Kapitel 3: Symmetrische Kryptographie

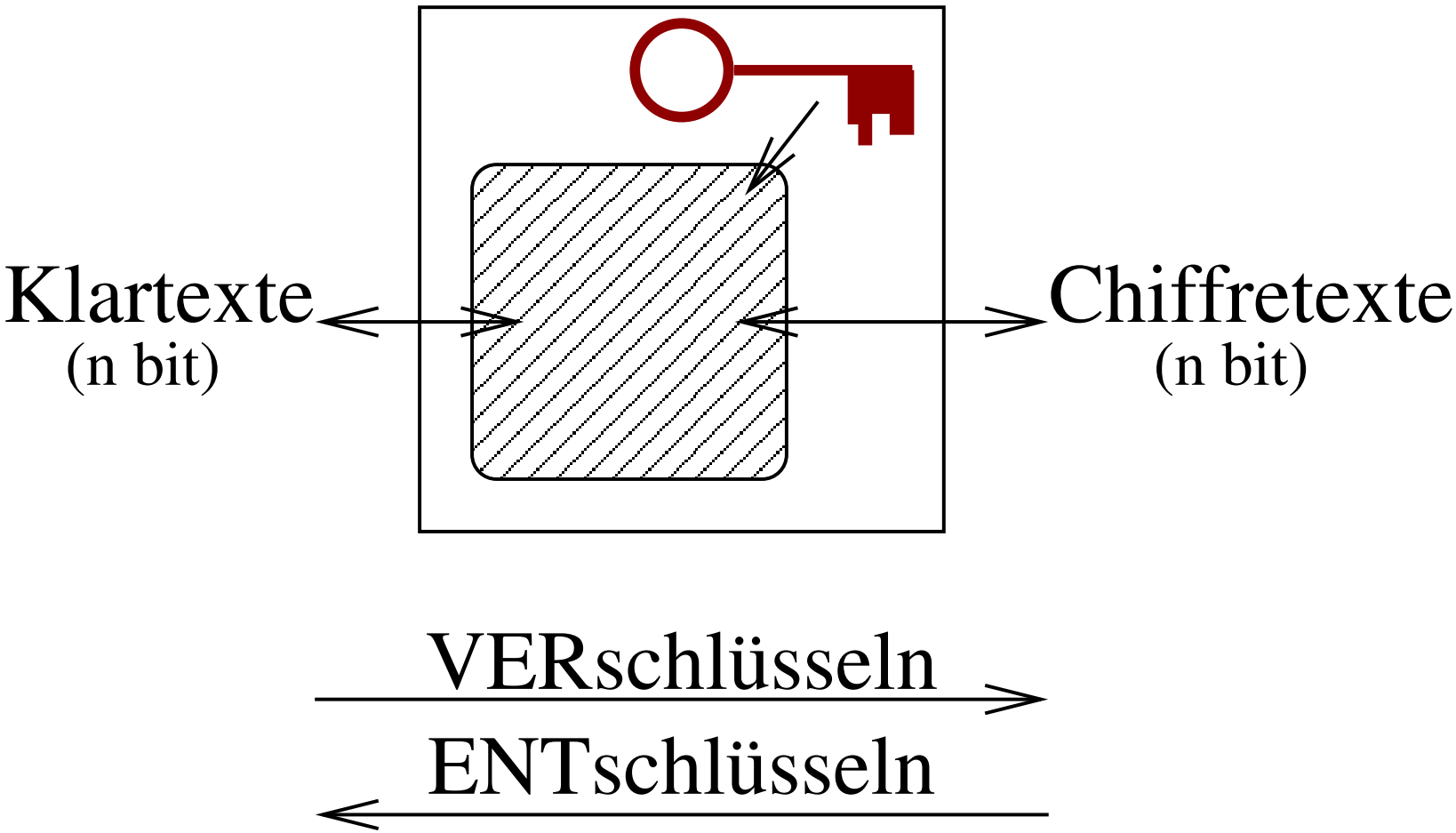
3: Symmetrische Kryptographie

- Bei der Symmetrischen Kryptographie verwenden Sender und Empfänger den gleichen Schlüssel.
- Dies kennen wir aus dem wahren Leben von Türen und Tresoren, welche auch mit dem gleichen Schlüssel ver- und aufgeschlossen werden.
- Die beiden zentralen symmetrischen Kryptoverfahren.
  1. Verschlüsselungsverfahren welches die Vertraulichkeit von Nachrichten sichern.
  2. MAC (Message Authentication Codes) welche die Integrität von Nachrichten schützen.
- Moderne symmetrische Verfahren basieren auf kryptographische Bausteine (Blockchiffren, Stromschiffren und Hashfunktionen).
- Sinnvolle Schlüssellängen: 128 bit, 192 bit und 256 bit.

Symmetrische Verschlüsselungsverfahren

- Mit einem symmetrischen Verschlüsselungsverfahren können viele Nachrichten unter dem selben kurzen Schlüssel **K** ver- und entschlüsselt werden.
- $C = \mathcal{E}_{\mathbf{K}}(M)$ für eine beliebig lange Nachrichten $M \in \{0,1\}^*$.
- $M = \mathcal{D}_{\mathbf{K}}(C)$ für eine beliebig langen Chiffretext $C \in \{0,1\}^*$.
- Für alle Klartexte gilt: $D_{\mathbf{K}}(\mathcal{E}_{\mathbf{K}}(M)) = M$

3.1: Blockchiffren



Abstrakte Blockchiffre

- $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$
- $D : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

- k: Schlüssellänge in Bits
- n: Blockgröße in Bits

- Für alle $\mathbf{K} \in \{0,1\}^k$ gilt: $E_{\mathbf{K}}(\cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ ist eine umkehrbare (bijektive) Funktion (Permutation).

- $D_{\mathbf{K}}(\cdot)$ ist daher die Umkehrfunktion von $E_{\mathbf{K}}(\cdot)$.
- Alternative Schreibweise für $D_{\mathbf{K}}(\cdot)$: $E_{\mathbf{K}}^{-1}(\cdot)$

- Für jede Blockchiffre gilt: $D_{\mathbf{K}}(E_{\mathbf{K}}(M)) = M$

- Typische Blockgrößen: 64 bit und 128 bit.

Sicherheit von Blockchifffren

Im folgenden Sei $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ eine n-bit Blockchiffre.

- E gilt als sicher, wenn kein bessere Angriff als das durchprobieren aller Schlüssel bekannt ist.
- Aufwand um alle möglichen k-Bit Schlüssel (Schlüsselraum) durchzuprobieren: 2^k Aufrufe von E
- Statistischer Aufwand um den richtigen Schlüssel zu finden: $2^k/2 = 2^{k-1}$ Aufrufe von E.
- Angenommen der Aufwand des besten bekanntesten Angriffs auf E beträgt 2^{x-1}, dann beträgt die Sicherheit von E x Bit. (E ist x-bit sicher)
- E ist sicher $\equiv$ E ist k-bit sicher.

Hardware Performance

Im folgenden Sei $E : \{0,1\}^k \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ eine 128-bit Blockchiffre welche in 32 Taktzyklen berechnet werden kann.

- Aktueller CPU Core (4 Ghz)
  - Operationen pro Sekunde: ca. 2^{32}
  - Aufrufe von E pro Sekunde: ca. 2^{27}
  - Aufrufe von E pro Jahr: ca. 2^{52}
  - 16 Milliarden Cores pro Jahr: ca. 2^{86}

- Aktuelle Bitcoin-Mining ASICs schaffen 112 Tera Hashes pro Sekunde das sind ca. 2^{47} Aufrufe einer Hashfunktion (ähnliche Konstruktion wie Blockchiffre) pro Sekunde.
  - Aufrufe von E pro Sekunde: ca. 2^{47}
  - Aufrufe von E pro Jahr: ca. 2^{72}
  - 256 Millionen ASICs pro Jahr: ca. 2^{100}

Schlüssellänge und deren Sicherheit

- 64-Bit: Zur kurz; ungefähr 250 high end PCs benötigen ein Jahr um 2^{63}-Schlüssel zu testen.
- 80-Bit: Zur kurz; 128 ASICs benötigen ca. 1 Jahr um 2^{79} Schlüssel zu testen.
- 100-Bit: 128 Millionen ASICs benötigen ca. 1 Jahre um 2^{99} Schlüssel zu testen.
  (Dies ist für Geheimdienste wie die NSA mit einem Budget von 10 Milliarden USD pro Jahr machbar.)
- 128-Bit: 32 Milliarden ASICs benötigen ca. 1 Million Jahre um 2^{127} Schlüssel zu testen.
- 192-Bit: 512 Trillionen ASICs benötigen ca. 2^{50} Jahre um 2^{191} Schlüssel zu testen. (Alter des Universums: $< 2^{34}$ Jahre).
- 256-Bit: Um 2^{255} Schlüssel zu testen wird mehr Energie benötigt als die Sonne in ihrer restlichen Laufzeit noch produziert.

Zentrale Designziele

Bei dem Design der Rundenfunktion gibt es zwei zentrale Designziele.

1. Konfusion

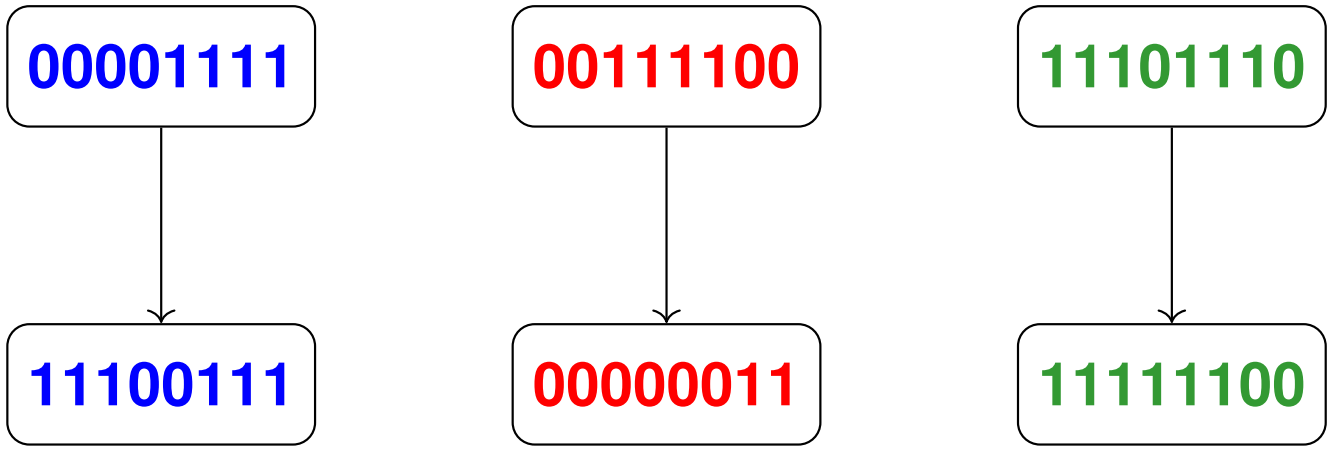
Verschleierung von Klartext-Chiffretext-Beziehungen.

2. Diffusion

Jedes Ausgabebit soll von allen Eingabebits abhängen.

Daneben soll die Chiffre noch möglichst schnell und sicher gegen alle bekannten Angriffstechniken sein.

Anmerkungen zur Konfusion



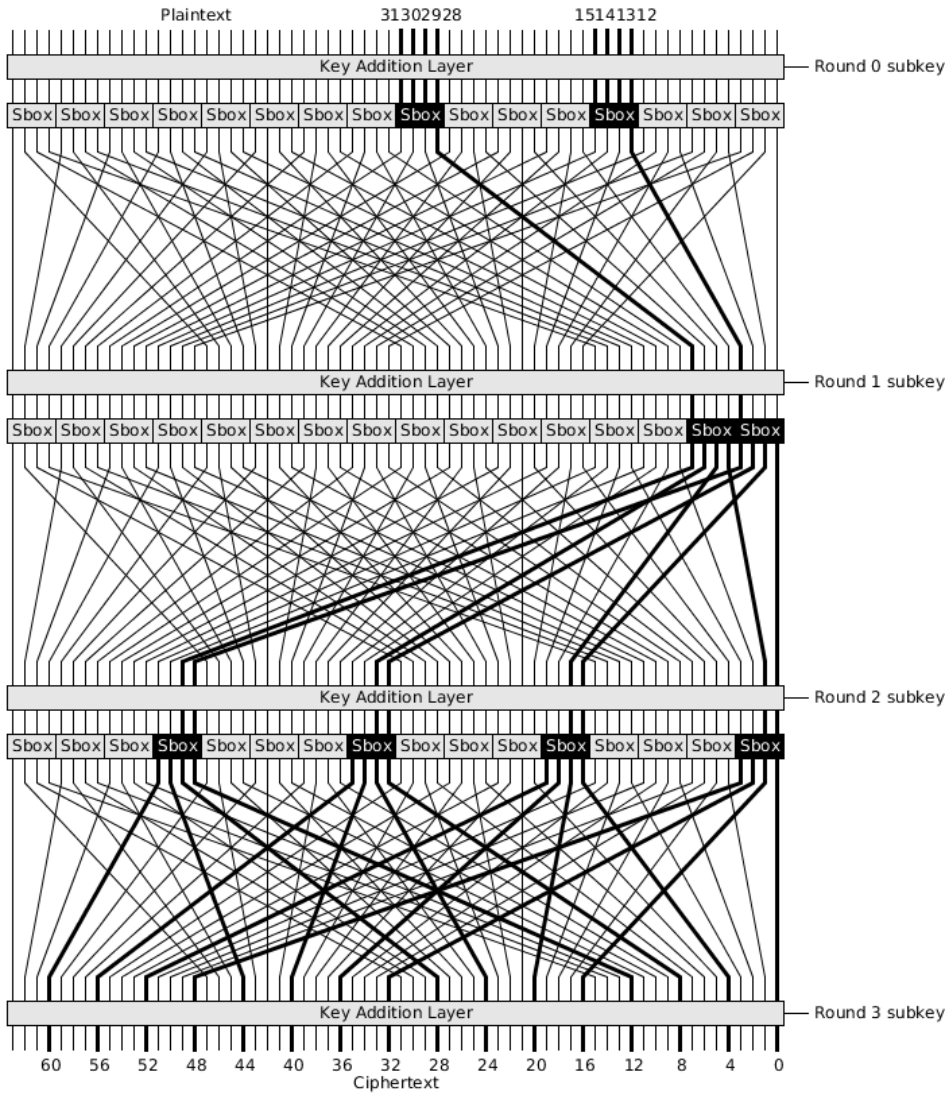
Eine Konfusion wird durch eine nicht lineare Operation realisiert.

