m=(a_1, a_2, a_3, \dots, a_l)$
- ⊗ e: $c=(m+k) \bmod n$ (zeichenweise)
- ⊗ d: $m=(c-k) \bmod n$

⌘ Beispiel

- ⊕ KRYPTOMACHTSPASS = Klartext
- ⊕ VABZEQTAWPNRTLKB = Schlüssel
- ⊗ Kryptanalyse:
 - ⊕ unmöglich (informationstheoretisch sicher)



Vernam-Chiffre (One-Time-Pad)

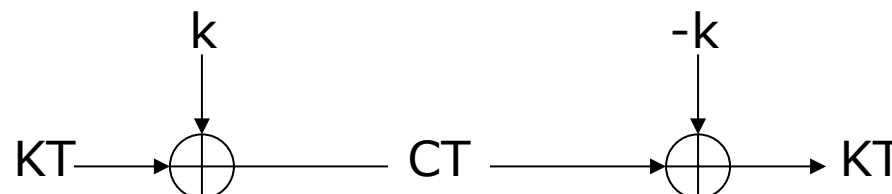
⌘ Informationstheoretisch sichere Konzelation:

- ⊗ Egal, was der Angreifer a priori an Information über den Klartext hat, er gewinnt durch die Beobachtung des Schlüsseltextes keine Information hinzu.

$$\forall s \in S \exists const \in N \forall x \in X : |\{k \in K \mid k(x) = s\}| = const$$

Für alle Schlüsseltexte s existiert eine konstante Anzahl von Schlüsseln k, die jeweils alle Klartexte x derart verschlüsseln, dass aus x jeder Schlüsseltext entstehen kann. $N = \{1, 2, 3, \dots\}$

- ⊗ »Hinter jedem Schlüsseltext kann sich jeder Klartext verbergen«



Visuelle Kryptographie

⌘ Symmetrisches Verfahren

- ⊗ Symmetrischer Schlüssel: Sender und Empfänger erzeugen sich Zufallsmuster aus zwei »Basismustern«:

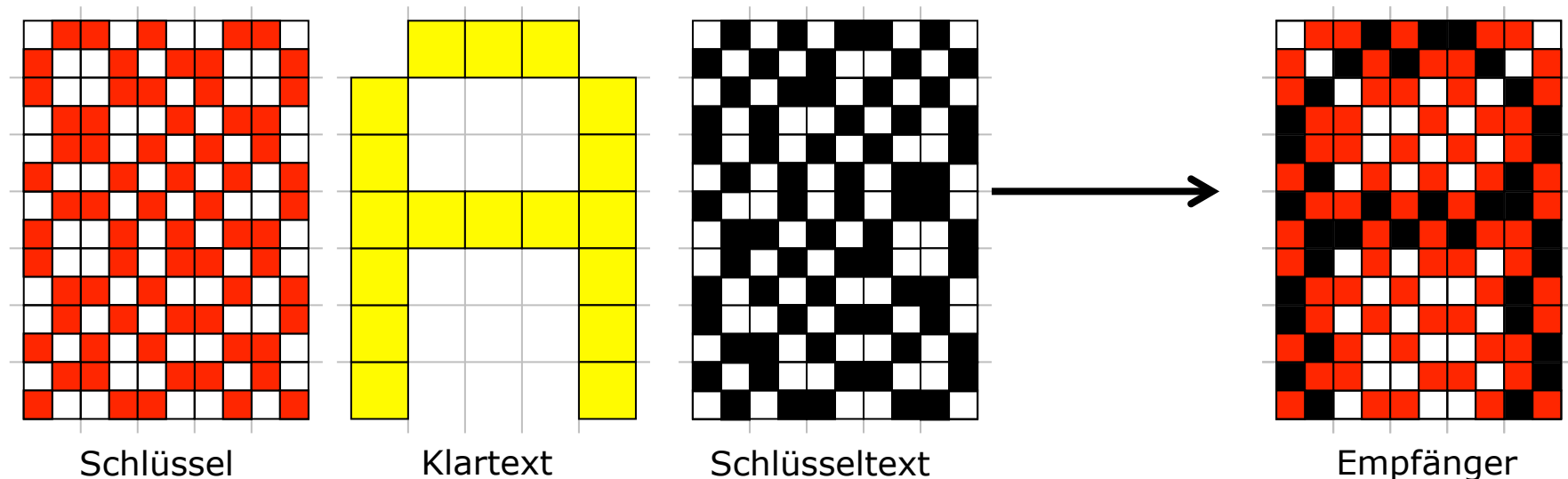


⌘ Visuelle Botschaft:

- ⊗ Sender verwendet negiertes Muster für schwarze Bildpunkte
- ⊗ Für »weiße« Bildpunkte: keine Veränderung

⌘ Schöne Demo:

- ⊗ <http://www.tcs.mu-luebeck.de/demos2003/vc.html>



⌘ Symmetrische Systeme

- ⊗ One-Time-Pad (mod 2)
- ⊗ Symmetrische Authentifikationscodes
- ⊗ DES (Data Encryption Standard)
- ⊗ IDEA (International Data Encryption Algorithm)
- ⊗ AES (Advanced Encryption Standard)

⌘ Praktischer Einsatz

- ⊗ Betriebsarten von Blockchiffren

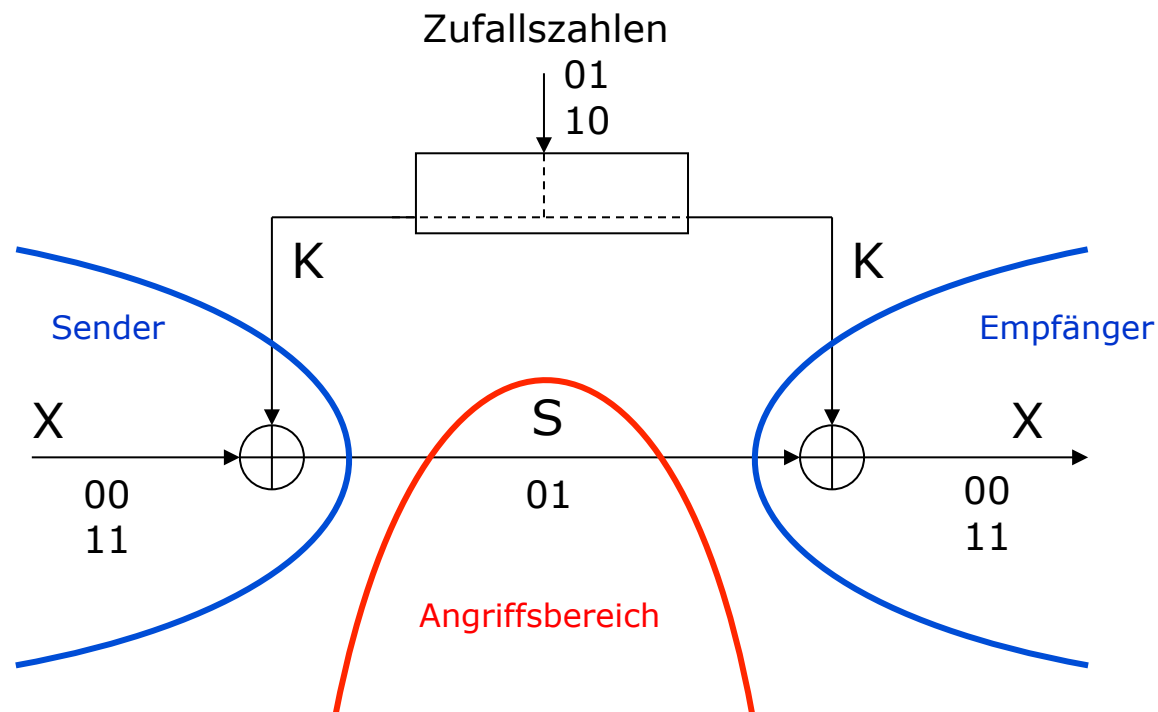
⌘ Asymmetrische Systeme

- ⊗ Diffie-Hellmann-Key-Exchange
- ⊗ El Gamal Kryptosystem
- ⊗ RSA zur Konzelation und Signatur
- ⊗ Blinde Signaturen mit RSA
- ⊗ Kryptosysteme auf Basis elliptischer Kurven

One-Time-Pad (mod 2)

$X \oplus K = S$		
0	0	0
0	1	1
1	0	1
1	1	0

- » Jedes Schlüsselbit darf nur einmal verwendet werden
- » Bits von K sind zufällig und unabhängig
- » Schlüssel genauso lang wie Klartext

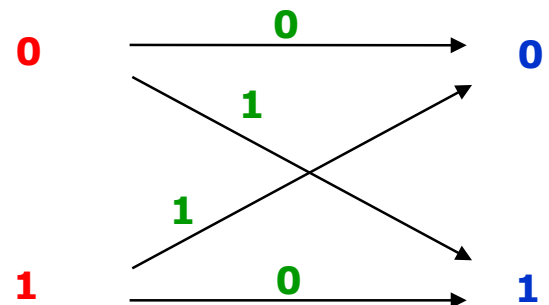


One-Time-Pad (mod 2)

$$\begin{array}{r} X \oplus K = S \\ \hline 0 \quad 0 \quad 0 \\ 0 \quad 1 \quad 1 \\ 1 \quad 0 \quad 1 \\ 1 \quad 1 \quad 0 \end{array}$$

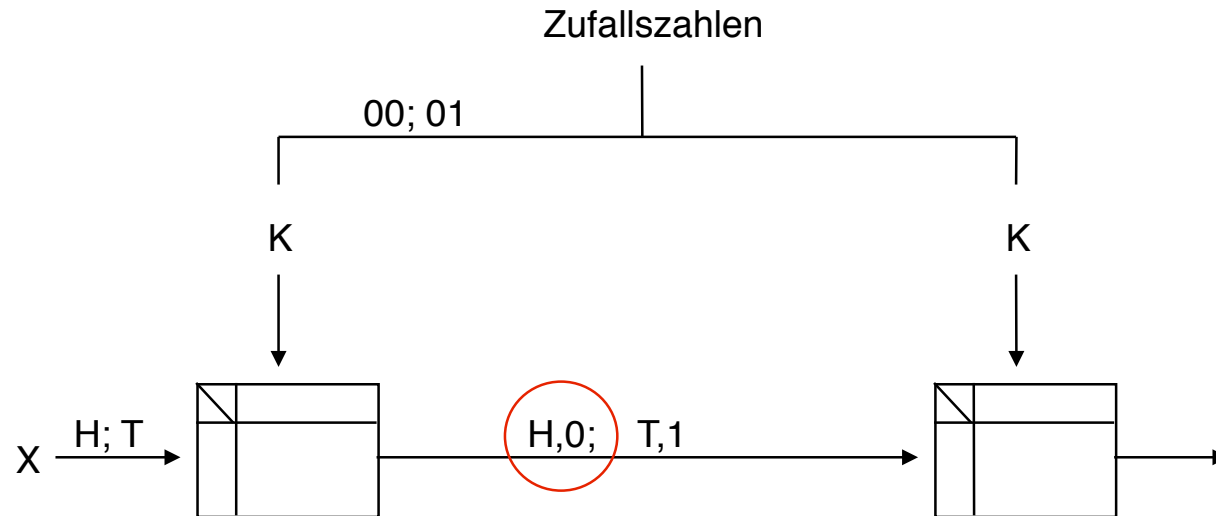
- » Jedes Schlüsselbit darf nur einmal verwendet werden
- » Bits von K sind zufällig und unabhängig
- » Schlüssel genauso lang wie Klartext

Angreifer sieht S: K kann sein: dann ist X gewesen:



- ⊗ Der Angreifer kann alle 4 Varianten durchrechnen, erhält dadurch aber keine zusätzliche Information über den Klartext.
- ⊗ Die Wahrscheinlichkeit, ein Kartextbit richtig zu raten, verändert sich durch die Beobachtung des Schlüsseltextes nicht, sondern bleibt $\text{const} = 0,5$.

Symmetrische Authentifikationscodes



x, MAC	$H,0$	$H,1$	$T,0$	$T,1$
k				
00	H	-	T	-
01	H	-	-	T
10	-	H	T	-
11	-	H	-	T

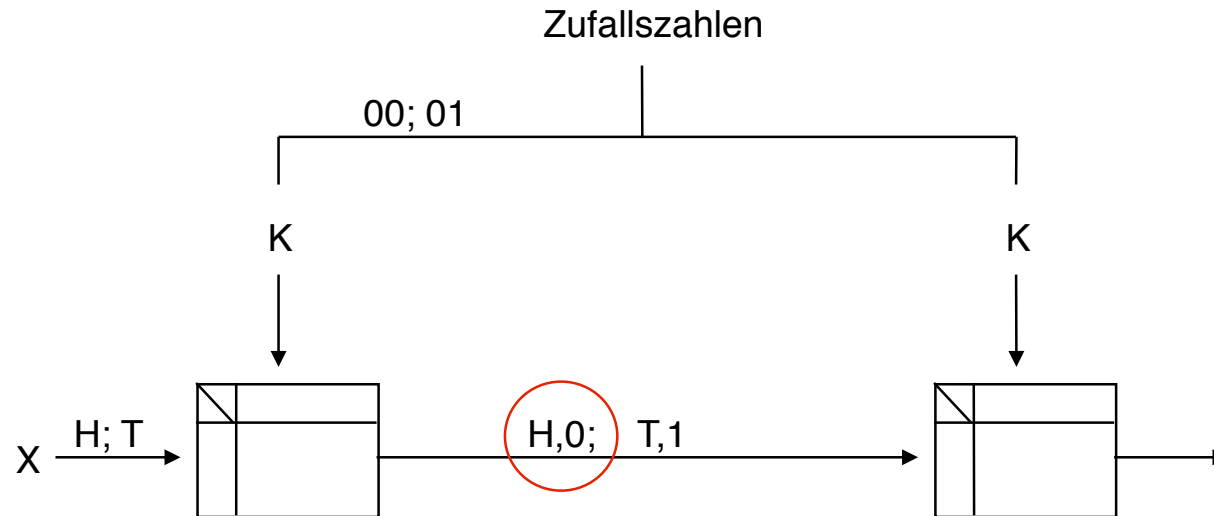
oder

k	x	H	T
00	0	0	0
01	0	0	1
10	1	1	0
11	1	1	1

MAC

$H := "0"$
 $T := "1"$

Symmetrische Authentifikationscodes



⌘ Angriff 1: (blind)

- ⊗ Angreifer will T senden
 - ⊕ erwischt richtigen MAC mit Wkt = 0,5

⌘ Angriff 2: (sehend)

- ⊗ Angreifer will H,0 in T ändern
 - ⊕ weiß: $k \in \{00,01\}$
 - ⊕ wenn $k = 00$ war, muß er T,0 senden
 - ⊕ wenn $k = 01$ war, muß er T,1 senden
 - ⊕ Wkt. ist immernoch 0,5

Symmetrische Authentifikationscodes

⌘ informationstheoretisch sicher

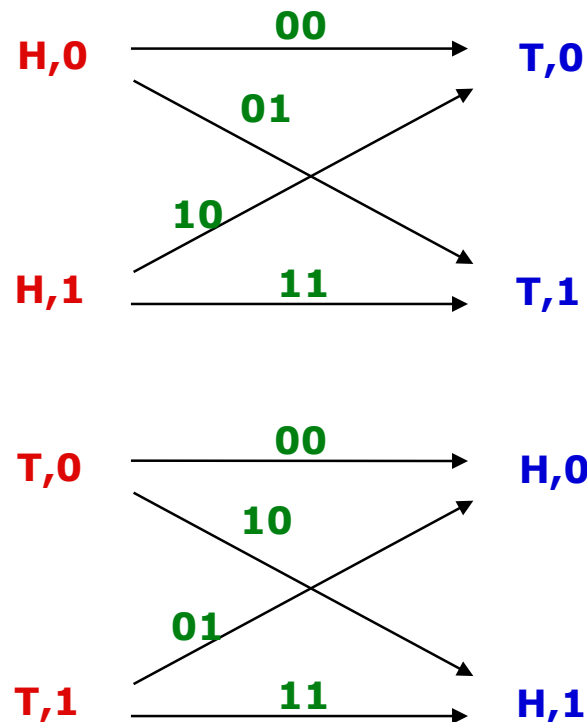
k \ x	H	T
00	0	0
01	0	1
10	1	0
11	1	1

MAC

Angreifer sieht:

K kann sein:

Angreifer will x fälschen
und sucht passenden MAC



Wkt., dass Angreifer
den richtigen MAC für
das Bit wählt, ist 0,5
(d.h. „Raten“)

DES (Data Encryption Standard)

⌘ Amerikanischer Verschlüsselungsstandard

- ⊗ 1977 vom National Bureau of Standards (NBS) der USA standardisiert

⌘ Blockchiffre

- ⊗ operiert auf Blöcken von jeweils 64 Bit

⌘ Feistel-Chiffre

- ⊗ iterierte Anwendung eines Verschlüsselungsschemas aus Permutationen, Substitutionen und Expansionen

⌘ $n=16$ Runden

- ⊗ mit jeweils unterschiedlichen Teilschlüsseln K_i

⌘ Schema ist selbstinvers

- ⊗ d.h. Entschlüsselung wie Verschlüsselung, jedoch umgekehrte Reihenfolge der Teilschlüssel

Gütekriterien für gute moderne symmetr. Chiffren

⌘ Höchstmaß an

- ⊗ Vollständigkeit
- ⊗ Avalanche
- ⊗ Nichtlinearität
- ⊗ Korrelationsimmunität

⌘ weitere Kriterien

- ⊗ gute Implementierbarkeit
- ⊗ Längentreue
- ⊗ Schnelligkeit

⌘ Vollständigkeit

⊗ Def.: Eine Funktion $F: \{0,1\}^n \rightarrow \{0,1\}^m$ ist dann vollständig, wenn jedes Bit des Outputs von jedem Bit des Inputs abhängt.

⊗ Beispiel:

$$\oplus y_1 = x_1x_2 + x_1x_3 + x_2x_3 + x_2 + x_3 + 1$$

$$\oplus y_2 = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_3 + 1$$

$$\oplus y_3 = x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1$$

⌘ Avalanche

⊗ Def.: Eine Funktion $F: \{0,1\}^n \rightarrow \{0,1\}^m$ besitzt dann den Avalanche-Effekt, wenn die Änderung eines Input-Bits im Mittel die Hälfte aller Output-Bits ändert.

