

# Advanced Encryption Standard (AES)

**Dozent:** Prof. Dr. Michael Eichberg

**Version:** 1.0.2

**Basierend auf:** *Cryptography and Network Security - Principles and Practice, 8th Edition, William Stallings*

**Quellen:** NIST FIPS PUB 197, "Advanced Encryption Standard (AES)"



1

**Folien:** **HTML:** <https://delors.github.io/sec-aes/folien.de.rst.html>

**PDF:** <https://delors.github.io/sec-aes/folien.de.rst.html.pdf>

**Fehler melden:**

<https://github.com/Delors/delors.github.io/issues>

# Arithmetik endlicher Körper

- Ein Körper ist eine Menge, in der wir Addition, Subtraktion, Multiplikation und Division durchführen können, ohne die Menge zu verlassen.
- Die Division ist mit der folgenden Regel definiert:  $a/b = a(b^{-1})$ .

## Beispiel

Ein endlicher Körper (mit einer endlichen Anzahl von Elementen) ist die Menge  $Z_p$ , die aus allen ganzen Zahlen  $\{0, 1, \dots, p - 1\}$  besteht, wobei  $p$  eine Primzahl ist und in dem modulo  $p$  gerechnet wird.

# Arithmetik endlicher Körper

- Der Einfachheit halber — und aus Gründen der Implementierungseffizienz — möchten wir mit ganzen Zahlen arbeiten, die genau in eine bestimmte Anzahl von Bits passen, ohne dass Bitmuster verschwendet werden.

Ganze Zahlen im Bereich 0 bis  $2^n - 1$ , die in ein n-Bit-Wort passen.

- Wenn eine Operation des verwendeten Algorithmus die Division ist, dann müssen wir Arithmetik anwenden, die über einem (ggf. endlichen) Körper definiert ist.

Division erfordert, dass jedes nicht-null-Element ein multiplikatives Inverses hat.

- Wenn wir modulare Arithmetik auf die Menge der ganzen Zahlen  $Z_{2^n}$  (mit  $n > 1$ ) anwenden, dann erhalten wir **keinen** Körper!

Zum Beispiel hat die ganze Zahl **2** keine multiplikative Inverse in  $Z_{2^n}$  (mit  $n > 1$ ), d. h. es gibt keine ganze Zahl  $b$ , so dass  $2b \bmod 2^n = 1$

- Ein endlicher Körper der  $2^n$  Elemente enthält, wird als  $GF(2^n)$  bezeichnet.

Jedes Polynom in  $GF(2^n)$  kann durch eine n-Bit-Zahl dargestellt werden.

# Arithmetik endlicher Körper in Hinblick auf AES

- Beim *Advanced Encryption Standard* (AES) werden alle Operationen mit 8-Bit-Bytes durchgeführt
- Die arithmetischen Operationen: Addition, Multiplikation und Division werden über den endlichen Körper  $GF(2^8)$  durchgeführt.

## Definition

AES verwendet das irreduzible Polynom:

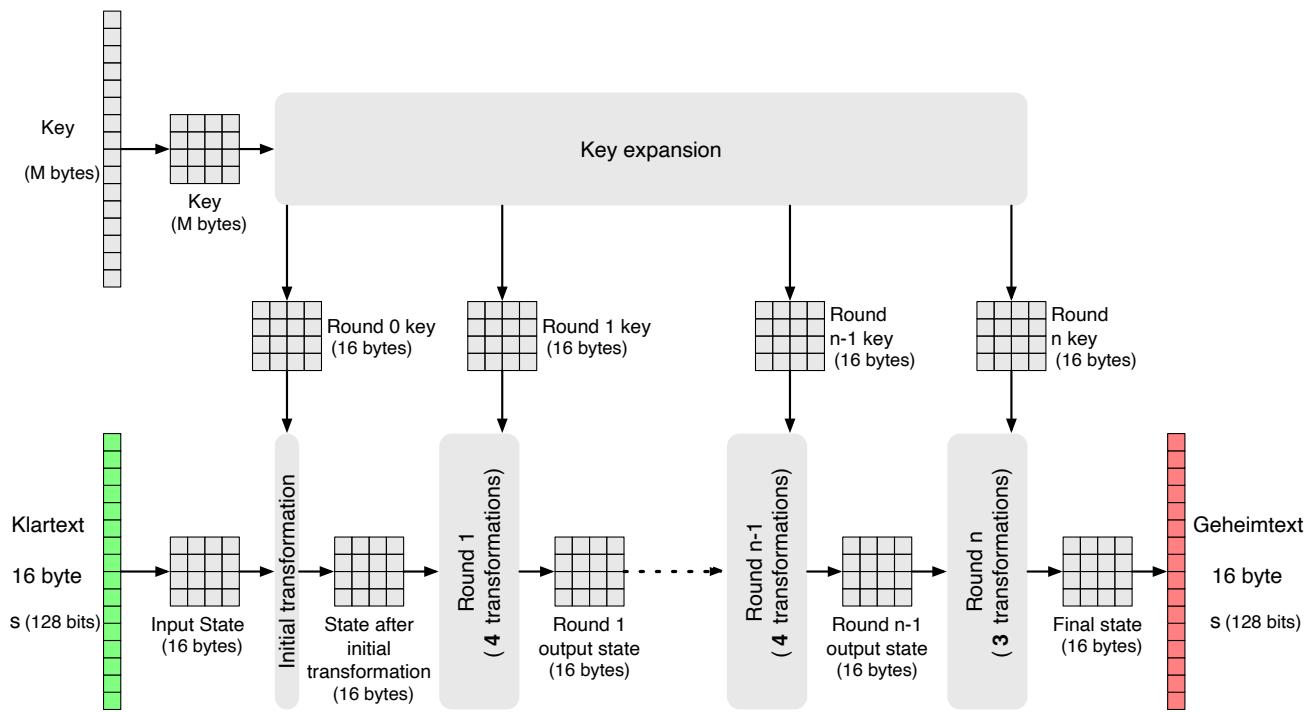
$$m(x) = x^8 + x^4 + x^3 + x + 1$$

# AES Schlüsseleigenschaften

- AES verwendet eine feste Blockgröße von 128 Bit.
- AES arbeitet mit einem 4x4-Array von 16 Bytes/128 Bits in Spaltenhauptordnung ( *column-major order*):  $b_0, b_1, \dots, b_{15}$  genannt *State* ( *Zustand*):

