AES – Eine Einführung in Cryptography

Andreas Hofmeier

15.01.2006

Übersicht

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.
- Öffentliches Netzwerk kann mit Postkarten verglichen werden

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.
- Öffentliches Netzwerk kann mit Postkarten verglichen werden
- Einfaches Kopieren

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.
- Öffentliches Netzwerk kann mit Postkarten verglichen werden
- Einfaches Kopieren
- Kopieren/Lesen hinterlässt keine Spuren

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.
- Öffentliches Netzwerk kann mit Postkarten verglichen werden
- Einfaches Kopieren
- Kopieren/Lesen hinterlässt keine Spuren
- mathematische Algorithmus, abhängig von Schlüssel

Einführung: Was ist Cryptography? Warum brauchen wir das?

- Ein Verfahren um Daten vor unbefugtem Zugriff zu schützen.
- Öffentliches Netzwerk kann mit Postkarten verglichen werden
- Einfaches Kopieren
- Kopieren/Lesen hinterlässt keine Spuren
- mathematische Algorithmus, abhängig von Schlüssel
- symmetrische Verschlüsselung

Summenverfahren

Beispiel 1:

- Verschlüsselung
  Klartext (6124627) + Schlüssel (4531773)

$$\begin{array}{r} 6124627 \\ +4531773 \\ \hline 10656400 \end{array}$$

Ergebnis: Chiffre (10656400)

Summenverfahren

Beispiel 1:

- Verschlüsselung
  Klartext (6124627) + Schlüssel (4531773)

$$\begin{array}{r} 6124627 \\ +4531773 \\ \hline 10656400 \end{array}$$

  Ergebnis: Chiffre (10656400)

- Entschlüsselung
  Chiffre (10656400) minus Schlüssel (4531773)

$$\begin{array}{r} 10656400 \\ -4531773 \\ \hline 6124627 \end{array}$$

  Ergebnis: Klartext (6124627)

Summenverfahren

Beispiel 2:

- Verschlüsselung
  Klartext (“TREFFEN UM SIEBEN”) + Schlüssel (“ZEBRA”)

```
   TREFFEN UM SIEBEN
 +            ZEBRA
---------------------
   ???????????
```

Summenverfahren

Beispiel 2:

- Verschlüsselung
  Klartext (“TREFFEN UM SIEBEN”) + Schlüssel (“ZEBRA”)

```
   TREFFEN UM SIEBEN
 +            ZEBRA
---------------------
         ???????????
```

- Jeder Buchstabe der Nachricht wird einzeln verschlüsselt.

Summenverfahren

Beispiel 2:

- Verschlüsselung
  Klartext (“TREFFEN UM SIEBEN”) + Schlüssel (“ZEBRA”)

```
   TREFFEN UM SIEBEN
 +            ZEBRA
---------------------
         ???????????
```

- Jeder Buchstabe der Nachricht wird einzeln verschlüsselt.
- Jedem Buchstabe wird aufgrund seine Position im Alphabet eine Zahl zugeordnet: `A = 0, B = 1, ... Z = 25, Freiraum = 26`.

Summenverfahren

Beispiel 2:

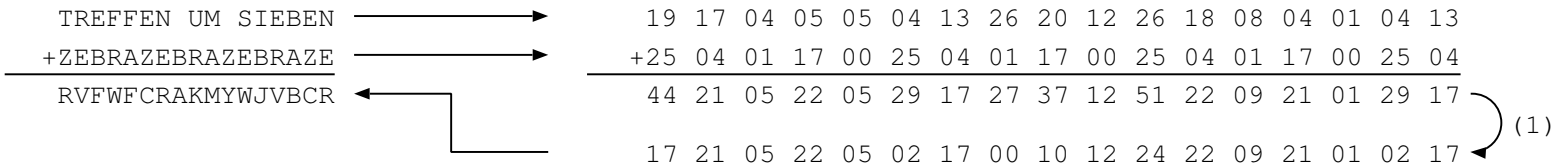
- Verschlüsselung
  Klartext (“`TREFFEN UM SIEBEN`”) + Schlüssel (“ZEBRA”)

```
   TREFFEN UM SIEBEN
 +            ZEBRA
--------------------
   ???????????
```

- Jeder Buchstabe der Nachricht wird einzeln verschlüsselt.
- Jedem Buchstabe wird aufgrund seine Position im Alphabet eine Zahl zugeordnet: `A = 0, B = 1, ... Z = 25, Freiraum = 26`.
- Ist die Nachricht länger als der Schlüssel, wird dieser wiederholt angewendet.

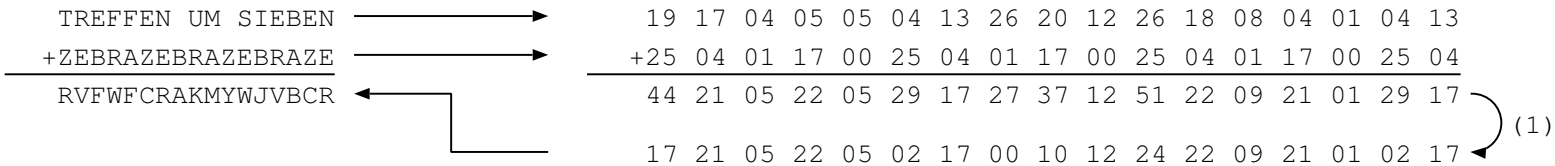
Summenverfahren

- Verschlüsseln

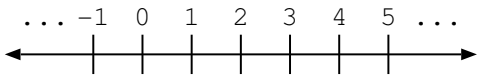


Summenverfahren

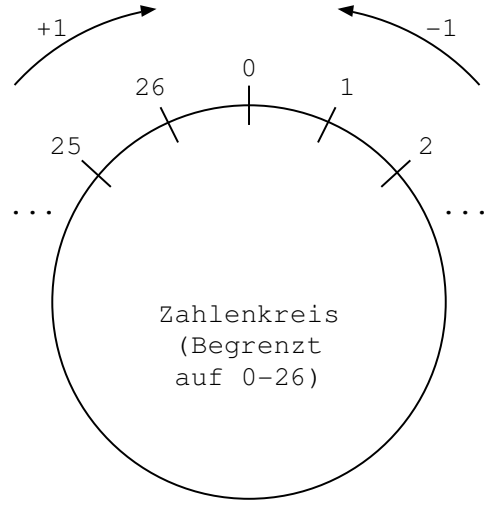
- Verschlüsseln



- Zahlenstrahl

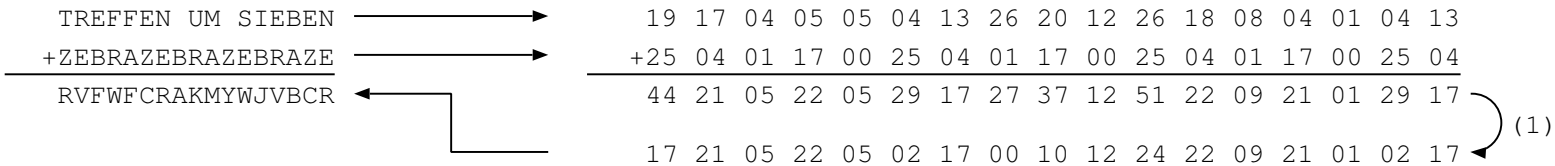


Zahlenstrahl (Unbegrenzt)

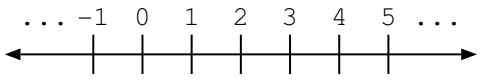


Summenverfahren

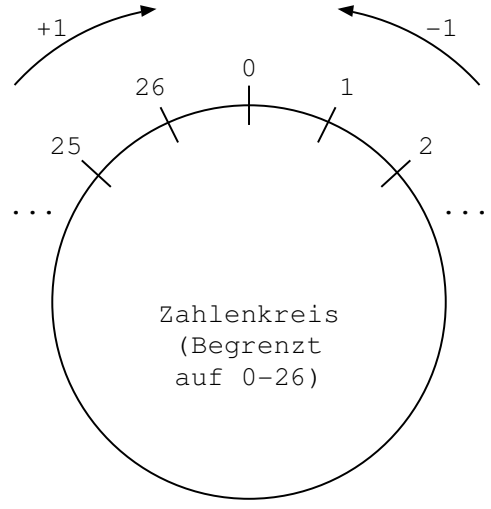
- Verschlüsseln



- Zahlenstrahl



Zahlenstrahl (Unbegrenzt)



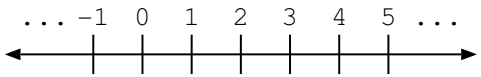
- Eigenschaft der Rechenwerke: Nur dass Ergebnis muss im Zahlenbereich liegen, damit die Rechnung korrekt ist.

Summenverfahren

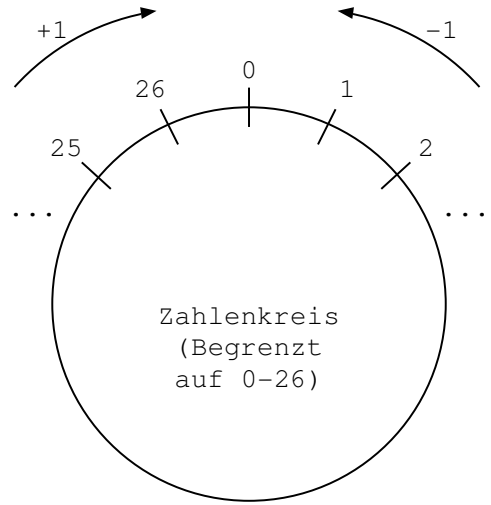
- Verschlüsseln

```
TREFFEN UM SIEBEN  ──→  19 17 04 05 05 04 13 26 20 12 26 18 08 04 01 04 13
+ZEBRAZEBRAZEBRAZE ──→ +25 04 01 17 00 25 04 01 17 00 25 04 01 17 00 25 04
RVFWFCRAKMYWJVBCR  ←──  44 21 05 22 05 29 17 27 37 12 51 22 09 21 01 29 17
                                                                        (1)
                        17 21 05 22 05 02 17 00 10 12 24 22 09 21 01 02 17
```

- Zahlenstrahl



Zahlenstrahl (Unbegrenzt)

