# Kryptographische Algorithmen

## Lerneinheit 4: Advanced Encryption Standard (AES)

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen


Wintersemester 2017/2018

# Einleitung

- Bei der Standardisierung von DES wurde vorgeschrieben, daß DES alle 5 Jahre bezüglich seiner Sicherheit evaluiert wird. Ende der 1990er Jahre ergab die Überprüfung, daß DES nicht mehr als sicher einzustufen ist.
- Um einen neuen Standard für symmetrische Verschlüsselung zu schaffen, schrieb das NIST im Jahre 1997 einen Wettbewerb zur Bestimmung des Advanced Encryption Standard (AES) aus.
- Anforderungen an den Kryptoalgorithmus waren:
  - Erfüllung aller heutigen Sicherheitsanforderungen
  - Öffentlich zugängliche Dokumentation der Spezifikation und Designkriterien
  - Weltweite kostenlose Nutzung

# Die 15 AES Kandidaten

| *Name* | *Entwickler* | *Typ* |
|---|---|---|
| CAST-256 | Entrust (CA) | Industrie |
| Crypton | Future Systems (KR) | Industrie |
| DEAL | Outbridge, Knudsen (USA/DK) | Forschung |
| DFC | ENS-CNRS (FR) | Forschung |
| E2 | NTT (JP) | Industrie |
| Frog | TecApro (CR) | Industrie |
| HPC | Schroeppel (USA) | Forschung |
| LOKI97 | Brown et al. (AU) | Forschung |
| Magenta | Deutsche Telekom (DE) | Industrie |
| Mars | IBM (USA) | Industrie |
| RC6 | RSA (USA) | Industrie |
| Rijndael | Daemen and Rijmen (BE) | Forschung |
| SAFER++ | Cyling (USA) | Industrie |
| Serpent | Anderson, Bihan, Knudson (UK/IL/DK) | Forschung |
| Twofish | Counterpane (USA) | Industrie |

# Auswahlkriterien

Bei der Auswahl des Standards wurden folgende Kriterien angewandt:

Sicherheit: Dies ist das wichtigste aber auch das am schwierigsten zu bewertende Kriterium.

Kosten: Man unterscheidet zwei Arten von Kosten:

- Geistiges Eigentum: Freie Verfügbarkeit des Algorithmus, aber auch Verzicht auf Patentansprüche auf Ideen in Vorschlägen der Konkurrenten
- Resourcenverbrauch: Rechenzeit, Speicherplatzbedarf, Größe des Programms, etc.

# Auswahlkriterien (Forts.)

Implementierungsaspekte: Folgende Punkte wurden beurteilt:

- Flexibilität: Ist der Algorithmus vielseitig einsetzbar?
- Schlüsselagilität: Wie effizient ist die Schlüssel-konfiguration?
- Einfachheit: Wie hoch ist der Aufwand zur Implementierung des Algorithmus?

Man beachte, daß diese Punkte schwer quantifizierbar sind.

In einem ersten Auswahlverfahren wurden aus den Kandidanten folgende fünf Finalisten ausgewählt: RC6, Rijndael, Twofish, MARS und Crypton.

# Und der Sieger ist ...Rijndael

Die fünf Finalisten wurden nochmals genau untersucht. Am 2. Oktober 2000 gab die NIST bekannt, daß Rijndael der Gewinner des Wettbewerbs und somit der zukünftige AES ist. Die Gründe für diese Entscheidung sind:

- Sicherheit: Rijndael erfüllt alle Sicherheitsanforderungen.
- Geschwindigkeit: Die Implementierung von Rijndael war die schnellste aller Kandidaten mit gleichmässig guten Performanz auf allen betrachteten Plattformen.
- Speicherbedarf: Rijndael benötigt wenig RAM- und ROM-Speicher.
- Implementierung in Hardware: Rijndael liefert die beste Performanz in Hardware-Implementierung.

# Wiederholung: Endliche Körper

Ein Körper ist eine algebraische Struktur $(A, \oplus, \odot)$ mit folgenden Eigenschaften:

- $(A, \oplus)$ und $(A \setminus \{0\}, \odot)$ sind kommutative Gruppen.
- Es gilt das Distributivgesetz, d.h., für alle $a, b, c$ gilt:

$$\begin{aligned} (a \oplus b) \odot c &= a \odot c \oplus b \odot c \\ c \odot (a \oplus b) &= c \odot a \oplus c \odot b \end{aligned}$$

Falls $\|A\| < \infty$, dann nennt man $(A, \oplus, \odot)$ einen endlichen Körper oder auch Galois-Feld.