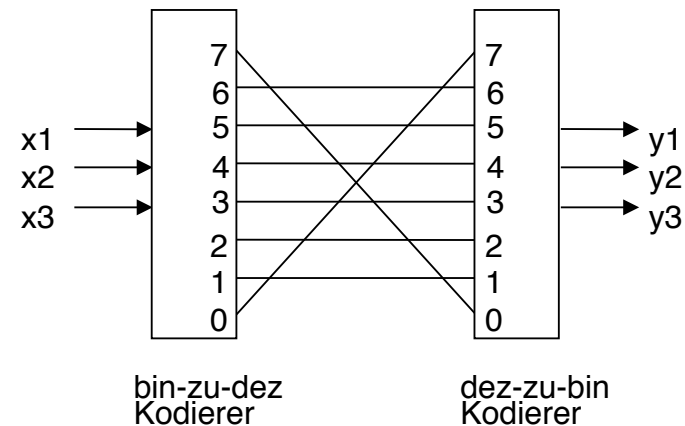
⊗ Wird durch Änderung eines Input-Bits jedes Output-Bit mit einer Wahrscheinlichkeit von 50% verändert, so erfüllt F das **strikte Avalanche-Kriterium**.

⊗ Satz: Erfüllt F das strikte Avalanche-Kriterium, so ist F stets vollständig.

⌘ Linearität

- ⊗ Def.: Eine Funktion $F: \{0,1\}^n \rightarrow \{0,1\}^m$ ist dann linear, wenn jedes Output-Bit y_j linear von den Input-Bits x_i abhängt.
- ⊗ Wenn wenigstens ein Output-Bit linear von den Input-Bits abhängt, bezeichnet man F als **partiell linear**.
- ⊗ Beispiel: (siehe Vollständigkeit)

X_{dez}	x_3	x_2	x_1	y_3	y_2	y_1	Y_{dez}
0	0	0	0	1	1	1	7
1	0	0	1	0	0	1	1
2	0	1	0	0	1	0	2
3	0	1	1	0	1	1	3
4	1	0	0	1	0	0	4
5	1	0	1	1	0	1	5
6	1	1	0	1	1	0	6
7	1	1	1	0	0	0	0



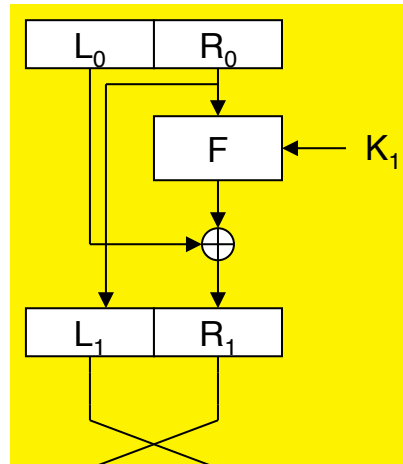
⌘ Korrelationsimmunität

- ⊗ Sei $f(x_1, \dots, x_n)$ eine boolesche Funktion in n Variablen.
- ⊗ f ist dann k -korrelationsimmun, wenn man aus Kenntnis einer beliebigen Menge von k Eingangswerten keine Informationen über den resultierenden Ausgangswert erhalten kann und umgekehrt.

- ⊗ Bedeutung:
 - ⊕ Jede Teilmenge der Output-Vektoren, die Rückschlüsse auf Teilmengen der Input-Vektoren zulässt, verringert den Aufwand für das vollständige Durchsuchen des Schlüsselraumes.

Feistel-Prinzip (1 Runde)

Verschlüsselung



$$L_1 = R_0 \quad (1)$$

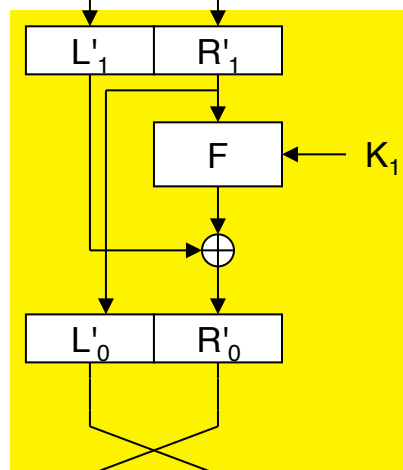
$$R_1 = f(R_0) \oplus L_0 \quad (2)$$

$$L'_1 = R_1 \quad (3)$$

$$R'_1 = L_1 \quad (4)$$

Schlüsseltext

Entschlüsselung



$$L'_0 = R'_1 \quad (5)$$

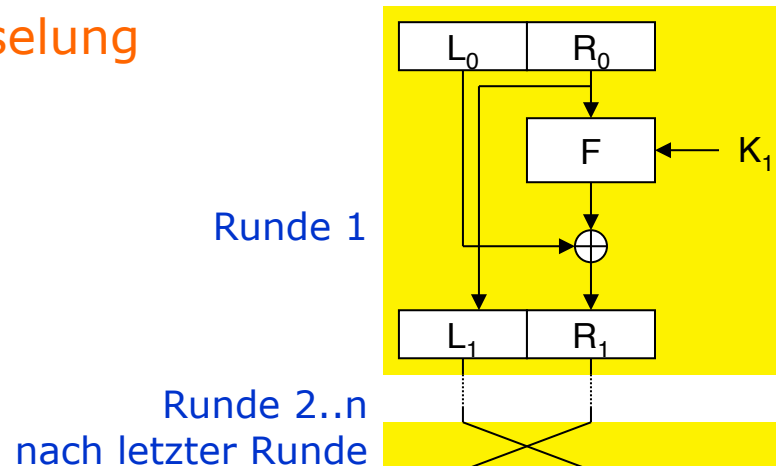
$$R'_0 = f(R'_1) \oplus L'_1 \quad (6)$$

Klartext

Funktion F kann
Einwegfunktion sein

Feistel-Prinzip (n Runden)

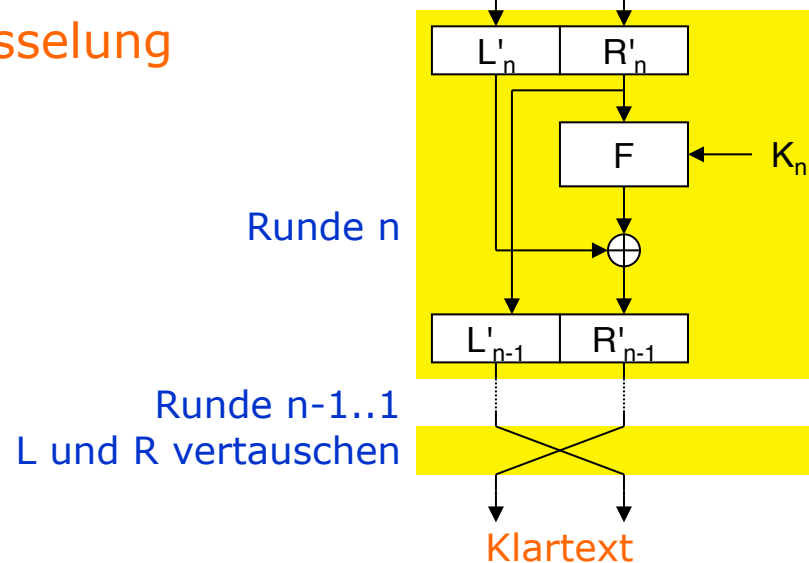
Verschlüsselung



$$L_n = R_{n-1}$$
$$R_n = f(R_{n-1}) \oplus L_{n-1}$$

$$L'_n = R_n$$
$$R'_n = L_n$$

Entschlüsselung

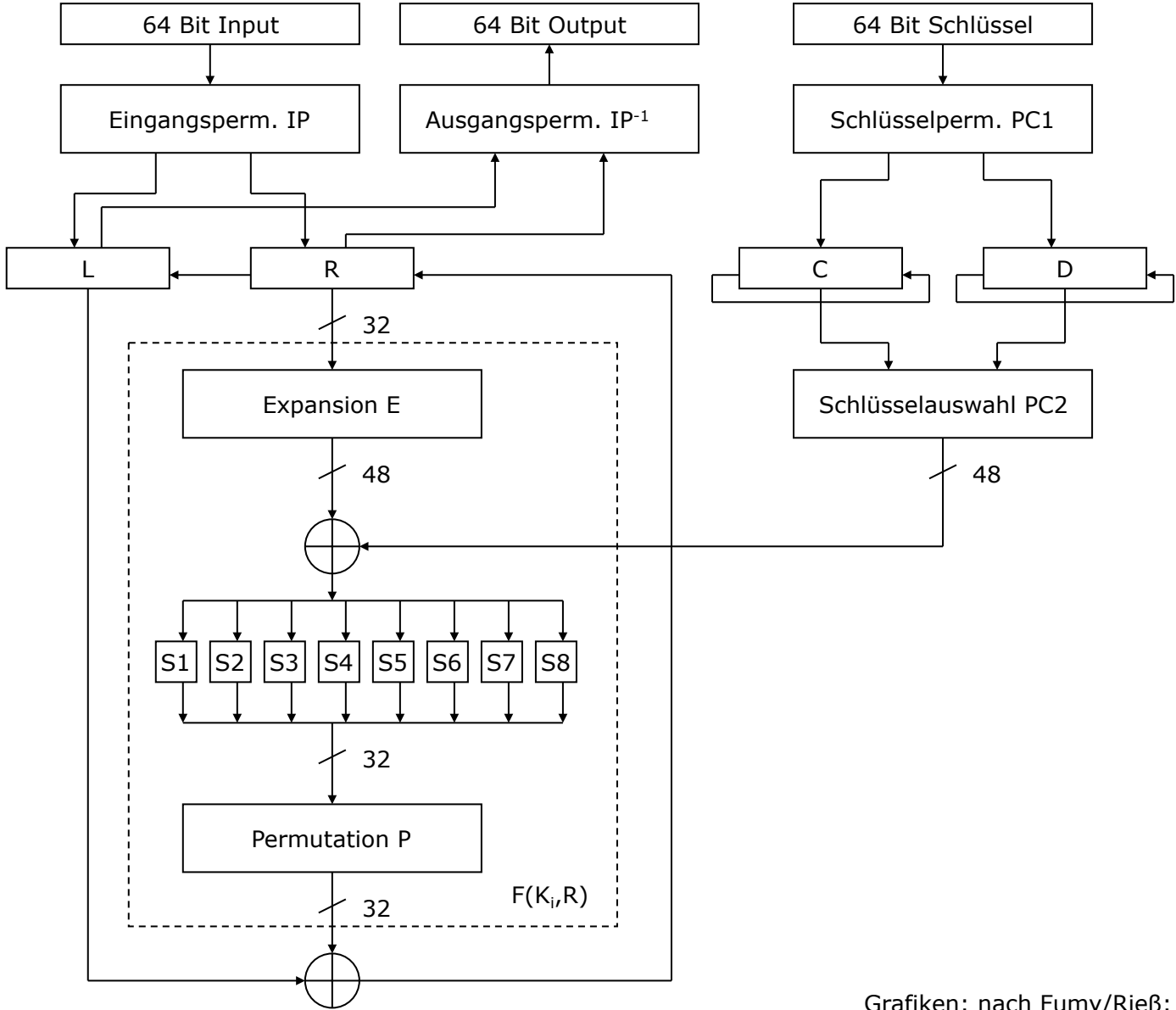


$$L'_{n-1} = R'_n$$
$$R'_{n-1} = f(R'_n) \oplus L'_n$$

DES (Data Encryption Standard)

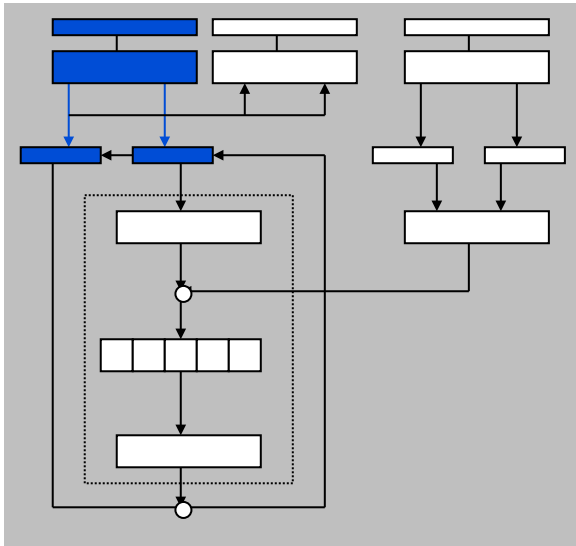
- ⌘ Symmetrische Blockchiffre mit $M \in \{0,1\}^{64}$, $K \in \{0,1\}^{56}$
 - ⊗ Feistel-Chiffre
 - ⊗ $n = 16$ Runden
 - ⊗ Schlüssel besteht aus 56 Bit + 8 Paritätsbits
 - ⊗ Teilschlüssel $K_1 \dots 16$ (jeweils 48 Bit) werden aus einem 56-Bit Schlüssel gewonnen
 - ⊗ Vor der ersten und nach der letzten Runde durchläuft der Datenblock eine Permutation IP bzw. IP^{-1} , die kryptographisch irrelevant ist.
- ⌘ Funktion $F(K_i, R_{i-1})$
 - ⊗ Expansionsabbildung von 32 auf 48 Bit
 - ⊗ 8 S-Boxen, jede S-Box: 6-Bit-Input, 4-Bit-Output
 - ⊗ 32-Bit-Permutation
- ⌘ Teilschlüsselgenerierung
 - ⊗ Permuted Choice 1 (Schlüsselpermutation)
 - ⊗ Zyklische Schiebeoperationen auf Registern C und D in Abhängigkeit der Rundenummer
 - ⊗ Permuted Choice 2 (Schlüsselauswahl 48 aus 56 Bit)

Data Encryption Standard (DES)



Grafiken: nach Fumy/Rieß: Kryptographie, 1988

DES: IP

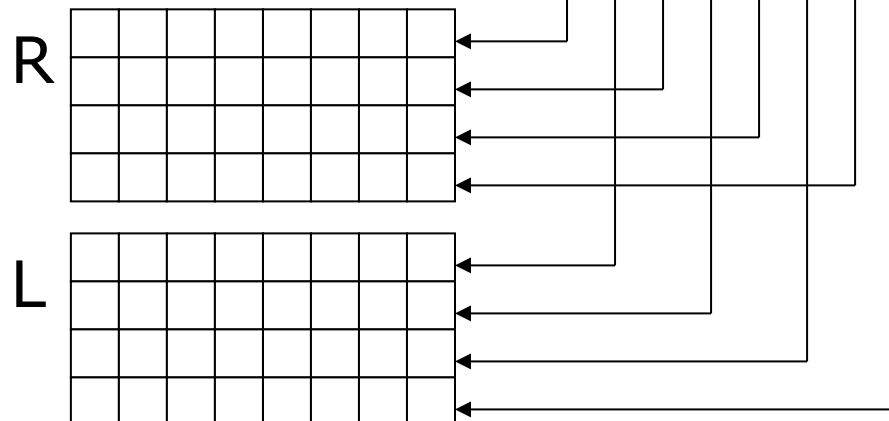


Inputpermutation IP

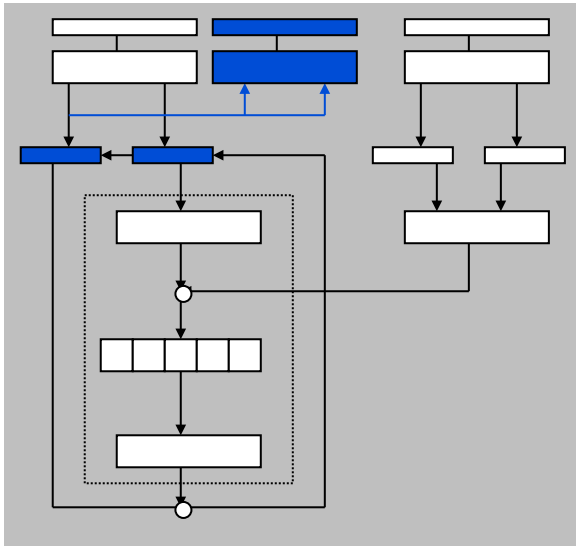
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

1	2	3	4	5	6	7	8
9	10	11	12	...			16
17							24
25							32
33							40
41							48
49							56
57	58	59	60	61	62	63	64

Dateninput/-output



DES: IP-1

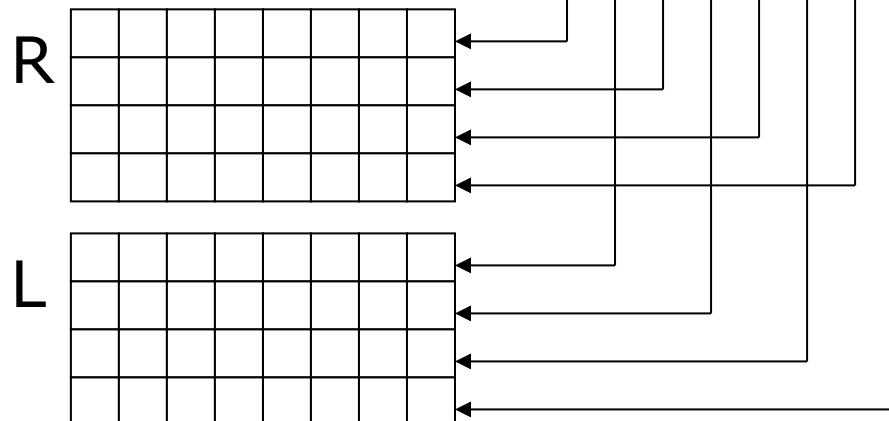


Outputpermutation IP-1

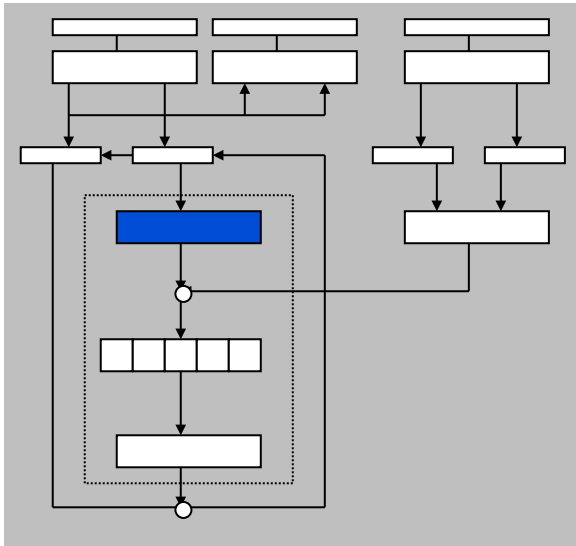
40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

1	2	3	4	5	6	7	8
9	10	11	12	...			16
17							24
25							32
33							40
41							48
49							56
57	58	59	60	61	62	63	64