- Ersetzung von Bits durch S-Boxen.
  S-Box ist ein Array: Input = Index und Output = Value.
- Modernere Realisierung: ARX
  Funktion auf Datenwörter mit Addition, Rotation und XOR.
  Beispiel: $f(a,b) = \{a+b; (b \lll 7) \oplus (a+b)\}$

Diffusion

- Bei der Diffusion soll durch das Kippen eines Eingabebits statistisch die Hälfte aller Ausgabebits kippen.
- Kleine Änderungen im Klartext sollen sich möglichst schnell ausbreiten (Lawineneffekt).
- Dieser Effekt lässt sich durch Permutation von Datenwörter erreichen.
- Natürlich müssen auch Bits zwischen einzelnen Datenwörtern ausgetauscht werden.

Lawineneffekt bei Present



Klassiche Blockchiffrenkonstruktionen

Es gibt zwei klassische Arten Blockchiffren zu konstruieren.

1. Feistelnetzwerk (balanciert)
   - Erfinder: Horst Feistel
   - Der Klartext wird in zwei gleich große Hälften zerlegt.
   - In jeder Runde wird auf einer Hälfte die Rundenfunktion angewandt.
   - Die Ausgabe wird mit einem anderen Hälfte verknüpft.

2. Substitutions-Permutations-Netzwerk
   - Der Klartext wird in gleichgroße Teile aufgeteilt.
   - Auf jeden Teil wird (quasi) parallel die Substitutionsoperation angewandt.
   - Im Anschluss wird eine Permutationsoperation angewandt welche die Teile vermischt.
   - Bei der letzten Operation kann die Permutationsoperation übersprungen werden.

Illustration: Feistelnetzwerk

$P_3(L, R)$:

1: $S \leftarrow L \oplus F_1(R)$
2: $Y \leftarrow R \oplus F_2(S)$
3: $X \leftarrow S \oplus F_3(Y)$
4: **return** (X, Y)

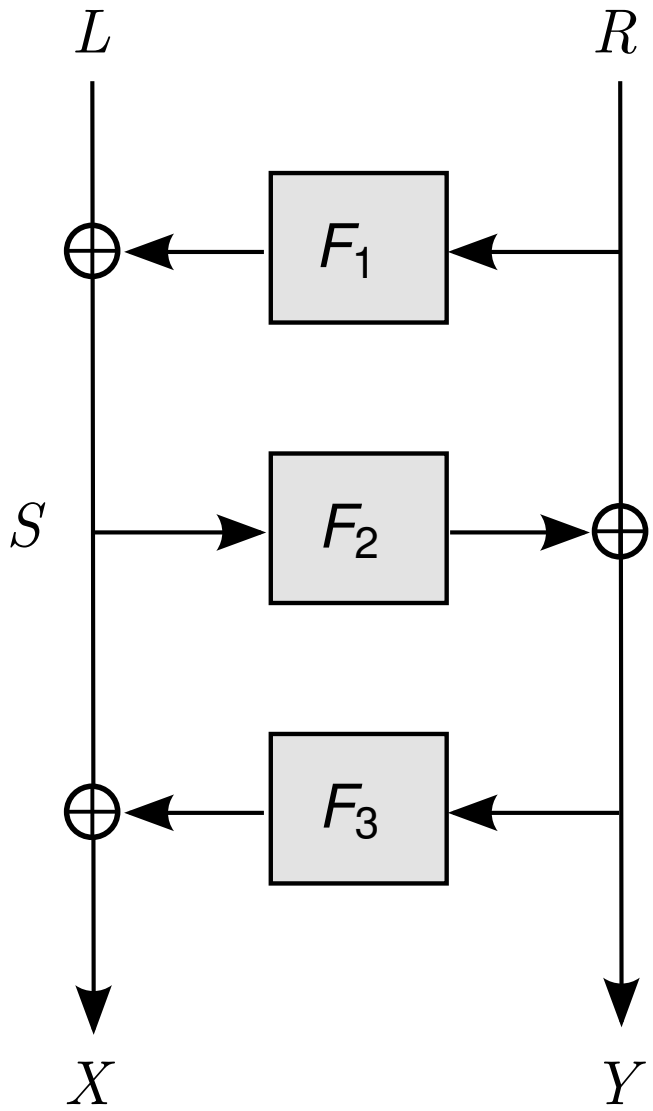
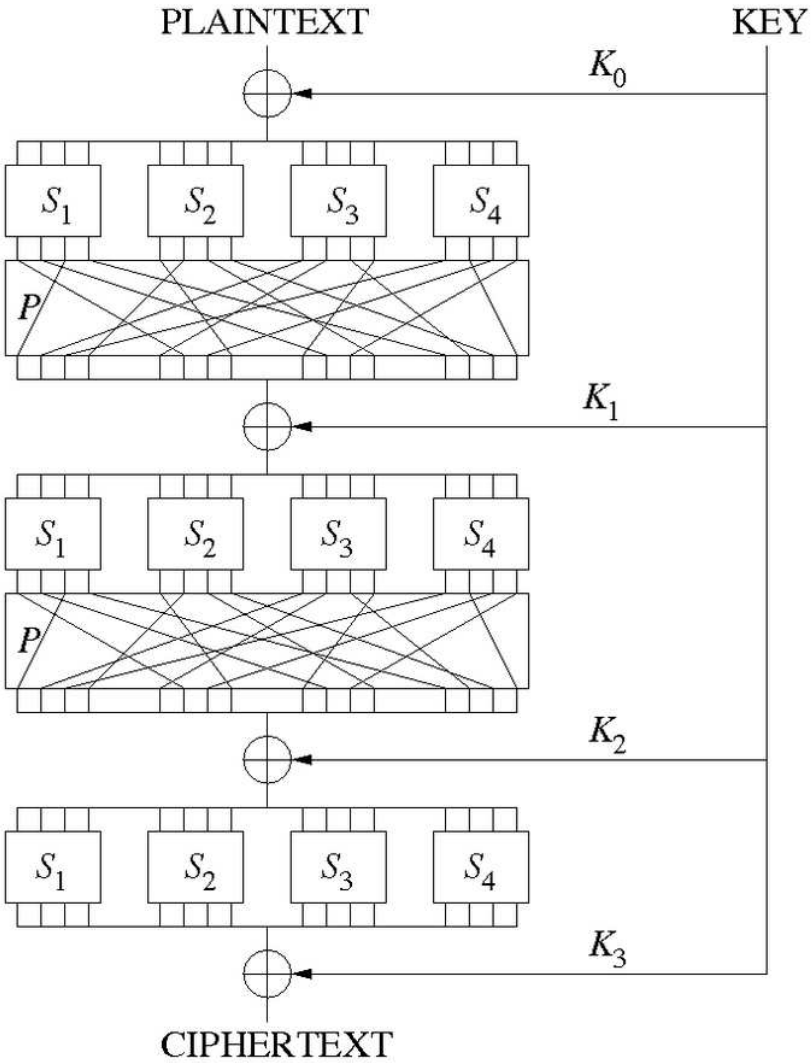


Illustration: Substitutions-Permutations-Netzwerk



3.2: Data Encryption Standard (DES)

- Designer: Horst Feistel (IBM)
- Ursprünglicher Name: Lucipher
- Design: 16-Runden Feistelnetzwerk
- Blockgröße: 64-bit
- Schlüssellänge: 56-bit.
- Gewinner einer Aussschreibung der amerikanischen Standardisierungsbehörde (NIST).
- Offizieller NIST Standard.

Geschichte des DES

1973-77 Zwei Ausschreibungen, ein geeigneter Kandidat ("Lucipher") nach Überarbeitung als DES (“Data Encryption Standard”) standardisiert:

Ab 1977 Kritik an Schlüssellänge.
Trotzdem große Akzeptanz und riesige Verbreitung.

Ab 1990 Differentielle und lineare Kryptanalyse.

1997 DES-Challenge (1000e von Rechnern, 4 Mon.).

Angriffe über die kurze Schlüssellänge

1980 Hellman time-memory-tradeoff
(Spezialrechner + Massenspeicher):
4 Mio. $, 2 Jahre Vorbereitungszeit, 100 Schlüssel/Tag.

1993 Wiener (Spezialrechner):
1 Mio. $, 7 Schlüssel/Tag.

1997 Erste DES-Challenge
(Internet und *idle time* tausender Rechner):
keine Kosten, 4 Monate/Schlüssel.

1998 DES-Cracker der EFF (Spezialrechner):
0.25 Mio. $, einige Tage/Schlüssel.

Vergleich: 1 Spionagesatellit 3 000 Mio. $ bis 6 000 Mio. $
(geschätzt).

3.3: Advanced Encryption Standard (AES)

- Designer: Joan Daemen und Vincent Rijmen
- Ursprünglicher Name: Rijndael
- Design: Substitutions-Permutations-Netzwerk (10-14 Runden)
- Blockgröße: 128 bit
- Schlüssellängen
  - 128 bit (10 Runden)
  - 192 bit (12 Runden)
  - 256 bit (14 Runden)
- Gewinner des AES-Wettbewerbs und offizieller NIST Standard
- Sicherer und effizienter als Triple-DES
- Hardwareunterstützung bei moderen CPUs (AES-NI)

Geschichte des AES (1)

1997 Ausschreibung des AES.

1998 1. AES-Konferenz; Präsentation von 15 Kandidaten.

“The Demolition Derby begins.”

Feistel-Netzwerk			**S.-P. Netzerk**		**Sonstige**
(wie DES)		(erweitert)	allgemein	Square-artig	
DEAL	DFC	Cast-256	SAFER+	Crypton	Frog
Loki97	E2	MARS	Serpent	Rijndael	HPC
Magenta	RC6				
Twofish					

Geschichte des AES (2)

1999 2. AES-Konferenz; danach Auswahl der 5 Finalisten

MajorAttacks *DEAL, Frog, HPC, Loki97, Magenta*

Finale **MARS, RC6, Rijndael, Serpent, Twofish**

Feistel-Netzwerk			**S.-P. Netzerk**		**Sonstige**
(wie DES)		(erweitert)	allgemein	Square-artig	
DEAL	DFC	Cast-256	SAFER+	Crypton	*Frog*
Loki97	E2	**MARS**	**Serpent**	**Rijndael**	*HPC*
Magenta	**RC6**				
Twofish					

Geschichte des AES (3)

April 2000 3. AES-Konferenz, Diskussion der Finalisten

Oktober 2000 **Rijndael** wird vorläufiger Standard.

April 2001 Standard endgültig bestätigt

Feistel-Netzwerk			**S.-P. Netzerk**		**Sonstige**
(wie DES)		(erweitert)	allgemein	Square-artig	
DEAL	DFC	Cast-256	SAFER+	Crypton	Frog
Loki97	E2	**MARS**	**Serpent**	**Rijndael**	HPC
Magenta	**RC6**				
Twofish					

Die Struktur des AES

- Vier Basis-Operationen
  1. Sub Bytes
  2. Shift Rows
  3. Mix Columns
  4. Key Addition

- Eine AES-Runde als Kombination der vier Basis-Operationen

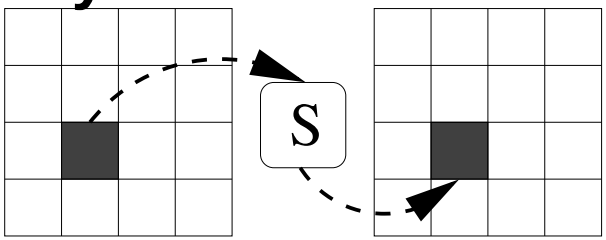
- Der "Key Schedule": Aus einem kurzen Chiffrier-Schlüssel werden 11–15 Rundenschlüssel (jeweils 128 bit).

  128-bit Schlüssel: 10 Runden
  192-bit Schlüssel: 12 Runden
  256-bit Schlüssel: 14 Runden

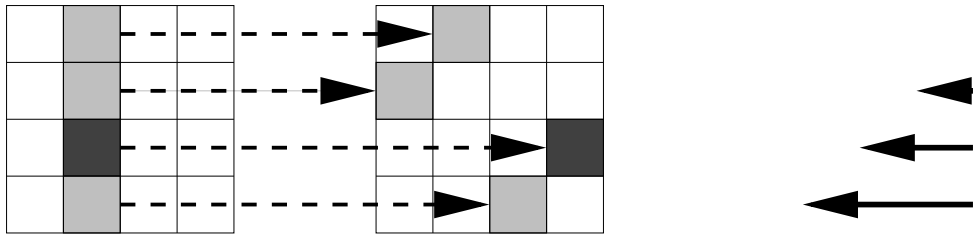
  Es gilt: Anzahl Rundenschlüssel = 1 + Anzahl Runden