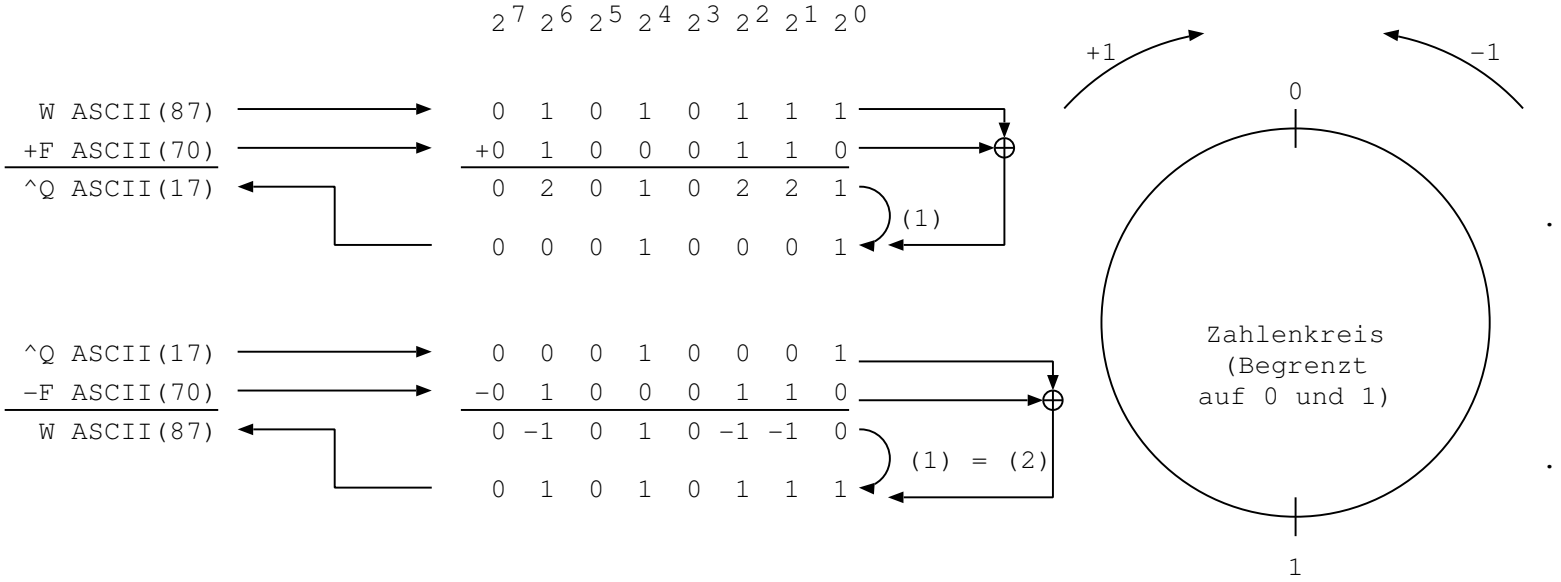


- Eigenschaft der Rechenwerke: Nur dass Ergebnis muss im Zahlenbereich liegen, damit die Rechnung korrekt ist.
- Entschlüsseln

```
RVFWFCRAKMYWJVBCR  ──→  17 21 05 22 05 02 17 00 10 12 24 22 09 21 01 02 17
−ZEBRAZEBRAZEBRAZE ──→ −25 04 01 17 00 25 04 01 17 00 25 04 01 17 00 25 04
TREFFEN UM SIEBEN  ←──  −08 17 04 05 05−23 13−01−07 12−01 18 08 04 01−23 13
                                                                        (2)
                        19 17 04 05 05 04 13 26 20 12 26 18 08 04 01 04 13
```

Summenverfahren

- Summenverfahren auf Bit-Ebene



XOR = $\oplus$

a	b	x
0	0	0
0	1	1
1	0	1
1	1	0

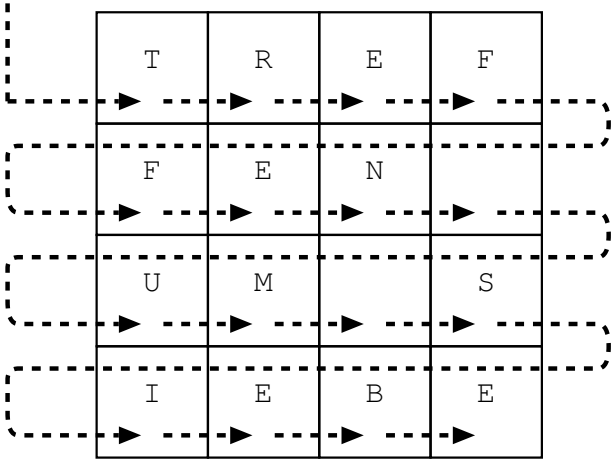
Permutationsverfahren

Permutationsverfahren

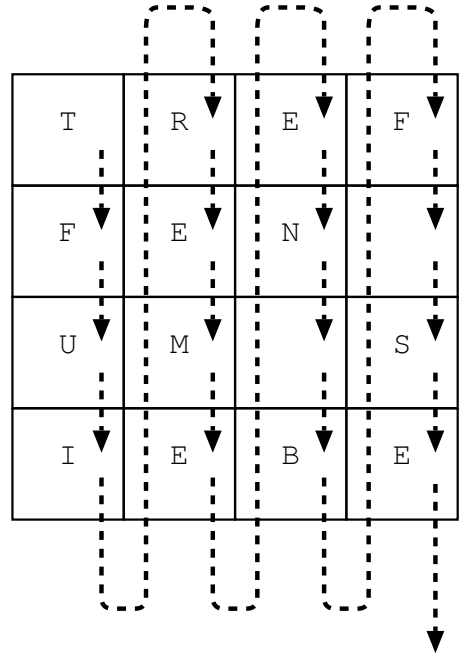
- Verschlüsseln

Zeichen werden in dieser Reiheinfolge in die Matrix geschrieben...

TREFFEN UM SIEBE



...und in dieser wieder ausgelesen



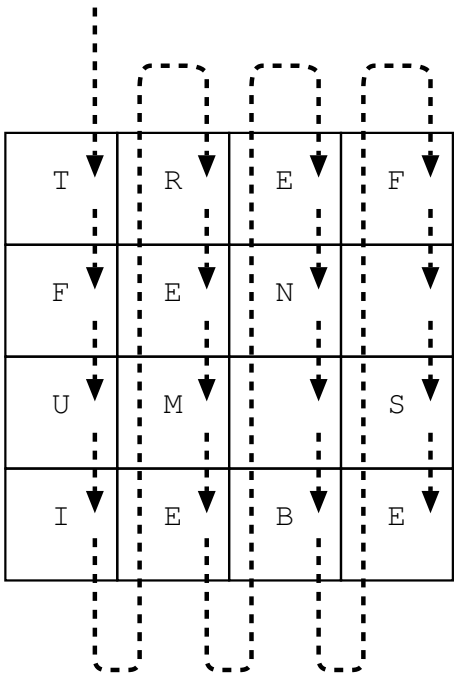
TFUIREMEEN BF SE

Permutationsverfahren

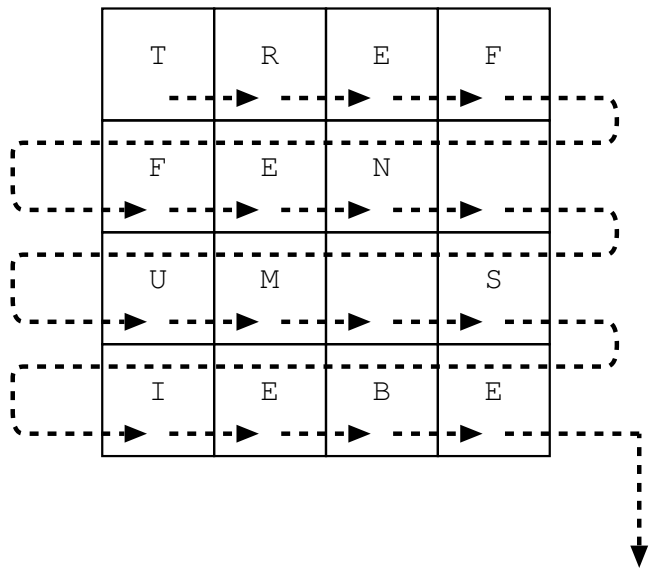
- Entschlüsseln

Zeichen werden in dieser Reiheinfolge in die Matrix geschrieben...

TFUIREMEEN BF SE



...und in dieser wieder ausgelesen



TREFFEN UM SIEBE

Permutationsverfahren

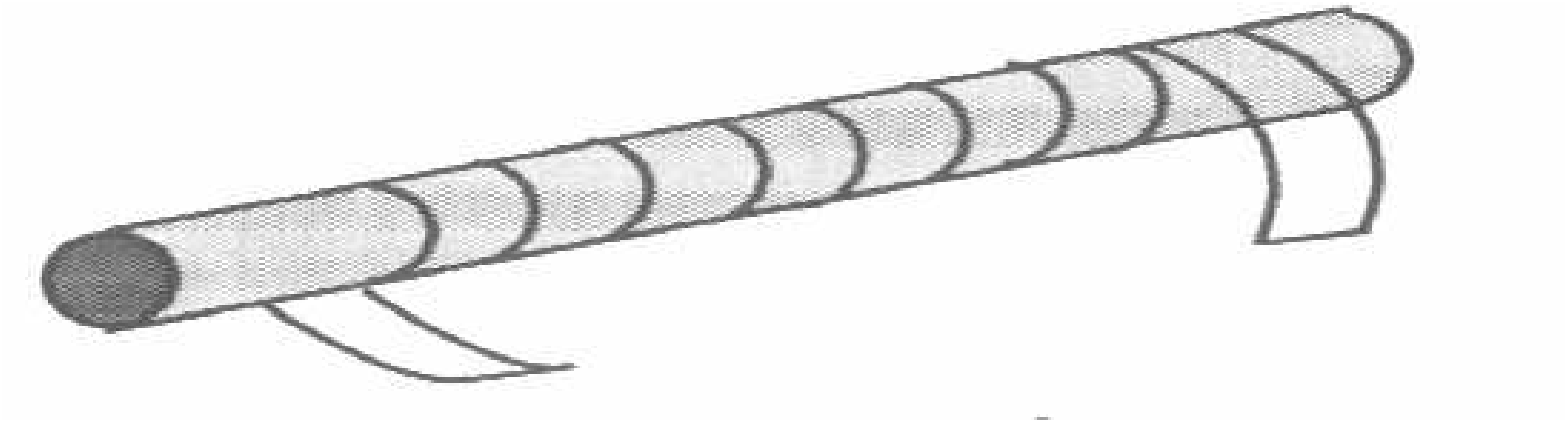
- Beispiel aus der Geschichte

Permutationsverfahren

- Beispiel aus der Geschichte
- (Vorführung!)

Permutationsverfahren

- Beispiel aus der Geschichte
- (Vorführung!)
- Die Regierung von Sparta zur hat bereits 500 v. Chr. mit Hilfe eines Holzstabes (Skytale) verschlüsselt.



Substitutionsverfahren

- Ersetzen einzelner Symbole durch andere.

Substitutionsverfahren

- Ersetzen einzelner Symbole durch andere.
- Historisches Beispiel: Cäsar-Chiffre

  A B C D E F G H I L M N O P Q R S T U V X

  U V X A B C D E F G H I L M N O P Q R S T

  (Das Lateinischen Alphabet verfügt nicht über die Buchstaben J, K, W, Y und Z.)

Substitutionsverfahren

- Ersetzen einzelner Symbole durch andere.
- Historisches Beispiel: Cäsar-Chiffre

  `A B C D E F G H I L M N O P Q R S T U V X`

  `U V X A B C D E F G H I L M N O P Q R S T`

  (Das Lateinischen Alphabet verfügt nicht über die Buchstaben J, K, W, Y und Z.)
- `TREFFEN UM SIEBEN` ← Klartext

  `QOBCCBI RH PFBVBI` ← Chiffre

Advanced Encryption Standard (AES)

- öffentliches Auswahlverfahren der U.S. National Institute of Standards and Technology (NIST.gov)

Advanced Encryption Standard (AES)

- öffentliches Auswahlverfahren der U.S. National Institute of Standards and Technology (NIST.gov)
- Nachfolger von Data Encryption Standards (DES), da DES angreifbar geworden ist

Advanced Encryption Standard (AES)

- öffentliches Auswahlverfahren der U.S. National Institute of Standards and Technology (NIST.gov)
- Nachfolger von Data Encryption Standards (DES), da DES angreifbar geworden ist
- Öffentlich, d.h., jeder konnte Kandidaten oder Ergebnisse einreichen

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit
  - Sicherheit ist nicht beweisbar

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit
  - Sicherheit ist nicht beweisbar
  - Brute-Force-Attack als Vergleich

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit
  - Sicherheit ist nicht beweisbar
  - Brute-Force-Attack als Vergleich
- Kosten

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit
  - Sicherheit ist nicht beweisbar
  - Brute-Force-Attack als Vergleich
- Kosten
  - Der Algorithmus sollte vollkommen (kosten-) frei verwendbar sein. (Frei von Patent- oder anderen Ansprüchen der Entwickler)

Anforderungen an AES

- wichtigste Auswahlkriterium: Sicherheit
  - Sicherheit ist nicht beweisbar
  - Brute-Force-Attack als Vergleich
- Kosten
  - Der Algorithmus sollte vollkommen (kosten-) frei verwendbar sein. (Frei von Patent- oder anderen Ansprüchen der Entwickler)
  - Die Implementierung und das Ver- bzw. Entschlüsseln sollte so billig wie möglich sein.