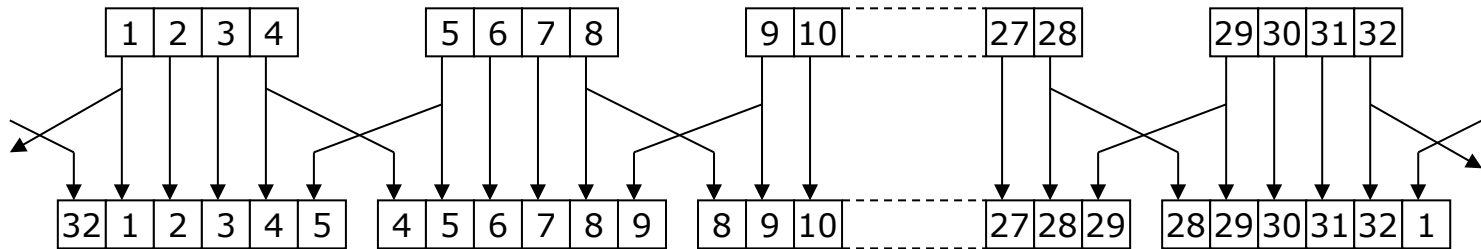
Dateninput/-output



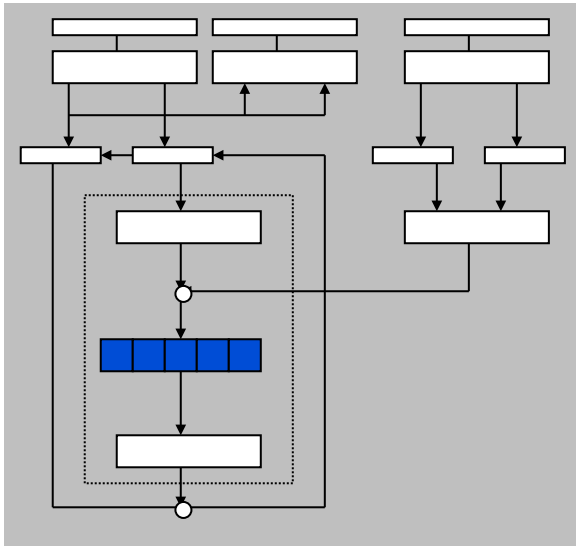
DES: Expansion E



32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

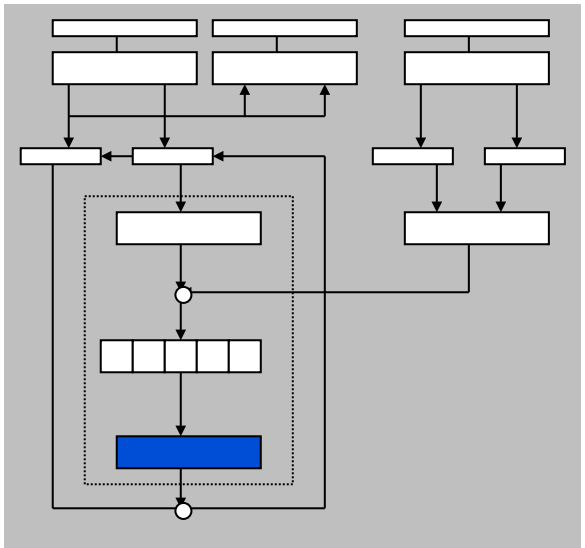


DES: S-Boxen S1 bis S8

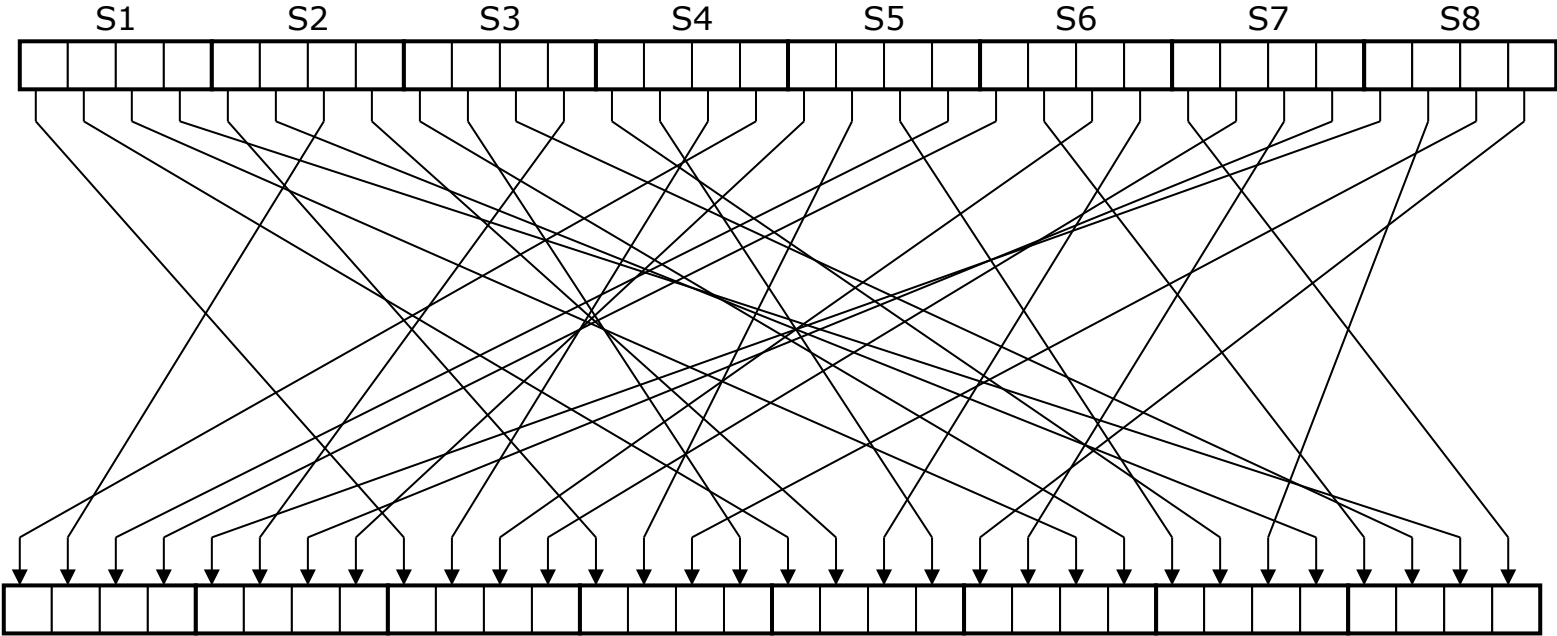


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
S1:	0:	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1:	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2:	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3:	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2:	0:	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1:	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2:	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3:	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3:	0:	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1:	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2:	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3:	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4:	0:	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1:	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2:	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3:	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5:	0:	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1:	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2:	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3:	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6:	0:	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1:	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2:	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3:	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7:	0:	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1:	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2:	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3:	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8:	0:	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1:	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2:	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3:	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

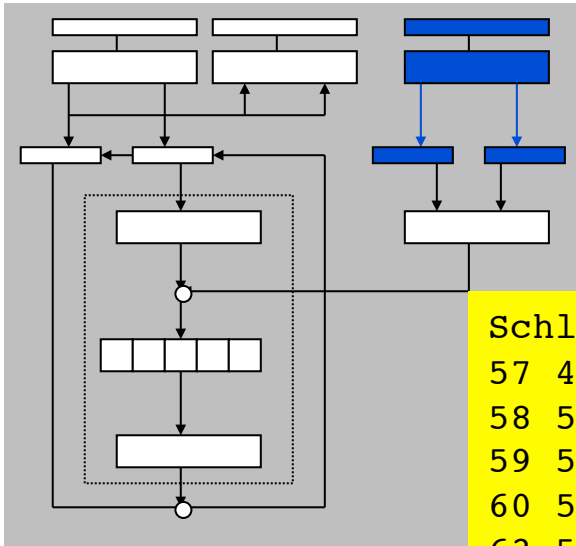
DES: Permutation P



16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25



DES: PC1



Schlüsselpermutation PC1

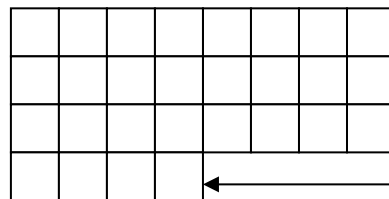
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36				
63	55	47	39	31	23	15	7
62	54	46	38	30	22	14	6
61	53	45	37	29	21	13	5
			28	20	12		4

MSB				LSB			
1	2	3	4	5	6	7	8
9	10	11	12	...			16
17							24
25							32
33							40
41							48
49							56
57	58	59	60	61	62	63	64

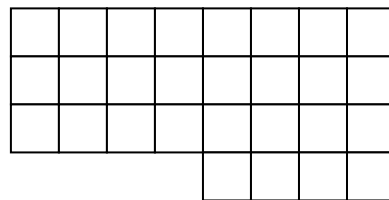
Externer Schlüssel

Paritätsbits

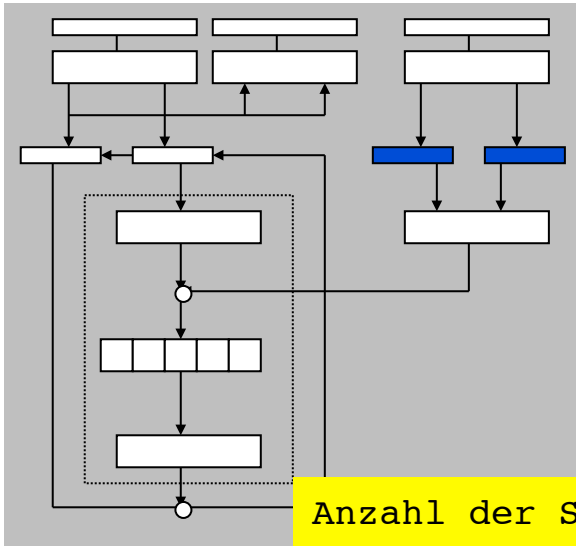
C



D

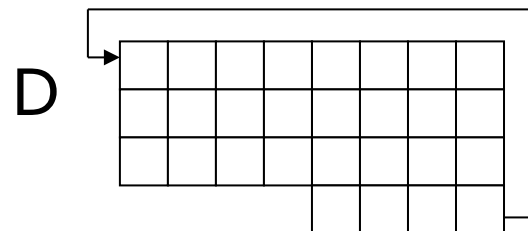
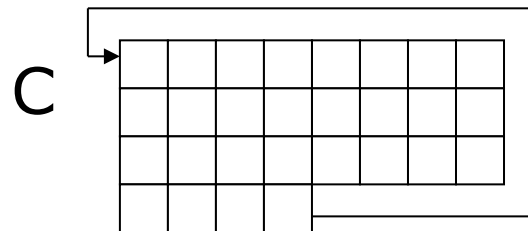


DES: Shifts bei Chiffrierung und Dechiffrierung

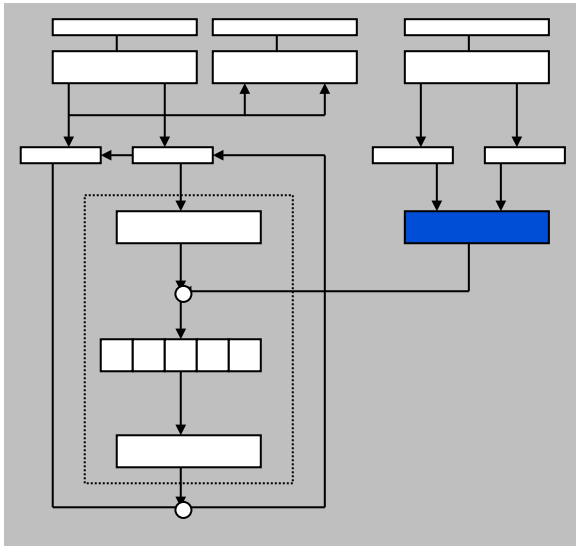


Anzahl der Shifts bei der Chiffrierung bzw. Deciffrierung

Rundennummer:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Links-Shifts:	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1 (Ver)
Rechts-Shifts:	0	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1 (Ent)



DES: PC2



Schlüsselauswahl (Permuted Choice 2, PC2)

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Eigenschaften des DES

- ⊗ Der DES ist **vollständig**: Jedes Output-Bit hängt von jedem Input-Bit ab.
- ⊗ Der DES ist derart **komplex**, dass keinerlei analytische Abhängigkeit zwischen Input und Output oder Schlüssel und Output feststellbar ist.
- ⊗ Der DES ist **invariant gegenüber Komplementbildung**, d.h.

$$\overline{\text{DES}(K, M)} = \text{DES}(\bar{K}, \bar{M})$$

- ⊗ Vier der 2^{56} Schlüssel sind **schwach**, d.h. $\text{DES}(K, \text{DES}(K, M)) = M$.

Externer Schlüssel	C-Register	D-Register
01 01 01 01 01 01 01 01	0000000	0000000
1F 1F 1F 1F 0E 0E 0E 0E	0000000	FFFFFFF
E0 E0 E0 E0 F1 F1 F1 F1	FFFFFFF	0000000
FE FE FE FE FE FE FE FE	FFFFFFF	FFFFFFF

⌘ Kritikpunkte

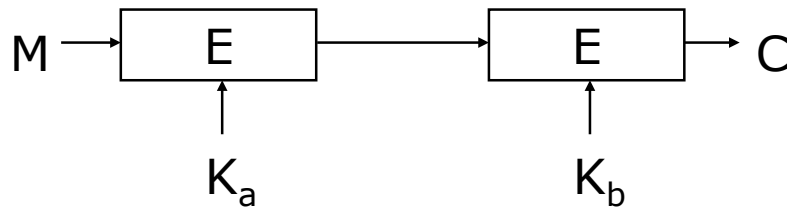
- ⊗ Designkriterien wurden nicht offengelegt (inzwischen bekannt)
- ⊗ wirksame Schlüssellänge heute viel zu gering (56 Bit)
- ⊗ nur ineffizient in Software implementierbar (wg. Permutationen)

⌘ 3-DES (Triple-DES)

- ⊗ Verbesserung der Sicherheit durch 3-fache Anwendung
 - » $S = \text{DES}(K1, \text{DES}(K2, \text{DES}(K1, M)))$

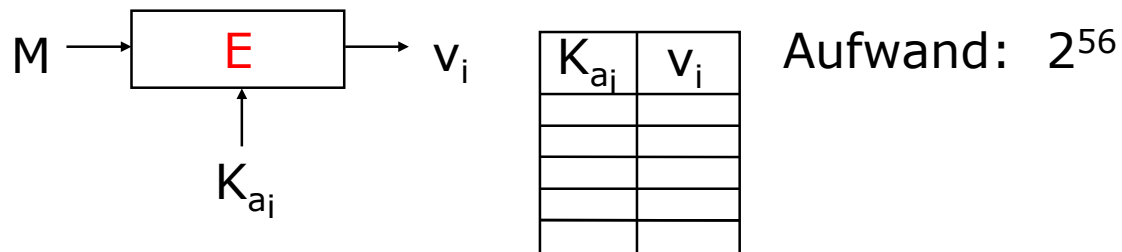
Möglicher Angriff bei »2-DES«

⌘ Ausgangssituation



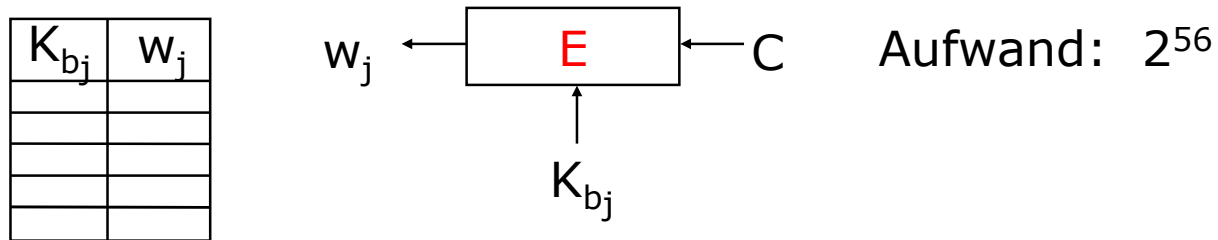
⌘ Known-plaintext-attack

1. Verschlüsse M für alle möglichen K_a und speichere die Schlüsseltexte v_i in einer Tabelle: $v_i = E(M, K_{a_i})$



Möglicher Angriff bei »2-DES«

2. Entschlüssele C für alle möglichen K_b und speichere die Klartexte w_j ebenfalls in einer Tabelle: $w_j = D(C, K_{b_j})$



3. Falls $v_i == w_j$ für ein bestimmtes Paar i und j , sind K_{a_i} und K_{b_j} die gesuchten Schlüssel, ggf. Probe mit weiteren M/C-Paaren machen!

⌘ Aufwand

⊗ $2^{56} + 2^{56} = 2 \cdot 2^{56} = 2^{57}$

⊗ **Sicherheitsgewinn wäre nur 1 Bit**

IDEA – International Data Encryption Algorithm

Lai, Massey 1991

⌘ Symmetrische Blockchiffre mit $M \in \{0,1\}^{64}$, $K \in \{0,1\}^{128}$

⌘ Operationen

⊕ bitweise Addition mod 2

⊕ Addition mod 2^{16}

⊙ Multiplikation mod $2^{16} + 1$ (0 wird durch 2^{16} dargestellt)

⌘ Ablauf

⊗ M wird in vier 16-Bit-Operanden $m_1 \dots m_4$ zerlegt.

⊗ Es werden $i=1\dots 8$ Runden durchlaufen.

⊗ Aus K werden sechs 16-Bit-Operanden $k_{i,1} \dots k_{i,6}$ erzeugt.

⌘ Teilschlüsselgenerierung

⊗ $K \rightarrow k_{1,1} \dots k_{1,6}, k_{2,1}, k_{2,2}$ (K wird in 8 Teile zerlegt.)