Die Basis-Operationen

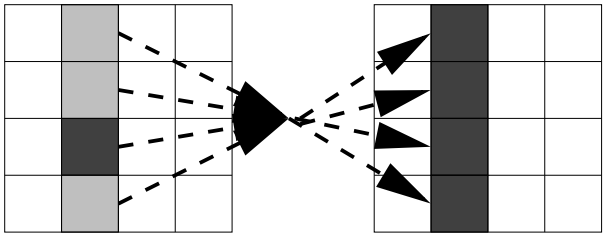
Sub Bytes



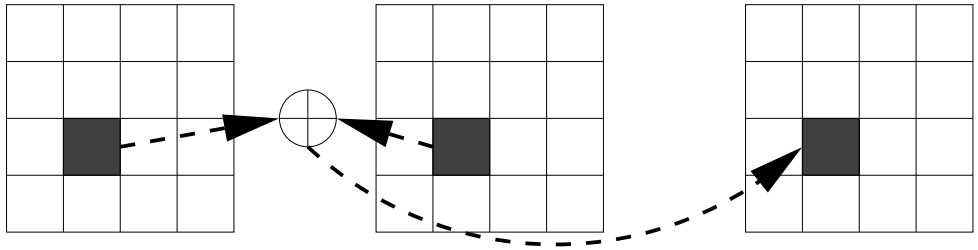
Shift Rows



Mix Columns



Key Addition



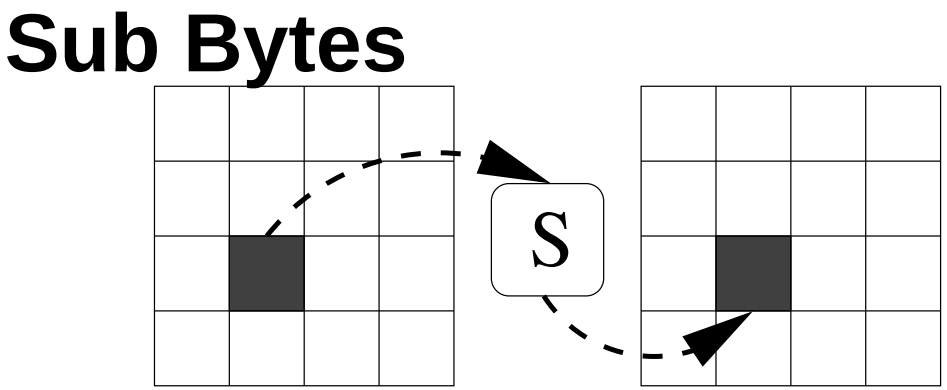
Sub Bytes

Anwendung einer invertierbaren S-Box

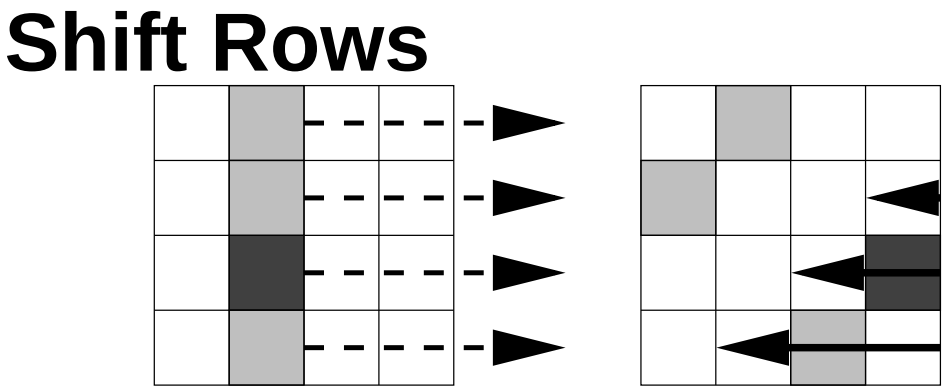
$$S : \{0,1\}^8 \rightarrow \{0,1\}^8$$

mit einer sehr einfachen algebraischen Struktur.

Trotz der einfachen algebraischen Struktur ist S eine nichtlineare Funktion.



Shift Rows



Mix Columns

$$\overbrace{(y_1, y_2, y_3, y_4)}^{\vec{y}} := \mathrm{MC}\overbrace{(x_1, x_2, x_3, x_4)}^{=\vec{x}}$$

- Die Spalten $\vec{x}$ und $\vec{y}$ fassen wir als Vektoren auf.
- Mix Columns ist eine Matrix-Operation.

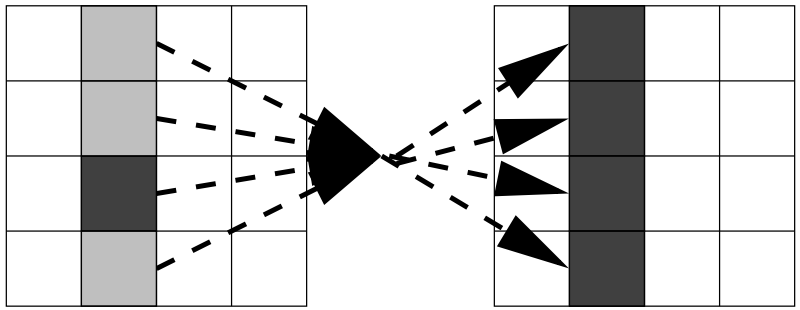
$$\vec{y} := \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \vec{x}.$$

Mix Columns: Eigenschaften

- Eine Differenz in *genau einem* der Eingabe-Werte x_i führt zu einer Differenz in **allen vier** Ausgabewerten y_1, y_2, y_3, y_4!
- Eine Differenz in *genau* $d > 0$ der Eingabe-Werte x_i führt zu einer Differenz in **mindestens** $5 - d$ der y_1, y_2, y_3, y_4!

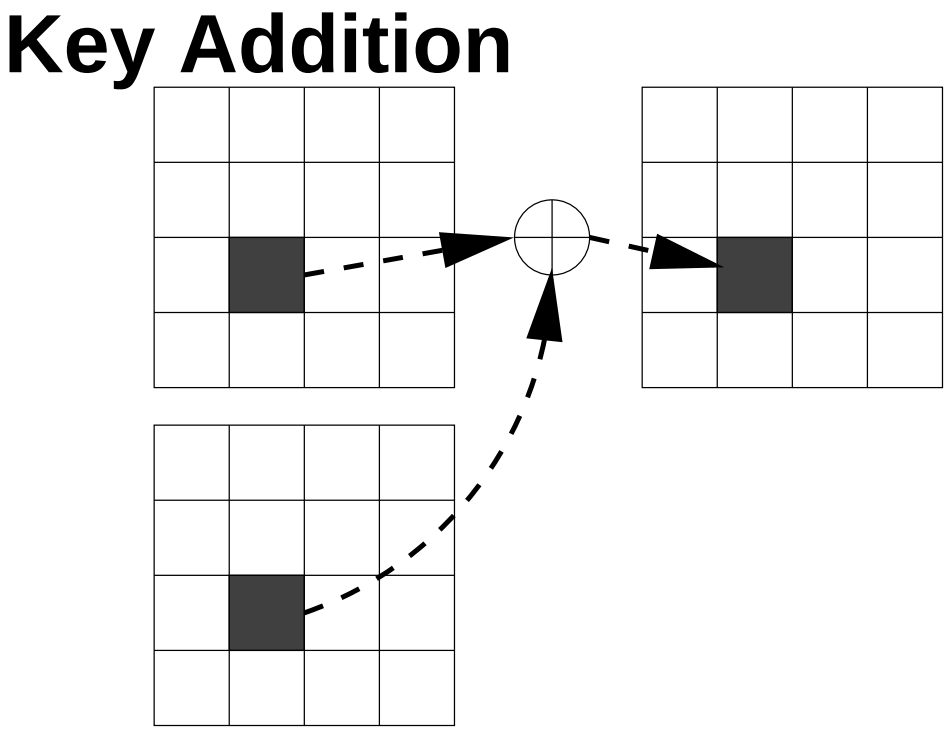
Die Matrix A ist invertierbar. Deshalb ist Mix Columns invertierbar.

Mix Columns

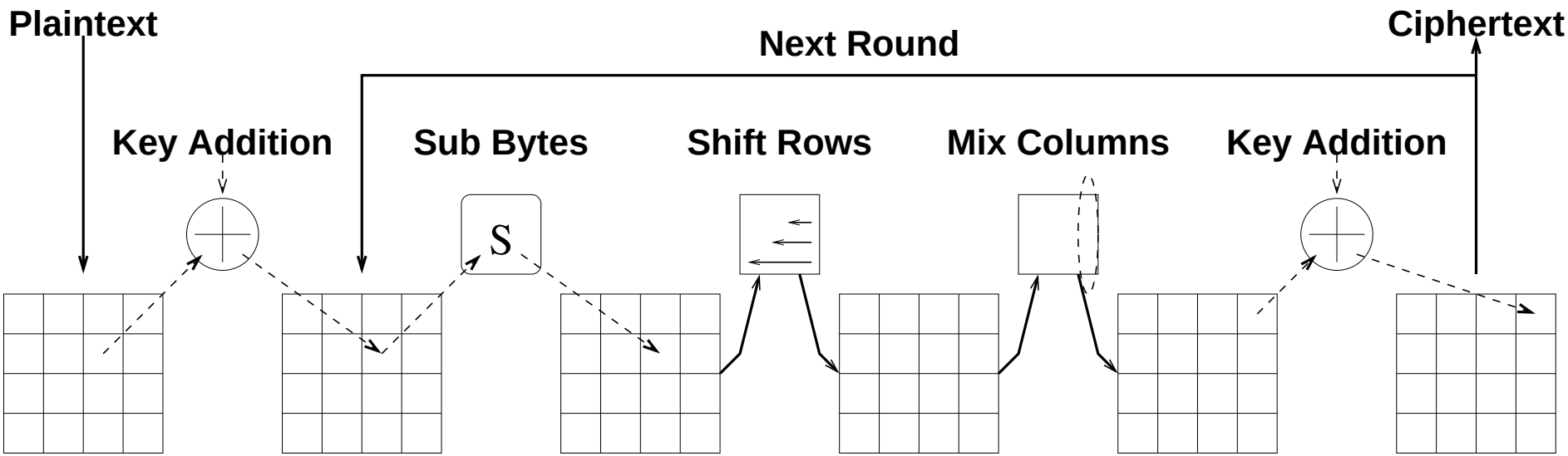


Key Addition

- Chiffrierschlüssel: k bit, $k \in \{128, 192, 256\}$
- Rundenschlüssel: $n + 1$ für n Runden, jeweils 128 bit
- Vor der ersten und nach jeder Runde:
  Addition (bitweise mod 2) eines Rundenschlüssel

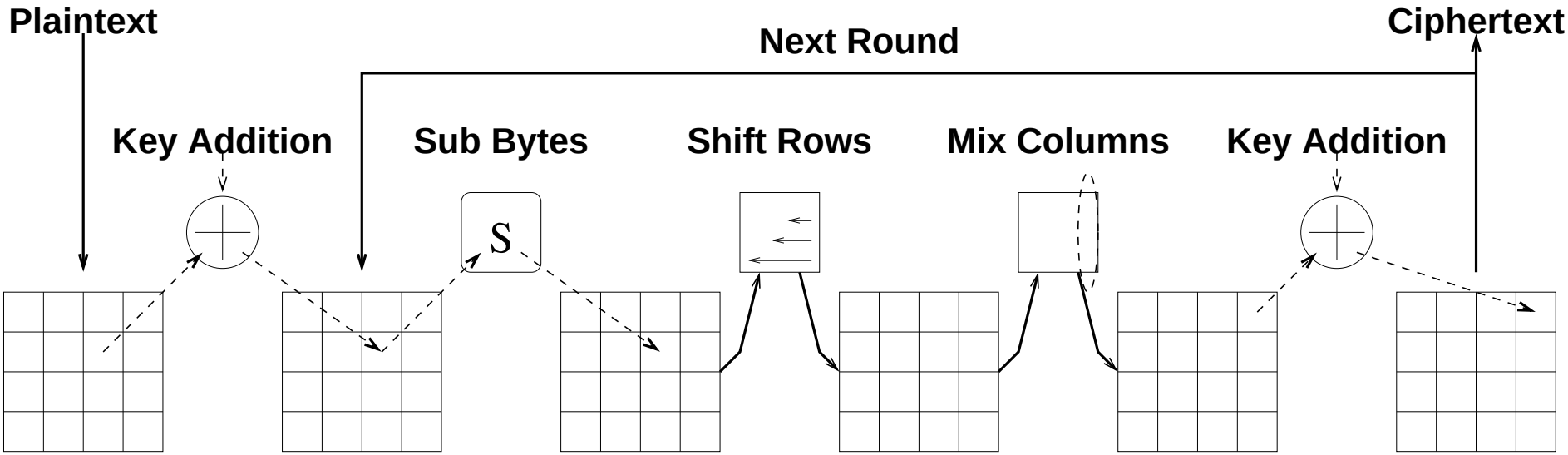


AES: Rundenstruktur



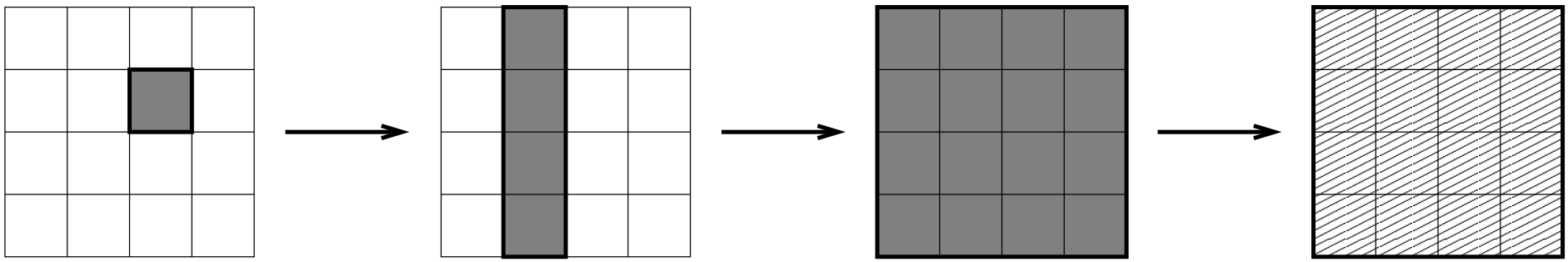
(Die letzte Runde ist ein Sonderfall, den wir hier nicht beachten.)