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

# AES Verschlüsselungsprozess

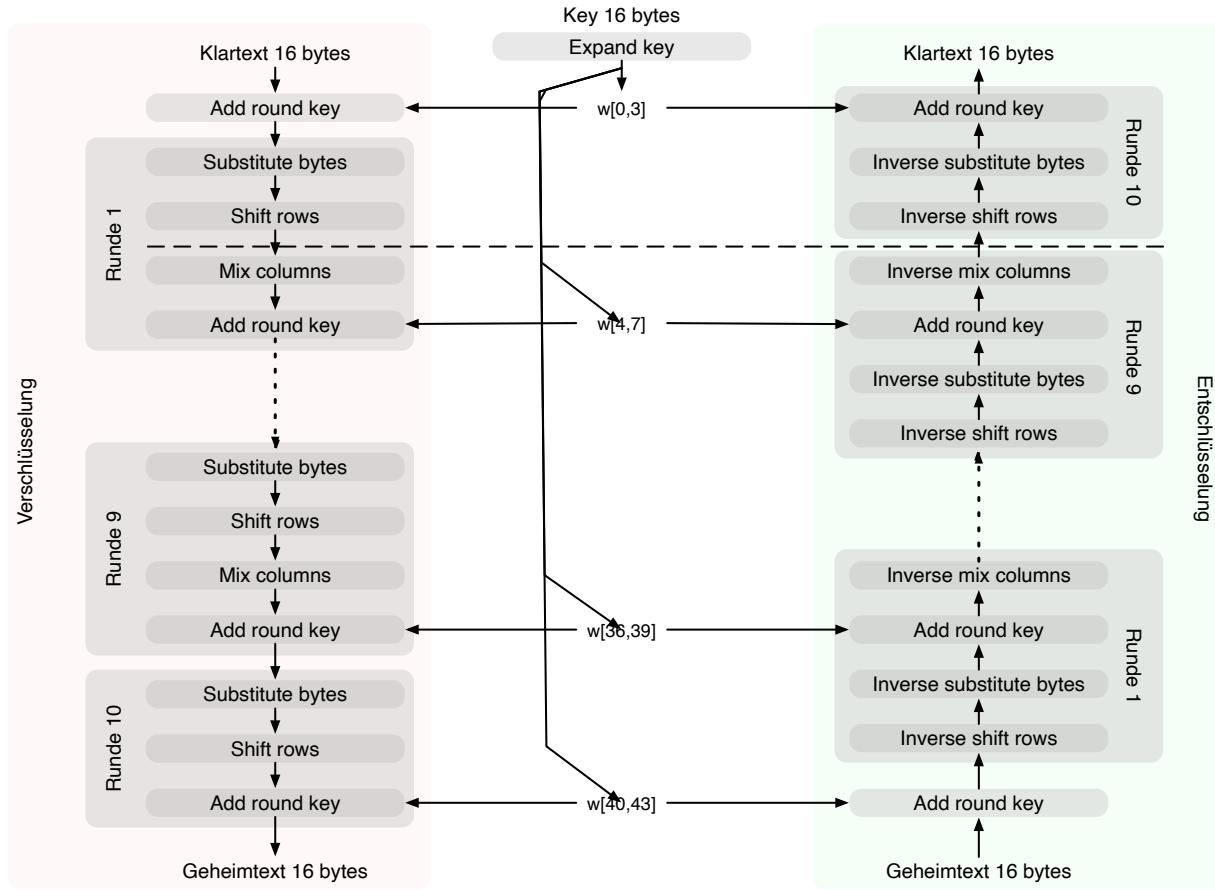


# AES Parameter

Schlüsselgröße (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Blockgröße ( <i>Block Size</i> ) (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Anzahl der Runden	10	12	14
Größe des Rundenschlüssels ( <i>RoundKeys</i> ) (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expandierte Schlüsselgröße (words/bytes)	44/176	52/208	60/240

# AES - Ver-/Entschlüsselungsprozess

(Key Size 128bits  $\Rightarrow$  10 Runden)



# AES Detaillierter Aufbau

- Verarbeitet in jeder Runde den gesamten Datenblock als eine einzige Matrix unter Verwendung von Substitutionen und Permutationen.
- Der als Eingabe bereitgestellte Schlüssel - bei 128 Bit Schlüsselgröße - wird in ein Array von vierundvierzig 32-Bit-Wörtern expandiert ( $w[i]$ )
- Die Chiffre beginnt und endet mit der AddRoundKey-Operation.
- Man kann sich die Chiffre als abwechselnde Operationen zwischen (a) der XOR-Verschlüsselung (AddRoundKey) eines Blocks vorstellen, gefolgt von (b) der Verwürfelung des Blocks (die anderen drei Stufen), gefolgt von der XOR-Verschlüsselung, und so weiter.
- Jede Stufe ist leicht umkehrbar.
- Der Entschlüsselungsalgorimus verwendet den expandierten Schlüssel in umgekehrter Reihenfolge, wobei der Entschlüsselungsalgorimus nicht mit dem Verschlüsselungsalgorimus identisch ist.
- Der Zustand (*State*) ist sowohl bei der Verschlüsselung als auch bei der Entschlüsselung derselbe.
- Die letzte Runde sowohl der Verschlüsselung als auch der Entschlüsselung besteht aus nur drei Stufen.