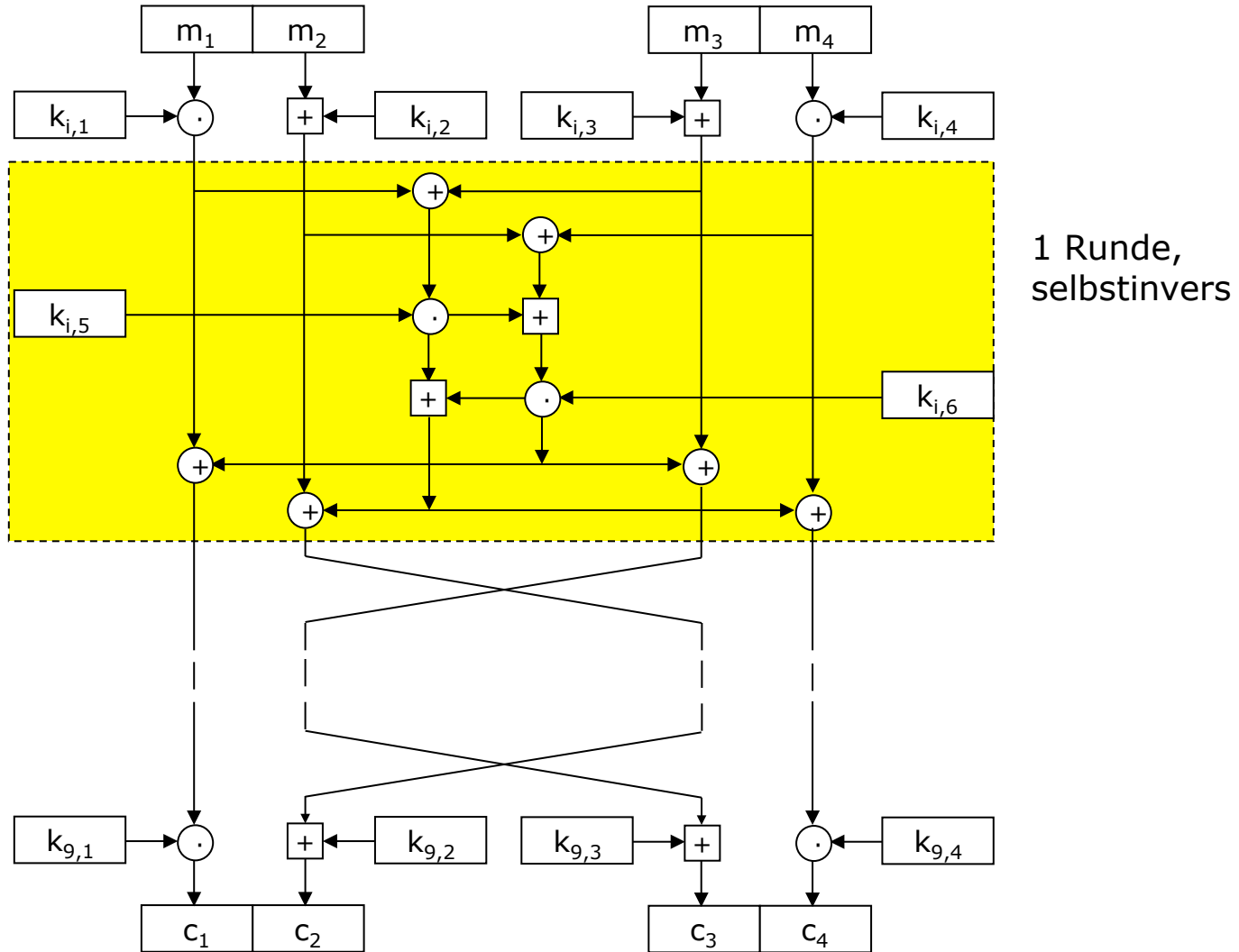
⊗ $\text{shiftLeft}(K, 25) \rightarrow k_{2,3} \dots k_{2,6}, k_{3,1} \dots k_{3,4}$

⊗ $\text{shiftLeft}(K, 25) \rightarrow k_{3,5}, k_{3,6}, k_{4,1} \dots k_{4,6}$

⊗ u.s.w

⊗ Nach jeder Erzeugung zyklische Linksverschiebung von K um 25 Bitstellen.

IDEA – International Data Encryption Algorithm



\oplus bitweise Addition mod 2

\boxplus Addition mod 2^{16}

\odot Multiplikation mod $2^{16} + 1$
(0 wird durch 2^{16} dargestellt)

IDEA – International Data Encryption Algorithm

⌘ Entschlüsselung

- ⊗ k_j sei Teilschlüssel zum Verschlüsseln in Runde j
- ⊗ d_j sei Teilschlüssel zum Entschlüsseln in Runde j
- ⊗ r_{\max} sei Rundenzahl (hier $r_{\max} = 8$)
- ⊗ $z = r_{\max} + 2$

$$\begin{array}{ll} d_{j,1} = (k_{z-j,1})^{-1} \bmod 2^{16}+1 & \text{mit } 1 \leq j \leq r_{\max} + 1 \\ d_{j,4} = (k_{z-j,4})^{-1} \bmod 2^{16}+1 & \text{mit } 1 \leq j \leq r_{\max} + 1 \\ d_{j,2} = (k_{z-j,2})^{-1} \bmod 2^{16} & \text{mit } j = 1, j = r_{\max} + 1 \\ d_{j,2} = (k_{z-j,3})^{-1} \bmod 2^{16} & \text{mit } 1 < j < r_{\max} + 1 \\ d_{j,3} = (k_{z-j,3})^{-1} \bmod 2^{16} & \text{mit } j = 1, j = r_{\max} + 1 \\ d_{j,3} = (k_{z-j,2})^{-1} \bmod 2^{16} & \text{mit } 1 < j < r_{\max} + 1 \\ d_{j,5} = (k_{z-(j+1),5}) & \text{mit } 1 \leq j \leq r_{\max} + 1 \\ d_{j,6} = (k_{z-(j+1),6}) & \text{mit } 1 \leq j \leq r_{\max} + 1 \end{array}$$

IDEA – International Data Encryption Algorithm

⌘ Designkriterien/Eigenschaften

- ⊗ Mischen verschiedenartiger Grundoperationen soll hohe Komplexität bereits nach wenigen Runden erreichen
- ⊗ Grundoperationen bewusst »inkompatibel« gewählt (erfüllen z.B. in keiner Kombination ein Distributiv- oder Assoziativgesetz)
- ⊗ hoher Grad an Immunität gegenüber differentieller Kryptanalyse (nach vier Runden immun)
- ⊗ bereits nach 1 Runde bzgl. der Inputbits vollständig, nach 2 Runden vollständig bzgl. der Schlüsselbits

⌘ Praktischer Einsatz

- ⊗ sehr gut in Hard- und Software implementierbar
- ⊗ sehr effizient
- ⊗ Für kommerzielle Anwendungen fallen Lizenzgebühren an.

Advanced Encryption Algorithm (AES)

⌘ Nachfolger des DES

- ⊗ Januar 1997 vom National Institute of Standards and Technology (NIST) als Nachfolger für DES initiiert
- ⊗ öffentliche internationale Ausschreibung

⌘ Neue Blockchiffre sollte folgende Kriterien erfüllen:

- ⊗ symmetrische Blockchiffre mit einer Blockgröße von 128 Bit und variabler Schlüssellänge von 128, 192 und 256 Bit.
- ⊗ AES soll für mindestens 30 Jahre Sicherheit bieten.
- ⊗ Weder Algorithmus noch Implementierung dürfen patentiert sein.
- ⊗ Spezifikation: <http://csrc.nist.gov/encryption/aes/>

⌘ August 1998 wurden 15 Kandidaten der Öffentlichkeit zur Begutachtung vorgelegt.

Advanced Encryption Algorithm (AES)

⌘ August 1999 wurden die 5 Finalisten vorgestellt:

- ⊗ MARS – IBM
- ⊗ RC6 – RSA Labs
- ⊗ Rijndael – Joan Daemen (Proton World Intl.), Vincent Rijmen (Katholieke Universiteit Leuven, Belgien)
- ⊗ Serpent – Ross Anderson (Univ of Cambridge), Eli Biham (Technion), Lars Knudsen (UC San Diego)
- ⊗ Twofish – Bruce Schneider, John Kelsey, Niels Ferguson (Counterpane Internet Security), Doug Whiting (Hi/fn, Inc.), David Wagner (UC Berkeley), Chris Hall (Princeton Univ.)

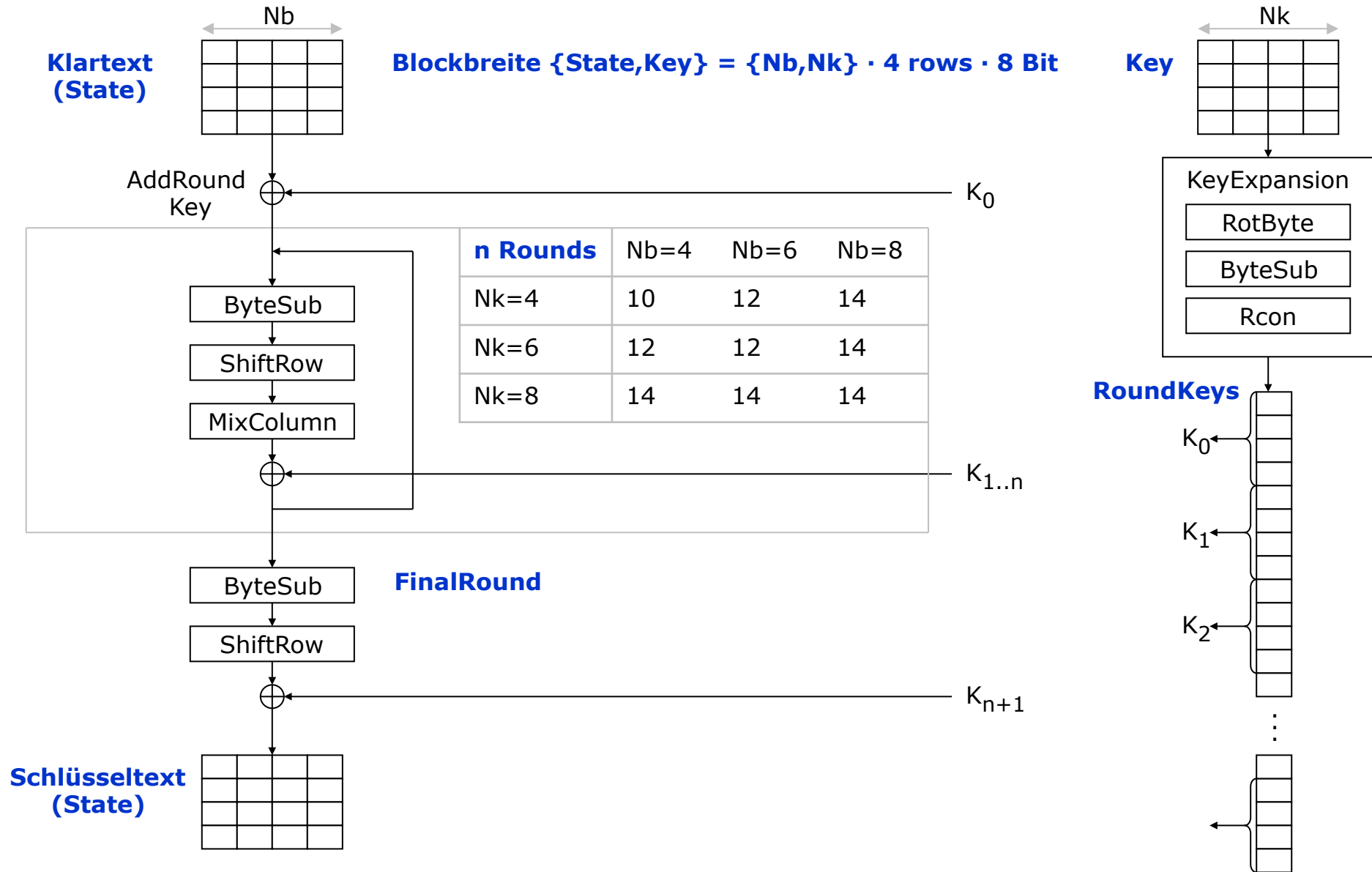
⌘ Oktober 2000:

- ⊗ Rijndael wird ausgewählt.

⌘ Begründung für Rijndael

- ⊗ Beste Kombination von Sicherheit, Leistungsfähigkeit, Effizienz und Implementierbarkeit sowohl in Software als auch in Hardware.

Advanced Encryption Standard (AES): Rijndael



Advanced Encryption Standard (AES)

⌘ Rijndael

- ⊗ <http://www.esat.kuleuven.ac.be/~rijmen/rijndael/>
- ⊗ sprich: "Rein-dahl"

⌘ Blockchiffre

- ⊗ keine Feistel-Chiffre, arbeitet aber in Runden
- ⊗ Rundentransformation besteht aus drei invertierbaren Transformationen
- ⊗ variable Blocklänge und variable Schlüssellänge, jeweils unabhängig wählbar aus {128 Bit, 192 Bit, 256 Bit}.
- ⊗ Blockbreite {Nachrichtenblock, Schlüssel} in Bit
= {**Nb**, **Nk**} · 8 Bit · 4 rows
- ⊗ Beispiel: Nb = 6 und Nk = 4

State

a _{0,0}	a _{0,1}	a _{0,2}	a _{0,3}	a _{0,4}	a _{0,5}
a _{1,0}	a _{1,1}	a _{1,2}	a _{1,3}	a _{1,4}	a _{1,5}
a _{2,0}	a _{2,1}	a _{2,2}	a _{2,3}	a _{2,4}	a _{2,5}
a _{3,0}	a _{3,1}	a _{3,2}	a _{3,3}	a _{3,4}	a _{3,5}

Cipher Key

k _{0,0}	k _{0,1}	k _{0,2}	k _{0,3}
k _{1,0}	k _{1,1}	k _{1,2}	k _{1,3}
k _{2,0}	k _{2,1}	k _{2,2}	k _{2,3}
k _{3,0}	k _{3,1}	k _{3,2}	k _{3,3}

Rijndael (AES)

⊗ Rundenzahl Nr ist eine Funktion von Nb und NK

Nr	Nb=4	Nb=6	Nb=8
Nk=4	10	12	14
Nk=6	12	12	14
Nk=8	14	14	14

```
Rijndael(State,CipherKey) {  
    KeyExpansion(CipherKey,ExpandedKey);  
    AddRoundKey(State,ExpandedKey);  
    For(i=1;i<Nr;i++)  
        Round(State,ExpandedKey+Nb*i);           // Pointer !  
    FinalRound(State,ExpandedKey+Nb*Nr);        // Pointer !  
}
```

⌘ Rundentransformationen

```
Round(State, RoundKey) {  
    ByteSub(State);  
    ShiftRow(State);  
    MixColumn(State);  
    AddRoundKey(State, RoundKey);  
}
```

```
FinalRound(State, RoundKey) {  
    // wie Round, aber ohne MixColumn  
    ByteSub(State);  
    ShiftRow(State);  
    AddRoundKey(State, RoundKey);  
}
```

⌘ ByteSub

- ⊠ operiert auf jedem Byte von State unabhängig
- ⊠ ist eine S-Box-Transformation

1. berechne das Multiplikative Inverse in $GF(2^8)$

2. berechne:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

⊠ Umkehroperation:

⊕ Inverse Tabelle und anschließend Berechnung des Multiplikativen Inversen in $GF(2^8)$

⊠ ByteSub kann als Tabelle vorberechnet werden.

Rijndael (AES)

⌘ ByteSub

⊠ ByteSub kann als Tabelle vorberechnet werden.

Input
unteres
Halbbyte →

Input
oberes
Halbbyte ↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Rijndael (AES)

⌘ ByteSub

⊠ substituiert die Bytes von State unabhängig voneinander

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Rijndael (AES)

⌘ ShiftRow

⊠ Anzahl der zyklischen Linksshifts in Abhängigkeit von Nb

	row 0	row 1	row 2	row 3
Nb=4	0	1	2	3
Nb=6	0	1	2	3
Nb=8	0	1	3	4

⊠ Beispiel: Nb=6

row 0: no shift	<table border="1"><tr><td>$a_{0,0}$</td><td>$a_{0,1}$</td><td>$a_{0,2}$</td><td>$a_{0,3}$</td><td>$a_{0,4}$</td><td>$a_{0,5}$</td></tr></table>	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	<table border="1"><tr><td>$a_{0,0}$</td><td>$a_{0,1}$</td><td>$a_{0,2}$</td><td>$a_{0,3}$</td><td>$a_{0,4}$</td><td>$a_{0,5}$</td></tr></table>	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$									
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$									
row 1: 1 shift	<table border="1"><tr><td>$a_{1,0}$</td><td>$a_{1,1}$</td><td>$a_{1,2}$</td><td>$a_{1,3}$</td><td>$a_{1,4}$</td><td>$a_{1,5}$</td></tr></table>	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	<table border="1"><tr><td>$a_{1,1}$</td><td>$a_{1,2}$</td><td>$a_{1,3}$</td><td>$a_{1,4}$</td><td>$a_{1,5}$</td><td>$a_{1,0}$</td></tr></table>	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$									
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$									
row 2: 2 shift	<table border="1"><tr><td>$a_{2,0}$</td><td>$a_{2,1}$</td><td>$a_{2,2}$</td><td>$a_{2,3}$</td><td>$a_{2,4}$</td><td>$a_{2,5}$</td></tr></table>	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	<table border="1"><tr><td>$a_{2,2}$</td><td>$a_{2,3}$</td><td>$a_{2,4}$</td><td>$a_{2,5}$</td><td>$a_{2,0}$</td><td>$a_{2,1}$</td></tr></table>	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$									
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$									
row 3: 3 shift	<table border="1"><tr><td>$a_{3,0}$</td><td>$a_{3,1}$</td><td>$a_{3,2}$</td><td>$a_{3,3}$</td><td>$a_{3,4}$</td><td>$a_{3,5}$</td></tr></table>	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	<table border="1"><tr><td>$a_{3,3}$</td><td>$a_{3,4}$</td><td>$a_{3,5}$</td><td>$a_{3,0}$</td><td>$a_{3,1}$</td><td>$a_{3,2}$</td></tr></table>	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$									
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$									
	vorher	nachher												

Rijndael (AES)

⌘ MixColumn

- ⊠ operiert auf allen Spalten von State
- ⊠ Berechne

$$b(x) = a(x) \otimes c(x) \pmod{x^4+1}$$

mit $c(x) = '03' x^3 + '01' x^2 + '01' x + '02'$

- ⊠ d.h.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$	$b_{0,4}$	$b_{0,5}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	$b_{2,5}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$