Fakt: Es gibt einen endlichen Körper mit $\|A\| = n$ genau dann, wenn $n$ eine Primzahlpotenz ist, d.h., $n = p^k$ für eine Primzahl $p$.

# Rechnen mit endlichen Körpern

Um über einem endlichen Körper $GF(p^k)$ rechnen zu können, muß zunächst die Addition und Multiplikation definiert werden.

Idee: Man stellt die Elemente als $k$-Tupel über $\mathbb{Z}_p$ dar und interpretiert diese als Polynome mit Grad $k-1$ mit Koeffizienten in $\mathbb{Z}_p$. Das Element $(a_{k-1}, \ldots, a_1, , a_0) \in (\mathbb{Z}_p)^k$ steht für das Polynom

$$a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots + a_1x + a_0$$

Beispiel: Das Element $(4, 0, 3, 2)$ aus $GF(5^4)$ steht für das Polynom $4x^3 + 3x + 2$.

# Die Addition über $GF(p^k)$

Die Addition über $GF(p^k)$ ist die Polynomaddition. Für

$$a(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots + a_1x + a_0$$

und

$$b(x) = b_{k-1}x^{k-1} + b_{k-2}x^{k-2} + \ldots + b_1x + b_0$$

ist $a(x) \oplus b(x) = c(x)$ wobei

$$c(x) = c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1x + c_0$$

und $c_i = (a_i + b_i) \bmod p$.

# Die Multiplikation über $GF(p^k)$

Will man die Multiplikation über $GF(p^k)$ auf analoge Weise, sprich als Polynommultiplikation, realisieren, dann trifft man auf das folgende ...

Problem: Das Produkt zweier Polynome mit Grad $k-1$ ist ein Polynom vom Grad $2(k-1)$ und somit kein Element von $GF(p^k)$.

Lösung: Man reduziert das Ergebnis der Multiplikation indem man modulo einem irreduziblem Polynom mit Grad $k$ rechnet. Man setzt hierzu die Polynomdivision mit Rest ein.

Ein Polynom ist irreduzibel, falls es nur durch 1 und sich selbst ohne Rest teilbar ist.

# Beispiel $GF(2^3)$ (Teil 1)

Die Elemente in $GF(2^3)$ sind:

| Tupel | Polynom | Zahl |
|---|---|---|
| $(0,0,0)$ | $0$ | $0$ |
| $(0,0,1)$ | $1$ | $1$ |
| $(0,1,0)$ | $x$ | $2$ |
| $(0,1,1)$ | $x+1$ | $3$ |
| $(1,0,0)$ | $x^2$ | $4$ |
| $(1,0,1)$ | $x^2+1$ | $5$ |
| $(1,1,0)$ | $x^2+x$ | $6$ |
| $(1,1,1)$ | $x^2+x+1$ | $7$ |

# Beispiel $GF(2^3)$ (Teil 2)

Die Addition von $x^2+x+1$ und $x^2+1$ liefert das Ergebnis

$$2x^2+x+2=x.$$

Um obige Werte zu multiplizieren, berechnet man zunächst

$$\begin{aligned}(x^2+x+1)(x^2+1) &= x^4+x^3+2x^2+x+1\\ &= x^4+x^3+x+1\end{aligned}$$

Anschließend dividiert man das Ergebnis mit dem irreduziblen Polynom $m(x)=x^3+x+1$.

# Beispiel $GF(2^3)$ (Teil 3)

Polynomdivision:

$$
\begin{array}{rcrcrcrcl}
(x^4 & + & x^3 & + & & & x & + & 1) \\
x^4 & + & & & x^2 & + & x & & \\
\hline
 & & x^3 & + & x^2 & + & & & 1 \\
 & & x^3 & + & & & x & + & 1 \\
\hline
 & & & & x^2 & + & x & &
\end{array}
\quad : (x^3 + x + 1) = x + 1
$$

$\Leftarrow$ Rest

Das Ergebnis des Produkts $(x^2 + x + 1) \odot (x^2 + 1)$ ist also $x^2 + x$.

Somit ist $(1, 1, 1) \odot (1, 0, 1) = (1, 1, 0)$.

# Algorithmen für Polynomarithmetik

Um endliche Körper in der Praxis einsetzen zu können, benötigt man effiziente Algorithmen zur Berechnung der Polynomarithmetik.

- Die Addition ist einfach zu implementieren. Sie ist die komponentenweise Addition modulo der Primzahlbasis $p$.
- Die Multiplikation ist mittels Polynommultiplikation und Polynomdivision berechenbar. Der Aufwand ist quadratisch im Grad der Polynome.
- Eine weitere wichtige Operation ist die Bestimmung von multiplikativen Inversen. Sie kann mittels des Extended Euklid Algorithmus berechnet werden.