Anforderungen an AES II

- Eigenschaften der Implementierung

Anforderungen an AES II

- Eigenschaften der Implementierung
  - Der Algorithmus sollte sowohl in Hardware als auch in Software mit geringsten möglichen Mitteln umgesetzt werden können, also Platformunabhängig gestaltet sein.

Anforderungen an AES II

- Eigenschaften der Implementierung
  - Der Algorithmus sollte sowohl in Hardware als auch in Software mit geringsten möglichen Mitteln umgesetzt werden können, also Platformunabhängig gestaltet sein.
  - Die Schlüssellänge sollte zwischen 128, 192 und 256 Bit gewählt werden können.

Anforderungen an AES II

- Eigenschaften der Implementierung
  - Der Algorithmus sollte sowohl in Hardware als auch in Software mit geringsten möglichen Mitteln umgesetzt werden können, also Platformunabhängig gestaltet sein.
  - Die Schlüssellänge sollte zwischen 128, 192 und 256 Bit gewählt werden können.
  - Die Blockgröße sollte 128 Bit betragen.

Anforderungen an AES II

- Eigenschaften der Implementierung
  - Der Algorithmus sollte sowohl in Hardware als auch in Software mit geringsten möglichen Mitteln umgesetzt werden können, also Platformunabhängig gestaltet sein.
  - Die Schlüssellänge sollte zwischen 128, 192 und 256 Bit gewählt werden können.
  - Die Blockgröße sollte 128 Bit betragen.
  - Das Verfahren sollte einfach und verständlich sein.

Das Auswahlverfahren

- Alle Kandidaten mussten bis zum 15. Mai 1998 eingereicht werden.

Das Auswahlverfahren

- Alle Kandidaten mussten bis zum 15. Mai 1998 eingereicht werden.
- August begann dann die erste Runde, welche im März 1999 mit der Auswahl der Finalisten endete.

Das Auswahlverfahren

- Alle Kandidaten mussten bis zum 15. Mai 1998 eingereicht werden.
- August begann dann die erste Runde, welche im März 1999 mit der Auswahl der Finalisten endete.
- Am 2. Oktober 2000, nach einer weiteren Auswahl-Runde, wurde schließlich der Gewinner gekürt: Rijndael.

Das Auswahlverfahren

- Alle Kandidaten mussten bis zum 15. Mai 1998 eingereicht werden.
- August begann dann die erste Runde, welche im März 1999 mit der Auswahl der Finalisten endete.
- Am 2. Oktober 2000, nach einer weiteren Auswahl-Runde, wurde schließlich der Gewinner gekürt: Rijndael.
- Viele Beobachter waren darüber erstaunt, dass ein Algorithmus gewonnen hat, der weder aus den USA stammt noch von einer Firma aus dem Verschlüsselungsgeschäft entwickelt wurde. Von vielen Seiten wurde aus diesem Grund auf ein objektives Auswahlverfahren geschlossen, welches am Schluss den besten Algorithmus zum Gewinner kürte.

Anwärter auf AES

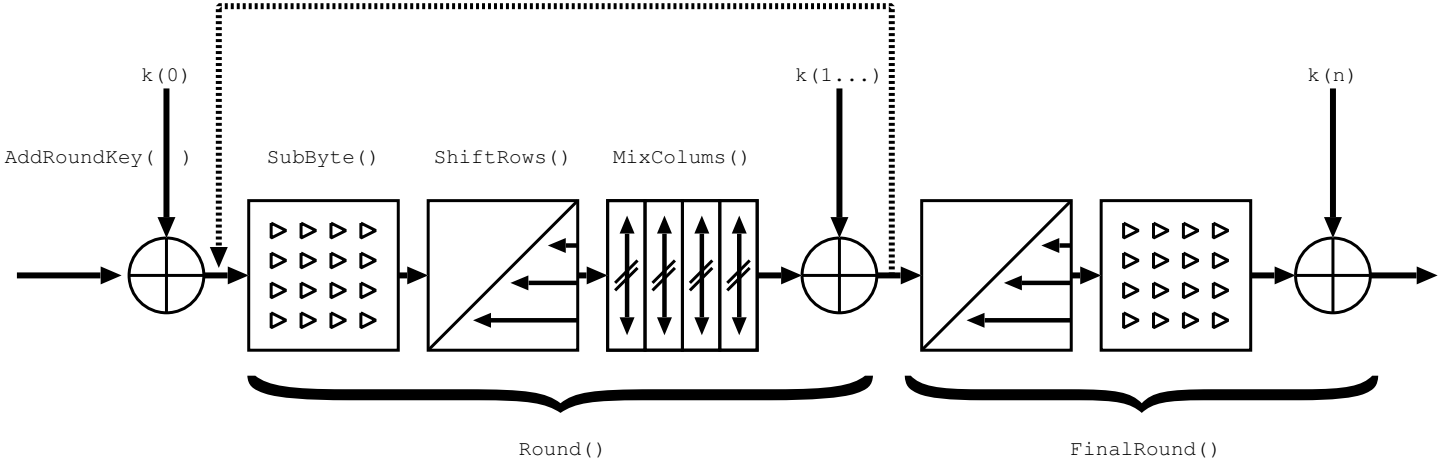
Name	eingereicht von		aus. Runde
CAST-256	Entrust (CA)	Company	1^{1}
Crypton	Future Systems (KR)	Company	1^{2}
DEAL	Outerbride, Knudsen (USA-DK)	Forscher	1^{sp}
DFC	ENS-CNRS (FR)	Forscher	1^{3}
E2	NTT (JP)	Company	1^{4}
Frog	TecApro (CR)	Company	1^{s}
HPC	Schroeppel (USA)	Forscher	1^{s}
LOKI97	Brown et al. (AU)	Forscher	1^{s}
Magenta	Deutsche Telekom (DE)	Company	1^{sp}
Mars	IBM (USA)	Company	2
RC6	RSA (USA)	Company	2
Rijndael	**Daemen and Rijmen (BE)**	**Forscher**	**WINNER**
SAFER+	Cylink (USA)	Company	1^{p}
Serpent	Anderson, Biham, Kundsen (UK-IL-DK)	Forscher	2
Twofish	Counterpane (USA)	Company	2

1) Ausw. 5 im März 1999 2) 2. Oktober 2000 ^{s}) Sicherheits ^{p}) Performace

^{1}) $\approx$ Serpent, impl. aufw. ^{2}) $\approx$ Rijndael, Twofish, ^{s} ^{3}) ^{s}, $\neq$ 64-Bit ^{4}) $\approx$ Rijndael, Twofish, impl. aufw.

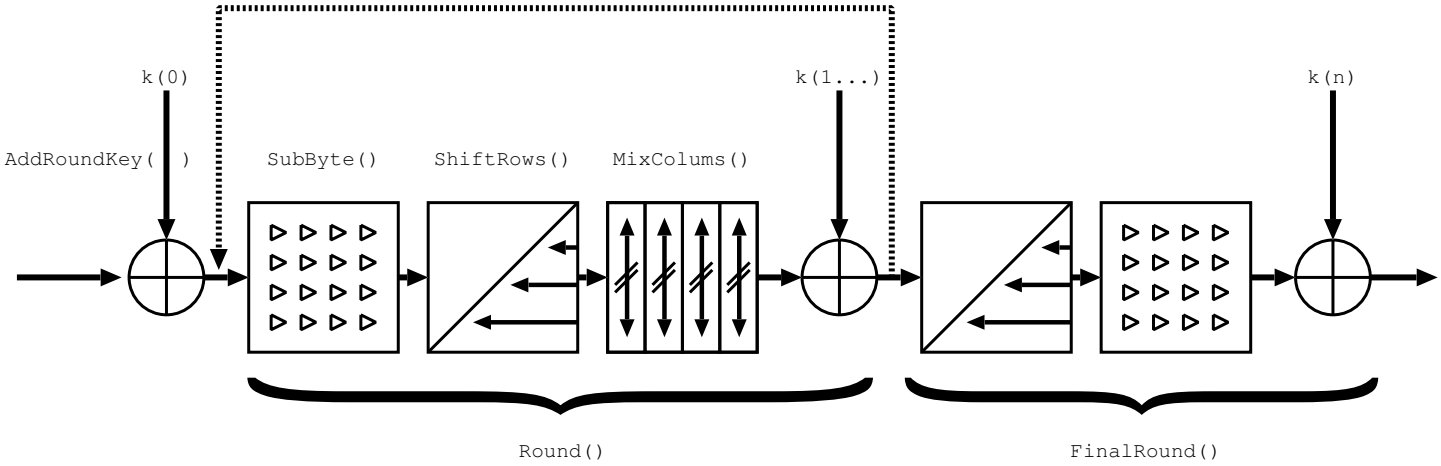
Aufbau von AES II

- Übersicht



Aufbau von AES II

- Übersicht



- Anzahl der Runden

Länge des Datenblockes	Länge des Schlüssels				
	128	160	**192**	224	**256**
128	**10**	11	**12**	13	**14**
160	11	11	12	13	14
192	12	12	12	13	14
224	13	13	13	13	14
256	14	14	14	14	14

Betrachtungsweise der Daten

- AES betrachtet die Blöcke in der folgenden Weise: Es werden jeweils vier Bytes von oben nach unten geschrieben. Beim fünften Byte wird nach rechts weitergegangen und wieder oben angefangen:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

128 Bit-Block (16-Byte)

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

256 Bit-Block (32 Byte)

Verwendung des Schlüssels

- Schlüssellänge: 128, 192 und 256 Bit (16, 24 und 32 Byte)

Verwendung des Schlüssels

- Schlüssellänge: 128, 192 und 256 Bit (16, 24 und 32 Byte)
- `KeyExpansion()` erweitert den Schlüssel auf (Runden + 1) * Blockgröße

Verwendung des Schlüssels

- Schlüssellänge: 128, 192 und 256 Bit (16, 24 und 32 Byte)
- `KeyExpansion()` erweitert den Schlüssel auf (Runden + 1) * Blockgröße
- und erzeugt recht unvorhersagbare aber reproduzierbare Werte aus dem Schlüssel:

```
00000000,00000000,00000000,00000000
62636363,62636363,62636363,62636363
9b9898c9,f9fbfbaa,9b9898c9,f9fbfbaa
90973450,696ccffa,f2f45733,0b0fac99
ee06da7b,876a1581,759e42b2,7e91ee2b
7f2e2b88,f8443e09,8dda7cbb,f34b9290
ec614b85,1425758c,99ff0937,6ab49ba7
21751787,3550620b,acaf6b3c,c61bf09b
0ef90333,3ba96138,97060a04,511dfa9f
b1d4d8e2,8a7db9da,1d7bb3de,4c664941

b4ef5bcb,3e92e211,23e951cf,6f8f188e
```

Verwendung des Schlüssels

- Schlüssellänge: 128, 192 und 256 Bit (16, 24 und 32 Byte)
- `KeyExpansion()` erweitert den Schlüssel auf (Runden + 1) * Blockgröße
- und erzeugt recht unvorhersagbare aber reproduzierbare Werte aus dem Schlüssel:

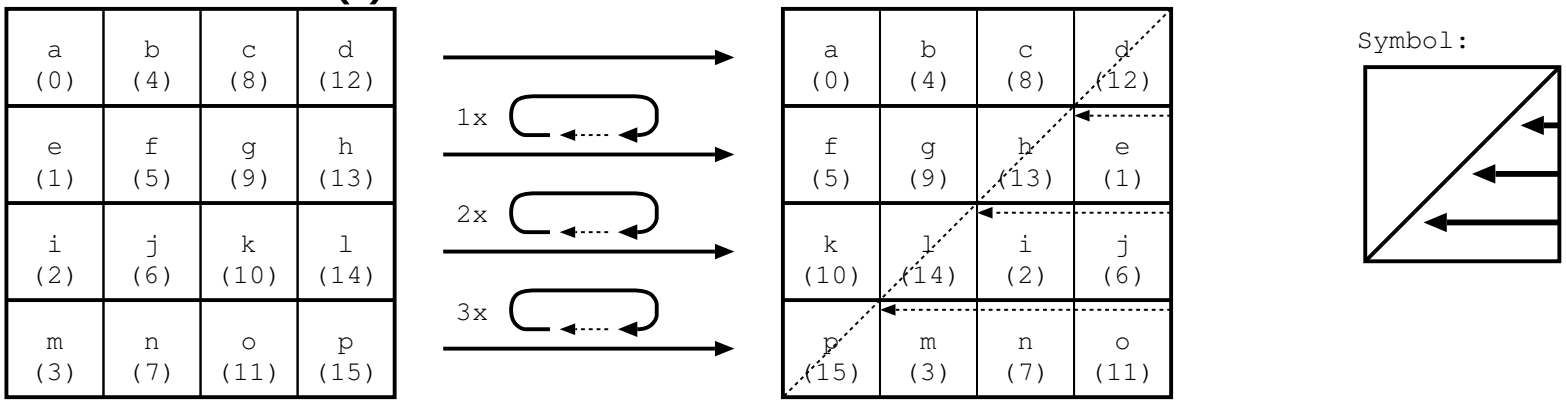
```
00000000,00000000,00000000,00000000
62636363,62636363,62636363,62636363
9b9898c9,f9fbfbaa,9b9898c9,f9fbfbaa
90973450,696ccffa,f2f45733,0b0fac99
ee06da7b,876a1581,759e42b2,7e91ee2b
7f2e2b88,f8443e09,8dda7cbb,f34b9290
ec614b85,1425758c,99ff0937,6ab49ba7
21751787,3550620b,acaf6b3c,c61bf09b
0ef90333,3ba96138,97060a04,511dfa9f
b1d4d8e2,8a7db9da,1d7bb3de,4c664941

b4ef5bcb,3e92e211,23e951cf,6f8f188e
```

- `AddRoundKey()` XORt die erweiterten Schlüssel mit dem Datenstrom

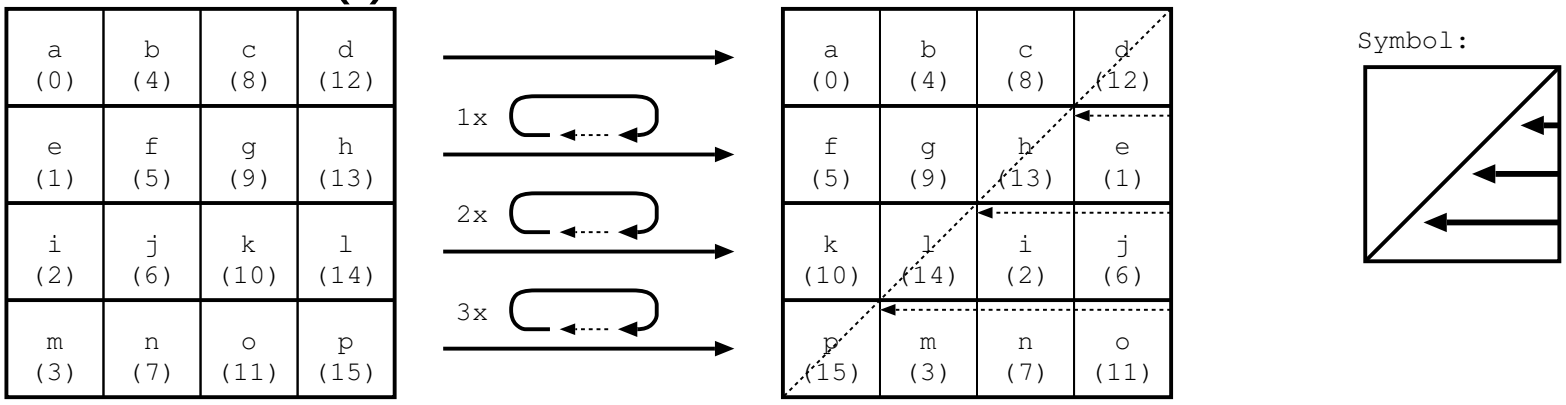
Permutationsschritt

- `ShiftRows()` verschiebt die Zeilen in einem Datenblock:



Permutationsschritt

- `ShiftRows()` verschiebt die Zeilen in einem Datenblock:

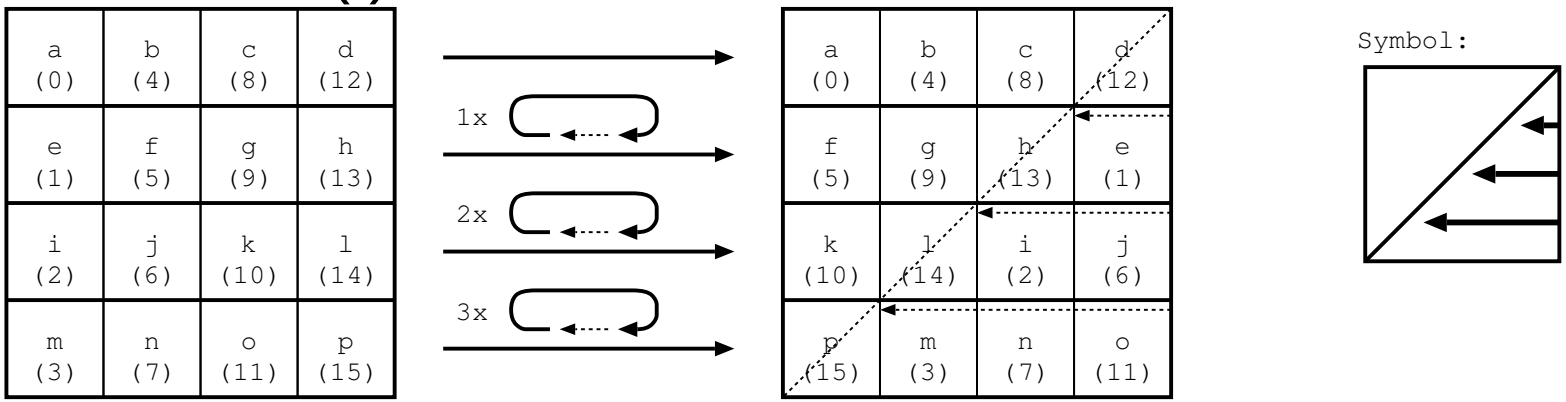


- Die Verschiebungsweite ist durch folgende Tabelle definiert:

Zeile	Länge des Datenblockes **128**	160	192	224	256
1	**0**	0	0	0	0
2	**1**	1	1	1	1
3	**2**	2	2	2	3
4	**3**	3	3	4	4

Permutationsschritt

- `ShiftRows()` verschiebt die Zeilen in einem Datenblock:



- Die Verschiebungsweite ist durch folgende Tabelle definiert:

Zeile	Länge des Datenblockes **128**	160	192	224	256
1	**0**	0	0	0	0
2	**1**	1	1	1	1
3	**2**	2	2	2	3
4	**3**	3	3	4	4

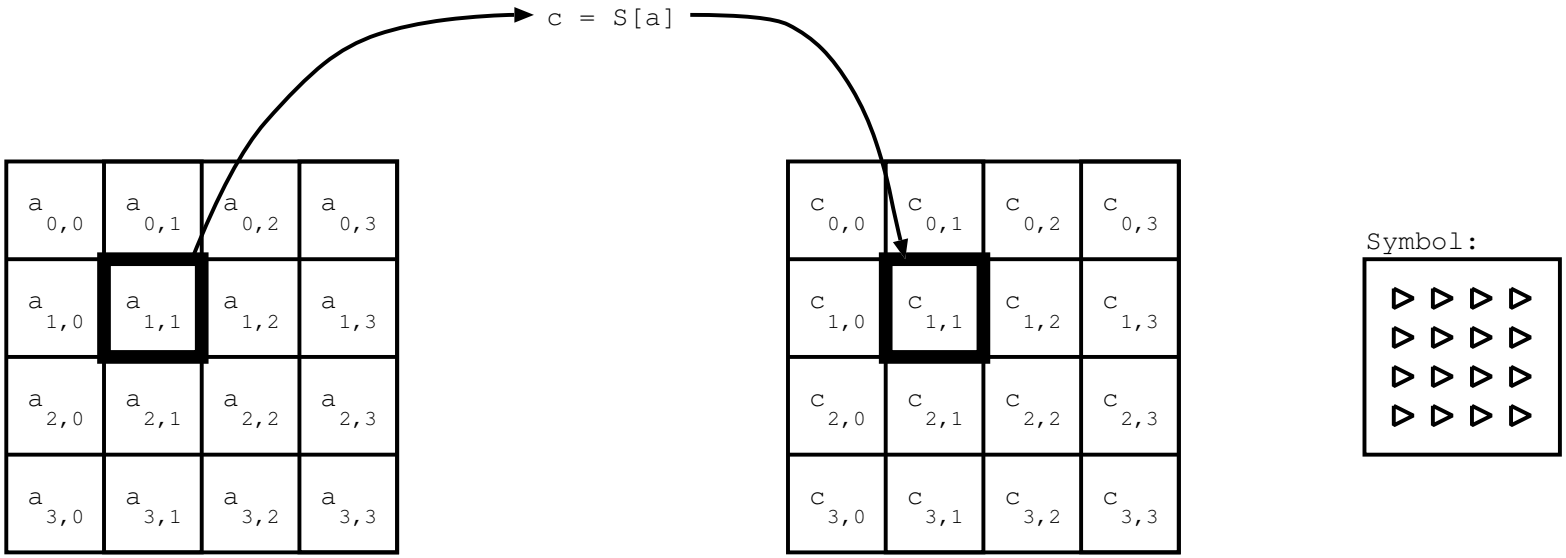
- Die inverse Funktion (`InvShiftRows()`) schiebt die Zeilen um die gleiche Anzahl von Bytes in die entgegengesetzte Richtung.

Substitutionsschritt 1: `SubBytes()`

- Substitution ist fest vorgeschrieben, nicht durch Schlüssel beeinflusst.