⌘ MixColumn

⊠ Inverse Operation:

$$a(x) = b(x) \otimes d(x) \pmod{x^4+1}$$

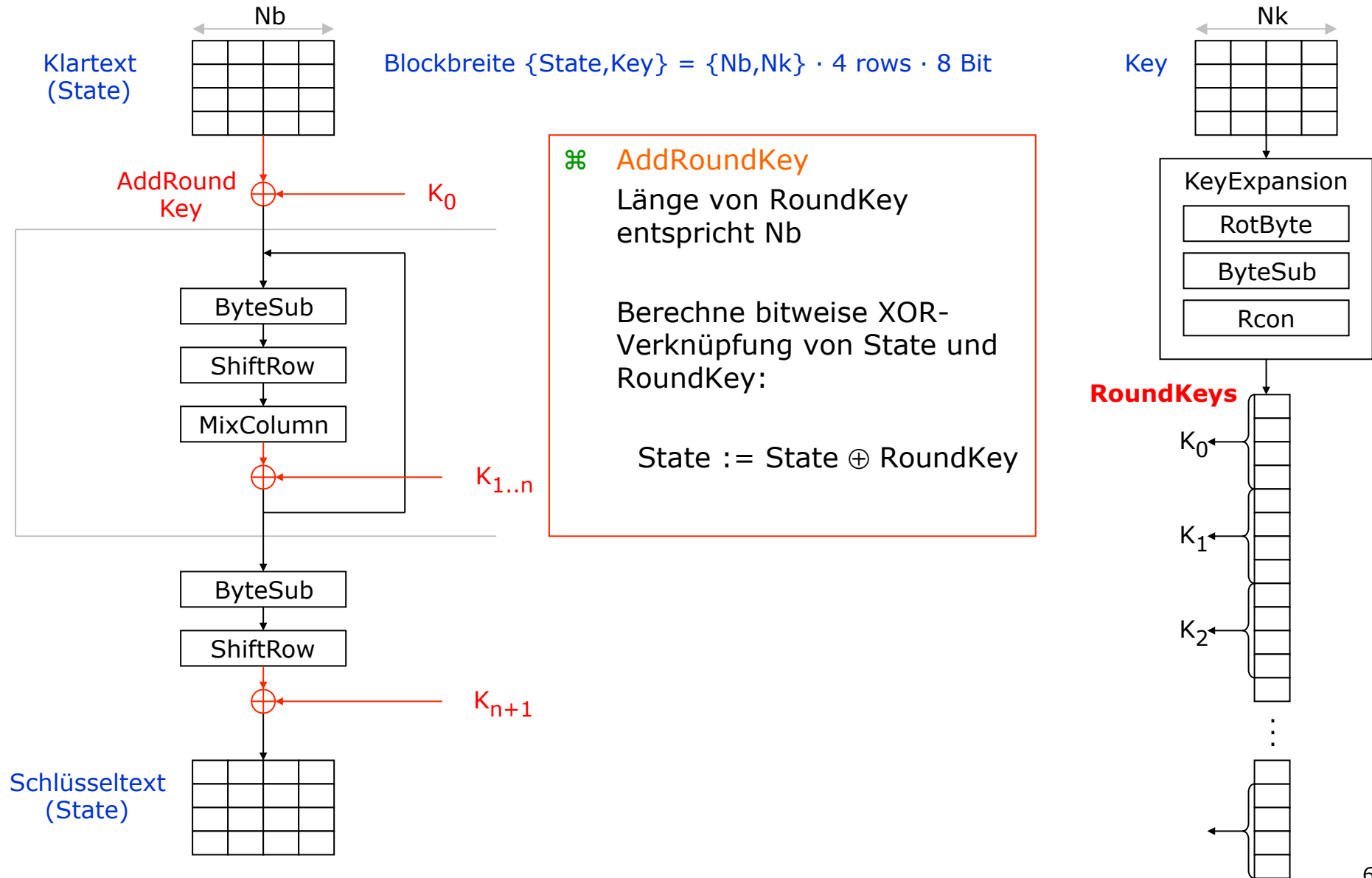
mit $d(x) = '0B' x^3 + '0D' x^2 + '09' x + '0E'$,
 da $('03' x^3 + '01' x^2 + '01' x + '02') \otimes d(x) = '01'$
 (neutrales Element bzgl. Mult.)

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$	$b_{0,4}$	$b_{0,5}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	$b_{2,5}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$



Rijndael (AES)



Rijndael (AES)

⌘ KeyExpansion

⊗ für $N_k \leq 6$:

```
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)]){
    for(i = 0; i < Nk; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = W[i - 1];
        if (i % Nk == 0)
            temp = ByteSub(RotByte(temp)) ^ Rcon[i / Nk];
        W[i] = W[i - Nk] ^ temp;
    }
}
```

⊗ für $N_k > 6$:

```
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)]) {
    for(i = 0; i < Nk; i++)
        W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++) {
        temp = W[i - 1];
        if (i % Nk == 0)
            temp = ByteSub(RotByte(temp)) ^ Rcon[i / Nk];
        else if (i % Nk == 4)
            temp = ByteSub(temp);
        W[i] = W[i - Nk] ^ temp;
    }
}
```

Rijndael (AES)

- ⊗ **RotByte**: zyklische Schiebeoperation (byteweise links)
- ⊗ **ByteSub** (wie bei Rundentransformation)
- ⊗ **Rcon**[i] = (RC[i], '00','00','00')
RC[1] = 1
RC[2] = x · (RC[i-1]) = x(i-1)

⌘ RoundKey Selection

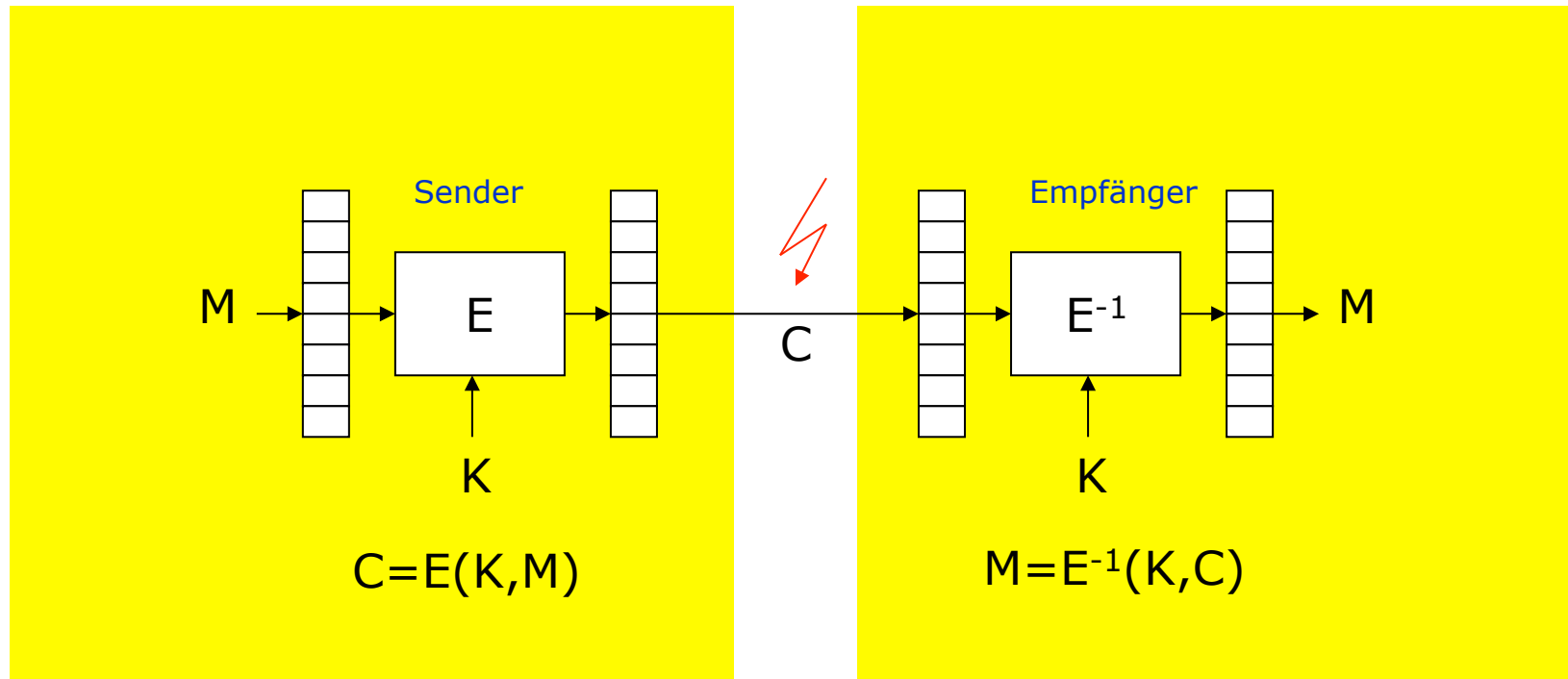
- ⊗ fortlaufende Auswahl
- ⊗ Beispiel für Nb = 6 und Nk = 4:

W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	...
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----

Round Key 0	Round Key 1	...
-------------	-------------	-----

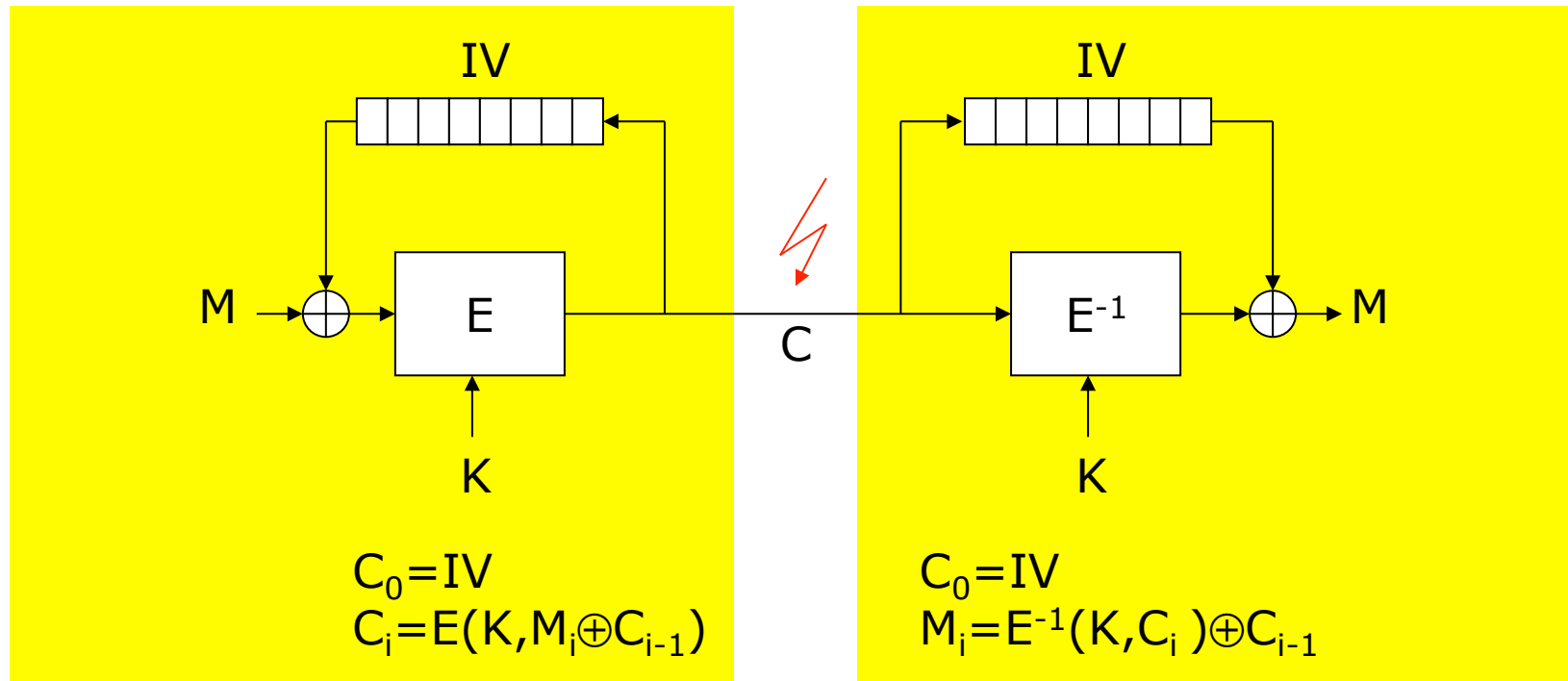
Betriebsarten von Blockchiffren

⌘ Electronic Code Book (ECB)



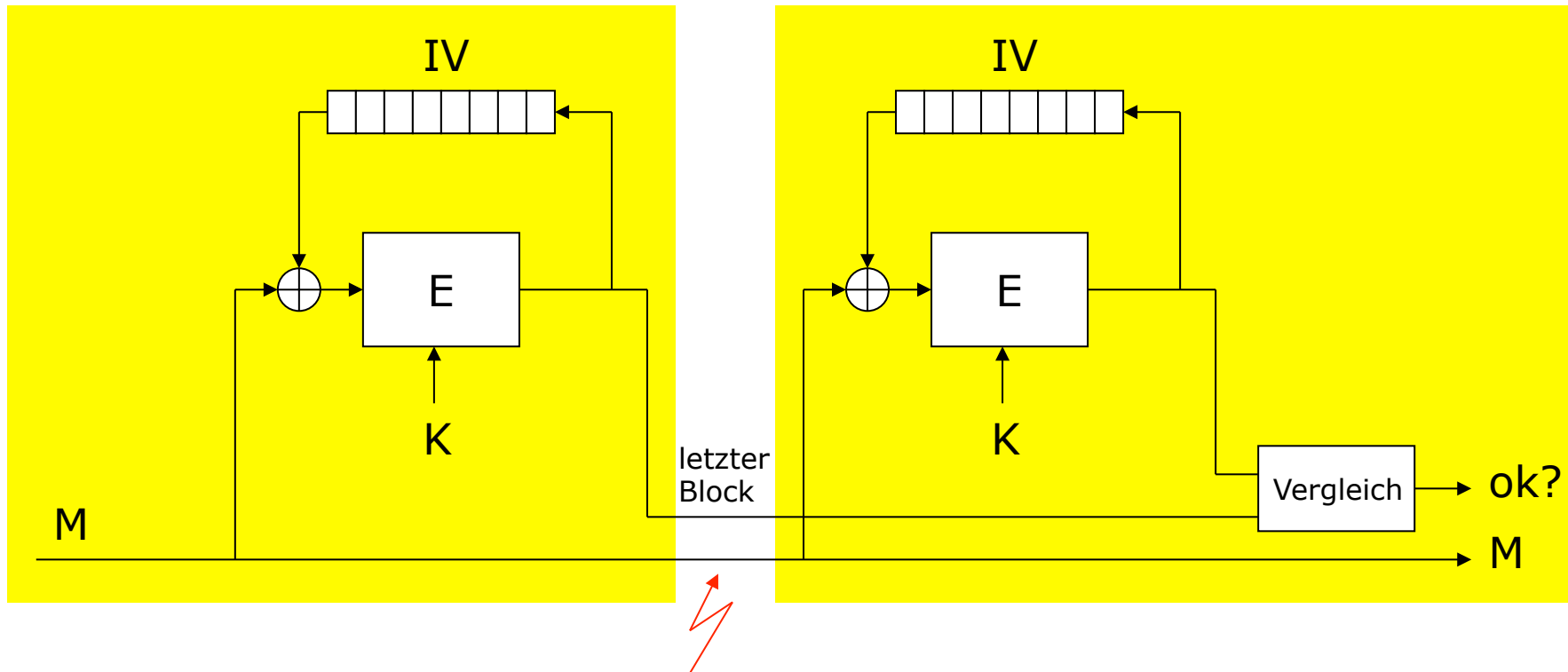
Betriebsarten von Blockchiffren

⌘ Cipher Block Chaining (CBC)