# Verbesserung der Effizienz

Ist bei einer Anwendung der endliche Körper $GF(m)$ fest gewählt, dann läßt sich die Multiplikation und Inversenberechnung deutlich effizienter gestalten.

Bekanntlich ist $(GF(m) \setminus \{0\}, \odot)$ eine Gruppe und besitzt somit ein erzeugendes Element $g$. D.h., für jedes $a \in GF(p^n) \setminus \{0\}$ existiert eine Zahl $i$ so daß $g^i = a$.

Um das Produkt $a \odot b$ berechnen, verwendet man den Generator $g$. Falls $a = g^i$ und $b = g^j$, dann ist

$$a \odot b = g^i \odot g^j = g^{i+j \bmod (m-1)}$$

Das Inverse von $a = g^i$ ist $a^{-1} = g^{m-1-i}$.

# Verbesserung der Effizienz (Forts.)

Die Berechnung der Multiplikation wurde also reduziert auf die Kombination der Funktion $\exp(i) = g^i$ und deren inversen Funktion $\log_g(x)$.

Liegen diese Funktionen in tabellarischer Form vor, dann beschränkt sich die Berechnung auf ganzzahlige Arithmetik und Suche von Tabelleneinträgen.

Berechnung der Multiplikation $a \odot b$:

1. Finde in der $\log_g$-Tabelle die Indizes $i$ und $j$ so daß $g^i = a$ und $g^j = b$
2. Berechne $\ell = i + j \bmod (m - 1)$
3. $a \odot b$ ist der $\ell$-te Eintrag in der $\exp_g$-Tabelle.

Analog: Berechnung von multiplikativen Inversen.

# Beispiel $GF(2^3)$

Das Element $(0,1,0) = x$ ist ein erzeugendes Element der Gruppe $GF(2^3) \setminus \{0\}$.

| $\exp_g$-Tabelle | | |
|---|---|---|
| $i$ | $x^i$ | $a$ |
| 0 | 1 | $(0,0,1)$ |
| 1 | $x$ | $(0,1,0)$ |
| 2 | $x^2$ | $(1,0,0)$ |
| 3 | $x+1$ | $(0,1,1)$ |
| 4 | $x^2+x$ | $(1,1,0)$ |
| 5 | $x^2+x+1$ | $(1,1,1)$ |
| 6 | $x^2+1$ | $(1,0,1)$ |

| $\log_g$-Tabelle | | |
|---|---|---|
| $a$ | Polynom | $i$ |
| $(0,0,1)$ | 1 | 0 |
| $(0,1,1)$ | $x+1$ | 3 |
| $(0,1,0)$ | $x$ | 1 |
| $(1,0,0)$ | $x^2$ | 2 |
| $(1,1,0)$ | $x^2+x$ | 4 |
| $(1,0,1)$ | $x^2+1$ | 6 |
| $(1,1,1)$ | $x^2+x+1$ | 5 |

# Beispiel $GF(2^3)$ (Forts.)

Zur Berechnung von $(1,1,1) \odot (1,0,1) = (x^2+x+1)(x^2+1)$ bestimmt man

$$\begin{aligned} i &= log_x[(1,1,1)] = 5 \text{ und} \\ j &= log_x[(1,0,1)] = 6. \end{aligned}$$

Somit ist $\ell = (5+6) \bmod 7 = 4$ und obiges Produkt gleich

$$\exp_x[4] = (1,1,0) = x^2 + x.$$

Um $(0,1,1)^{-1}$ bestimmen, setze $i = \log_x[(0,1,1)] = 3$ und berechne $\ell = 7-3 = 4$. $(0,1,1)^{-1}$ ist gleich $\exp_x[4] = (1,1,0)$.

# Einsatz von endlichen Körpern in Rijndael

- Viele der in Rijndael eingesetzten Transformationen basieren auf arithmetischen Operationen über $GF(2^8)$. Der Grund für die Wahl von $GF(2^8)$ ist, daß man damit Operationen auf Byte-Ebene definieren kann.
- Das irreduzible Polynom ist

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

- Multiplikation und Berechnung der multiplikativen Inversen werden mittels Exp- und Log-Tabellen implementiert. Jede dieser Tabellen hat eine Größe von 256 Byte.

# Das Rijndael Kryptosystem

Rijndael ist ein symmetrisches Kryptosystem mit folgenden Parametern:

- $\mathcal{P} = \mathcal{C} = \{0,1\}^{32 \cdot n_B}$, wobei $n_B = \{4, 6, 8\}$
- $\mathcal{K} = \{0,1\}^{32 \cdot n_K}$, wobei $n_K \in \{4, 6, 8\}$

Sowohl die Block- als auch die Schlüssellänge ist demnach variabel mit einer Länge von 128, 192 oder 256 Bit.