AES und differentielle Kryptanalyse



Wide Trail Strategy: Erzwinge viele *aktive S-Boxen!*

Beispiel über 3 Runden, wenn eine aktive S-Box in der ersten Runde:



AES: Zusammenfassung, Bemerkungen

- 128-bit Blockchiffre
- 3 verschiedene Schlüssellängen
  (128 bit, 192 bit, 256 bit)
- AES ging als Sieger aus einem mehrjährigen internationalen Wettbewerb hervor
- aktueller Standard
- inzwischen ähnlich intensiv analysiert wie der DES
- Empfehlung: Verwenden Sie AES falls möglich

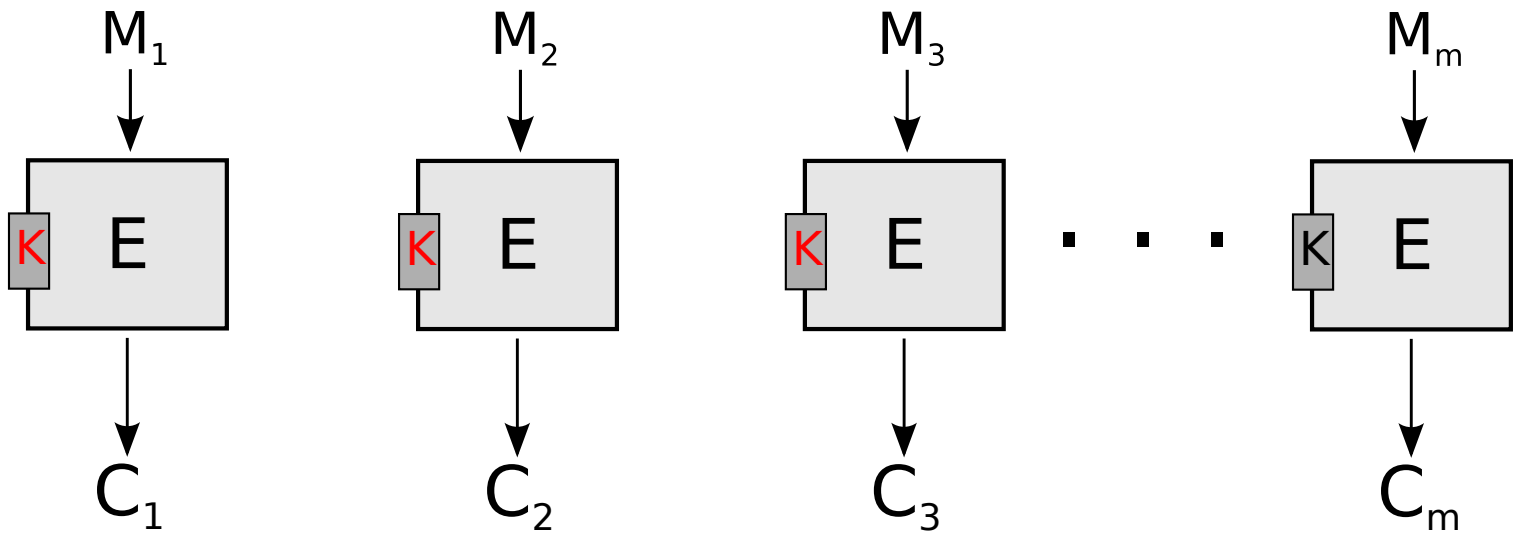
3.4: Betriebsarten für Blockchiffren

- Eine Blockchiffre kann immer nur n-Bit Nachrichten verarbeiten (z.B. DES mit $n = 64$ oder AES mit $n = 128$)
- Im folgenden beschäftigt sich mit symmetrischen Verschlüsselungsverfahren mit den folgenden Eigenschaften
  - Kurzer Schlüssel
  - Blockchiffre als zentraler **Baustein**
  - Verschlüsselung von Nachrichten *variabler* Länge
  - Verschlüsselung von multiplen Nachrichten unter dem gleichen Schlüssel
- Man nennt solche Verschlüsselungsverfahren auch **Betriebsmodus** oder **Betriebsart**.

Notation

- Im folgenden sei $M = M_1, \ldots, M_m$ mit $M_i \in \{0,1\}^n$.
- Wir bezeichnen M_i als den i-ten Nachrichtenblock.
- Im folgenden sei $C = \underbrace{C_0,}_{\textit{optional}} C_1, \ldots, C_m$ mit $C_i \in \{0,1\}^n$.
- Wir bezeichnen C_i als den i-ten Chiffretextblock.

Electronic Codebook (ECB)



$$C_i := E_{\mathbf{K}}(M_i).$$

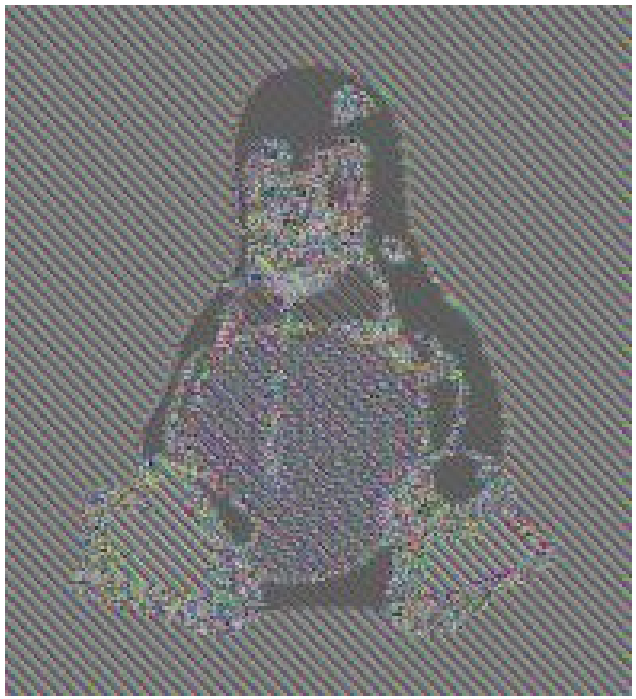
- Der ECB Modus ist einer von vier im Zusammenhang mit dem DES "offiziell standardisierten" “Modes of Operation” (“Betriebsarten”).
- In vielen *(schlechten!)* DES- oder AES-basierten Krypto-Produkten wird der ECB-Modus verwendet.

Der ECB Modus ist unsicher! (Warum?)

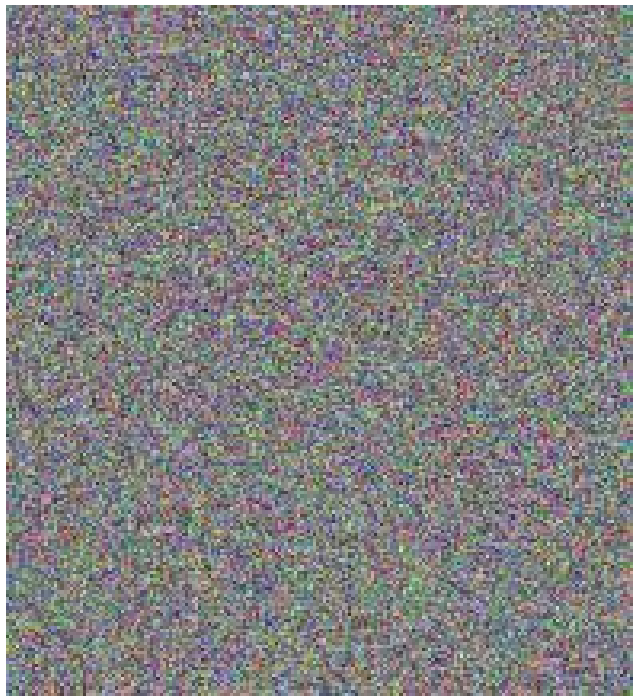
ECB-Verschlüsselung

Der Klartext:

ECB-Verschlüsselung:



Gute Verschlüsselung:

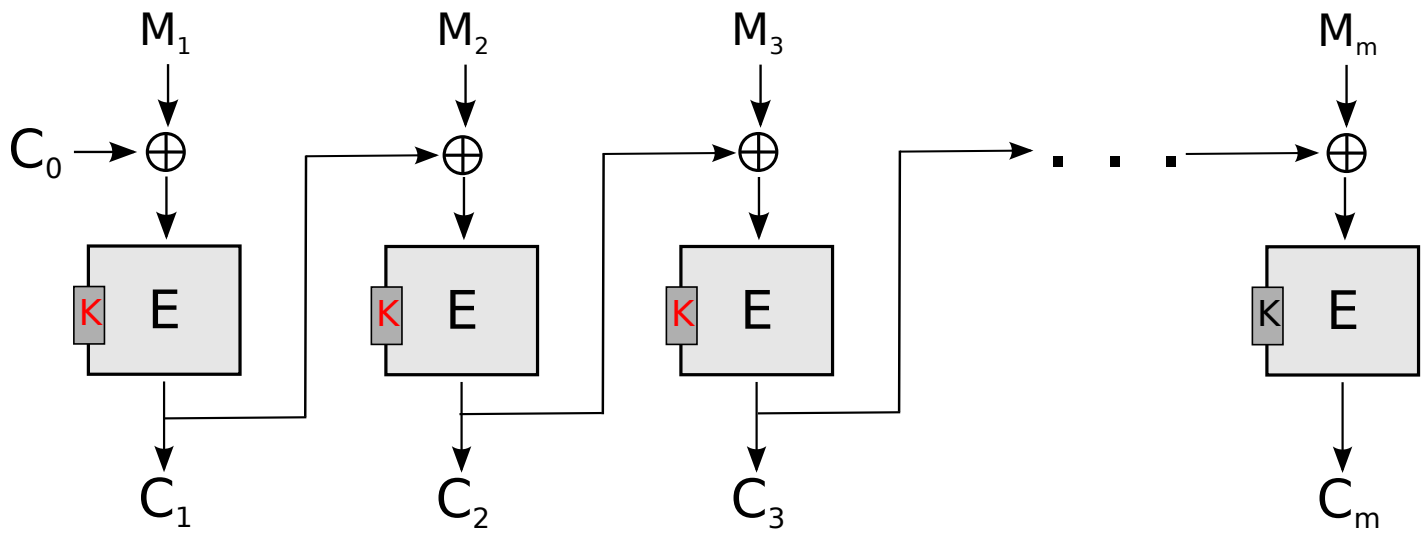


Folgerung

- Auch formal standardkonforme Kryptosysteme können unsicher sein. Electronic Codebook klingt gut, taugt aber nicht viel.

- Man muss genau analysieren, ob ein Kryptosystem etwas taugt und ob es für die eigenen Sicherheitsanforderungen eine geeignete Lösung liefert. Dazu muss man selbst seine eigenen Sicherheitsanforderungen kennen (“Bedrohungsanalyse”).

Cipher Block Chaining (CBC)

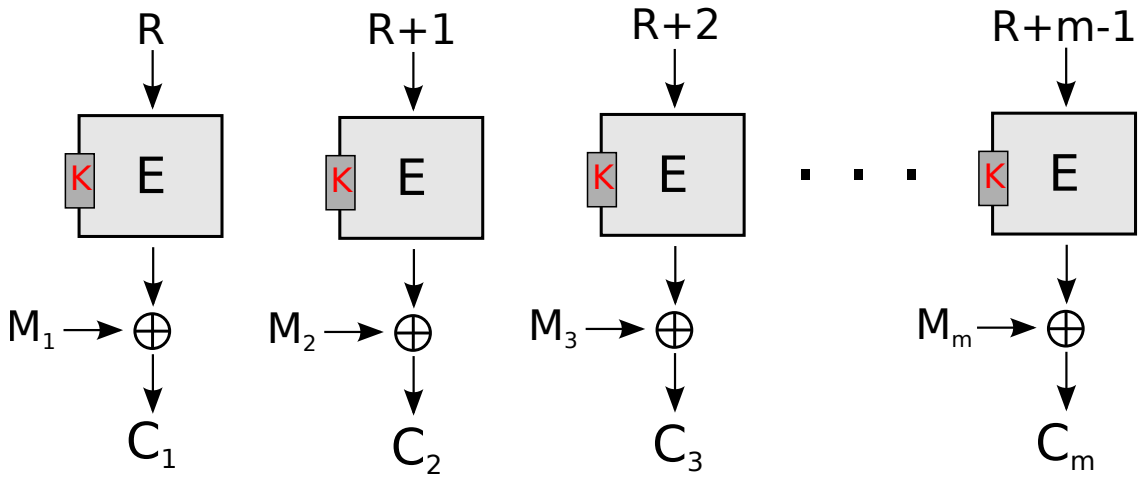


Zum Verschlüsseln der Nachricht $M = M_1, \ldots, M_m$:

1. Wähle einen zufälligen *Startwert* C_0
2. Für $1 \leq i \leq m$: $C_i := E_{\mathbf{K}}(M_i \oplus C_{i-1})$.
3. Der Chiffretext ist $(C_0, \ldots, C_n)$.

Und wie entschlüsselt man $(C_0, \ldots, C_n)$?

Counter-Mode



1. Wähle R zufällig.
2. Für $1 \leq i \leq m$: $C_i := M_i \oplus E_{\mathbf{K}}((i-1+R) \bmod 2^n)$
3. Chiffretext: $(R, C_1, \ldots, C_m)$.

Nebenbedingung: Die Zählerwerte dürfen sich nie wiederholen.

Eigenschaften von Ctr- und CBC-Mode

- Beides sind selbstsynchronisierender Betriebsmodus. ($\rightarrow$ Tafel)
- Der CBC- und Ctr-Modus sind mit zufälligen Startwert sicher
- CBC- und Ctr-Mode sind unsicher, falls ...
  1. ... der Angreifer Kontrolle über Initialization-Vector C_0 bzw. den Counter R hat.
  2. ... ein schon zuvor verwendeter Initialization-Vector oder Counter wiederverwendet wird.

Eigenschaften von Ctr- und CBC-Mode

- Selbstsynchronisierende Betriebsmodi. ($\rightarrow$ Tafel)
- Sind mit zufälligen Startwerten sicher
- Sind unsicher, falls . . .
  1. . . . der Angreifer Kontrolle über die Startwerte C_0 bzw. den Counter R hat.
  2. . . . ein schon zuvor verwendeter Initialization-Vector oder Counter wiederverwendet wird.

Sicherheit von Betriebsmodie

- Mit einem sicheren Betriebsmodus für eine sichere n-bit Blockchiffre $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ können Sie unter einem Schlüssel K bis zu $2^{n/2}$ Klartextblöcke verschlüsseln.

- Der Ctr-Mode wird beispielsweise unsicher, wenn zweimal der gleiche Input verschlüsselt wird (ECB-Problem).
  - Für $2^{n/2}$ zufällige Inputs $\{R_1, \ldots, R_{2^{n/2}}\}$ beträgt die Wahrscheinlichkeit einer Kollision mehr als 0.5.
  - Kollision: Es existieren $1 \leq i < j \leq 2^{n/2}$ mit $R_i = R_j$.
  - Grund: Geburtstagsparadoxon
  - In der Kryptographie wird $2^{n/2}$ als die Geburtstagsschranke bezeichnet.