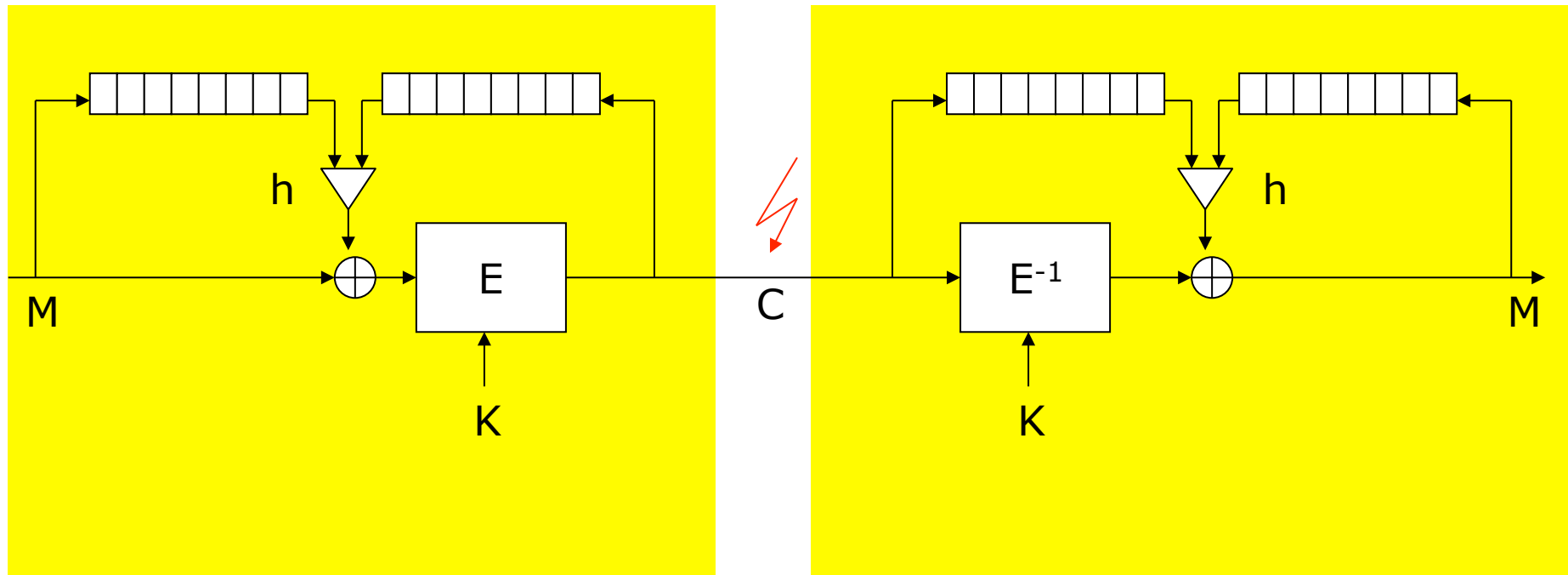
Betriebsarten von Blockchiffren

⌘ CBC zur Authentikation



Betriebsarten von Blockchiffren

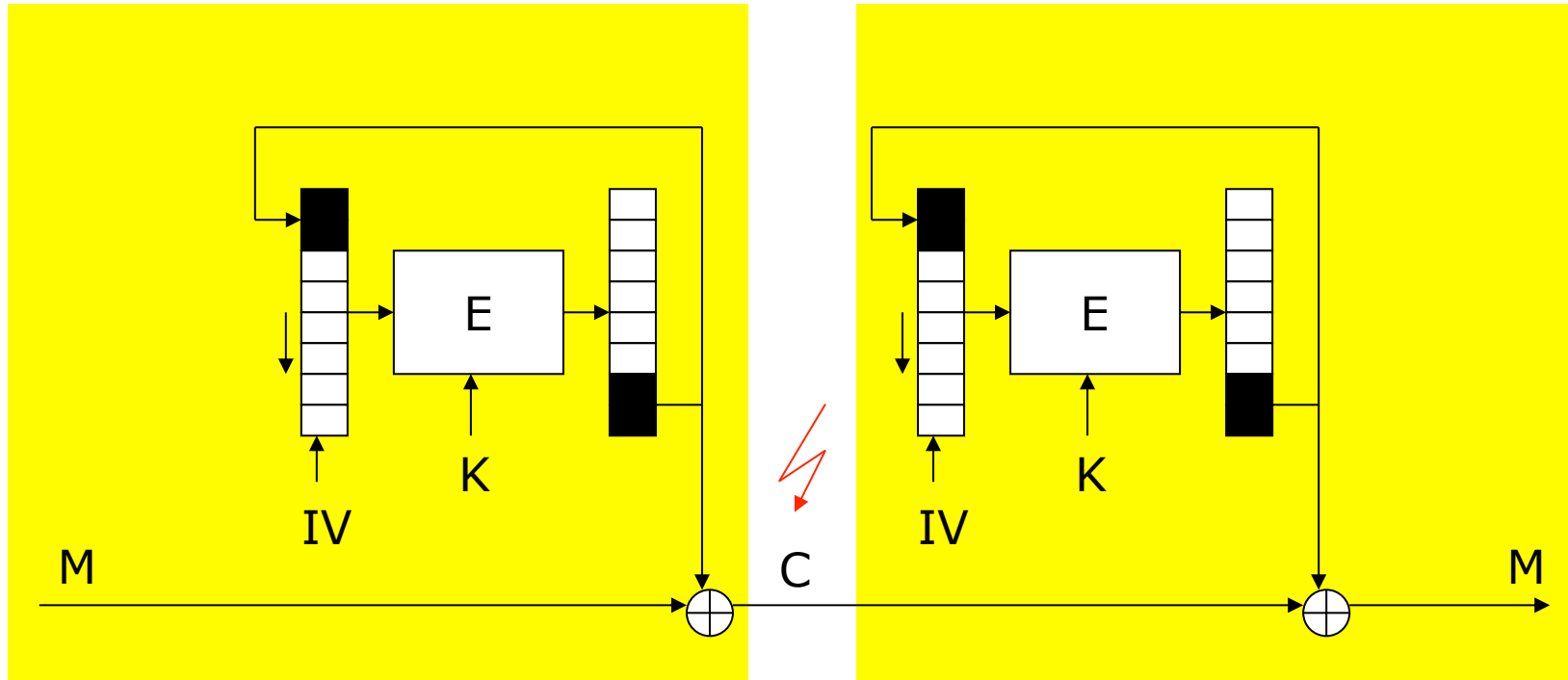
⌘ PCBC (Plain Cipher Block Chaining)



h : beliebige Funktion, z.B. Addition mod $2^{\text{Blocklänge}}$
gleichzeitig Authentisierung und Verschlüsselung
letzter Block ist Redundanzblock für Authentifikation

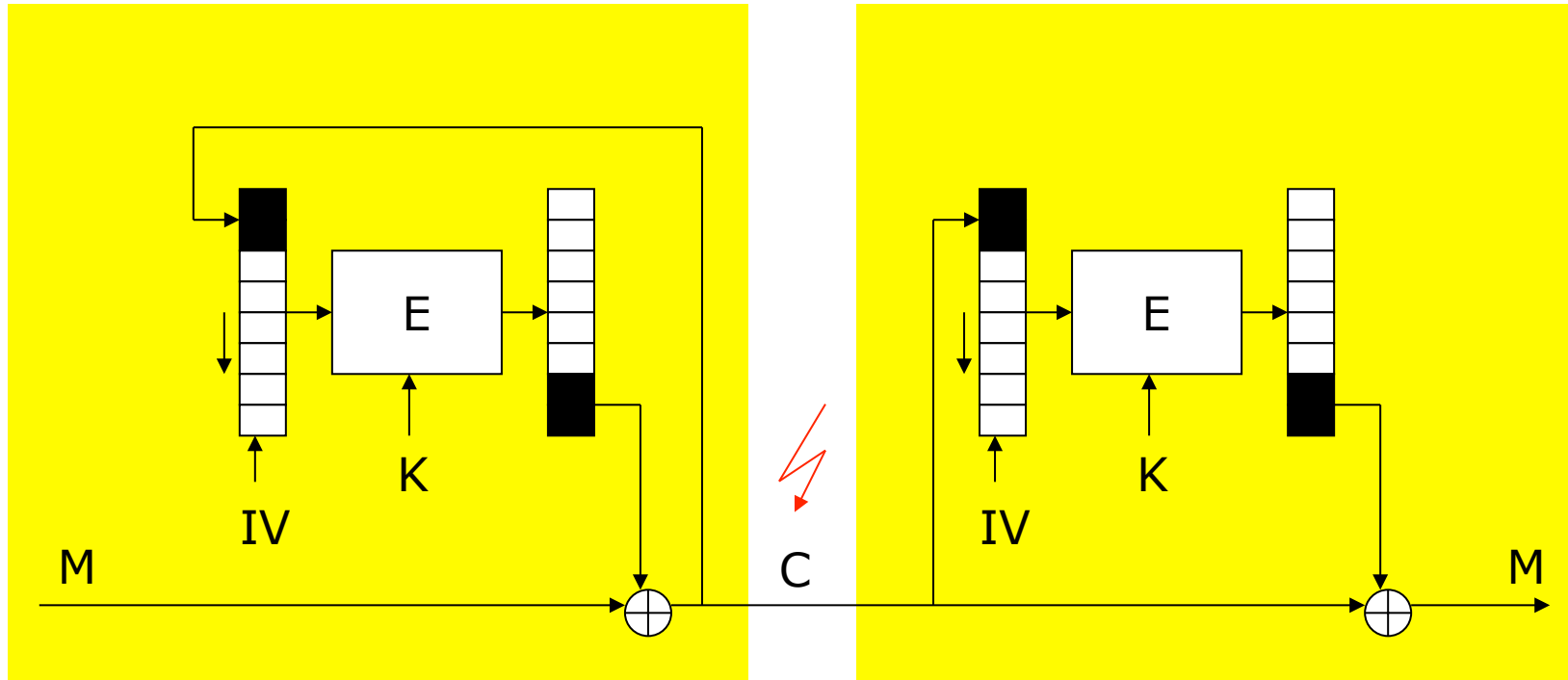
Betriebsarten von Blockchiffren

⌘ Output Feedback (OFB)



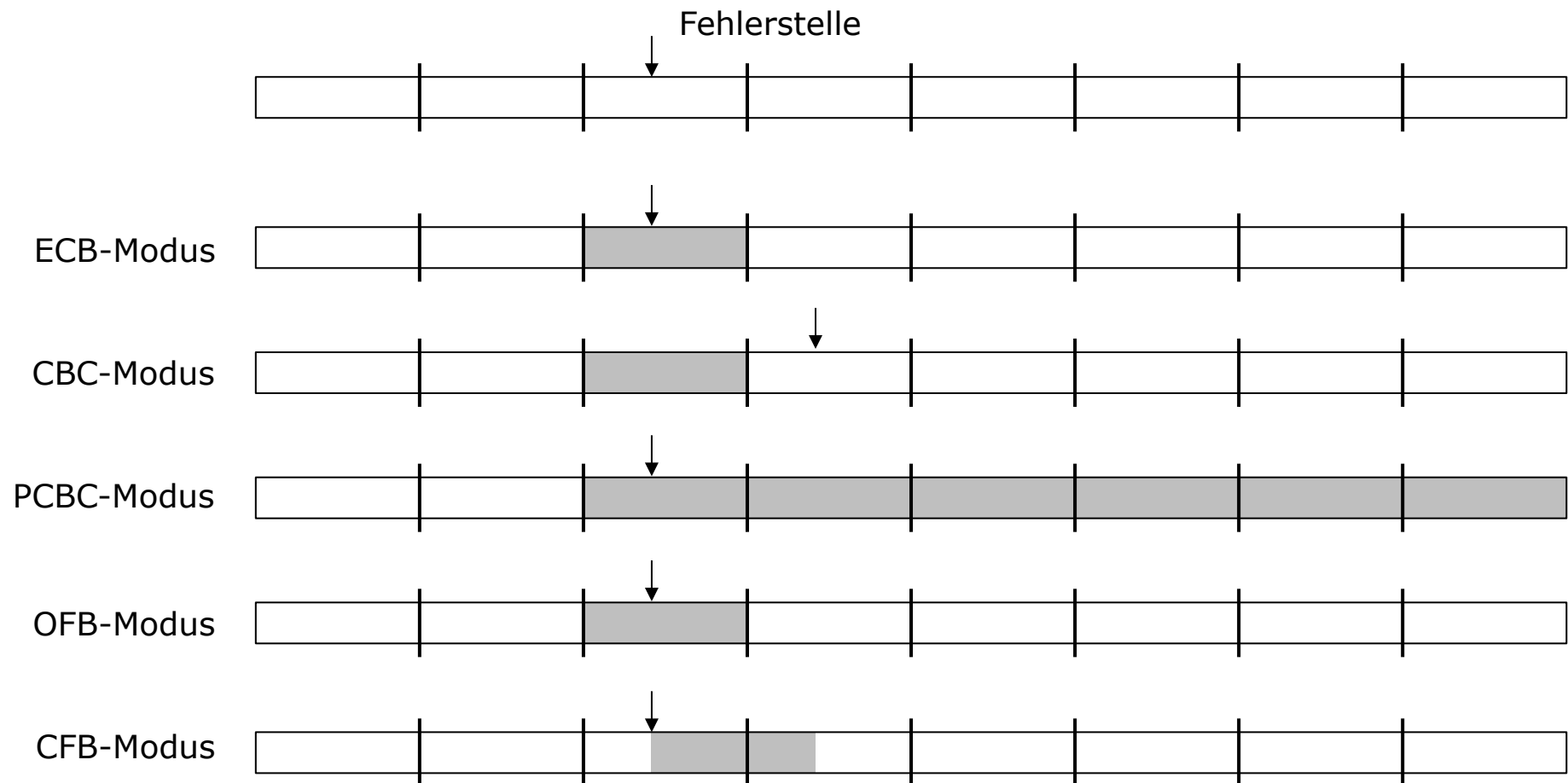
Betriebsarten von Blockchiffren

⌘ Cipher Feedback (CFB)



Betriebsarten von Blockchiffren

⌘ Fehlerfortpflanzung



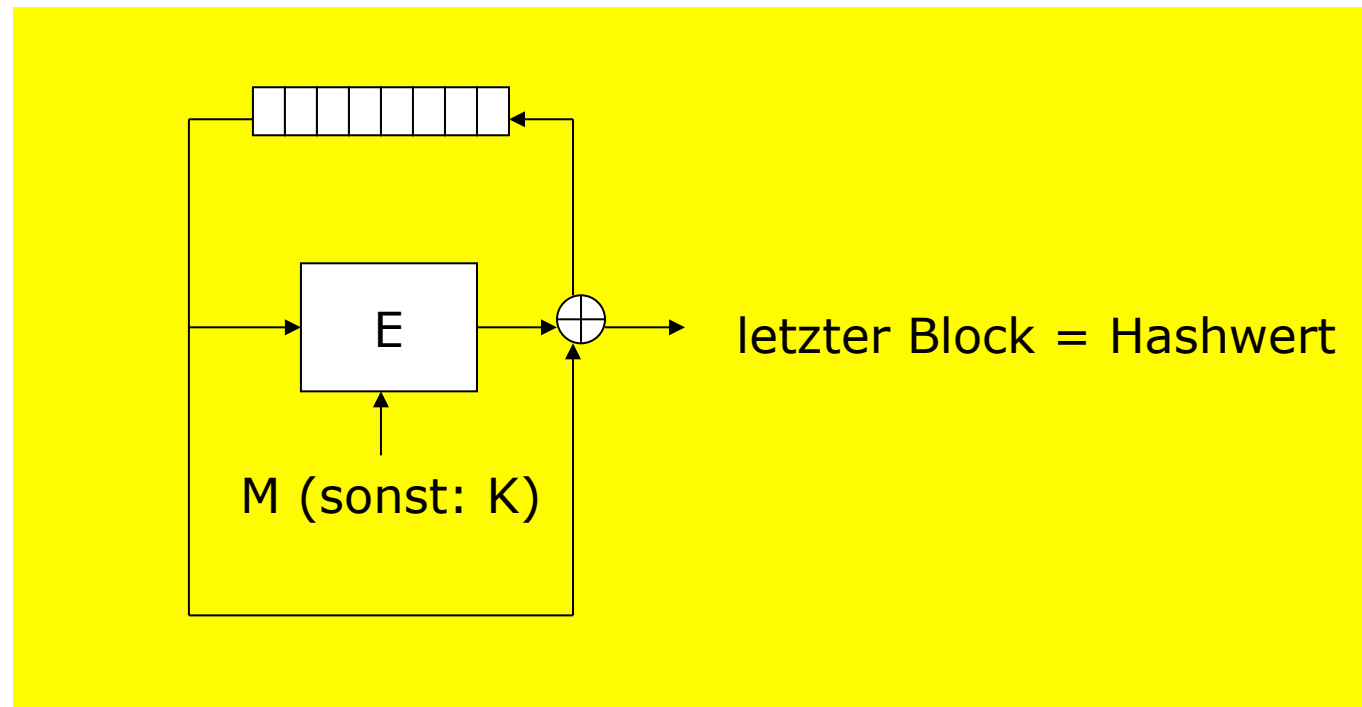
Betriebsarten von Blockchiffren

Modus	Vorteile	Nachteile
ECB	<ul style="list-style-type: none">• Direktzugriff möglich• keine Fehlerfortpflanzung bei additiven Fehlern	<ul style="list-style-type: none">• Fehlerfortpflanzung in alle nachfolgenden Blöcke bei Synchronisationsfehlern• gezieltes Einfügen und Entfernen von Blöcken möglich• gleiche Klartextblöcke liefern gleiche Chiffretextblöcke• Codebuchanalyse möglich
CBC	<ul style="list-style-type: none">• gleiche Klartextblöcke liefern unterschiedliche Chiffretextblöcke• Manipulationen sind erkennbar• Kryptoanalyse erschwert gegenüber ECB-Modus	<ul style="list-style-type: none">• kein Direktzugriff möglich• Fehlerfortpflanzung in den Folgeblock bei additiven Fehlern• anfällig gegen Synchronisationsfehler
OFB	<ul style="list-style-type: none">• keine Fehlerfortpflanzung bei additiven Fehlern	<ul style="list-style-type: none">• geringere Verschlüsselungsrate pro DES-Aufruf als ECB- und CBC-Modus (abh. von Bitbreite)• kein Direktzugriff möglich• unerkennbare Manipulationen sind u.U. möglich• anfällig gegen Synchronisationsfehler
CFB	<ul style="list-style-type: none">• Schlüsselstrom abhängig von Klartextstrom• Kryptoanalyse erschwert gegenüber OFB-Modus• Manipulationen sind erkennbar• selbstsynchronisierender Modus	<ul style="list-style-type: none">• geringere Verschlüsselungsrate pro DES-Aufruf als ECB- und CBC-Modus (abh. von Bitbreite)• kein Direktzugriff möglich• Fehlerfortpflanzung

Konstruktionen aus einer symmetrischen Blockchiffre

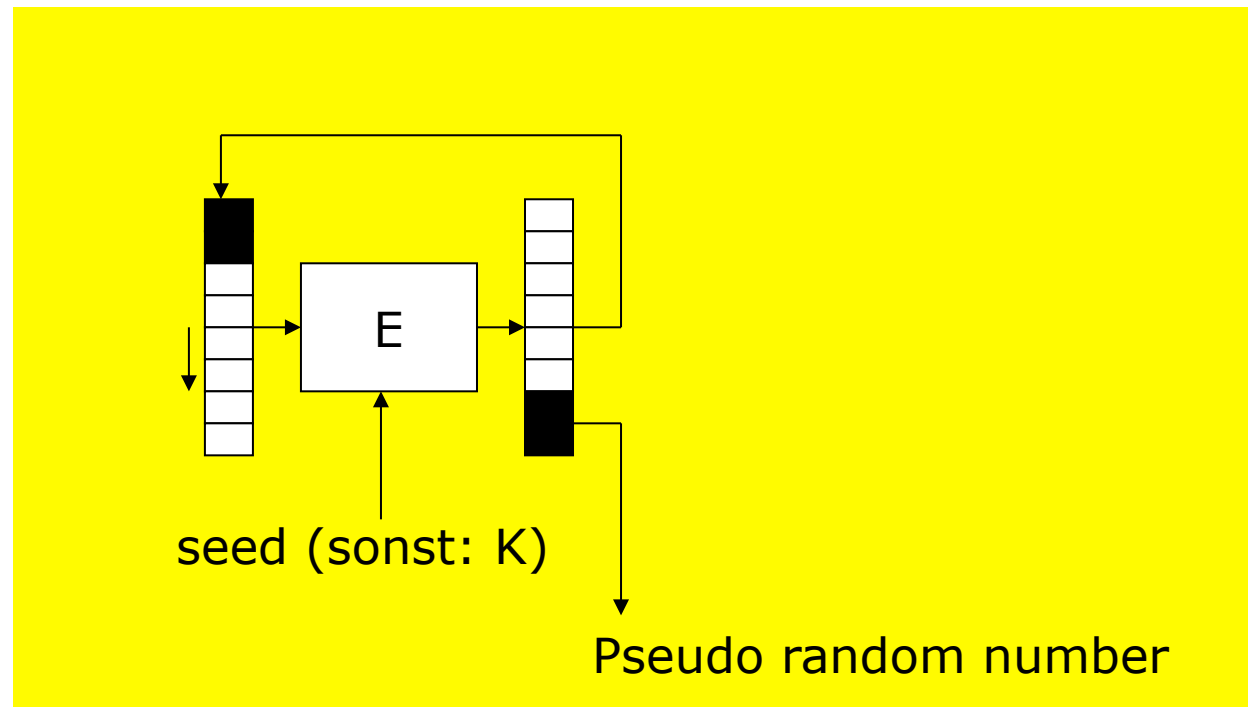
⌘ Hashfunktion

- ⊠ Aus Sicherheitsgründen sollte die Schlüssellänge nicht wesentlich länger sein als die Blocklänge



Konstruktionen aus einer symmetrischen Blockchiffre

⌘ Pseudozufallszahlengenerator



Mathematische Grundlagen asymmetrischer Systeme

⌘ Modulo-Rechnung

- ⊠ Grundlagen
- ⊠ Erweiterter Euklidischer Algorithmus

⌘ Systeme auf der Basis des diskreten Logarithmus

- ⊠ primitive Wurzel
- ⊠ diskreter Logarithmus Problem

⌘ Systeme auf der Basis der Faktorisierungsannahme

- ⊠ Faktorisierungsannahme
- ⊠ Primzahlzerlegung

Erweiterter Euklidischer Algorithmus

⌘ Anwendung

- ⊗ zur Bestimmung von $\text{ggT}(a,b)$ und
- ⊗ zur Ermittlung der **multiplikativen Inversen** der Zahl b im Restklassenring modulo a , d. h. zur Berechnung von b^{-1} aus der Beziehung $b \cdot b^{-1} = 1 \pmod{a}$.