Substitutionsschritt 1: `SubBytes()`

- Substitution ist fest vorgeschrieben, nicht durch Schlüssel beeinflusst.
- `SubBytes()` verarbeitet jedes Byte im Datenblock einzeln



Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:

Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:
  - Die Bits (a_7 bis a_0) in dem Byte (a) werden als Polynom betrachtet:
    $a = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 \neq 0 \Rightarrow$
    $f(x) = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^+ a_0$

Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:
  - Die Bits (a_7 bis a_0) in dem Byte (a) werden als Polynom betrachtet:
    $a = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 \neq 0 \Rightarrow$
    $f(x) = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^+ a_0$
  - Die Inverse dieses Polynoms wird in $GF(2^8) \Leftrightarrow GF(256)$ gebildet.
    $g(x) = f(x)^{-1}$

Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:
  - Die Bits (a_7 bis a_0) in dem Byte (a) werden als Polynom betrachtet:
    $a = a_7a_6a_5a_4a_3a_2a_1a_0 \neq 0 \Rightarrow$
    $f(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^+a_0$
  - Die Inverse dieses Polynoms wird in $GF(2^8) \Leftrightarrow GF(256)$ gebildet.
    $g(x) = f(x)^{-1}$
  - Das Polynom wird jetzt wieder als Byte betrachtet:
    $g(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^+b_0$
    $\Rightarrow b = b_7b_6b_5b_4b_3b_2b_1b_0$

Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:
  - Die Bits (a_7 bis a_0) in dem Byte (a) werden als Polynom betrachtet:
    $a = a_7a_6a_5a_4a_3a_2a_1a_0 \neq 0 \Rightarrow$
    $f(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^+a_0$
  - Die Inverse dieses Polynoms wird in $GF(2^8) \Leftrightarrow GF(256)$ gebildet.
    $g(x) = f(x)^{-1}$
  - Das Polynom wird jetzt wieder als Byte betrachtet:
    $g(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^+b_0$
    $\Rightarrow b = b_7b_6b_5b_4b_3b_2b_1b_0$
  - Für den nicht definierten Fall, dass a gleich 0 ist, wird angenommen, dass b ebenfalls 0 ist.

Substitutionsverfahren: `SubBytes()`

- Dies Substitution ist durch den folgenden mathematischen Zusammenhang bestimmt:
  - Die Bits (a_7 bis a_0) in dem Byte (a) werden als Polynom betrachtet:
    $a = a_7a_6a_5a_4a_3a_2a_1a_0 \neq 0 \Rightarrow$
    $f(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^+a_0$
  - Die Inverse dieses Polynoms wird in $GF(2^8) \Leftrightarrow GF(256)$ gebildet.
    $g(x) = f(x)^{-1}$
  - Das Polynom wird jetzt wieder als Byte betrachtet:
    $g(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^+b_0$
    $\Rightarrow b = b_7b_6b_5b_4b_3b_2b_1b_0$
  - Für den nicht definierten Fall, dass a gleich 0 ist, wird angenommen, dass b ebenfalls 0 ist.
  - ...

Substitutionsschritt 1: `SubBytes()`

- ...

Substitutionsschritt 1: `SubBytes()`

- ...
  - Das so gewonnene Bytes wird als Bit-Vektor betrachtet:

$$c = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Substitutionsschritt 1: `SubBytes()`

- ...
  - Das so gewonnene Bytes wird als Bit-Vektor betrachtet:

$$c = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

  - Die `InvSubBytes()` macht den so eben beschriebenen folgendermaßen Vorgang rückgängig:

$$b = \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Substitutionsschritt 1: `SubBytes()`

- ...
  - Das so gewonnene Bytes wird als Bit-Vektor betrachtet:

$$c = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

  - Die `InvSubBytes()` macht den so eben beschriebenen folgendermaßen Vorgang rückgängig:

$$b = \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

  - Das Ergebnis wird als Polynom betrachtet und invertiert.

Substitutionsschritt 1: `SubBytes()`

- ...
  - Das so gewonnene Bytes wird als Bit-Vektor betrachtet:

$$c = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

  - Die `InvSubBytes()` macht den so eben beschriebenen folgendermaßen Vorgang rückgängig:

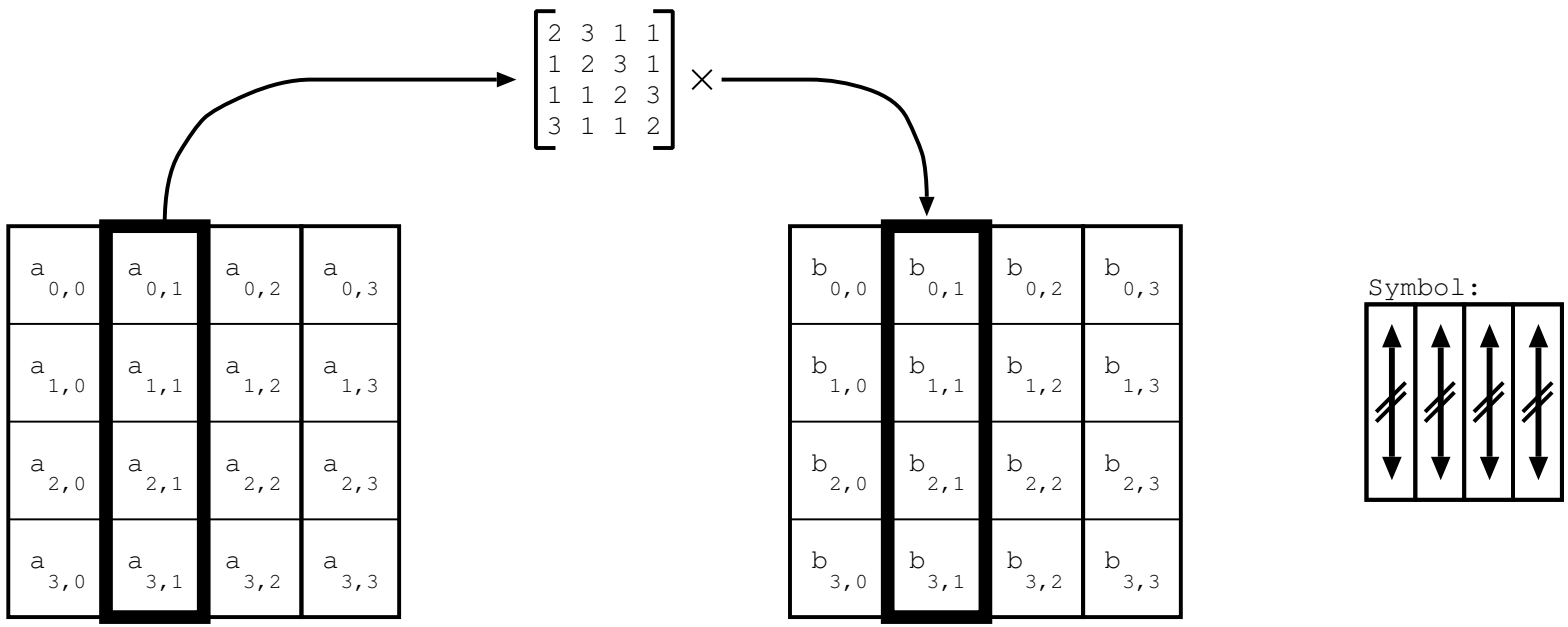
$$b = \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

  - Das Ergebnis wird als Polynom betrachtet und invertiert.
- Behandlung als Array: `output = S[input];`

Substitutionsschritt 2: `MixColums()`

- `MixColums()` wird jeweils auf eine 4-Bytes-Spalte angewendet und führt die folgende Substitution aus:

$$b = \begin{bmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{bmatrix}$$



Substitutionsschritt 2: `MixColums()`

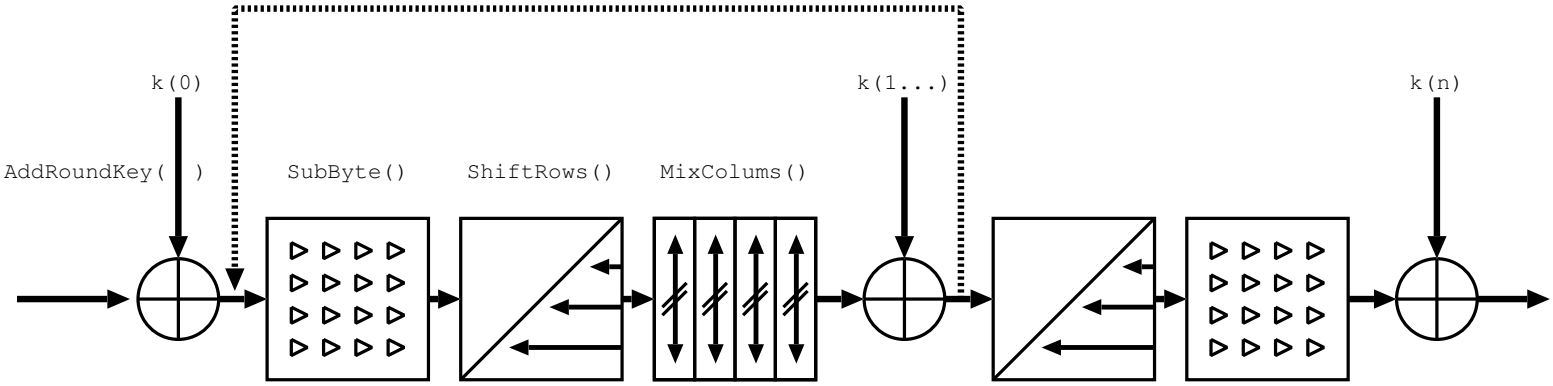
- Um diese Umwandlung rückgängig zu machen, wurde die Funktion `InvMixColums()` entwickelt, die folgende mathematische Operation ausführt:

$$a = \begin{bmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{bmatrix}$$

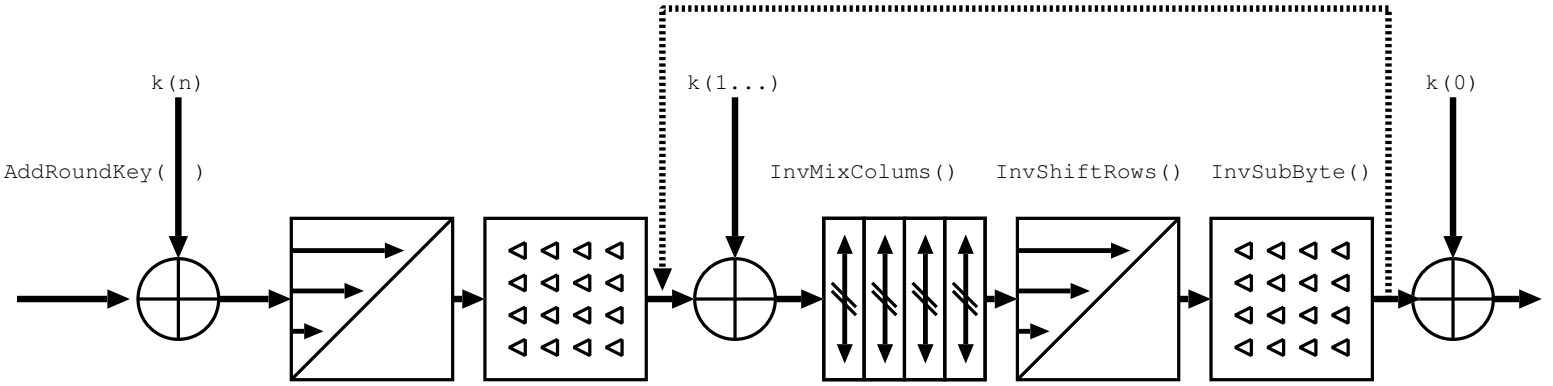
Entschlüsselung

- Anwendung der `Inv*()`-Funktionen:

Verschlüsselung



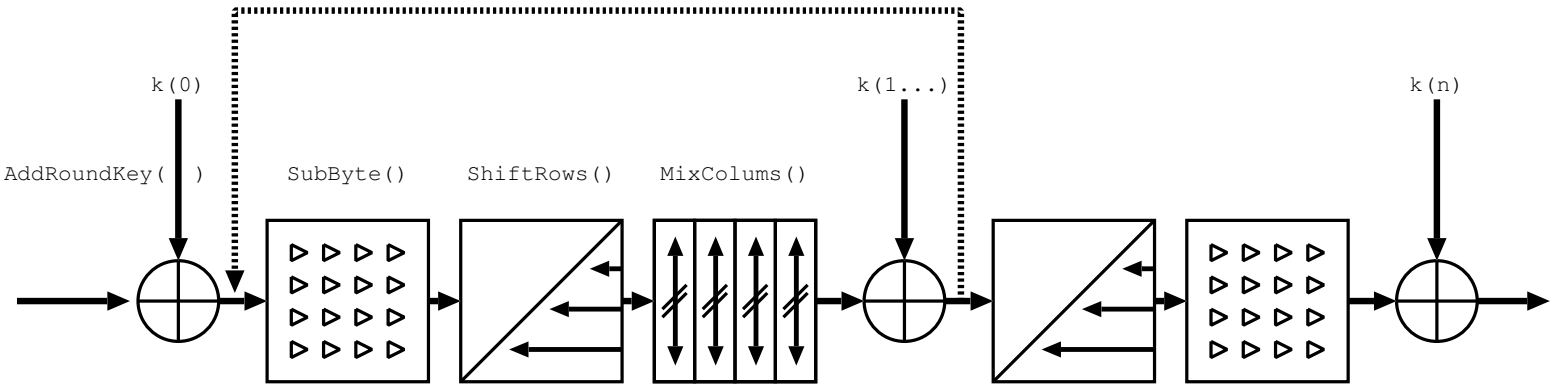
Entschlüsselung



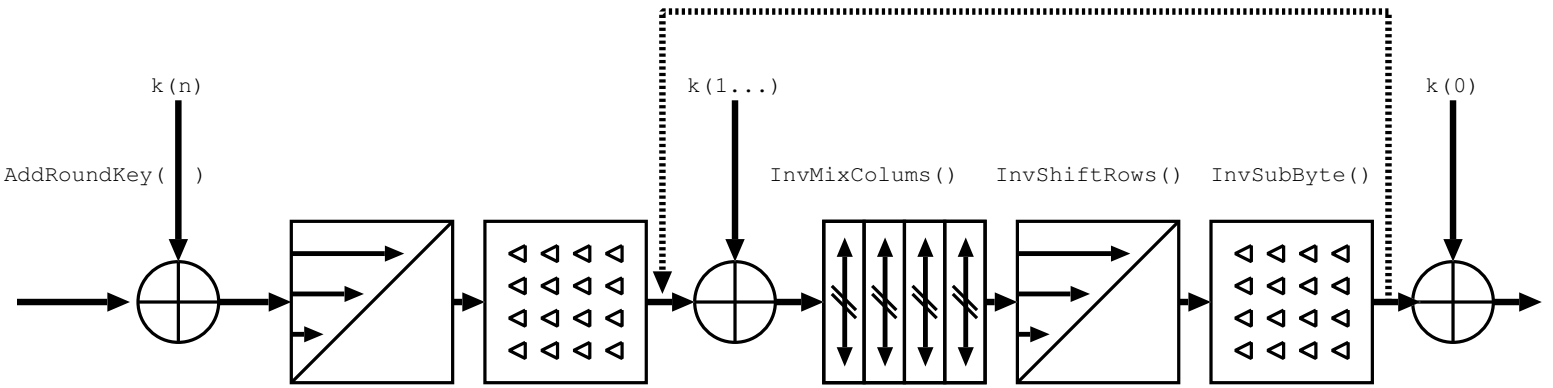
Entschlüsselung

- Anwendung der `Inv*()`-Funktionen:

Verschlüsselung



Entschlüsselung

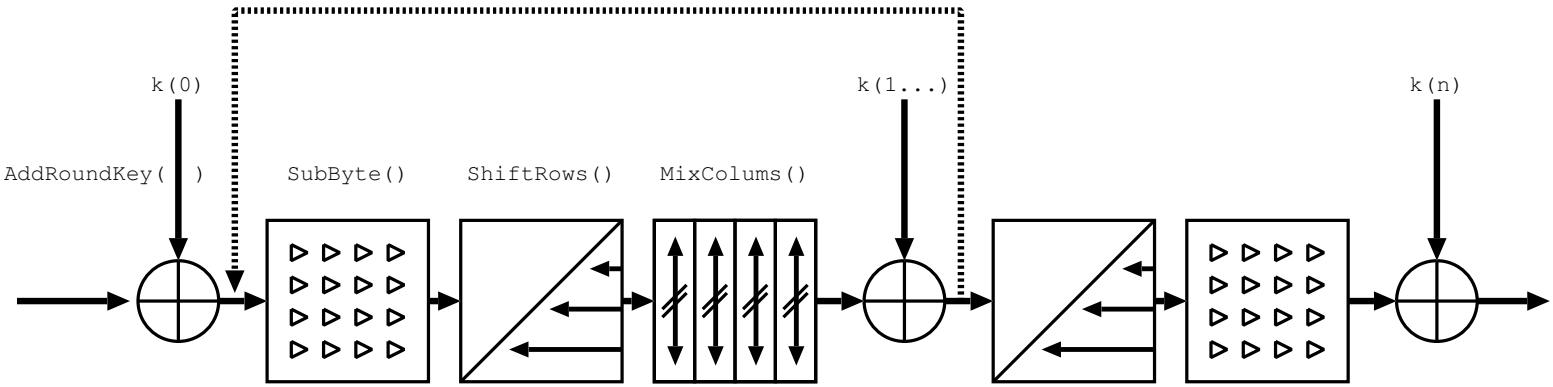


- Wer erkennt die vermeintliche Unstimmigkeit?

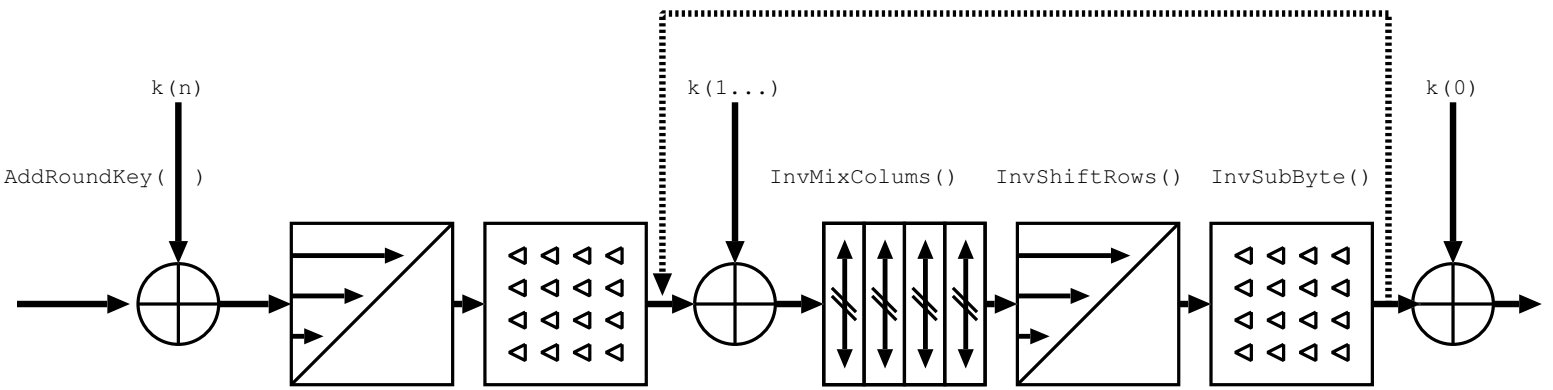
Entschlüsselung

- Anwendung der `Inv*()`-Funktionen:

`Verschlüsselung`



`Entschlüsselung`



- Wer erkennt die vermeintliche Unstimmigkeit?
- `SubBytes()` und `ShiftRows()` Vertauscht? Kein Problem, da ersteres auf jedes Byte einzeln angewendet wird. Dabei ist es egal, ob die Bytes verschoben wurden.

ECB – Electronic Code Book

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln

ECB – Electronic Code Book

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln
- Betrachtung als Substitution: $2^{128} \approx 3.4 * 10^{38}$ verschiedene Symbole.

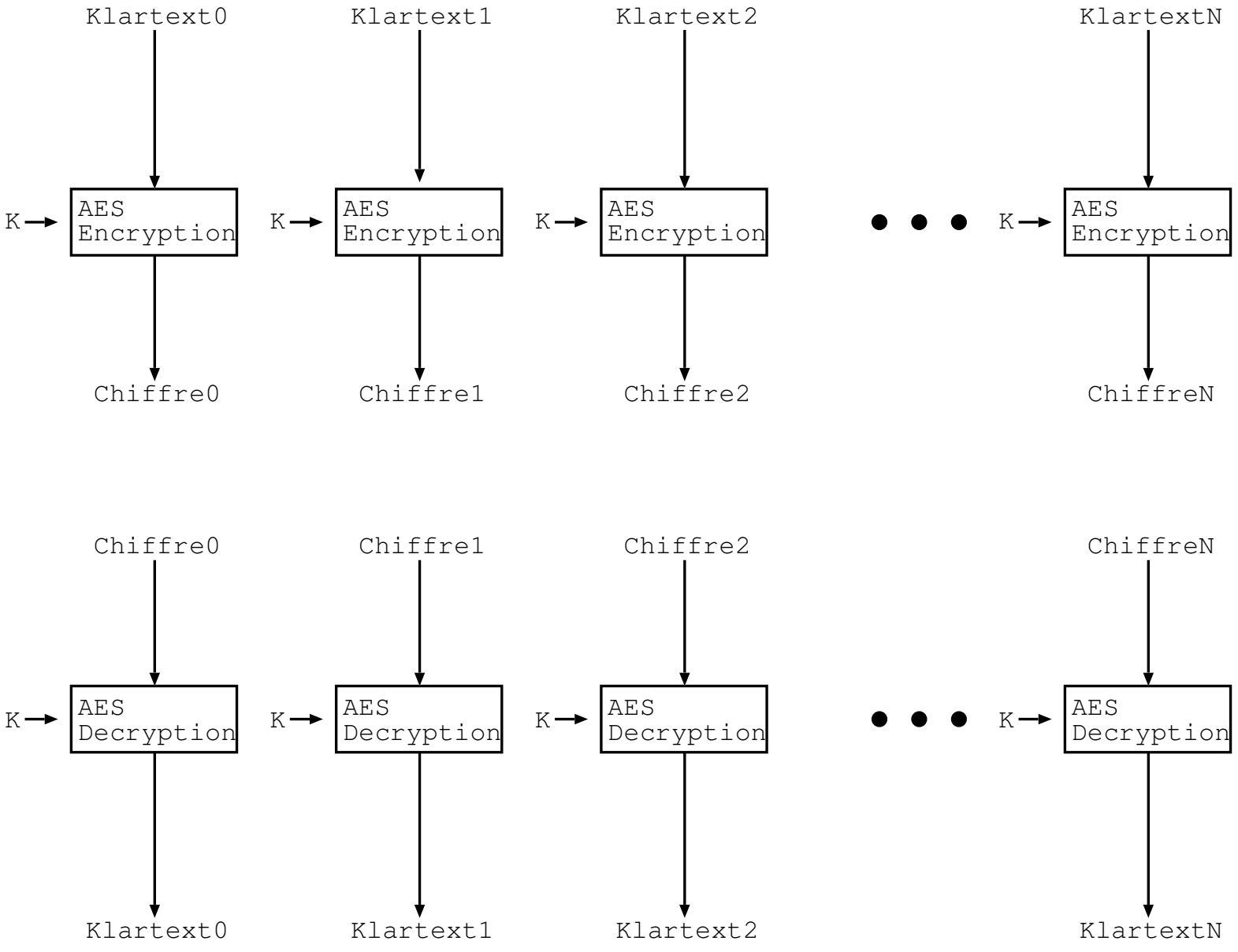
ECB – Electronic Code Book

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln
- Betrachtung als Substitution: $2^{128} \approx 3.4 * 10^{38}$ verschiedene Symbole.
- Nachricht muss vielfaches der Blockgröße lang sein.