Im AES Standard wurde die Blocklänge auf 128 Bit, d.h. $n_B = 4$ festgelegt.

Die Bezeichnung Rijndael wurde von den Namen seiner Erfinder Vincent Rijmen und Joan Daemen abgeleitet.

# Der interne Zustand von Rijndael

Der interne Zustand von Rijndael ist ein **Bytefeld**, dessen Länge mit der Blocklänge übereinstimmt.

Das Feld wird entweder als Array der Länge $4n_B$ oder als Matrix der Dimension $4 \times n_B$ angesprochen.

Die Konvertierung zwischen Arrayposition $n$ und Matrixindex $(i, j)$ erfolgt gemäß den folgenden Formeln:

$$i = n \bmod 4, \qquad j = \left\lfloor \frac{n}{4} \right\rfloor, \qquad \text{bzw.} \qquad n = i + 4 \cdot j.$$

Der Schlüsselblock wird auf analoge Weise interpretiert.

# Der Verschlüsselungsalgorithmus

RIJNDAEL$(x, k)$
**Input:** Klartext $x$ ($4 \cdot n_B$ Byte), Schlüssel $k$ ($4 \cdot n_K$ Byte)
**Output:** Geheimtext $y$ ($4 \cdot n_B$ Byte)

1 $s := x$;
2 KeyExpansion$(k, k_R[0, n_R])$;
3 AddRoundKey$(s, k_R[0])$;
4 **for** $i := 1$ **to** $n_R - 1$ **do** Round$(i, s, k_R[i])$;
5 FinalRound$(n_R, s, k_R[n_R])$;
6 **return** $s$;

# Anzahl der Runden

Die Anzahl der Runden $n_R$ ist abhängig von der Blocklänge $n_B$ und Schlüssellänge $n_K$.

| $n_R$ | $n_B = 4$ | $n_B = 6$ | $n_B = 8$ |
|---|---|---|---|
| $n_K = 4$ | 10 | 12 | 14 |
| $n_K = 6$ | 12 | 12 | 14 |
| $n_K = 8$ | 14 | 14 | 14 |

Für $n_B = 4$ und $n_K = 6$ ist beispielsweise $n_R = 12$.

# Aufbau einer Rundentransformation

Bei einer Verschlüsselsrunde von Rijndael kommen die folgenden Funktionen zum Einsatz:

- $\textsc{ByteSub}(s)$: Diese Operation ist eine nicht-lineare Substitution, die jedes Byte des Zustands unabhängig von den anderen verändert.
- $\textsc{ShiftRow}(s)$: Bei dieser Operation wird zeilenweise eine Linksverschiebung durchgeführt.
- $\textsc{MixColumns}(s)$: Diese Operation bearbeitet den Zustand spaltenweise.
- $\textsc{AddRoundKey}(s, k_R)$: Hier wird der Rundenschlüssel auf den Zustand bitweise addiert.

Jede dieser Funktionen ist für sich invertierbar.

# Round() und FinalRound()

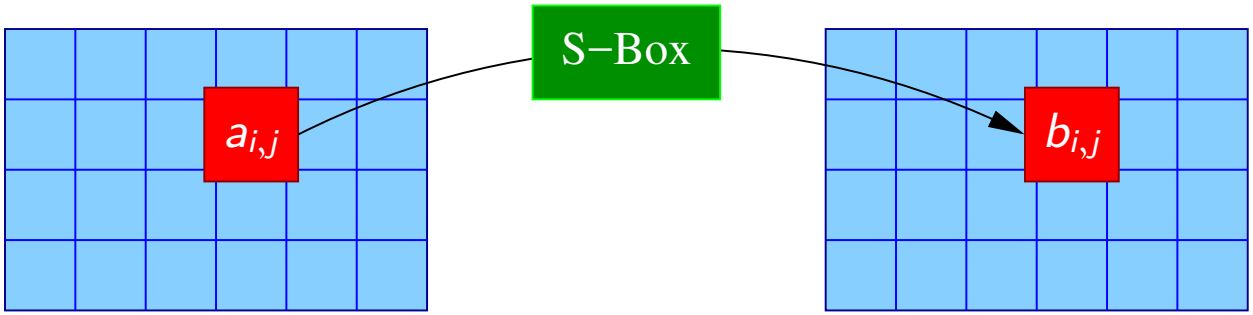
Round($s, k_R$)
**Input:** Zustand $s$, Rundenschlüssel $k_R$ (jeweils $4 \cdot n_B$ Byte)
**Output:** Zustand $s$ ($4 \cdot n_B$ Byte)