# AES verwendet vier verschiedene Stufen

## ***Substitute Bytes:***

verwendet eine S-Box, um eine byteweise Ersetzung des Blocks vorzunehmen

## ***ShiftRows:***

ist eine einfache Permutation

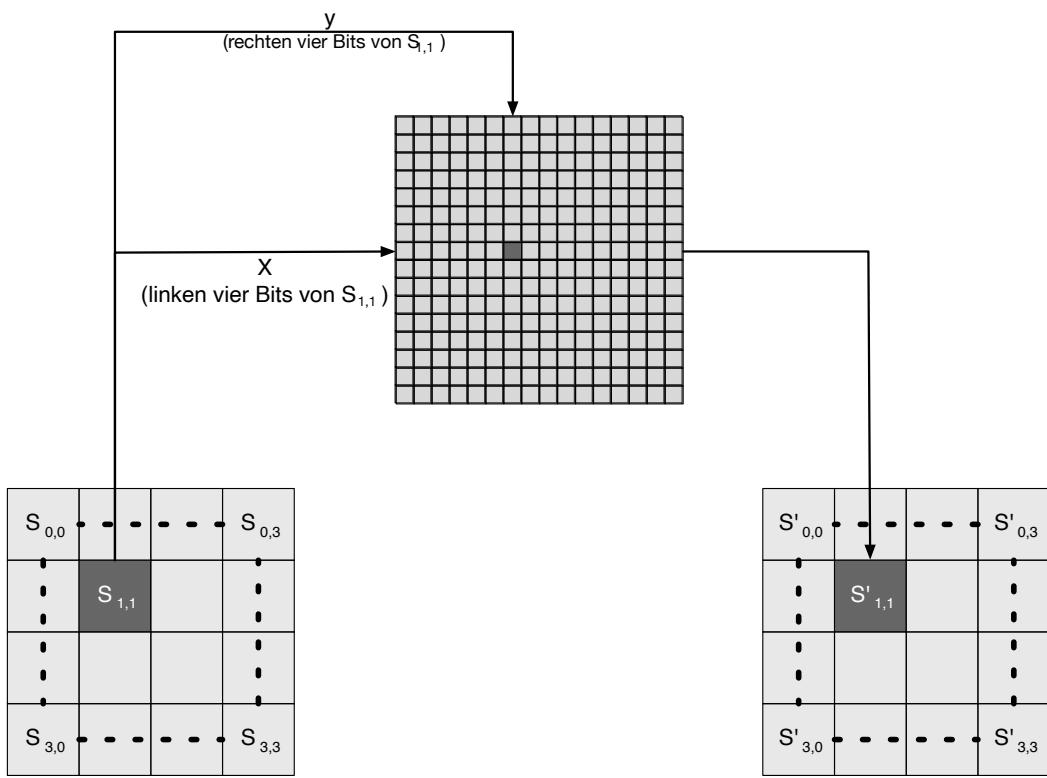
## ***MixColumns:***

ist eine Substitution, mit Hilfe von Polynomarithmetik über  $GF(2^8)$

## ***AddRoundKey:***

ist ein einfaches bitweises XOR des aktuellen Blocks mit einem Teil des expandierten Schlüssels

# AES Substitute Byte Transformation



# AES S-box

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

Jedes einzelne Byte des Zustands (*State*) wird auf folgende Weise auf ein neues Byte abgebildet: Die äußersten linken 4 Bits des Bytes werden als Zeilenwert und die äußersten rechten 4 Bits als Spaltenwert verwendet. Diese beiden Werte dienen als Indizes in der S-Box.

# AES Inverse S-box

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

## Beispiel

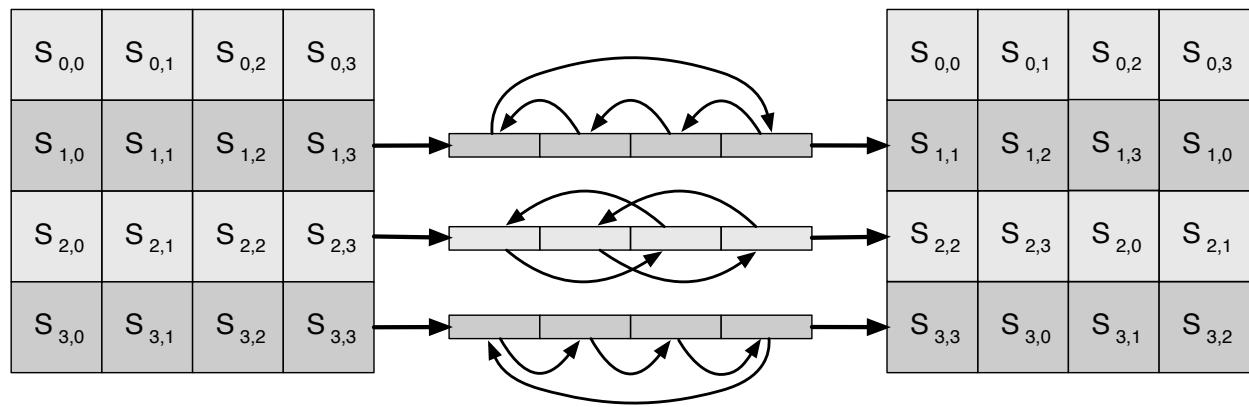
Der (Hex)Wert **0xA3** ( $x=A$  und  $y=3$ ) wird von der S-Box auf den (Hex)Wert **0x0A** abgebildet.

Die inverse S-Box bildet den Wert **0x0A** ( $x=0$  und  $y=A$ ) wieder auf den ursprünglichen Wert ab.

# S-Box Design Grundlagen

- Die S-Box ist so konzipiert, dass sie gegen bekannte kryptoanalytische Angriffe resistent ist.
- Die Rijndael-Entwickler suchten nach einem Design, das eine geringe Korrelation zwischen Eingabe- und Ausgabebits aufweist und die Eigenschaft hat, dass die Ausgabe keine lineare mathematische Funktion der Eingabe ist.
- Die Nichtlinearität ist auf die Verwendung der multiplikativen Inversen bei der Konstruktion der S-Box zurückzuführen.