ECB – Electronic Code Book

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln
- Betrachtung als Substitution: $2^{128} \approx 3.4 * 10^{38}$ verschiedene Symbole.
- Nachricht muss vielfaches der Blockgröße lang sein.
- Problem des “elektronischen Codebooks”

ECB – Electronic Code Book



CBC – Cipher Block Chaining

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln. Ergebnis vom vorherigen Block wird mit einbezogen

CBC – Cipher Block Chaining

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln. Ergebnis vom vorherigen Block wird mit einbezogen
- Jeder Chiffre-Block ist von Passwort, dem Klartext-Block und allen vorherigen Blöcken abhängig.

CBC – Cipher Block Chaining

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln. Ergebnis vom vorherigen Block wird mit einbezogen
- Jeder Chiffre-Block ist von Passwort, dem Klartext-Block und allen vorherigen Blöcken abhängig.
- Erster Block ist von IV (Initial Vector = Anfangswert), Passwort, und dem Klartext-Block abhängig.

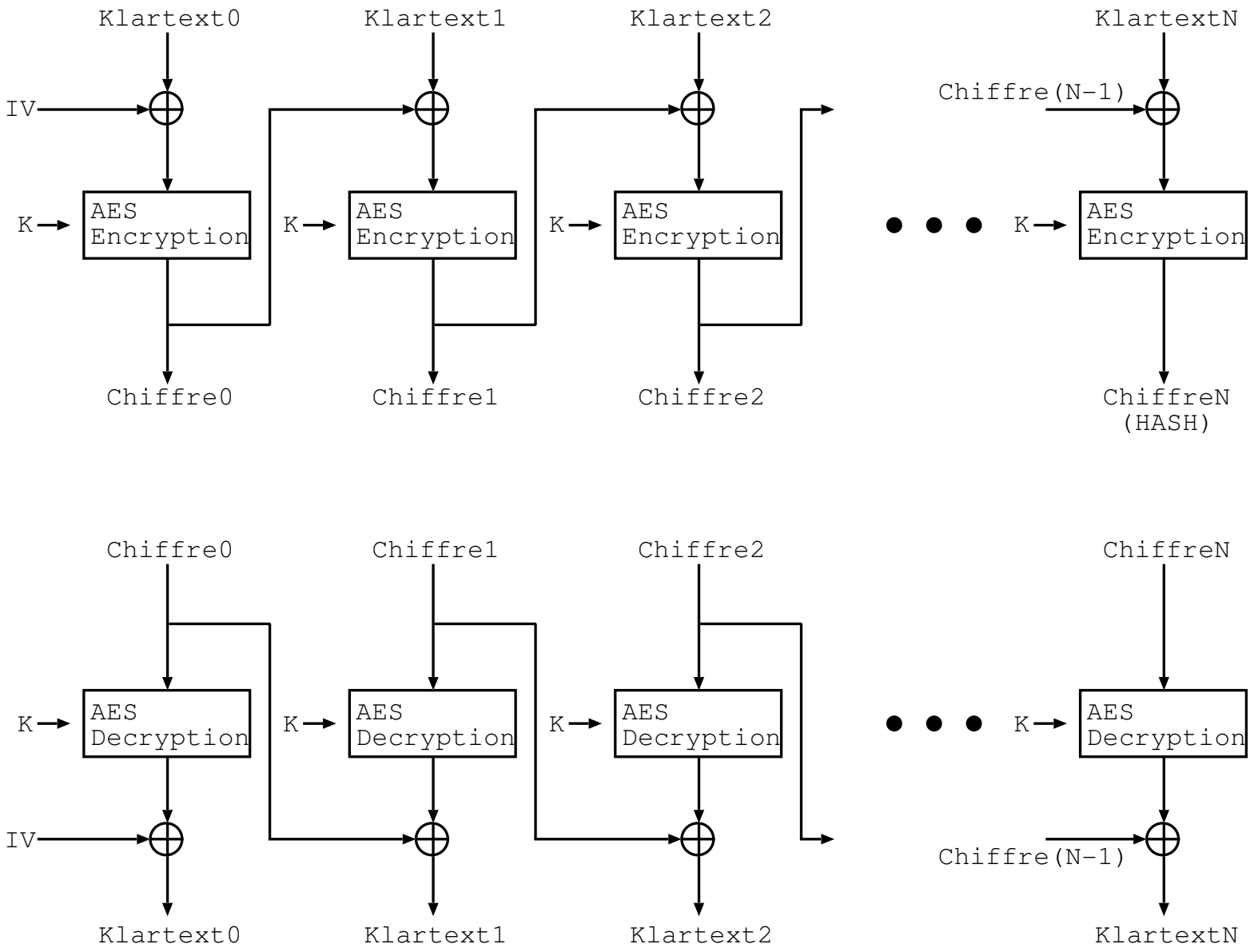
CBC – Cipher Block Chaining

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln. Ergebnis vom vorherigen Block wird mit einbezogen
- Jeder Chiffre-Block ist von Passwort, dem Klartext-Block und allen vorherigen Blöcken abhängig.
- Erster Block ist von IV (Initial Vector = Anfangswert), Passwort, und dem Klartext-Block abhängig.
- Da der letzte Block von allen Blöcken abhängig ist, kann dieser als HASH-Wert angesehen werden.

CBC – Cipher Block Chaining

- Datenstrom in Blöcke aufgeteilt. Anwendung des Algorithmus auf jeden Block einzeln. Ergebnis vom vorherigen Block wird mit einbezogen
- Jeder Chiffre-Block ist von Passwort, dem Klartext-Block und allen vorherigen Blöcken abhängig.
- Erster Block ist von IV (Initial Vector = Anfangswert), Passwort, und dem Klartext-Block abhängig.
- Da der letzte Block von allen Blöcken abhängig ist, kann dieser als HASH-Wert angesehen werden.
- Alle vorherigen Blöcke müssen vorhanden sein, um die Nachricht zu entschlüsseln.

CBC – Cipher Block Chaining



Key-Stream-Generatoren

- AES wird benutzt um einen Schlüssel-Stream zu erzeugen

Key-Stream-Generatoren

- AES wird benutzt um einen Schlüssel-Stream zu erzeugen
- Dieser Key-Stream ist abhängig vom Passwort und vom IV (Initial Vector = Anfangswert)

Key-Stream-Generatoren

- AES wird benutzt um einen Schlüssel-Stream zu erzeugen
- Dieser Key-Stream ist abhängig vom Passwort und vom IV (Initial Vector = Anfangswert)
- Die Daten werden durch XOR mit dem Schlüssel-Stream verschlüsselt.

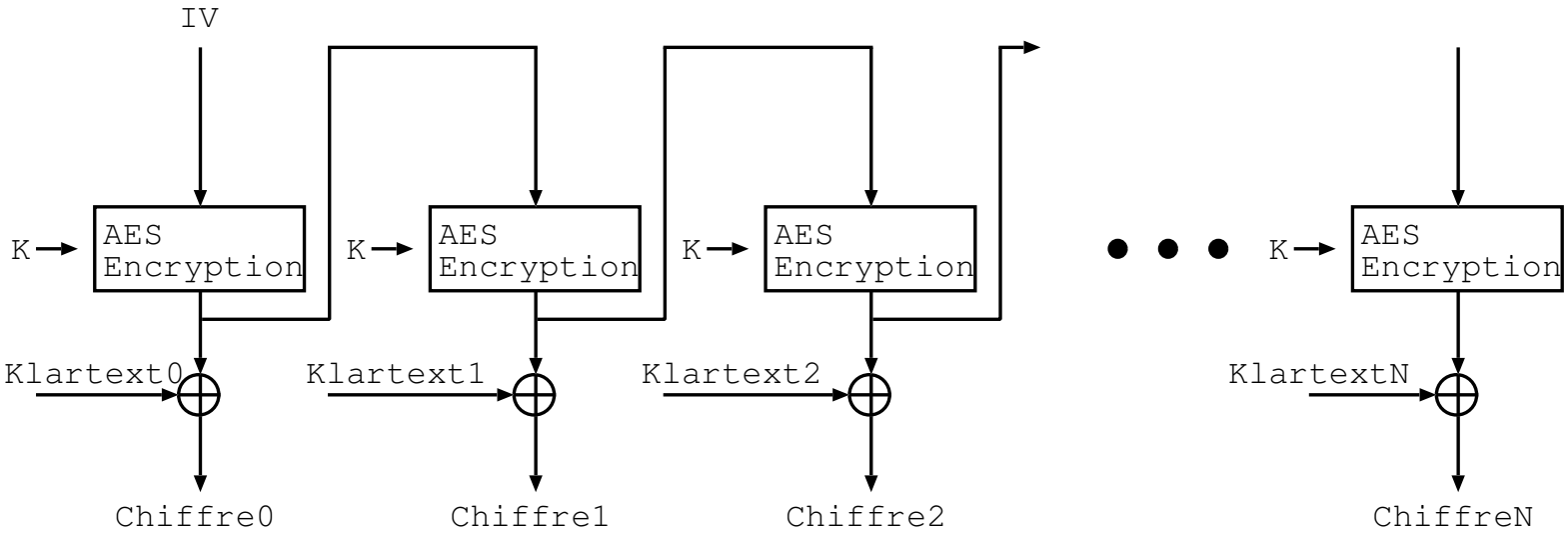
Key-Stream-Generatoren

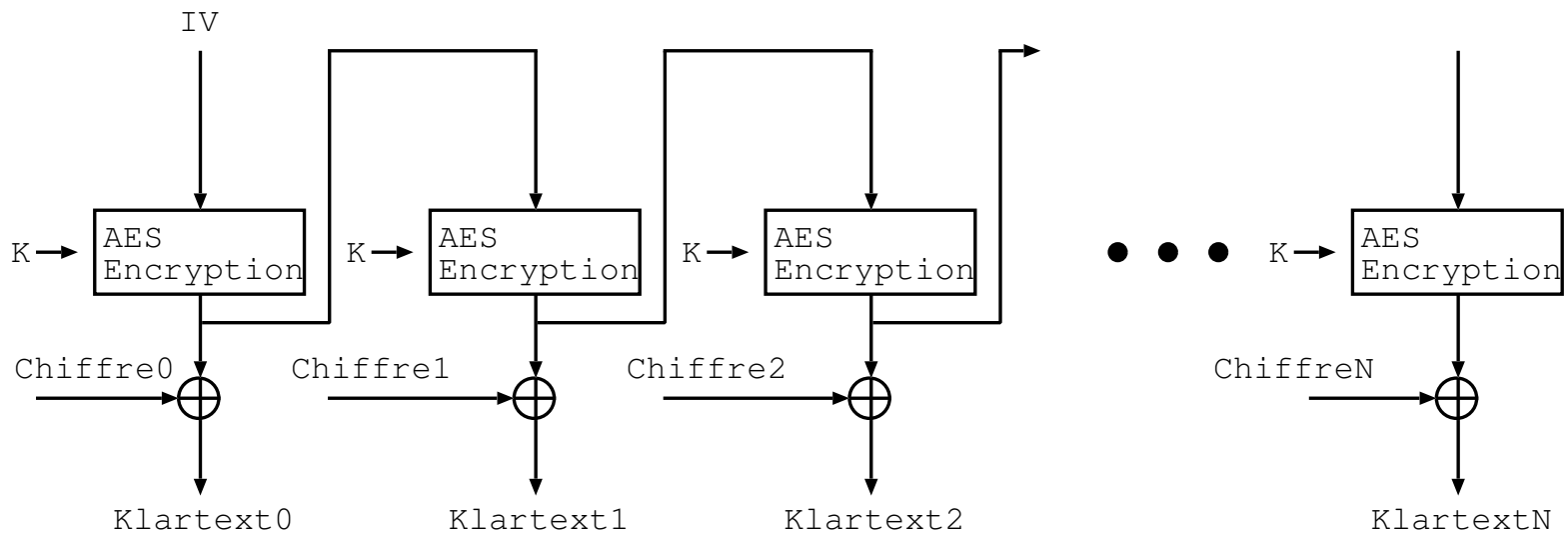
- AES wird benutzt um einen Schlüssel-Stream zu erzeugen
- Dieser Key-Stream ist abhängig vom Passwort und vom IV (Initial Vector = Anfangswert)
- Die Daten werden durch XOR mit dem Schlüssel-Stream verschlüsselt.
- AES-Entschlüsselungsalgorithmus wird nicht gebraucht.