1 ByteSub($s$);
2 ShiftRow($s$);
3 MixColumn($s$);
4 AddRoundKey($s, k_R$);

FinalRound($s, k_R$)
**Input:** Zustand $s$, Rundenschlüssel $k_R$ (jeweils $4 \cdot n_B$ Byte)
**Output:** Zustand $s$ ($4 \cdot n_B$ Byte)

1 ByteSub($s$);
2 ShiftRow($s$);
3 AddRoundKey($s, k_R$);

# Die ByteSub Transformation




Die S-Box wird auf jedes Byte des Zustands angewandt. Sie ist eine Komposition aus zwei biijektiven Abbildungen.

1. Zuerst wird das zu berechnende Byte als Element von $GF(2^8)$ interpretiert und auf sein multiplikatives Inverses abgebildet. Das Element 0 wird auf sich selbst abgebildet.
2. Der zweite Schritt besteht in der Berechnung einer affinen Abbildung, wobei über $GF(2)$ gerechnet wird.

# Die ByteSub Transformation (Forts.)

Die affine Funktion $f: \{0,1\}^8 \mapsto \{0,1\}^8$ ist wie folgt definiert:

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

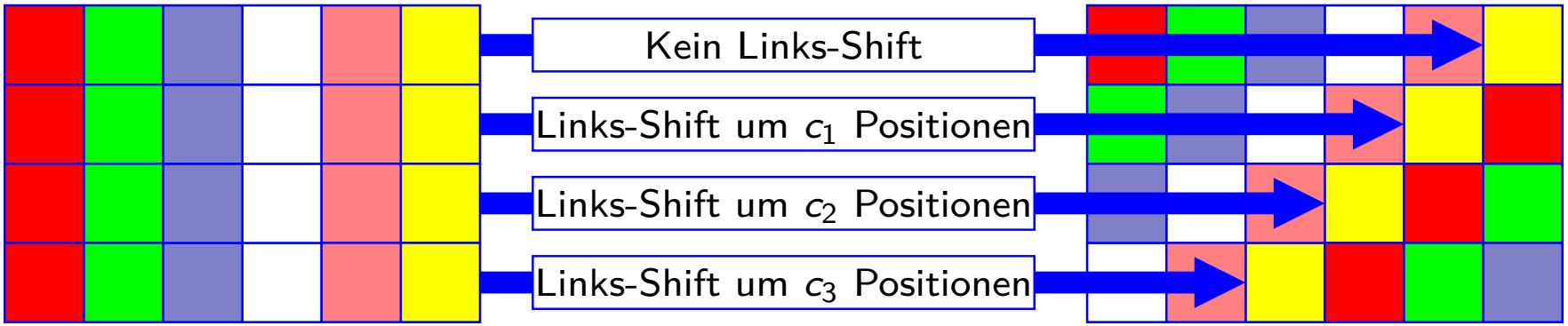
Das zu transformierende Byte wird als Element in $(\mathbb{Z}_2)^8$ interpretiert.

# Die ByteSub Transformation (Forts.)

Die zu $f$ inverse Abbildung $f^{-1} : \{0,1\}^8 \mapsto \{0,1\}^8$ ist ebenfalls eine affine Funktion und wie folgt definiert:

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

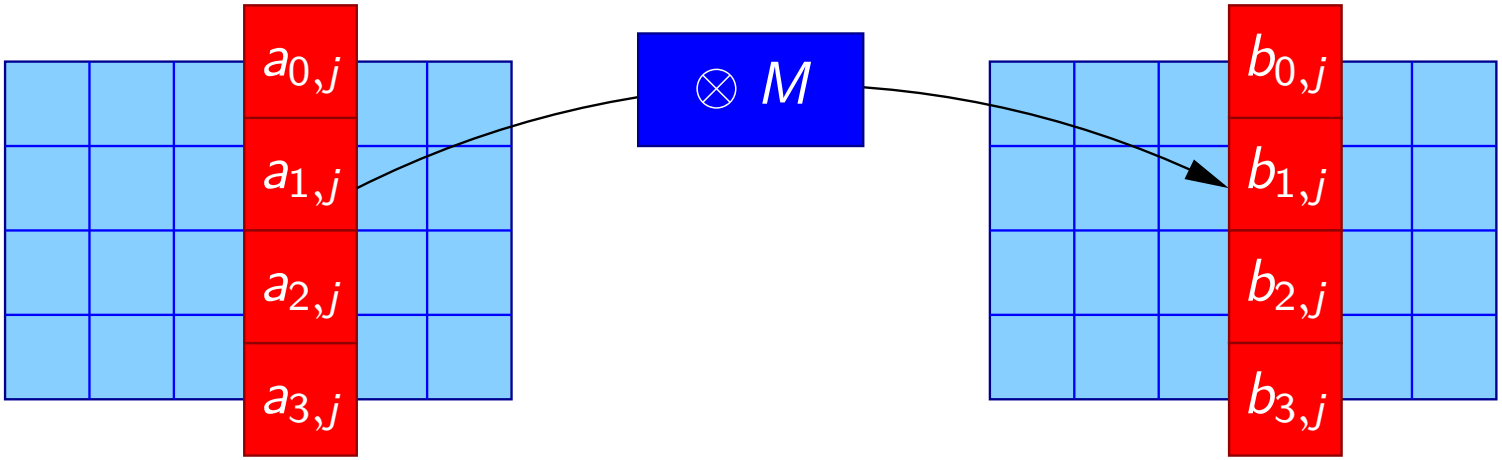
# Die ShiftRow Transformation




Die ShiftRow-Operation ist eine zeilenweise Links-Verschiebung. Die Offsetwerte der Verschiebung sind abhängig von der Blocklänge:

| $n_B$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| 4 | 1 | 2 | 3 |
| 6 | 1 | 2 | 3 |
| 8 | 1 | 3 | 4 |

# Die MixColumn Transformation




Die MixColumn Transformation bearbeitet den Zustand spaltenweise. Eine Spalte wird als Vektor interpretiert und mit einer $4 \times 4$ Matrix multipliziert. Die Berechnungen erfolgen über $GF(2^8)$.

# Die MixColumn Transformation (Forts.)

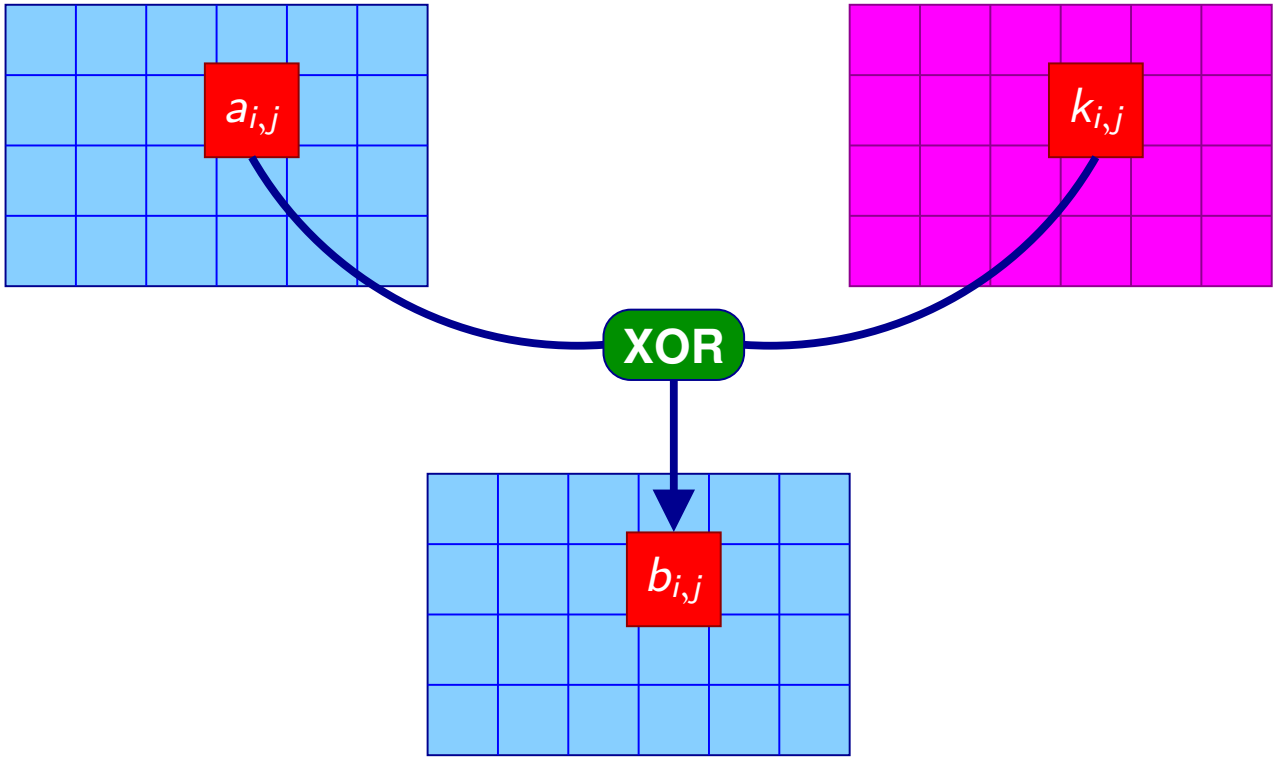
$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Die hierzu inverse Transformation ist:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Beachte: Man rechnet über dem Körper $GF(2^8)$.