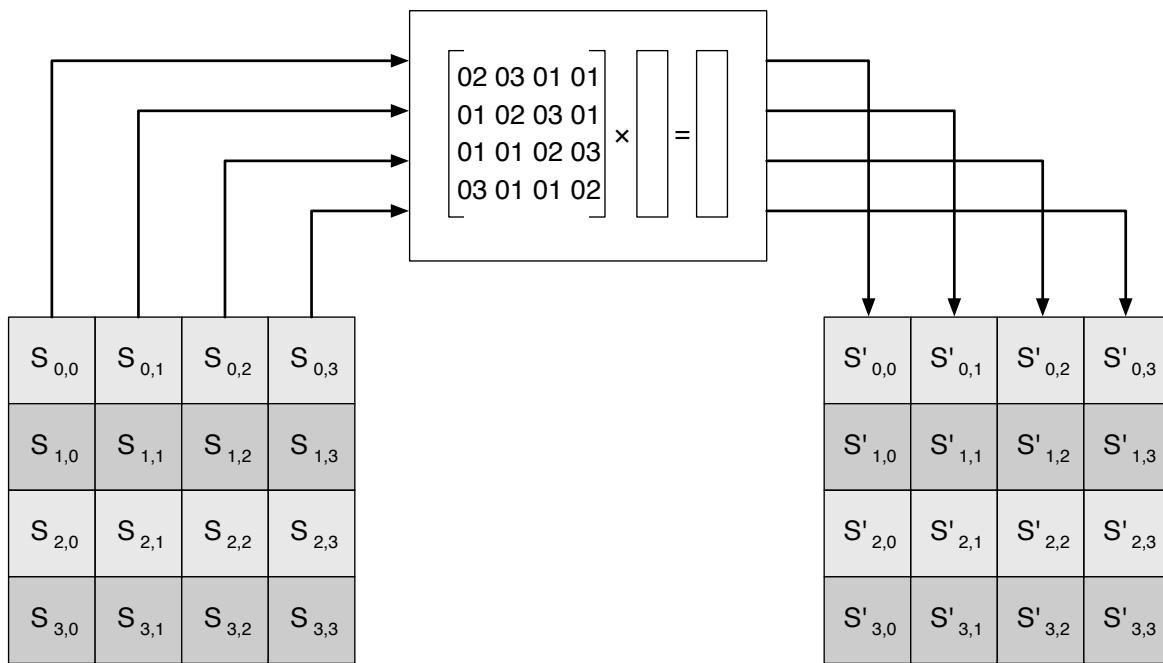
# Shift Row Transformation



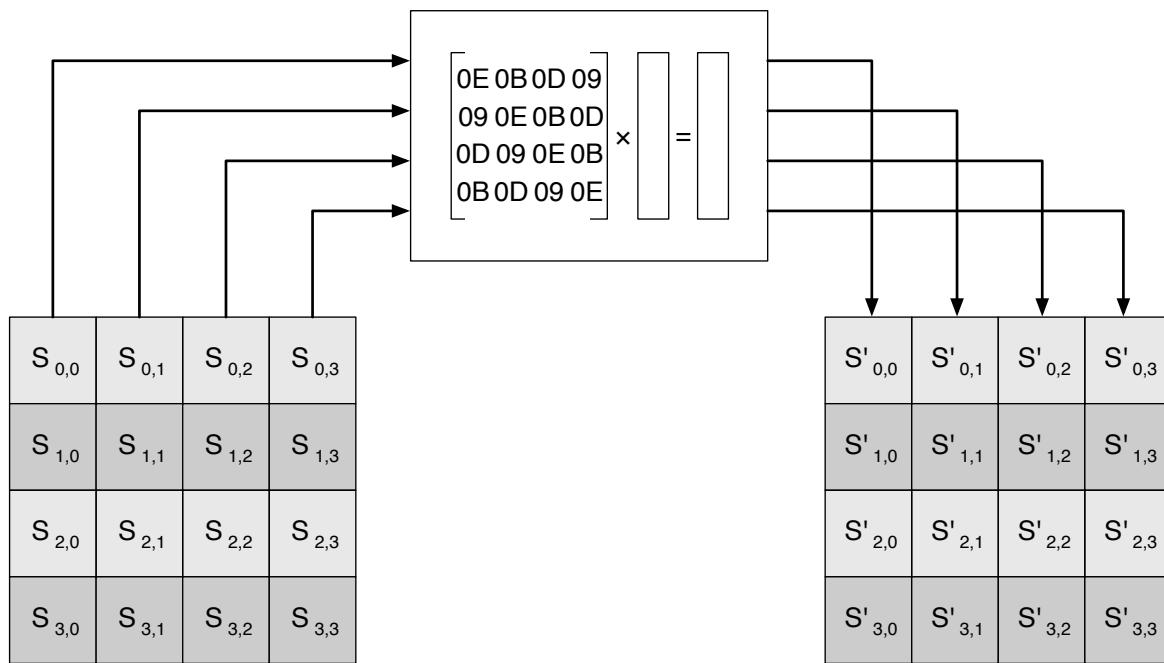
# **Shift Row Transformation - Begründung**

- Wesentlicher als es auf den ersten Blick scheint!
- Der Zustand (*State*) wird ebenso wie die Chiffrierein- und -ausgabe als Array aus vier 4-Byte-Spalten behandelt.
- Bei der Verschlüsselung werden die ersten 4 Bytes des Klartextes in die erste Spalte vom Zustands (*State*) kopiert, und so weiter.
- Der Rundenschlüssel wird spaltenweise auf den Zustand (*State*) angewendet.
- Bei einer Zeilenverschiebung wird also ein einzelnes Byte von einer Spalte in eine andere verschoben, was einem linearen Abstand von einem Vielfachen von 4 Byte entspricht.
- Die Transformation sorgt dafür, dass die 4 Bytes einer Spalte auf vier verschiedene Spalten verteilt werden.

# Mix Column Transformation



# Inverse Mix Column Transformation



# Mix Column Transformation - Beispiel

Gegeben

Ergebnis

Beispiel für die Berechnung von  $S'_{0,0}$ :

87	F2	4D	97	47	40	A3	4C
6E	4C	90	EC	37	D4	70	9F
46	E7	4A	C3	94	E4	3A	42
A6	8C	D8	95	ED	A5	A6	BC

$$S'_{0,0} = 02 \times S_{0,0} \oplus 03 \times S_{1,0} \oplus 01 \times S_{2,0} \oplus 01 \times S_{3,0}$$

$$(02 \times 87) \oplus (03 \times 6E) \oplus (46) \oplus (A6) = 47.$$

## Hilfsrechnungen

$$\begin{aligned} (02 \times 87) &= (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101) \\ (03 \times 6E) = 6E \oplus (02 \times 6E) &= (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010) \\ 46 &= (0100\ 0110) \\ A6 &= (1010\ 0110) \\ &\hline (0100\ 0111) \end{aligned}$$

## Warnung

$(03 \times 6E) = 6E \oplus (02 \times 6E)$  und **ist nicht**  $6E \oplus 6E \oplus 6E$ , da wir hier Polynomarithmetik in  $GF(2^8)$  nutzen und **03** dem Polynom:  $x + 1$  entspricht.