Geburtstagsschranke

- Bei dem DES handelt es sich um eine 64-bit Blockchiffre.
  - Wegen dem Geburtstagsparadoxon können mit einem DES basierten Verschlüsselungsverfahren maximal $2^{64/2} = 2^{32}$ Klartextblöcke verschlüsselt werden.
  - $2^{32} \cdot 64\,\text{bit} = 2^{32} \cdot 2^6\,\text{bit} = 2^{38}\,\text{bit} = 2^{35}, \text{byte} 2^5\,\text{GB} = 32\,\text{GB}$.
  - Mit Sicherheitsmarge: Schlüsselwechsel nach 1 GB.

- Bei dem AES handelt es sich um eine 128-bit Blockchiffre.
  - Wegen dem Geburtstagsparadoxon können mit einem AES basierten Verschlüsselungsverfahren maximal $2^{128/2} = 2^{64}$ Klartextblöcke verschlüsselt werden.
  - $2^{64} \cdot 128\,\text{bit} = 2^{64} \cdot 2^7\,\text{bit} = 2^{71}\,\text{bit} = 2^{68}\,\text{byte} = 256\,\text{EB}$
  - Mit Sicherheitsmarge: Schlüsselwechsel nach 1 EB.

3.5: Exkurs: Das Geburtstagsparadoxon

Wie wahrscheinlich ist es, dass **von 23 Leuten** auf einem Fußballfeld (beide Teams und der Schiedsrichter) **zwei am gleichen Tag Geburtstag haben**

Etwa 50.7 %. **Überrascht?**

⟨http://de.wikipedia.org/wiki/Geburtstagsparadoxon⟩

Wie rechnet man das aus?

- $\mathbf{n} \leq 365$ Bälle, die jeweils zufällig in einen von 365 Körben geworfen werden.
- Die Wahrscheinlichkeit $\mathbf{p_n}$, dass von $\mathbf{n}$ Bällen in jedem Korb höchstens ein Ball ist:
  - $\mathbf{p_1} = (365/365) = 1$
  - $\mathbf{p_2} = (364/365)$
  - $\mathbf{p_3} = (364/365) * (363/365)$
  - $\mathbf{p_4} = (364/365) * (363/365) * (362/365)$
  - ...

$$\mathbf{p_n} = \prod_{0 \leq i < \mathbf{n}} \frac{365 - i}{365}.$$

- Die Wahrscheinlichkeit, dass in mindestens einem Korb mehr als ein Ball liegt, ist natürlich $\mathbf{1 - p_n}$, siehe Tabelle.

Anz.	$\mathbf{1 - p_n}$
1	0.0000
2	0.0027
3	0.0082
4	0.0164
⋮	⋮
20	0.4114
21	0.4437
22	0.4757
23	**0.5073**
24	0.5383
25	0.5687
26	0.5982
27	0.6269
28	0.6545

Verallgemeinerung des Geburtstagsproblems

- $\mathbf{k}$ Körbe und (wie bisher) $\mathbf{n} \leq \mathbf{k}$ Bälle.
- Die Wahrscheinlichkeit, dass Bälle in verschiedenen Körben landen:

$$\mathbf{p_n} = \prod_{0 \leq i < \mathbf{n}} \frac{\mathbf{k} - i}{\mathbf{k}}.$$

- Für "große" $\mathbf{k}$ erwartet man eine Kollision bei

$$\mathbf{n} = \sqrt{\frac{\pi}{2}} \cdot \sqrt{\mathbf{k}}$$

- Es gilt $\sqrt{\frac{\pi}{2}} \approx 1.25$.
- Ist $\mathbf{n} = \mathbf{c} \cdot \sqrt{\frac{\pi}{2}} * \sqrt{\mathbf{k}}$, dann sind $\mathbf{c^2}$ Kollisionen zu erwarten.

Anwendung Kryptographie

Sei $f \xleftarrow{\$} \mathcal{F}_n$ eine n-Bit Zufallsfunktion.

Frage: Nach wie vielen Anfragen an f ist mit einer Kollision zu rechen?

Antwort: Nach ca. $1.25 \cdot \sqrt{\mathbf{2^n}} \approx \sqrt{\mathbf{2^n}} = \mathbf{2^{n/2}}$ Anfragen.

Frage: Warum sollte die Länge eines mit dem Counter-Modus generierten Schlüsselstroms immer weit kleiner al $n \cdot 2^n$ Bits sein? Was wenn nicht?

Klassiche Verschlüsselungsverfahren

- Die folgenden klassichen Betriebsmodi sind für **zufällige** Startwerte (IV, Nonce, Zustand, C_o) sicher.
  - Cipher Block Chaining (CBC)
  - Cipher Feedback (CFB)
  - Counter (Ctr)
  - Output Feedback (OFB)
- Verwenden Sie diese Betriebsmodi mit AES-256, falls möglich.
- Im späteren Verlauf dieses Kapitel werden bessere Modis vorgestellt.

3.6: Fehlende Authentizität

- Viele Laien gehen unbewusst von der Annahme aus, dass Daten, die (durch gute Verschlüsselung) sicher vor Mithörern sind, auch sicher vor Veränderungen sind, zumindest sicher vor gezielten Veränderungen.

→ **Kryptographie sollte man nur einsetzen, wenn man zuvor eine Bedrohungsanalyse gemacht hat. Sonst kann es sein, dass man hervorragende Schutzmaßnahmen einsetzt, die überhaupt nicht vor den tatsächlichen Gefahren schützen.**

Als **Ausweg** bietet sich der Einsatz kryptographischer “Message Authentication Codes” (MACs) an (demnächst mehr dazu).

Fehlende Authentizität kann sogar die Vertraulichkeit gefährden

- Drahtlos: Verschlüsseltes Datenpaket $D = (Ctr_{\mathbf{K}}(R_H, H), Ctr_{\mathbf{K}}(R_M, M))$, bestehend aus Header H und Nutzdaten M.
- Gateway: Entschlüssele D und leite den Klartext (H, M) (drahtgebunden) an die in H angegebene IP-Adresse.
- Keine überprüfung der Authentizität durch das Gateway. Vielleicht kann der Empfänger erkennen, wenn M manipuliert wurde – das ist nicht das Problem des Gateways!
- Eve
  - hat $D = (Ctr_{\mathbf{K}}(R_H, H), Ctr_{\mathbf{K}}(R_M, M))$ abgehört und will M wissen,
  - weiss (oder vermutet), dass D an die Adresse **B** gerichtet ist (eigene Adresse **E**),
  - addiert deshalb $\mathbf{B} \oplus \mathbf{E}$ zur Adresse im Header
  - und schickt das manipulierte Datenpaket D' an das Gateway.
- Das Gateway entschlüsselt D' und leitet den Klartext an die Adresse **E** von Eve.

3.7: Message Authentication Codes (MACs)

- Eine **MAC** wird def. durch drei Mengen
  1. $\mathcal{M}$: Nachrichtenmenge
  2. $\mathcal{T}$: Tagmenge (kryptographische Prüfsumme)
  3. $\mathcal{K}$: Schlüsselmenge
- und zwei (bzw. drei) effiziente Algorithmen
  1. $\mathcal{G} : \emptyset \to \mathcal{K}$ (Schlüssel erzeugen)
  2. $\text{MAC} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ (Prüfsumme generieren)
  3. $\mathcal{D} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{\text{SUCCESS}, \text{FAILURE}\}$ (Prüfsumme Verifizieren)
- Für alle Nachrichten Schlüssel Tuple $(M, \mathbf{K}) \in \mathcal{M} \times \mathcal{K}$ gilt: $\text{Verify}_{\mathbf{K}}(M, \text{MAC}_{\mathbf{K}}(M)) = \text{SUCCESS}$
- Bei einem sicheren MAC ist es ohne Kenntnis von **K** praktisch unmöglich ein Tuple (M, T) zu finden für das gilt: $\text{Verify}_{\mathbf{K}}(M, T) = \text{SUCCESS}$

Aufgabe und Sicherheit von MACs

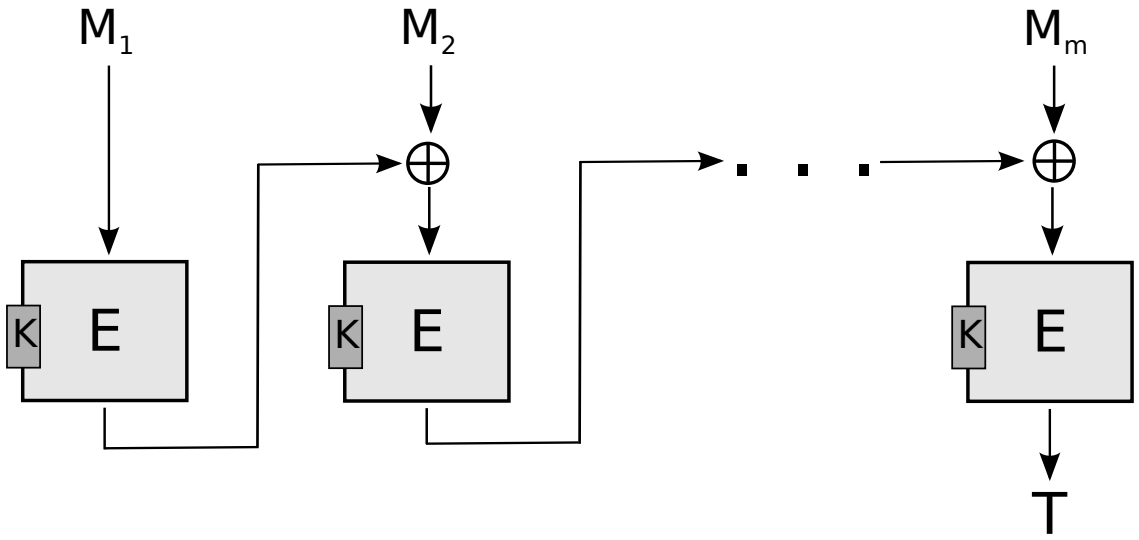
- MACs sichern die Echtheit (Authentizität/Integrität) von Nachrichten.
  - Schutz von Bank-Transaktionen vor Verfälschung
  - Schutz von Computerprogrammen vor Viren
  - ...
- Ein MAC ist sicher, wenn es ohne den Schlüssel nicht möglich ist für eine frische Nachricht M einen Tag T zu generieren mit $\text{Verify}_{\mathbf{K}}(M, T) = \text{SUCCESS}$

Verify

Die Verifikationsfunktion $Verify_K$ wird häufig wie folgt implementiert.

Verify$_K(M, T)$:
1: **if** $MAC_K(M) = T$ **then**
2: **return** SUCCESS
3: **else**
4: **return** FAILURE
5: **end if**

Beispiel für einen unsicheren MAC: CBC-MAC



Zum Authentisieren der Nachricht $M = M_1, \ldots, M_m$:

- Setze $C_0 = 0$
- Für $1 \leq i \leq m$: $C_i \leftarrow E_{\mathbf{K}}(M_i \oplus C_{i-1})$
- Es gilt CBC-MAC$(M) := C_n$ und damit $T \leftarrow C_n$

Wie verifiziert man ein Tupel (M, T)

Angriff auf den CBC-MAC (2)

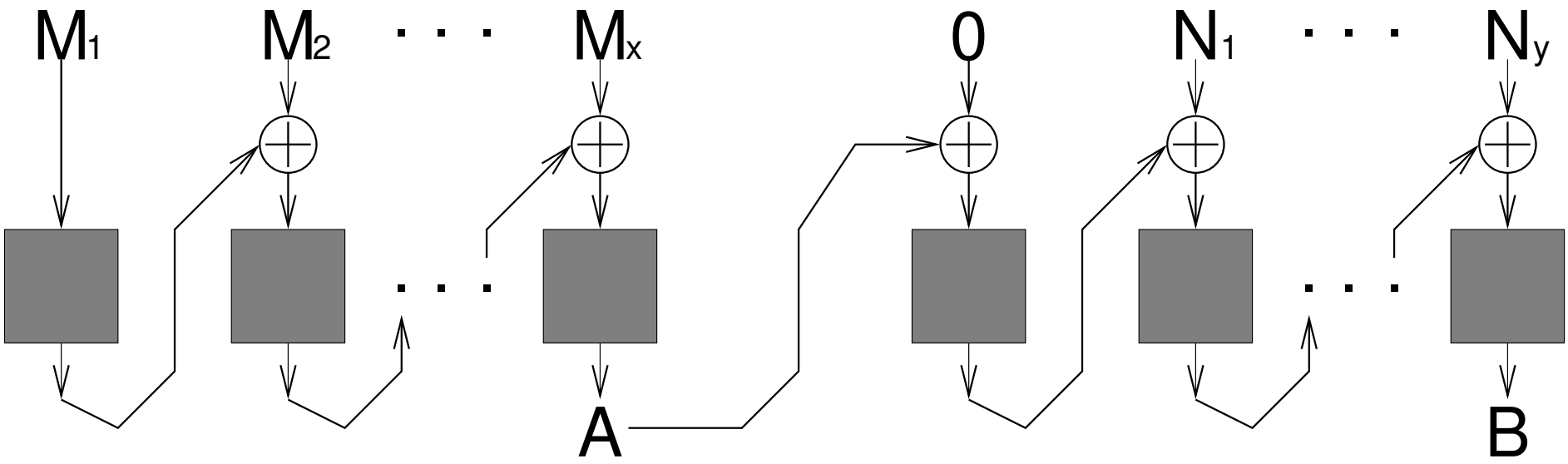
“Splicing Attack” für Nachrichten variabler Länge

- Angreifer erfragt $\boxed{A \leftarrow \mathrm{MAC}(M)}$ mit $M = (M_1, \ldots, M_x)$.
- Angreifer erfragt $\boxed{B \leftarrow \mathrm{MAC}(N)}$ mit $N = (A, N_1, \ldots, N_y)$.
- Berechne die Fälschung (M', B) mit

$$M' = (M_1, \ldots, M_x, 0, N_1, \ldots, N_y)$$

“Splicing”