# AddRoundKey Transformation




In der AddRoundKey Transformation wird der Rundenschlüssel auf den Zustand addiert. Hierzu wird jedes Byte des Zustand mit dem entsprechenden Byte des Rundenschlüssels via XOR verknüpft.

# Entschlüsselungsalgorithmus

Da jede der Basisoperationen für sich invertierbar ist, besteht die Entschlüsselung in der Anwendung der inversen Transformationen in umgekehrter Reihenfolge.

Die Umkehrung von $\textsc{Round}()$ ist beispielsweise:

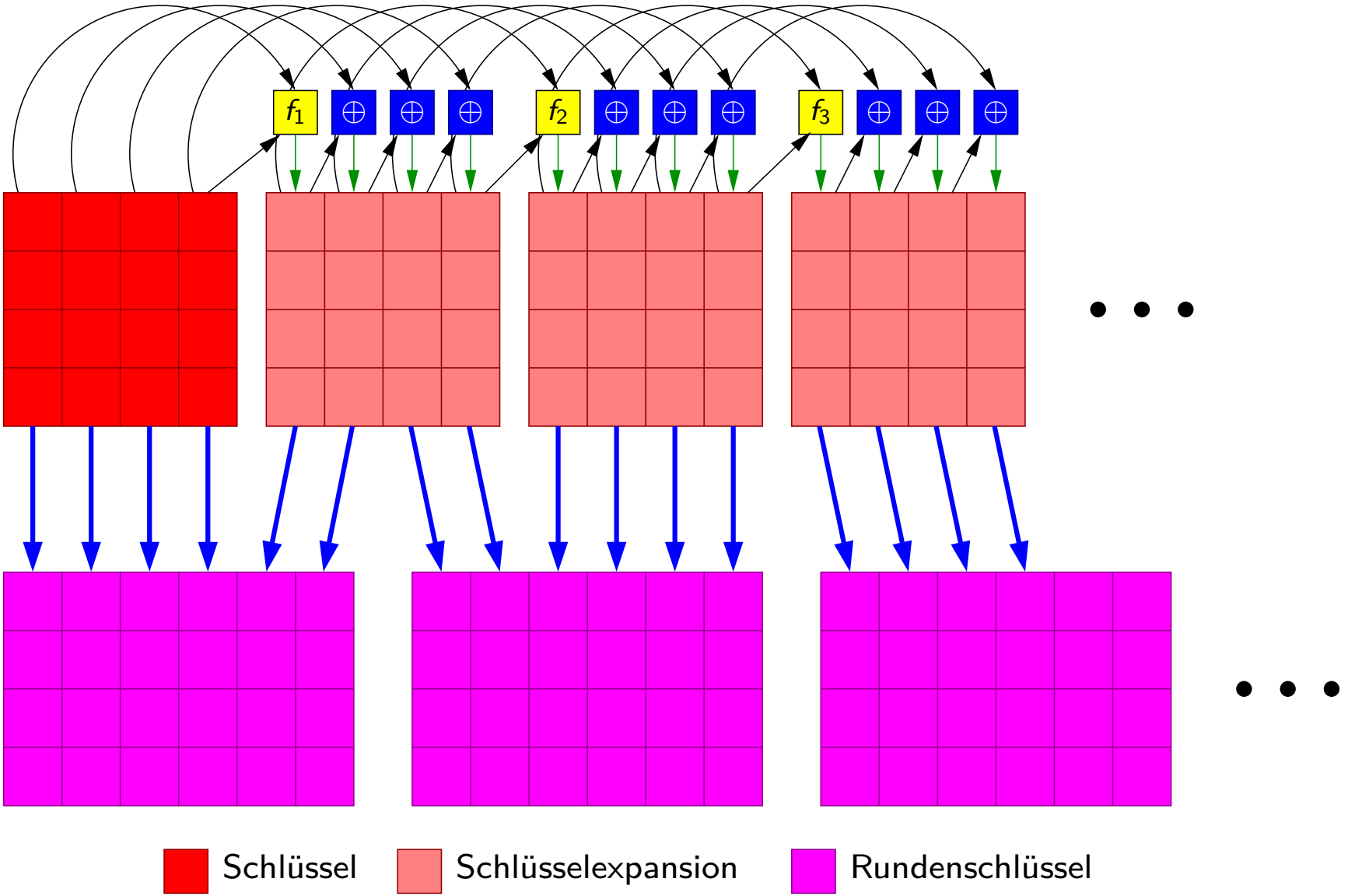
$\textsc{InvRound}(s, k_R)$
**Input:** Zustand $s$, Rundenschlüssel $k_R$ (jeweils $4 \cdot n_B$ Byte)
**Output:** Zustand $s$ ($4 \cdot n_B$ Byte)