## **Mix Column Transformation - Begründung**

- Die Koeffizienten einer Matrix, die auf einem linearen Code mit maximalem Abstand zwischen den Codewörtern basiert, gewährleisten eine gute Mischung zwischen den Bytes jeder Spalte.
- Die *Mix Column Transformation* (~ *Vermischung der Spalten*) - kombiniert mit der *Shift Row Transformation* ( *Zeilenverschiebung*) - stellt sicher, dass nach einigen Runden alle Ausgangsbits von allen Eingangsbits abhängen.

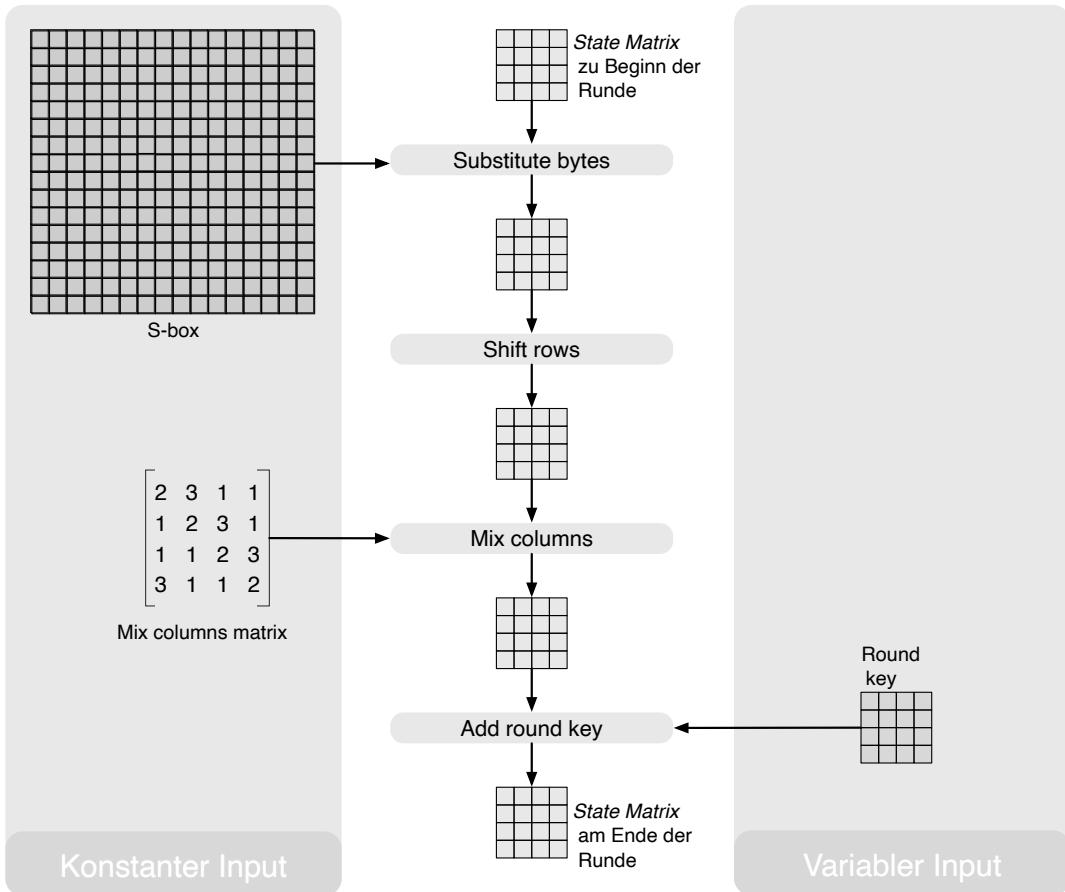
# AddRoundKey Transformation

- Die 128 Bits des Zustands (*State*) werden bitweise mit den 128 Bits des Rundenschlüssels XOR-verknüpft.
- Die Operation wird als spaltenweise Operation zwischen den 4 Bytes einer Spalte des Zustands (*State*) und einem Wort des runden Schlüssels betrachtet.
- *Kann auch als eine Operation auf Byte-Ebene betrachtet werden.*

## Designbegründung

- Sie ist so einfach wie möglich und betrifft jedes Bit des Staates.
- Die Komplexität der runden Schlüsselexpansion plus die Komplexität der anderen Stufen von AES sorgen für Sicherheit!