⌘ Algorithmus

- ⊗ Seien $a, b \in \mathbf{N}+1$, $a > b$.

- ⊗ Setze
$$\begin{array}{lll} r_{-2} = a & s_{-2} = 0 & t_{-2} = 1 \\ r_{-1} = b & s_{-1} = 1 & t_{-1} = 0. \end{array}$$

- ⊗ Berechne c_k, r_k, s_k und t_k nach folgenden Beziehungen:

$$\begin{aligned} c_k &= r_{k-2} \text{ DIV } r_{k-1} \\ r_k &= r_{k-2} \text{ MOD } r_{k-1} \\ s_k &= c_k s_{k-1} + s_{k-2} \\ t_k &= c_k t_{k-1} + t_{k-2}. \end{aligned}$$

- ⊗ Abbruchbedingung : $r_k = 0$.

- ⊗ Es gilt: $b \cdot s_{k-1} - a \cdot t_{k-1} = (-1)^k \cdot r_{k-1}$

- ⊗ Ergebnisse:
$$\begin{aligned} r_{k-1} &= \text{ggT}(a,b) \\ s_{k-1} \cdot b &= (-1)^k \pmod{a}, \text{ falls } \text{ggT}(a,b) = 1. \end{aligned}$$

Erweiterter Euklidischer Algorithmus

$$c_k = r_{k-2} \text{ DIV } r_{k-1} \quad r_k = r_{k-2} \text{ MOD } r_{k-1} \quad s_k = c_k s_{k-1} + s_{k-2} \quad t_k = c_k t_{k-1} + t_{k-2}$$

⌘ Beispiel 1

⊠ Gegeben: $a=10$ $b=4$

⊠ Gesucht: $\text{ggt}(a,b)$

k	c_k	r_k	s_k	t_k
-2		10 = a	0	1
-1		4 = b	1	0
0	2	2 = ggt	2	1
1	2	0 (Abbruch)		

Erweiterter Euklidischer Algorithmus

$$c_k = r_{k-2} \text{ DIV } r_{k-1} \quad r_k = r_{k-2} \text{ MOD } r_{k-1} \quad s_k = c_k s_{k-1} + s_{k-2} \quad t_k = c_k t_{k-1} + t_{k-2}$$

⌘ Beispiel 2

⊗ Gegeben:

$$p=53 \quad q=61 \quad n=p \cdot q=3233 \quad \Phi(n)=(p-1) \cdot (q-1)=3120 \quad c=523$$

⊗ Gesucht:

$c \cdot c^{-1} = 1 \pmod{\Phi(n)}$, d.h. Multiplikatives Inverses zu $c \pmod{\Phi(n)}$

k	c_k	r_k	s_k	t_k
-2		3120 = a = Φ	0	1
-1		523 = b = c	1	0
0	5	505	5	1
1	1	18	6	1
2	28	1 = ggt	173	29
3	18	0 (Abbruch)		

Erweiterter Euklidischer Algorithmus

⌘ Beispiel 2 (Forts.)

$$\begin{array}{rcll} \boxtimes \text{ Es gilt: } & s_{k-1} \cdot b & = & (-1)^k \pmod{a} \\ & 173 \cdot 523 & = & (-1)^3 \pmod{3120} \\ & 90479 & = & -1 \pmod{3120} \\ & 3119 & = & -1 \pmod{3120} \\ & -1 & = & -1 \pmod{3120} \end{array}$$

⊗ Da $(-1)^k = (-1)^3 = -1$, muss noch mit -1 multipliziert werden.

$$\begin{array}{rcll} & -s_{k-1} \cdot b & = & -(-1)^k \pmod{a} \\ & c & = & -s_{k-1} = -173 = -173 + a \\ & c & = & \underline{\underline{2947}} \end{array}$$

Eulersche- Φ -Funktion

⌘ Def. Eulersche Φ -Funktion

⊗ Für eine beliebige ganze Zahl n bildet die Menge \mathbf{Z}_n^* der ganzen Zahlen modulo n , die zu n teilerfremd sind, eine multiplikative Gruppe. Die Ordnung dieser Gruppe ist $\Phi(n)$.

⊕ **Beispiel:** $\Phi(9) = 6$ $\mathbf{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$

⊗ Für den Sonderfall $n = p \in \mathbf{P}$ gilt $\Phi(p) = p-1$.

⌘ Satz von Euler

⊗ Für ein beliebiges x mit $(1 \leq x < n)$, das zu n teilerfremd ist bzw. $x \in \mathbf{Z}_n^*$, gilt $x^{\Phi(n)} = 1 \pmod n$.

⌘ Euler-Fermat-Identität (kleiner Satz von Fermat)

⊗ Mit $\Phi(p) = p-1$ ($p \in \mathbf{P}$) folgt aus dem Satz von Euler:
 $x^{p-1} = 1 \pmod p$ $(1 \leq x \leq p-1)$.

⌘ Wenn n das Produkt zweier Primzahlen $n = p \cdot q$ ist, gilt

$$x^{\Phi(p \cdot q)} = x^{(p-1) \cdot (q-1)} \pmod{p \cdot q}.$$

Primitive Wurzel

⌘ Definition

- ⊗ Eine beliebige ganze Zahl im Bereich $1 \leq a < n$ heißt primitive Wurzel, wenn gilt
 $\text{ggT}(a, n) = 1$ und
 $a^d \neq 1 \pmod{n}$ für alle d mit der Bedingung $1 \leq d < \Phi(n)$

⌘ Theorem

- ⊗ Die ganze Zahl n hat genau dann eine primitive Wurzel, wenn $n = 1, 2$ oder 4 ist oder die Form p^k oder $2p^k$ hat, wobei p eine ungerade Primzahl ist.
- ⊗ Wenn n eine primitive Wurzel hat, dann hat n genau $\Phi(\Phi(n))$ primitive Wurzeln.

⌘ Vermutung

- ⊗ Jede ganze Zahl, die keine Quadratzahl ist, ist die primitive Wurzel einer Primzahl.

Diskreter Logarithmus

⌘ Definition diskreter Logarithmus

- ⊗ Sei p eine beliebige ganze Zahl, die eine primitive Wurzel a hat. Wenn für ein beliebiges c mit $0 \leq c < \Phi(p)$

$$b = a^c \pmod{p}$$

gilt, dann ist c der diskrete Logarithmus zur Basis a modulo p oder auch

$$c = \log_a b \pmod{p}.$$

⌘ Problemstellung für den Angreifer

- ⊗ Öffentlich bekannt sind a , b , p .
- ⊗ Geheim ist c . Erfährt ein Angreifer c , ist das System gebrochen.
- ⊗ Folglich möchte ein Angreifer c ermitteln.

⌘ Beispiel

- ⊕ $p = 3137$ und $a = 577$ öffentlich; $c = 1374$ geheim
- ⊕ $b = a^c \pmod{p} = 858$ öffentlich
- ⊕ $c = \log_a b \pmod{p} = \log_{577} 858 \pmod{3137} = ?$ (Angreifersicht)

Diskreter Logarithmus

⌘ Algorithmen zur Berechnung des diskreten Logarithmus

⊠ Baby-Step, Giant-Step, Index-Calculus-Alg., Adleman-Alg.

⊠ Laufzeit zur Berechnung des diskreter Log. mod p mit $p \in \mathbf{P}$
 $e^{(1+O(1))(\log p \cdot \log(\log p))^{1/2}}$

⊠ Rechenzeiten bei 10^8 Operationen pro sek. für verschiedene Größenordnungen von p :

p	Anzahl Ops.	benötigte Zeit in Jahren
$\approx 10^{100}$	$7,9 \cdot 10^{22}$	$2,5 \cdot 10^7$
$\approx 10^{200}$	$1,8 \cdot 10^{34}$	$5,7 \cdot 10^{19}$
$\approx 10^{300}$	$1,8 \cdot 10^{43}$	$5,7 \cdot 10^{28}$
$\approx 10^{400}$	$9,5 \cdot 10^{50}$	$4,8 \cdot 10^{36}$
$\approx 10^{500}$	$7,4 \cdot 10^{57}$	$4,8 \cdot 10^{43}$

Vergleich: Logarithmus

$\log_a b$ ($a > 0$; $a \neq 1$; $b > 0$) ist diejenige reelle Zahl c , für die gilt $a^c = b$.

$c = \log_a b$ wird z.B. gelöst mit $\log_a b = \frac{\log_x b}{\log_x a}$

- ⌘ **Beispiel 1:** Wieviel Bit benötigt man, um die Zahlen zwischen 0 und 255 binär zu kodieren?

$$\log_2 256 = \frac{\ln 256}{\ln 2} = 8 \text{ Bit}$$

- ⌘ **Beispiel 2:** Wieviel Bit benötigt man, um die Zahlen bis 10^{200} im Binärcode darzustellen?

$$10^{200} \leq 2^x \quad x \geq \log_2 10^{200} = \frac{\log_{10} 10^{200}}{\log_{10} 2} = \frac{200}{\log_{10} 2} \quad x \geq 665 \text{ Bit}$$

- ⌘ **Beispiel 3:** $a = 577$; $b = 858$; $c = ?$

$$c = \log_a b = \log_{577} 858 = \frac{\ln 858}{\ln 577} = 1,0624$$

Faktorisierungsannahme

⌘ Seien

p und q zwei große, unabhängig und zufällig gewählte Primzahlen. p und q werden geheimgehalten. $|p| \approx |q| \approx 500 \dots 1500$ Bit.

Das Produkt n aus p und q wird veröffentlicht: $n = p \cdot q$.

Die öffentlich ausführbare Funktion (Verschlüsseln bzw. Signaturtest) kommt mit dem öffentlichen n aus. Die private Funktion (Entschlüsseln bzw. Erzeugen einer Signatur) nutzt p und q .

⌘ Annahme

Es ist zwar mit vernünftigem Aufwand möglich, Primzahlen p und q zu finden und diese zu multiplizieren. Es ist aber nicht mit vernünftigem Aufwand möglich, die Primfaktoren von n zu finden.

Für jeden »schnellen« Faktorisierungsalgorithmus $F(n)$ wird die Wahrscheinlichkeit, dass $F(n)$ eine Zahl $n=p \cdot q$ tatsächlich faktorisieren kann, schnell kleiner, je größer die Länge $|p|$ und $|q|$ der Faktoren ist.

⌘ Dass Faktorisierung schwer ist, ist bisher nicht bewiesen.

Diffie-Hellman-Key-Exchange

A will B die Nachricht m schicken.

B:

$p_B \in \mathcal{P}$ und a primitive Wurzel von p_B wählen

x_B mit $1 \leq x_B \leq p_B - 1$ wählen

x_B bleibt geheim!

$y_B = a^{x_B} \bmod p_B$ berechnen

a, p_B und y_B sind auf key server veröffentlicht

A:

liest Eintrag für B: a, p_B und y_B

x_A mit $1 \leq x_A \leq p_B - 1$ geheim wählen

$y_A = a^{x_A} \bmod p_B$ berechnen

Key Agreement:

$k_{AB} = y_B^{x_A} \bmod p_B$ berechnen

Verschlüsselung:

$s = E(k_{AB}, m)$

s und y_A

B:

$k_{BA} = y_A^{x_B} \bmod p_B$ berechnen

$m = E^{-1}(k_{BA}, s)$ entschlüsseln

Diffie-Hellmann-Key-Exchange

⌘ Berechnung des Kommunikationsschlüssels

k_{AB} bzw. k_{BA} erfolgt durch

$$k_{AB} = y_B^{x_A} \bmod p_B \text{ bei A und}$$

$$k_{BA} = y_A^{x_B} \bmod p_B \text{ bei B.}$$

⌘ Nachweis

$$k_{AB} = y_B^{x_A} = (a^{x_B})^{x_A} = (a^{x_A})^{x_B} = y_A^{x_B} = k_{BA} \pmod{p_B}$$

⌘ Angreifer muss zum Brechen x_A oder x_B ermitteln, d.h. er muss berechnen:

$$x_A = \log_a y_A \pmod{p_B} \quad \text{oder}$$

$$x_B = \log_a y_B \pmod{p_B}$$

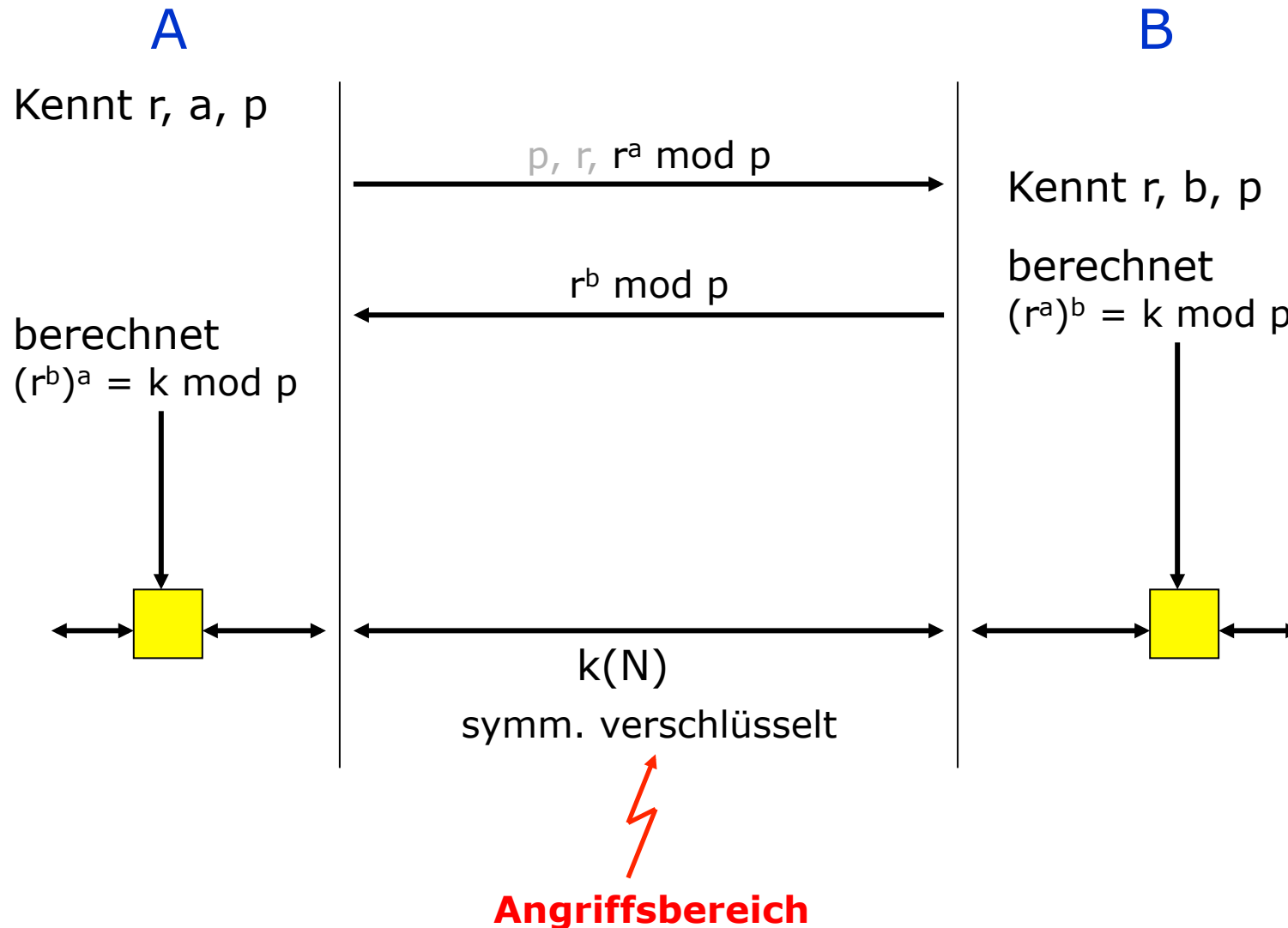
⌘ Sicherheit

⊗ Verfahren ist sicher gegen einen passiven Angreifer.

⊗ Verfahren ist unsicher gegen einen aktiven Angreifer (Maskerade).