1 $\textsc{AddRoundKey}(s, k_R)$;
2 $\textsc{InvMixColumn}(s)$;
3 $\textsc{InvShiftRow}(s)$;
4 $\textsc{InvByteSub}(s)$;

# Erzeugen der Rundenschlüssel

- Für eine komplette Ver- bzw. Entschlüsselung werden $n_R + 1$ Rundenschlüssel mit jeweils $4n_B$ Byte benötigt.
- Mit anderen Worten: Aus den $4n_K$ Byte des Schlüssels müssen also $4n_B(n_R + 1)$ Byte für die Rundenschlüssel genereriert werden.
- Notation: $K[i]$ und $W[i]$ bezeichnen $4 \times i$-Matrizen über $GF(2^8)$. Die Spalten der Matrizen werden als Vektoren interpretiert.
- Ziel: Berechne aus $K[n_K]$ die Matrix $W[n_B(n_R + 1)]$ und zerlege diese in die Rundenschlüssel.

# Schlüsselexpansion (Idee)



# Hilfsfunktionen

- Die Abbildung $\mathcal{S} : GF(2^8) \mapsto GF(2^8)$ ist identisch mit der S-Box der ByteSub-Transformation.
- Die Funktion $RC : GF(2^8) \mapsto GF(2^8)$ ist rekursiv definiert:

$$\begin{aligned} RC(0) &= x^0 \quad (= 1) \\ RC(1) &= x^1 \quad (= 2) \\ RC(i) &= x^{i-1} \quad \text{für } i \geq 2 \end{aligned}$$

- Vektoraddition: $\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \oplus \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_0 \oplus y_0 \\ x_1 \oplus y_1 \\ x_2 \oplus y_2 \\ x_3 \oplus y_3 \end{pmatrix}$

# Hilfsfunktionen (Forts.)

- Abbildung $f$:

$$f(i, \vec{x}, \vec{y}) = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \oplus \begin{pmatrix} \mathcal{S}(y_1) \\ \mathcal{S}(y_2) \\ \mathcal{S}(y_3) \\ \mathcal{S}(y_0) \end{pmatrix} \oplus \begin{pmatrix} RC(i) \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Abbildung $g$:

$$g(\vec{x}, \vec{y}) = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \oplus \begin{pmatrix} \mathcal{S}(y_0) \\ \mathcal{S}(y_1) \\ \mathcal{S}(y_2) \\ \mathcal{S}(y_3) \end{pmatrix}$$

# Schlüsselexpansion für $n_K \leq 6$

KeyExpansion($K[n_K], W[n_B(n_R+1)]$)
**Input:** Schlüssel $K$ ($4n_K$ Byte) wobei $n_K \leq 6$
**Output:** Rundenschlüssel ($4n_B(n_R+1)$ Byte)

```
1  for j := 0 to n_K − 1 do
2    W[j] := K[j];
3  for (j := n_K to n_B(n_R + 1) − 1) do
4    if j mod n_K = 0 then
5      W[j] := f(j/n_K, W[j − n_K], W[j − 1])
6    else
7      W[j] := W[j − n_K] ⊕ W[j − 1];
```

# Schlüsselexpansion für $n_K > 6$

KeyExpansion($K[n_K], W[n_B(n_R+1)]$)
**Input:** Schlüssel $K$ ($4n_K$ Byte) wobei $n_K > 6$
**Output:** Rundenschlüssel ($4n_B(n_R+1)$ Byte)

```
1  for j := 0 to n_K − 1 do
2    W[j] := K[j];
3  for j := n_K to n_B(n_R + 1) − 1 do
4    if (j mod n_K = 0) then
5      W[j] := f(j/n_K, W[j − n_K], W[j − 1])
6    elseif (j mod n_K = 4) then
7      W[j] := g(W[j − n_K], W[j − 1]);
8    else
9      W[j] := W[j − n_K] ⊕ W[j − 1];
```

# Abschließende Bemerkungen

- Weitere Details zur Spezifikation und Implementierung von Rijndael sowie Quellcode in C, C++ und Java findet man unter

  `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`

- Zusätzliche Informationen zum AES und den Standardisierungsprozess findet man bei der NIST:

  `http://csrc.nist.gov/CryptoToolkit/aes/`