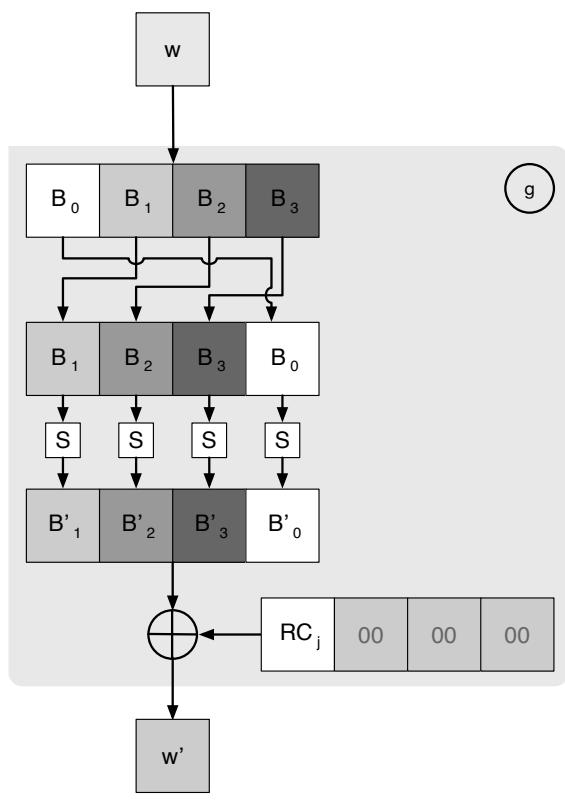
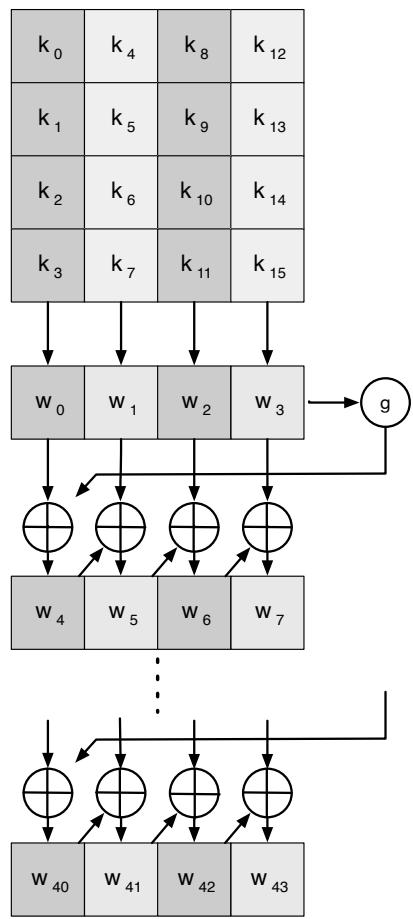
## Eingabe für eine einzelne AES-Verschlüsselungsrounde



# AES Schlüsselexpansion

- Nimmt als Eingabe einen (hier: 128-Bit) Schlüssel mit vier Wörtern (16 Byte) und erzeugt ein lineares Array mit 44 Wörtern (176 Byte).
- Dies liefert einen vier Worte umfassenden Rundenschlüssel für die initiale *AddRoundKey*-Stufe sowie für jede der folgenden 10 Runden der Chiffre.
- Der Schlüssel wird in die ersten vier Wörter des erweiterten Schlüssels kopiert.
- Der Rest des expandierten Schlüssels wird in Blöcken von jeweils vier Wörtern aufgefüllt.
- Jedes hinzugefügte Wort  $w[i]$  hängt vom unmittelbar vorangehenden Wort  $w[i - 1]$  und dem vier Positionen zurückliegenden Wort  $w[i - 4]$  ab.
- In drei von vier Fällen wird ein einfaches XOR verwendet.
- Für ein Wort dessen Position im Array  $w$  ein Vielfaches von 4 ist, wird die komplexere Funktion  $g$  angewandt.

# AES Schlüsselexpansion - Visualisiert



# AES Round Key Berechnung

$$\begin{aligned} r_i &= (r_{c_i}, 00, 00, 00) \\ r_{c_1} &= 01 \\ r_{c_{i+1}} &= \text{xtime}(r_{c_i}) \end{aligned}$$

## xtime Function

$$y_7y_6y_5y_5y_4y_3y_2y_1y_0 = \text{xtime}(x_7x_6x_5x_5x_4x_3x_2x_1x_0) \quad (x_i, y_i \in \{0, 1\})$$

$$y_7y_6y_5y_5y_4y_3y_2y_1y_0 = \begin{cases} x_6x_5x_5x_4x_3x_2x_1x_00, & \text{if } x_7 = 0 \\ x_6x_5x_5x_4x_3x_2x_1x_00 \oplus 00011011, & \text{if } x_7 = 1 \end{cases}$$

## Die Round Key Werte sind:

$$r_{c_1} = 01, r_{c_2} = 02, r_{c_3} = 04, r_{c_4} = 08, r_{c_5} = 10$$

$$r_{c_6} = 20, r_{c_7} = 40, r_{c_8} = 80, r_{c_9} = 1B = 00011011, r_{c_{10}} = 36$$

Die *xtime* Funktion ist eine Multiplikation im endlichen Körper  $GF(2^8)$  und ist die Polynommultiplikation mit dem Polynom  $x$ .

# AES Schlüsselexpansion - Beispiel (Runde 1)

Gegeben:

$$w[0] = (54, 68, 61, 74)$$

$$w[1] = (73, 20, 6D, 79)$$

$$w[2] = (20, 4B, 75, 6E)$$

$$w[3] = (67, 20, 46, 75)$$

■  $g(w[3])$ :

- zirkuläre Linksverschiebung von  $w[3]$ :  $(20, 46, 75, 67)$
- Bytesubstitution mit Hilfe der s-box:  $(B7, 5A, 9D, 85)$
- Addition der Rundenkonstante  $(01, 00, 00, 00) \Rightarrow g(w[3]) = (B6, 5A, 9D, 85)$
- $w[4] = w[0] \oplus g(w[3]) = (E2, 32, FC, F1)$
- $w[5] = w[4] \oplus w[1] = (91, 12, 91, 88)$
- $w[6] = w[5] \oplus w[2] = (B1, 59, E4, E6)$
- $w[7] = w[6] \oplus w[3] = (D6, 79, A2, 93)$
- Der erste Rundenschlüssel ist:  $w[4] \quad || \quad w[5] \quad || \quad w[6] \quad || \quad w[7]$

# AES Schlüsselexpansion - Begründung

- Die Rijndael-Entwickler haben den Expansionsschlüssel-Algorithmus so konzipiert, dass er gegen bekannte kryptoanalytische Angriffe resistent ist.
- Die Einbeziehung einer rundenabhängigen Rundenkonstante beseitigt die Symmetrie, die sonst bei der Erzeugung der Rundenschlüssel in den verschiedenen Runden entstehen würde.