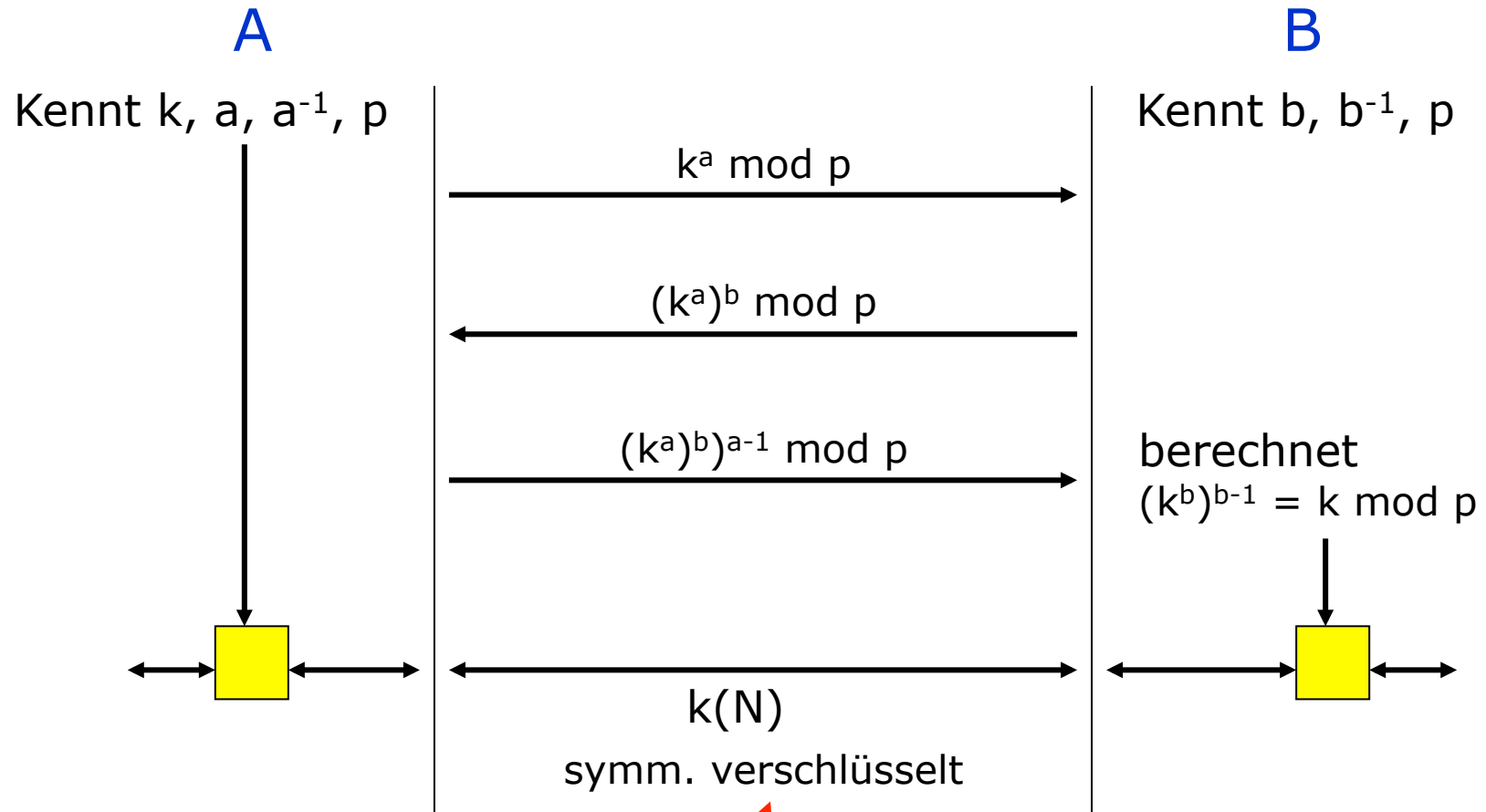
Asymmetrische Schlüsselvereinbarung (v1)

⌘ Wert von k kann nicht festgelegt werden (ist zufällig)



Asymmetrische Schlüsselvereinbarung (v2)

⌘ Teilnehmer A kann k festlegen



Angriffsbereich

Konzelationssystem nach El Gamal

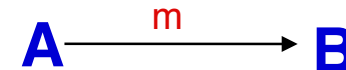
basiert auf der Schwierigkeit der Berechnung des diskreten Log

⌘ Schlüsselgenerierung

- | | | |
|---------------------|---|--------------------------------|
| ⊗ wähle global: | <ul style="list-style-type: none">• $p \in \mathbf{P}$• a primitive Wurzel von p | öffentlich
öffentlich |
| ⊗ jeder Tln. wählt: | <ul style="list-style-type: none">• geheimen Schlüssel k_i ($k_i < p-1$)
relativ prim zu $p-1$, d.h. $\text{ggT}(k_i, p-1)=1$
(nur Signatursystem nach El Gamal)• berechnet $-k_i \bmod (p-1)$• $y_i = a^{-k_i} \bmod p$ (*) | geheim
geheim
öffentlich |

⌘ Verschlüsselung

- ⊗ A will Nachricht m ($m < p$) an B schicken
- ⊗ A besorgt sich p, a, y_B
- ⊗ A wählt Zufallszahl z ($z < p$)
- ⊗ A berechnet $c = y_B^z \cdot m \bmod p$
- ⊗ A sendet an B: $a^z \bmod p, c$



⌘ Entschlüsselung

- ⊗ B berechnet $m^* = (a^z)^{k_B} \cdot c \bmod p$

Konzelationssystem nach El Gamal: Beispiel

basiert auf der Schwierigkeit der Berechnung des diskreten Log

⌘ Schlüsselgenerierung

⊗ Global öffentlich: $p = 3137$ und $a = 577$

⊗ Teilnehmer B: $k_B = 1762$ geheim

$-k_B \bmod (p-1) = -1762 \bmod 3136 = -1762 + 3136 = 1374$
geheim

$y_B = a^{-k_B} \bmod p = 577^{1374} \bmod 3137 = 858$

⌘ Verschlüsselung

⊗ **A** will **B** vertraulich die Nachricht $m = 2115$ schicken.

⊗ **A** wählt $z = 932$ geheim.

⊗ berechnet $a^z \bmod p = 577^{932} \bmod 3137 = 1852$

⊗ berechnet $y_B^z \bmod p = 858^{932} \bmod 3137 = 749$

⊗ berechnet $c = y_B^z \cdot m \bmod p = 749 \cdot 2115 \bmod 3137 = 3087$

⊗ schickt $a^z = 1852$ und $c = 3087$ an **B**

⌘ Entschlüsselung

⊗ **B** berechnet

$(a^z)^{k_B} \cdot c \bmod p = 1852^{1762} \cdot 3087 \bmod 3137 = 2115$

Signaturssystem nach El Gamal

basiert auf der Schwierigkeit der Berechnung des diskreten Log

⌘ Schlüsselgenerierung

⊗ wähle global:

• $p \in \mathbf{P}$ öffentlich

• a primitive Wurzel von p öffentlich

⊗ jeder Tln. wählt:

• $x_i \in \mathbf{Z}_p^*$ geheim

• berechnet $y_i = a^{x_i} \bmod p$ öffentlich

⌘ Signatur

⊗ A wählt:

• Zufallszahl k mit $k \in \mathbf{Z}_{p-1}^*$
bzw. k relativ prim zu $p-1$, d.h. $\text{ggT}(k, p-1)=1$

⊗ A berechnet:

• $k^{-1} \bmod (p-1)$

• $r = a^k \bmod p$

• $h(m)$ (Hash-Wert von m ; $h(m) < p$)

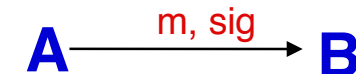
• löst die Kongruenz

$h(m) = (x_A \cdot r + k \cdot s) \bmod (p-1)$ nach s auf:

$s = k^{-1}(h(m) - x_A \cdot r) \bmod (p-1)$

⊗ A bildet $\text{sig} = (s, r)$

⊗ A sendet m, sig



⌘ Test

⊗ B berechnet:

• $t_1 = a^{h(m)} \bmod p$

• $t_2 = y_A^r \cdot r^s \bmod p$

⊗ B vergleicht:

• $t_1 = t_2 \rightarrow$ gültige Signatur

• $t_1 \neq t_2 \rightarrow$ ungültige Signatur

RSA-Verfahren (Rivest, Shamir, Adleman, 1978)

basiert auf der Faktorisierungsannahme

⌘ Schlüsselgenerierung

- ⊗ wähle unabh. und zufällig $p, q \in \mathbf{P}$ mit $|p| \approx |q|$ und $p \neq q$
- ⊗ berechne $n = p \cdot q$
- ⊗ wähle c mit $3 \leq c < \Phi(n)$ und
 $\text{ggT}(c, \Phi(n))=1$ mit $\Phi(n) = (p-1)(q-1)$
- ⊗ berechne d mittels p, q, c als multiplikatives Inverses von c mod $\Phi(n)$

$$c \cdot d \equiv 1 \pmod{\Phi(n)}$$

	Konzelationssystem	Signaturssystem
öffentl.	c, n	d (hier meist t genannt), n
geheim	d, p, q	c (hier meist s genannt), p, q

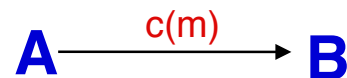
RSA-Verfahren (Rivest, Shamir, Adleman, 1978)

basiert auf der Faktorisierungsannahme

⌘ Ein Sicherheitsbeweis von RSA ist bisher nicht bekannt.

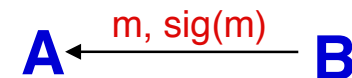
Verschlüsselung:

A will Nachricht m ($1 < m < n$)
an B schicken



Signatur:

A will Signatur einer Nachricht
 m ($1 < m < n$) von B testen



A besorgt sich öffentliche Parameter von B: c bzw. t , sowie n

naiv: $c(m) := m^c \bmod n$

$\text{sig}_s(m) := m^s \bmod n$

sicher: $c(m) := (z, m, h(z, m))^c \bmod n$

$\text{sig}_s(m) := (h(m))^s \bmod n$

Entschlüsselung:

naiv: $m^* = (m^c)^d \bmod n$

$m^* = (m^s)^t \bmod n$
 $m^* =? m' \rightarrow \text{out(ok)}$

sicher: $(z^*, m^*, y) = c(m)^d \bmod n$
 $y =? h(z^*, m^*) \rightarrow \text{out}(m)$

$h(m)^* = ((h(m))^s)^t \bmod n$
 $h(m)^* =? h(m') \rightarrow \text{out(ok)}$

RSA-Verfahren: Angriffe

⌘ Raten von Klartextblöcken

- ⊠ Angreifer kann wahrscheinliche Klartextblöcke raten, mit c verschlüsseln und mit abgefangenen Schlüsseltexten vergleichen.

⌘ Verhinderung:

- ⊠ Zufallszahl in Klartext hineinkodieren



RSA besitzt multiplikative Struktur

⌘ Passiver Angriff auf naives Signatursystem (Davida)

⊗ Angenommen, Angreifer kennt zwei Signaturen m_1^s und m_2^s sowie die Nachrichten m_1 und m_2 und kann eine dritte Signatur m_3^s bilden. m_3 ist jedoch nicht beliebig wählbar.

$$m_3^s = m_1^s \cdot m_2^s \pmod{n}$$

$$m_3 = m_1 \cdot m_2 \pmod{n}$$

$$\text{gilt, da } m_1^s \cdot m_2^s = (m_1 \cdot m_2)^s$$

⌘ Homomorphismus bezüglich Multiplikation

$$\begin{array}{ccc} (m_1, m_2) & \xrightarrow{\cdot} & m_1 \cdot m_2 \\ \downarrow x^s & & \downarrow x^s \\ (m_1^s, m_2^s) & \xrightarrow{\cdot} & m_1^s \cdot m_2^s = (m_1 \cdot m_2)^s \end{array}$$

RSA-Verfahren: Angriffe

⌘ Aktiver Angriff zum selektiven Brechen von RSA nach Judy Moore

- Angreifer möchte Schlüsseltextblock S_3 entschlüsselt haben
- wählt Zufallszahl r mit $1 \leq r < n$
- berechnet multiplikatives Inverses mod n : r^{-1}
- berechnet $S_2 := S_3 \cdot r^c \bmod n$
- lässt S_2 entschlüsseln, d.h. Angreifer erhält S_2^d
- Er weiß $S_2^d \equiv (S_3 \cdot r^c)^d \equiv S_3^d \cdot r^{c \cdot d} \equiv S_3^d \cdot r \bmod n$
- berechnet $S_3^d \equiv S_2^d \cdot r^{-1} \bmod n$

Angriff wird «ausgenutzt»
für blinde Signaturen
(Chaum 1985)

⌘ Verhinderung der Angriffe (aktiv und passiv)

- Konzelenation: Hinzunahme eines Redundanzprädikates, z.B. einer Zufallszahl, so dass das Multiplizieren zweier Klartextblöcke keinen dritten Klartextblock mit passender Redundanz ergibt; alternativ: vor Verschlüsselung Hashwert an Nachricht anhängen
- Signatur: Signatur des Hashwertes $h(m)$ der Nachricht m .
 h ist eine kollisionsfreie Hashfunktion. Das Finden einer Kollision, d.h. $h(m) = h(m^*)$ mit $m \neq m^*$ ist ein schwieriges Problem.

Blinde Signatur mit RSA-Verfahren

⌘ Teilnehmer möchte Nachricht m signiert haben, ohne dass der Signierer die Nachricht m selbst zur Kenntnis bekommt

⊗ wählt Zufallszahl r mit $1 \leq r < n$

⊗ berechnet multiplikatives Inverses mod n : r^{-1}

⊗ «Blendet» die Nachricht m , d.h. berechnet

$$m^{\sim} := m \cdot r^t \pmod n$$

⊗ läßt m^{\sim} signieren, d.h. erhält $m^{\sim s}$

⊗ Er weiß

$$m^{\sim s} \equiv (m \cdot r^t)^s \equiv m^s \cdot r^{t \cdot s} \equiv m^s \cdot r \pmod n$$

⊗ «Entblendet» die Nachricht, d.h. berechnet

$$\text{sig}(m) \equiv m^s \cdot r \cdot r^{-1} \pmod n$$

⌘ Anwendungsbeispiel: Anonyme digitale Zahlungssysteme

Blinde Signatur mit RSA-Verfahren

- ⌘ Anwendungsbeispiel: Anonyme digitale Zahlungssysteme
- ⌘ Signierer = Bank
- ⌘ Bank erfährt nichts über Zahlungsflüsse ähnl. Bargeld
 1. Kunde schickt «geblendete digitale Banknote» m^{\sim} an Bank
 2. Bank belastet Konto des Kunden mit Gegenwert
 3. Bank signiert m^{\sim} und schickt $m^{\sim s}$ zurück an Kunden
 4. Kunde «entblendet» Banknote und erhält $\text{sig}(m)$
 5. Kunde kauft bei Händler ein und bezahlt mit Banknote $\text{sig}(m)$
 6. Händler löst $\text{sig}(m)$ bei Bank ein
 7. Bank prüft $\text{sig}(m)$ auf Gültigkeit (korrekte Signatur und nicht bereits eingelöst)
 8. Bank schreibt Händler Gegenwert auf seinem Konto gut

Kryptographie auf Basis Elliptischer Kurven

ECC: Elliptic Curve Cryptography

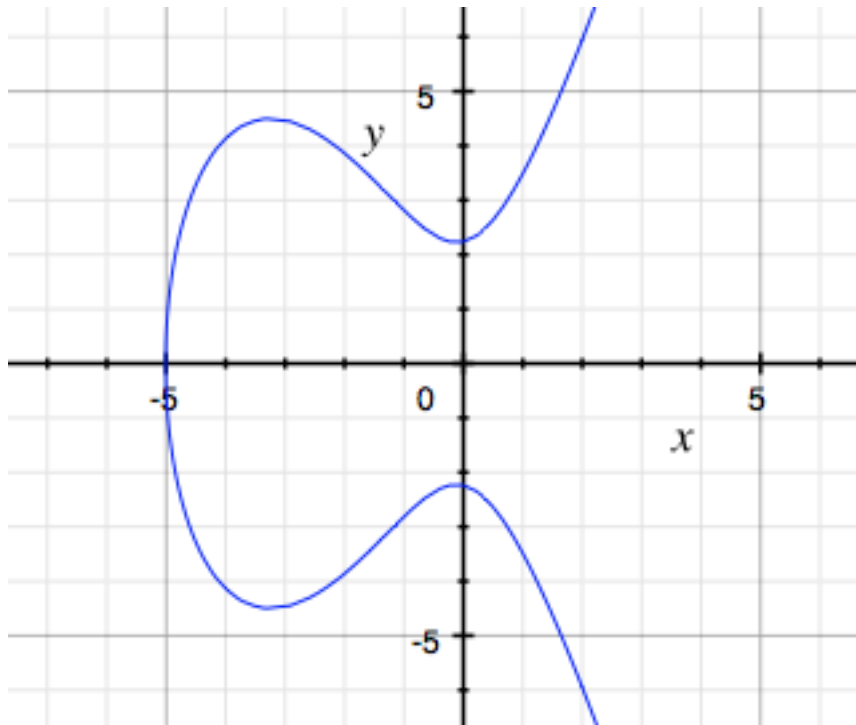
⌘ Elliptische Kurve

- ⊠ Eine elliptische Kurve $E(K)$ über einem Körper K ist die Menge aller Punkte der Ebene $K \times K$, deren Koordinaten x und y einer kubischen Gleichung der folgenden Form genügen:

$$F(x,y): y^2 = x^3 + ax^2 + bx + c \text{ mit } a,b,c \in \mathbf{Z}$$

⌘ Beispiel

$$y^2 = x^3 + 5x^2 + x + 5$$



Kryptographie auf Basis Elliptischer Kurven

ECC: Elliptic Curve Cryptography

⌘ Spezielle Form für Kryptosysteme

⊗ $F(x,y): y^2 = x^3 + ax + b \pmod{p}$ mit $a,b \in \mathbf{Z}$ und $p \in \mathbf{P}$

⊗ Beispiel: $E(\text{GF}(p))$ mit $a=2, b=4, p=5$ besitzt folgende Punkte:
– $\{(0,2),(0,3),(2,1),(2,4),(4,1),(4,4), o\}$

⊗ Gruppenstruktur über den Punkten $P_i = (x_i, y_i)$ definiert:

⊕ Abgeschlossenheit: $P_1 + P_2 = P_3$

⊕ Assoziativgesetz: $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$

⊕ Neutrales Element: $P_i + o = P_i$ mit o definiert als $o = (\infty, \infty)$

⊕ Inverses Element $P_i^{-1} = A$: $P_i + A = o$

⊕ Kommutativgesetz (nur abelsche Gruppe): $P_1 + P_2 = P_2 + P_1$

Kryptographie auf Basis Elliptischer Kurven

ECC: Elliptic Curve Cryptography

⌘ Anwendung von elliptischen Kurven für Kryptographie

- ⊗ Nicht alle elliptischen Kurven sind gleich gut für ECC geeignet.
- ⊗ Ordnung (Punkteanzahl) von $E(K)$ ist u.a. wichtig für Sicherheit.
- ⊗ Methoden zum Finden geeigneter elliptischer Kurven:
 - ⊕ Wähle a, b, p , berechne die Gruppenordnung und überprüfe Sicherheit
 - ⊕ Wähle p und Gruppenordnung und bestimme a und b (schnellere Methode)

⌘ Vorteile

- ⊗ bei vergleichbarem Sicherheitsniveau kürzere Schlüssel (≥ 200 Bit)
- ⊗ erzeugen deutlich kürzere Signaturen
- ⊗ weniger Rechenaufwand
- ⊗ Hoffnung: Alternative Kryptoalgorithmen, wenn z.B. Faktorisierungsannahme nicht halten sollte

⌘ Verfügbare Algorithmen

- ⊗ ECDSA: Standardisiertes Signatursystem auf Basis elliptischer Kurven
- ⊗ ECElGamal: ElGamal auf Basis elliptischer Kurven