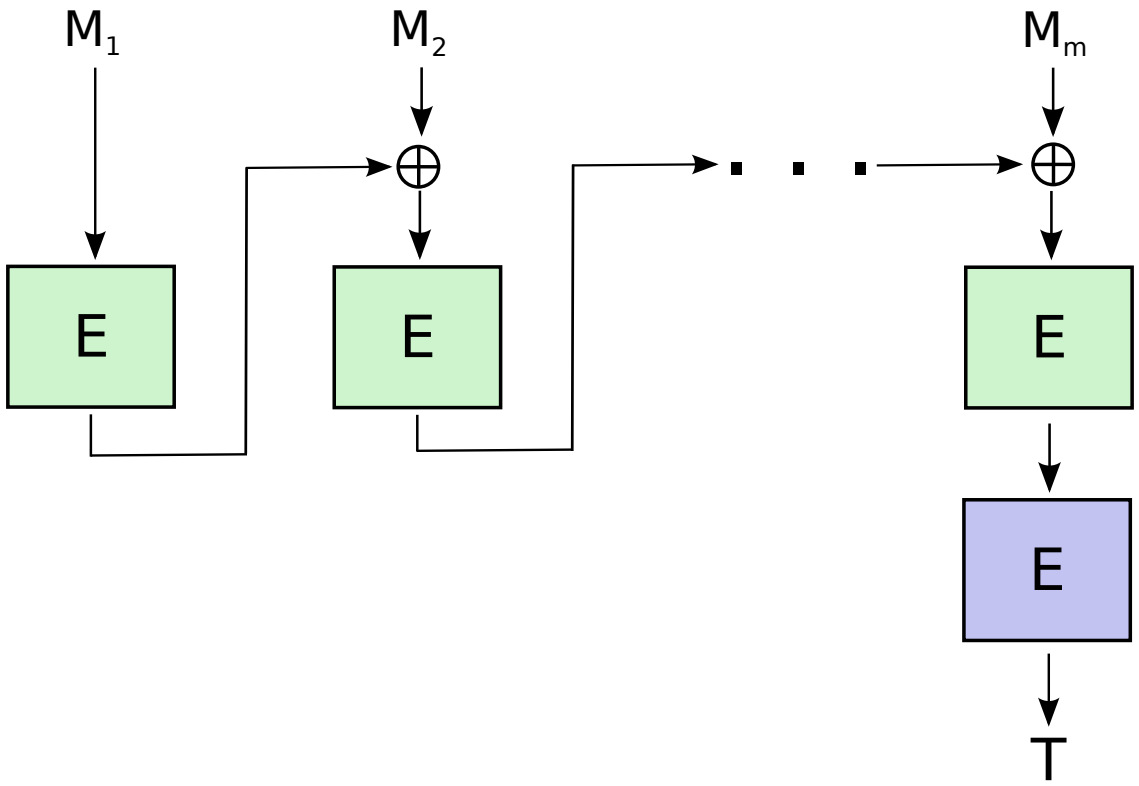
Graphische Darstellung des Angriffs



Encrypted CBC-MAC (CMAC)

Eine sichere Variante des CBC-MAC

$$\text{MAC}_{K_1,K_2}(M_1,\ldots,M_m) = E_{K_2}(\text{CBC-MAC}_{K_1}(M_1,\ldots,M_m))$$



MAC Empfehlungen

Die folgenden MACs sind sicher und empfehlenswert

- CMAC-AES
- HMAC-SHA256
- HMAC-SHA512
- HMAC-SHA3
- Poly1305-AES

3.8: Authentifizierte Verschlüsselung

- Bei einer authentifizierte Verschlüsselung wird die
  - die Vertraulichkeit (durch Verschlüsselung) und
  - die Integrität (durch kryptographische Prüfsumme)

  eines Klartextes geschützt.

- Ein Betriebsmodus welcher authentifiziert verschlüsselt wird Authenticated Encryption Scheme (AE-Scheme) gennant.

- Ein AE-Scheme gibt einen Chiffretext C und die kryptographische Prüfsumme (Tag) T zurück.

- Ja, nach AE-Scheme handelt es sich bei T, um einen MAC des Klar- oder Chiffretextes, oder sogar um einen MAC über Klar- und Chiffretext

Associated Data (Header)

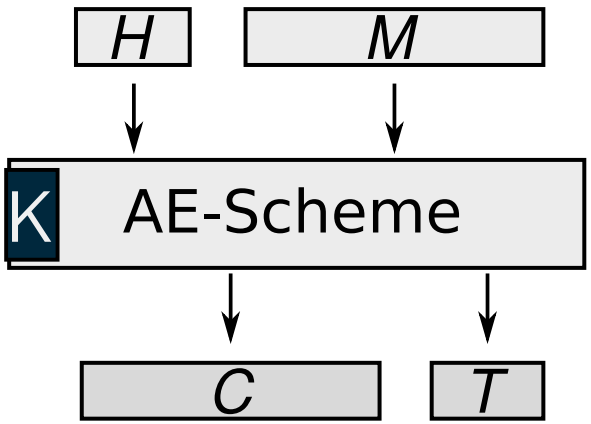
- Im Gegensatz zu klassichen Betriebsmode verarbeiten die meisten AE-Schemes noch Associated Data (AD).
- AD sind zusätzliche Daten, die noch zum Klartext gehören aber nicht verschlüsselt werden sollen.
- Ein AE-Scheme schützt nur die Integrität der AD nicht deren Vertraulichkeit.
- Oftmals handelt es sich bei AD um den Header eines verschlüsselten Netzwerkpaketes.
- Header eine alternative Bezeichung für Associated Data.
- Die Sicherheit eines AE-Schemes beruht darauf, dass der gleiche Header nicht mehr als einmal verwendet wird.
- Beim Header handelt es sich wie beim IV um eine Nonce (Number used only once)

AE-Scheme

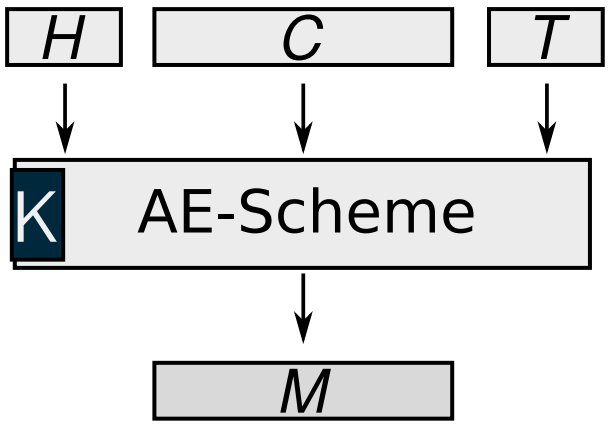
- Sei $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ eine n-bit Blockchiffre, eine AE-Scheme $\mathcal{E} : \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^n$ das auf E basiert können viele Nachrichten unter einem k-bit Schüssel **K** authentisiert verschlüsselt werden.
- $(C, T) = \mathcal{E}_{\mathbf{K}}(H, M)$ für eine beliebig lange Nachrichten $M \in \{0,1\}^*$ und einen beliebig langen Header $H \in \{0,1\}^*$.
- Gilt für das Tupel (H, M, C, T) die Gleichung $(C, T) = \mathcal{E}_{\mathbf{K}}(H, M)$, dann ist (H, C, T) valide.
- Ist (H, C, T) valide, dann gilt: $M = \mathcal{D}_{\mathbf{K}}(H, C, T)$
- Ist (H, C, T) nicht valide, dann gilt: $\perp = \mathcal{D}_{\mathbf{K}}(H, C, T)$
- Ist ein AE-Scheme $\mathcal{E}$ sicher, dann ist es ohne Kenntnis des Schlüssels **K** praktisch nicht möglich ein valides und frisches Triple (H, C, T) zu generieren.

AE-Scheme: Illustration

Verschlüsselung



Entschlüsselung



H: Header

M: Nachricht (Klartext)

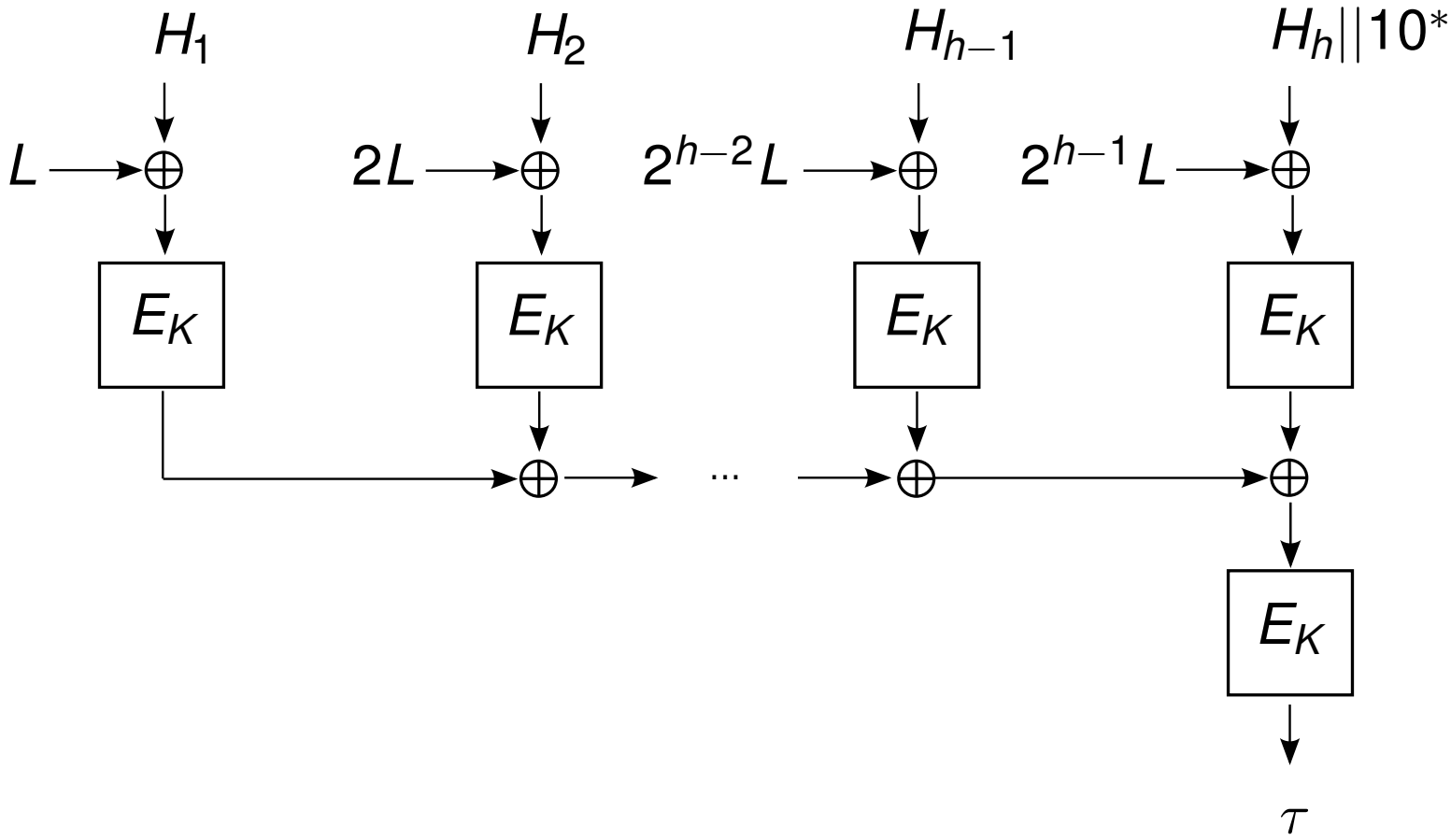
C: Verschlüsselte Nachricht (Chiffretext)

T: Kryptographische Prüfsumme (Tag)

AE-Scheme: POET

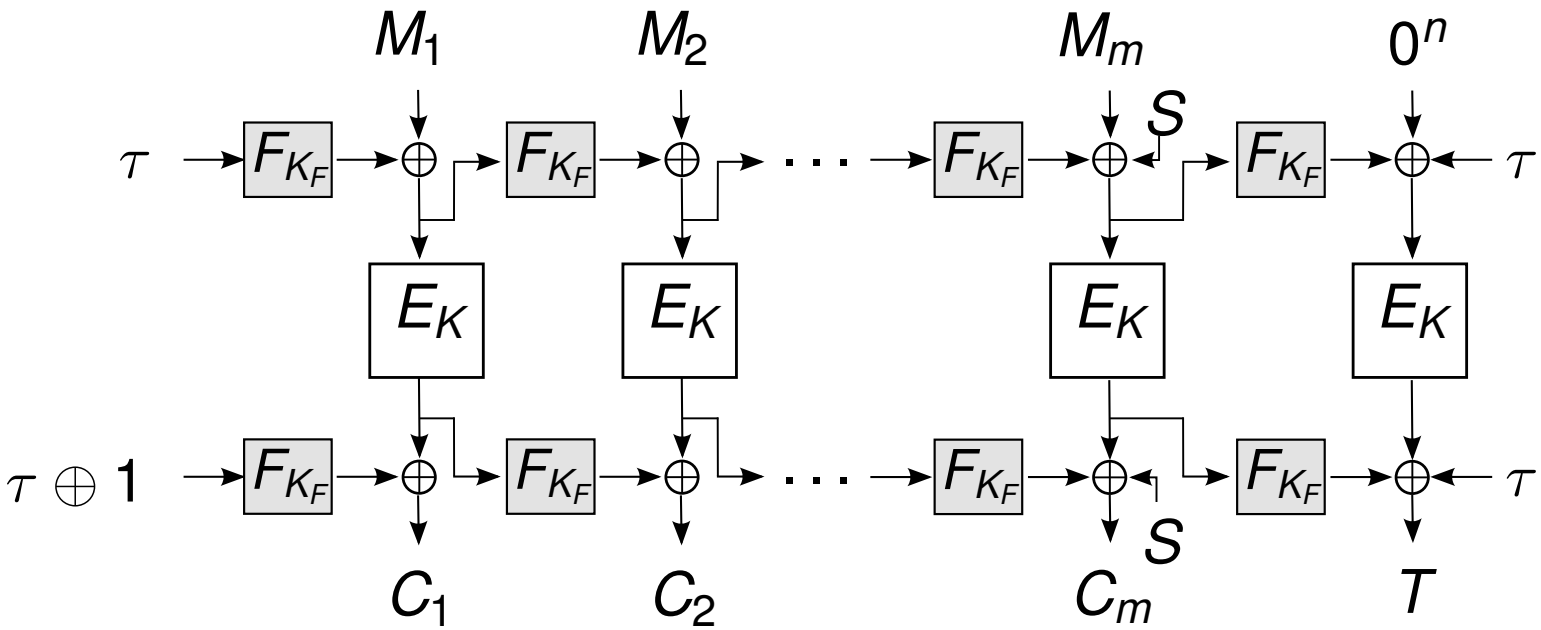
- Bei POET handelt es sich um ein AES-128 basiertes AE-Scheme.
- Es wurde 2016 von Abed, Fluhrer, Forler, List, Lucks, McGrew und Wenzel vorgestellt.
- POET wurde entwickelt transparent Daten in einem optische Transportnetz (OTN) zu verschlüsseln. (Datenrate > 100 GBit/s)
- POET war ein Kandidat für die Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) und hat es in die zweite Runde geschafft. `https://competitions.cr.yp.to/caesar-submissions.html`

POET: Header-Verarbeitung (PMAC Variante)



- H ist der Header
- $L = E_{\mathbf{K}}(1)$
- $K = E_{\mathbf{K}}(0)$
- τ ist der MAC von H

POET: Klartext-Verarbeitung



- $K = E_{\mathbf{K}}(0)$
- $K_F = E_{\mathbf{K}}(2)$
- F ist AES-128 oder AES4.
- AES4 steht für Runden AES.