Designziele:

- Kenntnis eines Teils des Chiffrierschlüssels oder des Rundenschlüssels ermöglicht nicht die Berechnung vieler anderer Bits des Rundenschlüssels
- Eine invertierbare Transformation
- Performance auf einer breiten Palette von CPUs
- Verwendung von Rundenkonstanten zur Beseitigung von Symmetrien
- Diffusion der Chiffrierschlüsselunterschiede in die Rundenschlüssel
- Ausreichende Nichtlinearität, um die vollständige Bestimmung von Rundenschlüsselunterschieden nur aus Chiffrierschlüsselunterschieden zu verhindern
- Einfachheit der Beschreibung

## Lawineneffekt in AES: Änderung im Klartext

Round		# unterschiedlicher Bits
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d6890175070485e62	59
4	f867aeee8b437a5210c24c1974cfffeabc 43efd697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36ald891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

## Lawineneffekt in AES:

### Änderung im Schlüssel

Runde	# unterschiedlicher Bits
	0
0	1
1	22
2	58
3	67
4	63
5	81
6	70
7	74
8	67
9	59
10	53

# Äquivalente inverse Chiffre

AES-Entschlüsselung ist nicht identisch mit der Verschlüsselung.

- Die Abfolge der Umwandlungen ist unterschiedlich, obwohl die Schlüsselableitung die gleiche ist.
- Dies hat den Nachteil, dass für Anwendungen, die sowohl Verschlüsselung als auch Entschlüsselung erfordern, zwei separate Software- oder Firmware-Module benötigt werden.

Zwei unabhängige, separate Änderungen sind erforderlich, um die Entschlüsselungsstruktur mit der Verschlüsselungsstruktur in Einklang zu bringen:

1. Die ersten beiden Stufen der Entschlüsselungsrounde müssen vertauscht werden.
2. Die zweiten beiden Stufen der Entschlüsselungsrounde müssen vertauscht werden.

# Vertausch von *InvShiftRows* und *InvSubBytes*

## ***InvShiftRows:***

beeinflusst die Reihenfolge der Bytes im Zustand (*State*), ändert aber nicht den Inhalt der Bytes und ist nicht vom Inhalt der Bytes abhängig, um seine Transformation durchzuführen.

## ***InvSubBytes:***

beeinflusst den Inhalt von Bytes im Zustand (*State*), ändert aber nicht die Byte-Reihenfolge und hängt nicht von der Byte-Reihenfolge ab, um seine Transformation durchzuführen.

Diese beiden Operationen sind kommutativ und soweit vertauschbar.

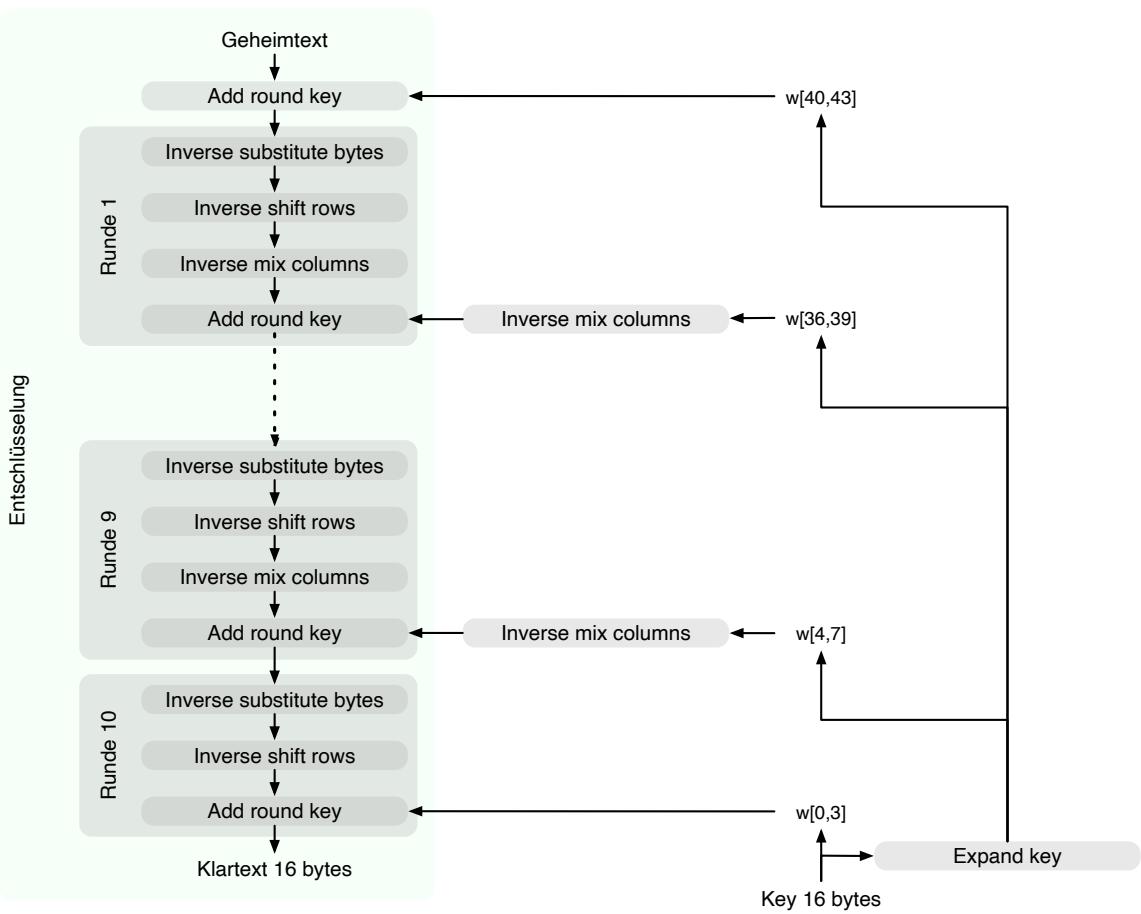
## Vertausch von *AddRoundKey* und *InvMixColumns*

- Die Transformationen *AddRoundKey* und *InvMixColumns* ändern die Reihenfolge der Bytes im Zustand (*State*) nicht.
- Betrachtet man den Schlüssel als eine Folge von Wörtern, so wirken sowohl *AddRoundKey* als auch *InvMixColumns* jeweils nur auf eine Spalte des Zustands (*State*).
- Diese beiden Operationen sind linear in Bezug auf die gegebene Spalte.