Key-Stream-Generatoren

- AES wird benutzt um einen Schlüssel-Stream zu erzeugen
- Dieser Key-Stream ist abhängig vom Passwort und vom IV (Initial Vector = Anfangswert)
- Die Daten werden durch XOR mit dem Schlüssel-Stream verschlüsselt.
- AES-Entschlüsselungsalgorithmus wird nicht gebraucht.
- Nachrichten können beliebige Längen aufweisen

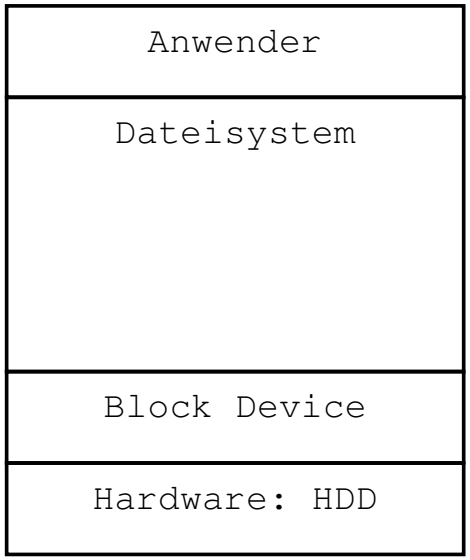
Key-Stream-Generatoren (Generelle Beispiel)



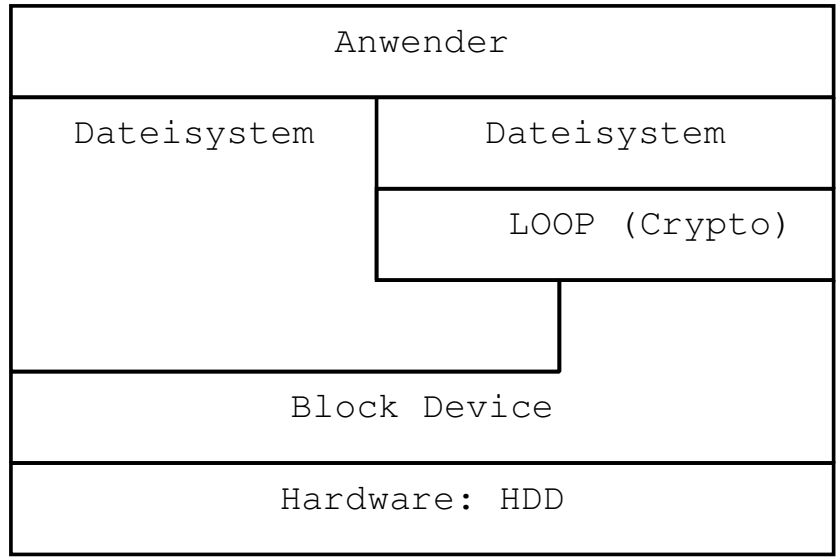


Konkrete Anwendung: Linux CryptoLOOP

Dateisystem direkt auf HDD:



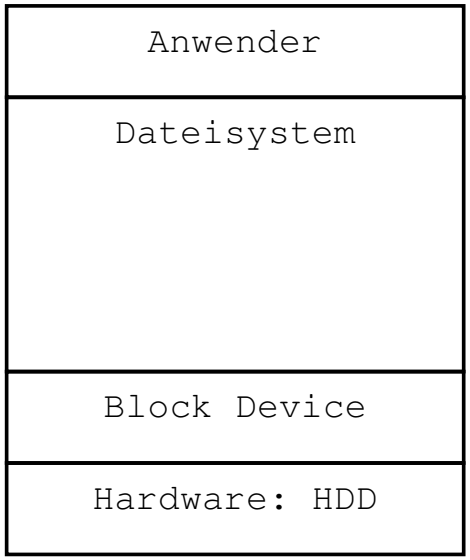
Dateisystem auf LOOP-Driver:



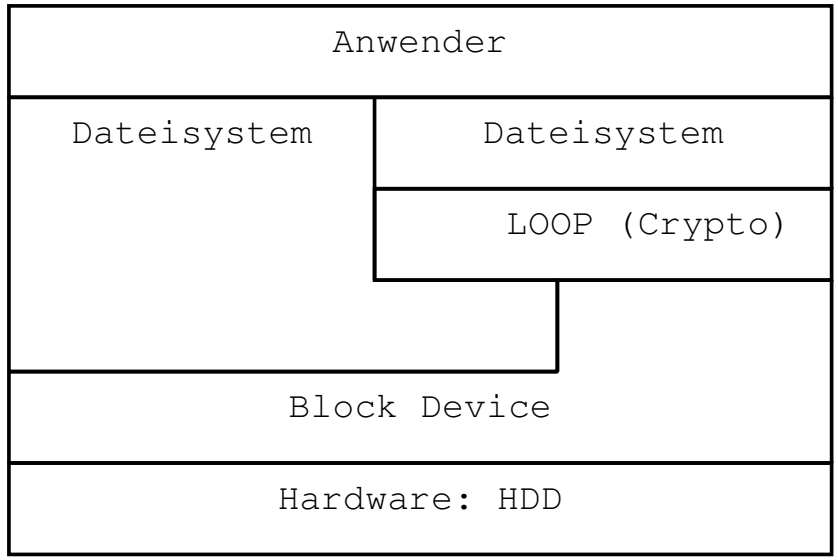
- Dateisysteme nur auf Block Devices

Konkrete Anwendung: Linux CryptoLOOP

Dateisystem direkt auf HDD:



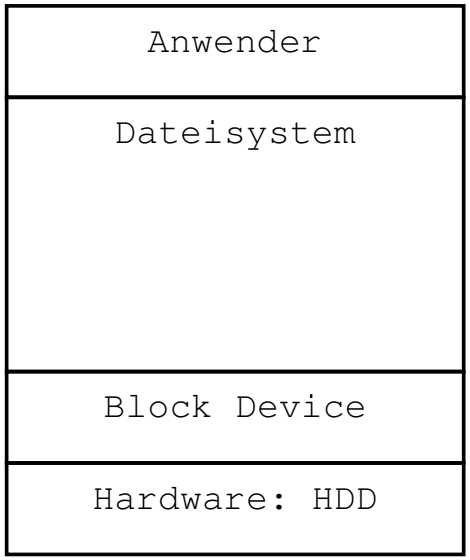
Dateisystem auf LOOP-Driver:



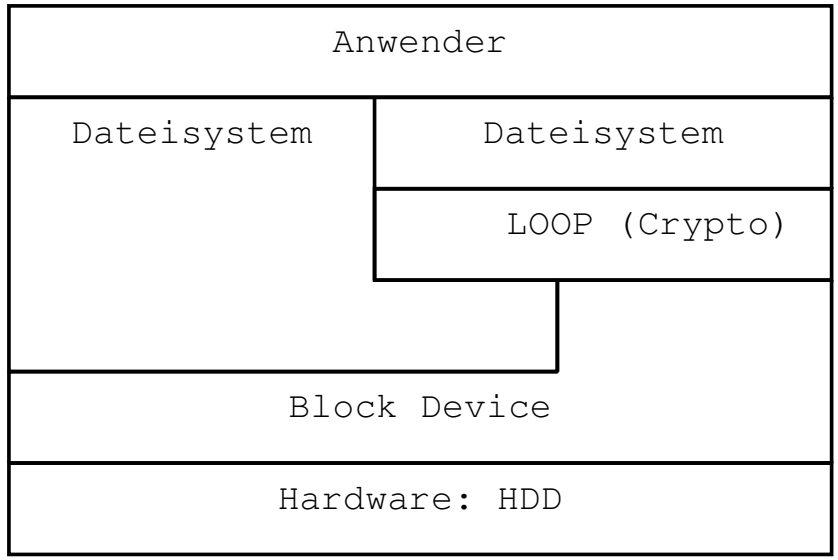
- Dateisysteme nur auf Block Devices
- Loop wandelt Dateien in Blockdevices (Images mounten)

Konkrete Anwendung: Linux CryptoLOOP

Dateisystem direkt auf HDD:



Dateisystem auf LOOP-Driver:



- Dateisysteme nur auf Block Devices
- Loop wandelt Dateien in Blockdevices (Images mounten)
- Loop-Treiber kann AES

Eigenschaften von CryptoLOOP

- Blockgröße 512 Bytes

Eigenschaften von CryptoLOOP

- Blockgröße 512 Bytes
- Verschlüsselung jedes Blockes in CBC-Mode

Eigenschaften von CryptoLOOP

- Blockgröße 512 Bytes
- Verschlüsselung jedes Blockes in CBC-Mode
- Optimierung in Assembler

Eigenschaften von CryptoLOOP

- Blockgröße 512 Bytes
- Verschlüsselung jedes Blockes in CBC-Mode
- Optimierung in Assembler
- Im “normalem” Modus Password SHA gehasht

Eigenschaften von CryptoLOOP

- Blockgröße 512 Bytes
- Verschlüsselung jedes Blockes in CBC-Mode
- Optimierung in Assembler
- Im “normalem” Modus Password SHA gehasht
- Wiederholtes verschlüsseln des Passwortes

Geschwindigkeit

- Pentium III 850 MHz: 11485591 Byte/Sekunde ca. 11 MByte/s.

Geschwindigkeit

- Pentium III 850 MHz: 11485591 Byte/Sekunde ca. 11 MByte/s.
- $\Rightarrow$ 22411 512-Byte-Blöcke/s $\Rightarrow$ 717152 16-Byte-(128-Bit)-Blöcke/s.

Geschwindigkeit

- Pentium III 850 MHz: 11485591 Byte/Sekunde ca. 11 MByte/s.
- $\Rightarrow$ 22411 512-Byte-Blöcke/s $\Rightarrow$ 717152 16-Byte-(128-Bit)-Blöcke/s.
- Brute-Force-Attack:
$$\frac{2^{256} Moeglichkeiten}{717152 \frac{16-Byte-(128-Bit)-Bloecke}{s}} * \frac{1}{60*60*24*7*53} \approx 5 * 10^{63} Jahre$$

Bibliographie und Referenzen

- Bibliographie und Referenzen
- Vorführung