- $S = E_{\mathbf{K}}(|M|)$
- $|M|$ Länge von M in Bits.
- τ ist der MAC von H
- T ist der Tag.

POET: API (1/4)

```
void keysetup(poet_ctx_t *ctx, const byte_t key[KEYLEN]);
```

- Initialisierung des POET-Kontextes `ctx` mit Hilfe eines 128-bit (16-byte) Schlüssels `key`.
- Bei der Initalisierung werden unter anderem die Maske L für und Rundenschlüssel für AES berechnet.

```
void process_header(poet_ctx_t *ctx, const byte_t *header,
                    uint64_t header_len);
```

- Parameter `*ctx`: Zeiger auf einen bereits initialisierten POET-Kontext.
- Parameter `header`: Der zu verarbeitende Header.
- Parameter `len`: Die Länge des Headers in Bytes.

POET: API (2/4)

```
void encrypt_block(poet_ctx_t *ctx, const block mi, block
    ci);
```

- Parameter `*ctx`: Zeiger auf einen bereits initialisierten POET-Kontext.
- Parameter `mi`: Zu verschlüsselter Klartextblock
- Parameter `ci`: Chiffretextblock (Verschlüsselung von `mi`)

```
void decrypt_block(poet_ctx_t *ctx, const block in, block
    out);
```

- Parameter `*ctx`: Zeiger auf einen bereits initialisierten POET-Kontext.
- Parameter `ci`: Zu entschlüsselder Chiffretextblock
- Parameter `mi`: Klartextblock (Entschlüsselung von `ci`)

POET: API (3/4)

```
void encrypt_final(poet_ctx_t *ctx,
                   const byte_t *m, uint64_t mlen,
                   byte_t *c, byte_t tag[TAGLEN]);
```

- Mit diesem Aufruf wird die Verschlüsselung eines Klartextes abgeschlossen.
- Parameter `*ctx`: Zeiger auf einen bereits initialisierten POET-Kontext.
- Parameter `m`: Der letzte Teil des zu verschlüsselten Klartextes `m`
- Parameter `mlen`: Länge von `m` in Bytes
- Parameter `c`: Die letzten `m` Bytes des Chiffretextes.
- Parameter `tag`: Kryptographische Pürfsumme über Header, Klartext und Chiffretext.

POET: API (4/4)

```
int decrypt_final(poet_ctx_t *ctx,
                  const byte_t *c,  uint64_t clen,
                  const byte_t tag[TAGLEN], byte_t *m);
```

- Diese Funktion gibt bei Erfolg 0 zurück, d. h. sie gibt 0 zurück wenn die Entschlüsselung des Chiffretext erfolgreich war.
- Parameter `*ctx`: Zeiger auf einen bereits initialisierten POET-Kontext.
- Parameter `c`: Der letzte Teil des zu entschlüsselten Chiffretextes `c`
- Parameter `clen`: Länge von `c` in Bytes
- Parameter `tag`: Kryptographische Pürfsumme.
- Parameter `m`: Die letzten `m` Bytes des Klartextes

POET: Ablauf Verschlüsselung

```
#include <stdint.h>
#include <poet.h>
...
poet_ctx_t ctx;
keysetup(&ctx, key);
process_header(&ctx, header, hlen)
int len = BLOCKLEN;
while(len==BLOCKLEN) {
   len = read(fdm, mi, BLOCKLEN);
   if(len == BLOCKLEN)  encrypt_block(&ctx, mi, ci)
   else  encrypt_final(&ctx, mi, (uint64_t) len, ci, tag);
   write(fdc, ci, len);
}
...
```

POET: Ablauf Entschlüsselung

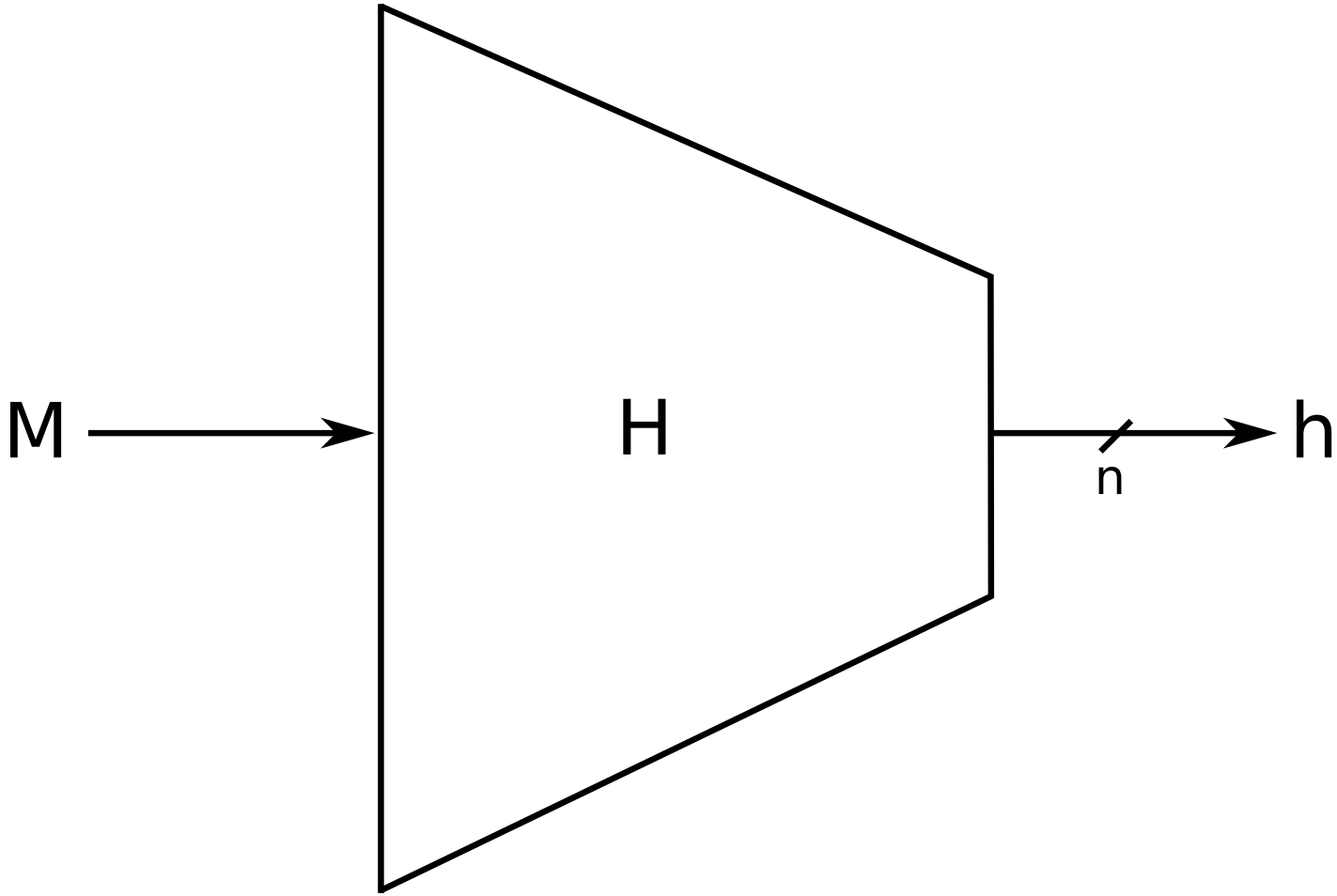
```
#include <stdint.h>
#include <poet.h>
...
poet_ctx_t ctx;
keysetup(&ctx, key);
process_header(&ctx, header, hlen)
int len = BLOCKLEN;
while(len==BLOCKLEN) {
   len = read(fdc, ci, BLOCKLEN);
   if(len == BLOCKLEN)  decrypt_block(&ctx, ci, mi)
   else  r = decrypt_final(&ctx, ci, (uint64_t) len,
                           tag, mi);
   write(fdm, mi, len);
}
if (r) delete(plaintext);
...
```

AE-Schemes Empfehlungen

1. RIV (Abed, Forler, List, Lucks, Wenzel 2016)
2. SIV (Rogaway, Shrimpton 2006)
3. COLM (Andreeva et. al 2016)
4. POET (Abed, Fluhrer, Forler, List, Lucks, McGrew, Wenzel 2016)
5. McOE-G (Fleischmann, Forler, Lucks, [Wenzel] 2012)
6. OCB (Krovetz, Rogaway 2010)
7. CCM (Whiting, Housley, Ferguson 2002)
8. GCM (McGrew, Viegan 2004)

Verwenden Sie einen diese Betriebsmodie, falls möglich.

3.9: Kryptographische Hashfunktionen



Was ist eine Hashfunktion

Hashfunktion

Eine Hashfunktion H ist eine Funktion, deren welche Bitstrings beliebiger Länge auf Bitstrings fester Länge abbildet.

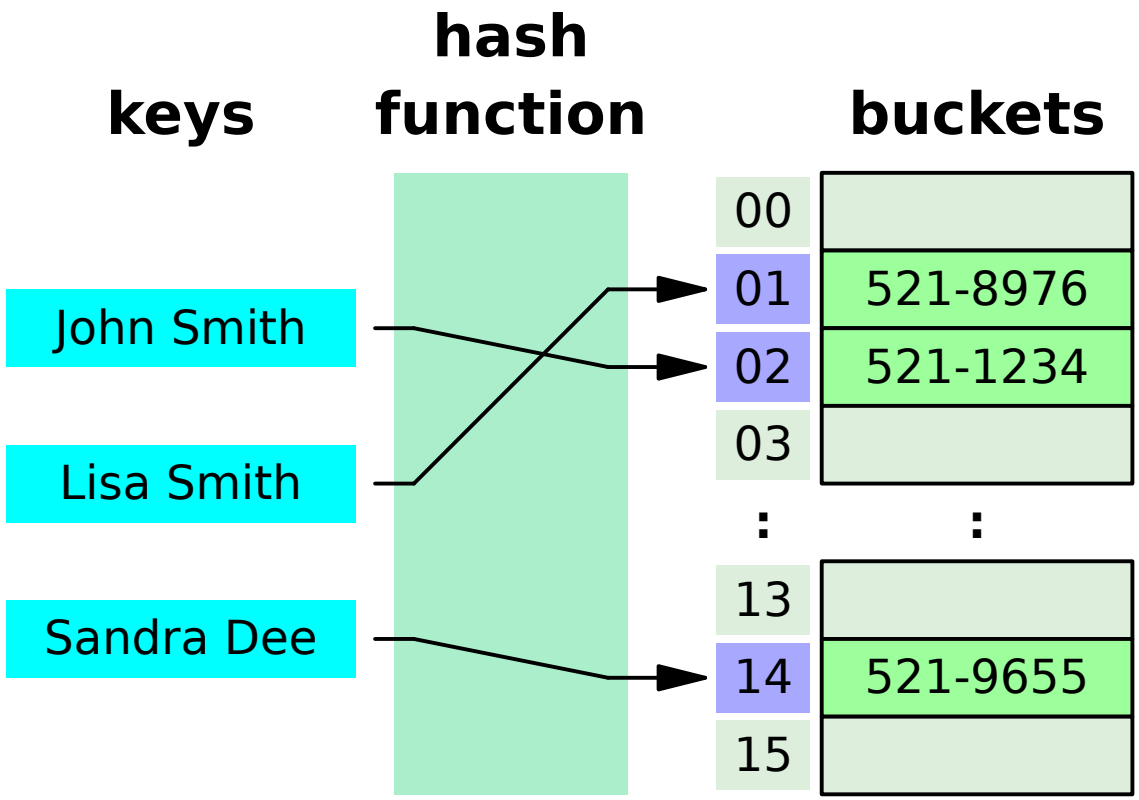
n-bit Hashfunktion

Eine n-Bit Hashfunktion $H : \{0,1\}^* \rightarrow \{0,1\}^n$ bildet eine beliebig Großen Bitstring auf einen n-Bit Wert ab.

Zentrale Anwendungsgebiete von Hashfunktionen

- Effiziente Datenstruktur (z.B. Hashtabellen)
- Kryptographie
  - Kryptographische Prüfsummen (z.B. HMAC)
  - Digitale Unterschriften (z.B. Full Domain Hash)
  - Password-Hashing (z.B. Argon2)
  - Proof-of-Work (z.B. Blockchain)
  - ...

Eigenschaften von Hashtabellen



Quelle: `https://en.wikipedia.org/wiki/Hash_table`

- Vorteil: Schneller Zugriff
- Nachteil: Es können Kollisionen auftreten.