Das heißt, für einen bestimmten Zustand  $S_i$  und einen bestimmten Rundenschlüssel  $w_j$ :

$$InvMixColumns(S_i \oplus w_j) = InvMixColumns(S_i) \oplus InvMixColumns(w_j)$$

# Äquivalente Inverse Chiffre



# Aspekte der Umsetzung auf 8-bit Prozessoren

AES kann sehr effizient auf einem 8-Bit-Prozessor implementiert werden:

## AddRoundKey:

ist eine byteweise XOR-Operation.

## ShiftRows:

ist eine einfache Byte-Verschiebeoperation.

## SubBytes:

arbeitet auf Byte-Ebene und benötigt nur eine Tabelle von 256 Bytes.

## MixColumns:

erfordert eine Matrixmultiplikation im Körper  $GF(2^8)$ , was bedeutet, dass alle Operationen mit Bytes durchgeführt werden.

## **Aspekte der Umsetzung auf 32-bit Prozessoren**

AES kann effizient auf einem 32-Bit-Prozessor implementiert werden:

- Die einzelnen Schritte können so umdefiniert werden, dass sie 32-Bit-Wörter verwenden.
  - Es ist möglich 4 Tabellen für die *MixColumns* Transformation mit je 256 Wörtern vorzuberechnen.
    - Dann kann jede Spalte in jeder Runde mit 4 Tabellen-Lookups + 4 XORs berechnet werden.
    - Die Kosten für die Speicherung der Tabellen belaufen sich auf „4Kb“.
  - Die Entwickler glauben, dass die Möglichkeit einer effizienten Implementierung ein Schlüsselfaktor für die Wahl der AES-Chiffre zum neuen Standard war.

# Übung (AES-128) - Berechnung des *RoundKey*

Sei der folgende *RoundKey* gegeben:

$$rk_1 = w[4] \parallel w[5] \parallel w[6] \parallel w[7] =$$

$$-w[4]----- \quad -w[5]----- \quad -w[6]----- \quad -w[7]-----$$

$$\begin{array}{ccccccccc} E2 & 32 & FC & F1 & 91 & 12 & 91 & 88 & B1 & 59 & E4 & E6 & D6 & 79 & A2 & 93 \end{array}$$

In Hinblick auf die Berechnung von  $rk_2$ ; d. h. den Rundschlüssel (*Roundkey*) für die zweite Runde, führe folgende Schritte durch.

## Formeln für die Berechnung des *RoundKey*

Bevor Sie die konkrete Berechnung durchführen, schreiben Sie zunächst die Formeln für:  $w[8] = \dots \oplus \dots$     $w[9] = \dots \oplus \dots$     $w[10] = \dots \oplus \dots$     $w[11] = \dots \oplus \dots$  auf.

**Berechne  $w[8]$  und  $w[9]$ .**

## Formeln für die Berechnung des *RoundKey*

Bevor Sie die konkrete Berechnung durchführen, schreiben Sie zunächst die Formeln für:

$w[8] = \dots \oplus \dots$      $w[9] = \dots \oplus \dots$      $w[10] = \dots \oplus \dots$      $w[11] = \dots \oplus \dots$  auf.

**Berechne  $w[8]$  und  $w[9]$ .**

# Übung (AES-128)

Nehmen wir an, dass der Zustand (*State*) folgendermaßen sei:

00 3C 6E 47

1F 4E 22 74

0E 08 1B 31

54 59 0B 1A

Führen Sie den ***Substitute Bytes*** Schritt durch (Anwendung der S-box Transformation).

Führen Sie die ***Shift Rows Transformation*** auf dem Ergebnis des vorherigen Schrittes durch.

Führen Sie den ***Substitute Bytes*** Schritt durch (Anwendung der S-box Transformation).

Führen Sie die *Shift Rows Transformation* auf dem Ergebnis des vorherigen Schrittes durch.

## Mix Columns Transformation

Nehmen wir an, dass der Zustand (*State*) folgendermaßen sei:

6A	59	CB	BD
4E	48	12	A0
98	9E	30	9B
8B	3D	F4	9B

Führen Sie die *Mix Columns Transformation* durch für das fehlende Feld ( $S'_{0,0}$ ):

??	C9	7F	9D
CE	4D	4B	C2
89	71	BE	88
65	47	97	CD

## Mix Columns Transformation

Nehmen wir an, dass der Zustand (*State*) folgendermaßen sei:

6A	59	CB	BD
4E	48	12	A0
98	9E	30	9B
8B	3D	F4	9B

Führen Sie die *Mix Columns Transformation* durch für das fehlende Feld ( $S'_{0,0}$ ):

??	C9	7F	9D
CE	4D	4B	C2
89	71	BE	88
65	47	97	CD

## RoundKey Anwendung

Wenden Sie den folgenden *RoundKey*:

$-w[x]-----$      $-w[x+1]-----$      $-w[x+2]-----$      $-w[x+3]-----$

D2 60 0D E7    15 7A BC 68    63 39 E9 01    C3 03 1E FB

auf die folgende Zustandsmatrix (*State*):

AA 65 FA 88

16 0C 05 3A

3D C1 DE 2A

B3 4B 5A 0A

## RoundKey Anwendung

Wenden Sie den folgenden *RoundKey*:

$w[x]$  -----  $w[x+1]$  -----  $w[x+2]$  -----  $w[x+3]$  -----  
D2 60 0D E7 15 7A BC 68 63 39 E9 01 C3 03 1E FB

auf die folgende Zustandsmatrix (*State*):

AA 65 FA 88  
16 0C 05 3A  
3D C1 DE 2A  
B3 4B 5A 0A

# Übung (AES-128)

## Nachgehakt

Fragen Sie sich, was passiert, wenn Sie einen Block, der nur aus **0x00** Werten besteht, mit einem Schlüssel verschlüsseln, der ebenfalls nur aus **0x00** Werten besteht?

## Nachgehakt

Fragen Sie sich, was passiert, wenn Sie einen Block, der nur aus **0x00** Werten besteht, mit einem Schlüssel verschlüsseln, der ebenfalls nur aus **0x00** Werten besteht?