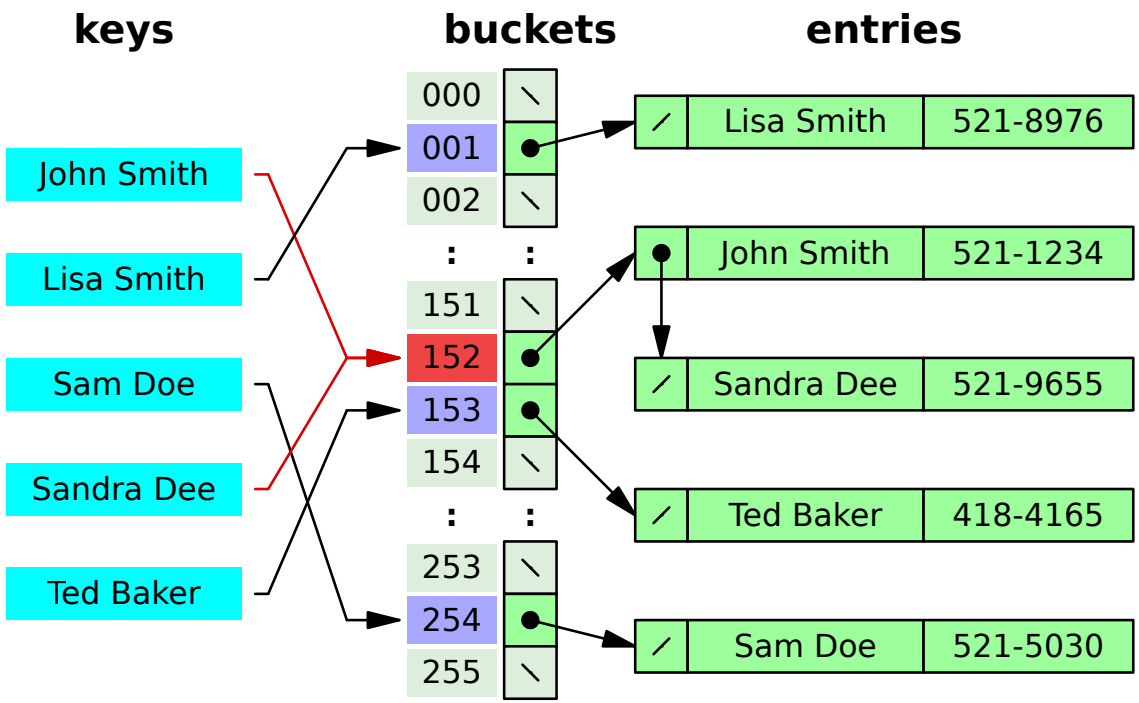
Kollisionen

Kollision

Sei H eine Hashfunktion, dann haben wir eine Kollision falls gilt:

$$H(x) = H(x') \qquad x \neq x'.$$

Kollisionen bei Hashtabellen



Quelle: `https://en.wikipedia.org/wiki/Hash_table_coll`

Kollision wirken sich negativ auf die Performance von Hashtabellen aus. (**Warum?**)

Kryptographische Hashfunktionen

Wir nennen eine n-bit Hashfunktion H eine sichere kryptographische Hashfunktion, wenn Sie die folgenden zwei Eigenschaften hat.

- Kollisionsresistenz (*engl. collision resistance*)
  Es ist praktisch unmöglich eine Kollision für H zu finden.
  Formal: Aufwand sollte $2^{(n/2)}$ Aufrufe von H entsprechen.

- Einwegeigenschaft (*engl. preimage resistance*)
  Es ist praktisch unmöglich für einen gegebenen Ausgabewert $y \in \{0,1\}^n$ eine passenden Eingabewert x mit $H(x) = y$ zu finden.
  Formal: Aufwand sollte 2^n Aufrufe von H entsprechen.

Sind die folgenden Hashfunktionen sicher?

- $H_1(M) = M \bmod 256$
- $H_2(M) = (3 \cdot M + 11) \bmod 256$
- $H_3(M) = \bigoplus_{i=1}^{m} M_i$

Sicherheitseigenschaft von Hashfunktionen

Theorem 4.1 (Sicherheit von Hashfunktionen)

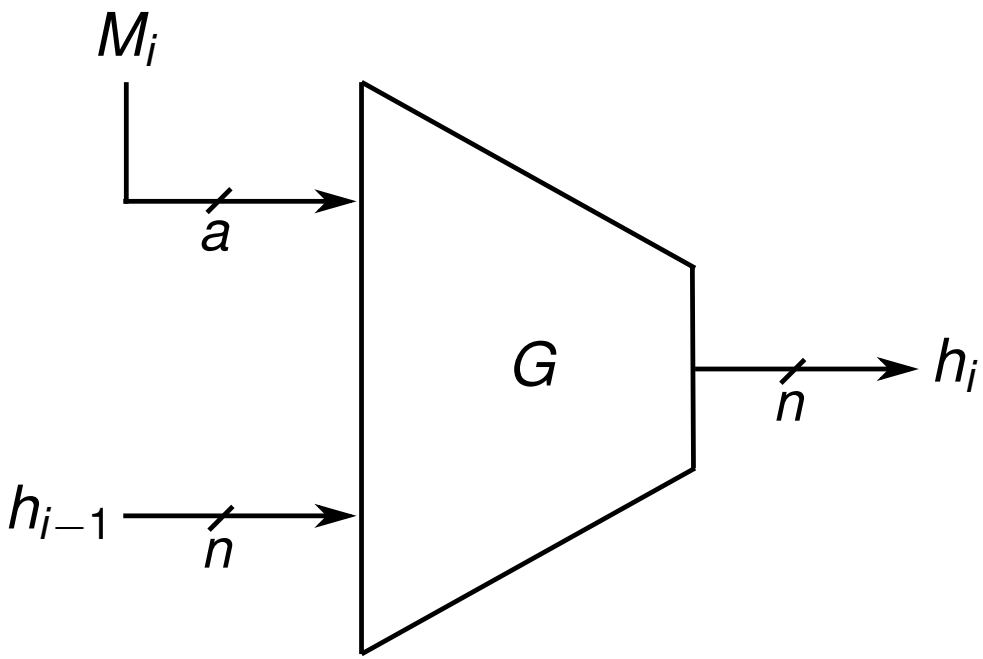
Sei **H** *Eine Hashfunktion die nicht preimage sicher ist, dann gilt:* **H** *ist nicht kollisionsresistenz.*

($\rightarrow$ Tafel)

Sichere kryptographische Hashfunktionen

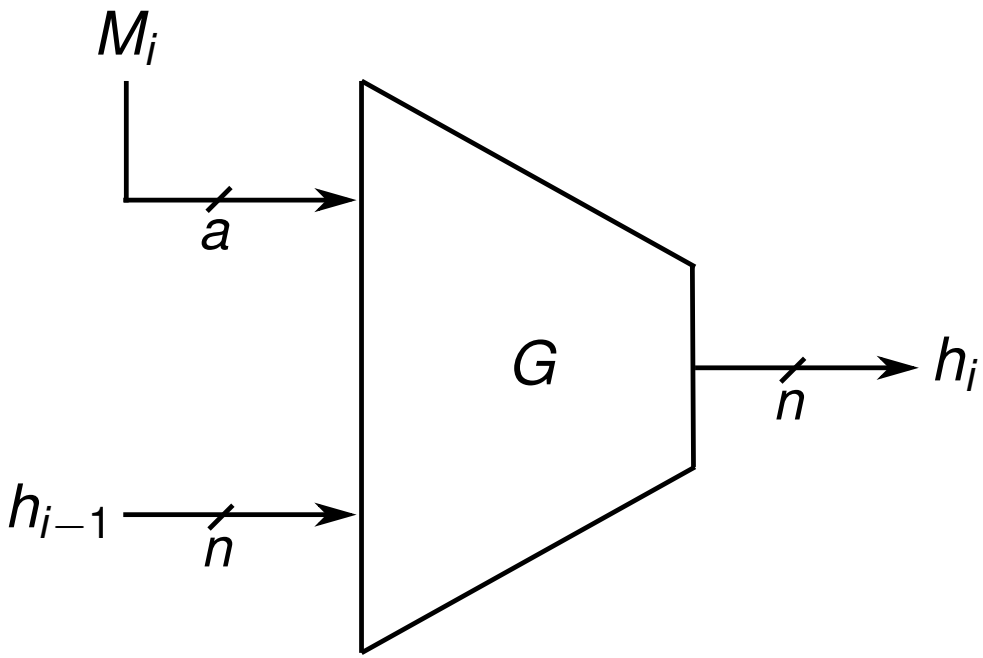
- Sichere kryptographische Hashfunktionen
  - SHA-256
  - SHA-512
  - SHA-3
  - Blake2
  - Skein
- Unsichere kryptographische Hashfunktionen
  - MD5 (n=128)
    - Kollision: Aufwand 2^{39} [Marc Stevens]
    - Preimage: Aufwand $2^{123.4}$ [Yu Sasaki, Kazumaro Aoki]
  - SHA-1 (n=160; Kollision mit Aufwand 2^{61} [Marc Stevens])

Kompressionsfunktion (1/2)



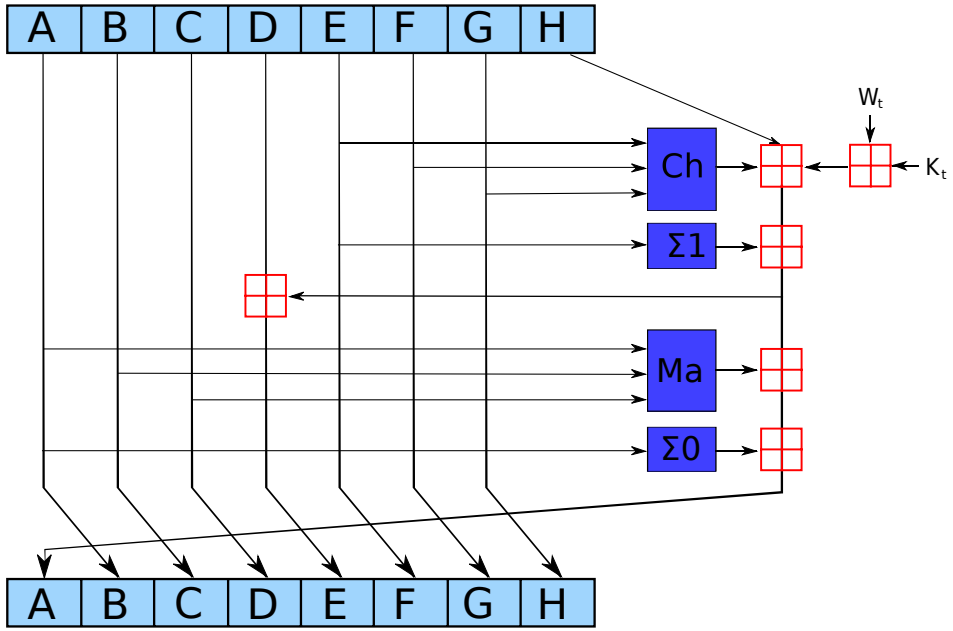
Ein Hashfunktion ist ein Betriebsmodus für eine Kompressionsfunktion

Kompressionsfunktion (2/2)



- Die Kompressionsfunktion komprimiert einen a-bit Nachrichtenblock M_i und einen n-bit Zustand h_{i-1} zu einem n-bit Ausgabewert h_i.
- Es gilt: $h_i = G(h_{i-1}, M_i)$
- Initaler Verkettungswert h_0

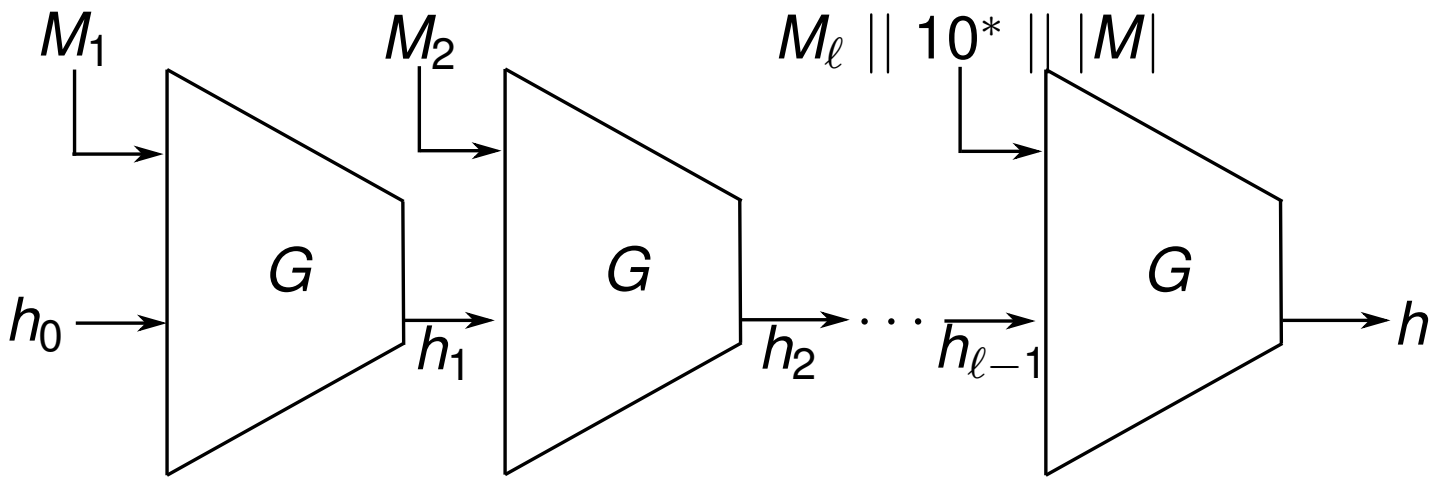
SHA-512 Kompressionsfunktion



Quelle: `https://en.wikipedia.org/wiki/SHA-2`

- 80-Runden unsymmetrisches (*engl. unbalanced*) Feistel-Netzwerk mit einer Wortgröße von 64 bit.
- Die Ch, Σ1, Ma, Σ0 Funktionen bestehen aus den folgenden Bitoperationen: $\wedge$, $\oplus$, $\neg$ und $\ggg$.
- Größe eines Nachrichtenblocks: 1024 bit.

Merkle Damgård Konstruktion



- Merkle Damgård Konstruktionen: MD5, SHA-1 und SHA-256/512.
- Der letzte Nachrichtenblock wird gepadded.
- **Frage**: Warum ist das Padding relevant für die Sicherheit?

Zusammenfassung: Symmetrische Kryptographie

Sie sollten . . .

- . . . wissen was symmetrische Kryptographie ist
- . . . das Konzept der Blockchiffre verstanden haben
- . . . die funktionsweise von AES verstanden haben
- . . . das Konzept der Betriesmodi verstanden haben
- . . . wissen was ein MAC ist
- . . . wissen was ein AE-Scheme ist
- . . . wissen wann eine Hashfunktion sicher ist