CRYPTO2 – Übersicht: VL03+04: AES/Rijndael / „Was macht AES?!"

Rijndael:
- Blocklänge: $128 \le \lambda_b \le 256 \wedge 32 \mid \lambda_b$
- Schlüssellänge: $128 \le \lambda_k \le 256 \wedge 32 \mid \lambda_k$

$\Rightarrow \lambda_b, \lambda_k \in \{128, 160, 192, 224, 256\}$

AES:
- Blocklänge: $\lambda_b = 128$ (bit)
- Schlüssellänge: $\lambda_k \in \{128, 192, 256\}$ (bit) (in VL nur: $\lambda_k = 128$)
- Rundenanzahl: je nach Schlüssellänge: 10 Runden bei AES-128

Runden 1-9: MixColumns ∘ ShiftRows ∘ SubBytes
Runde 10: ShiftRows ∘ SubBytes („Schlussrunde")

Enc: ↓ Dec: ↑ (I)SB & (I)SR kommutieren: ↑ $IMC \circ (+k_i) = (+IMC(k_i)) \circ IMC$: ↑

	Enc: ↓	Dec: ↑	(I)SB & (I)SR kommutieren: ↑
	$+k_0$ ↓	$+k_0$	$+k_0$
Verschlüsselungsrunde #1	SB	ISB	ISR
	SR	ISR	ISB
	MC	IMC	IMC
	$+k_1$	$+k_1$	$+k_1$
Verschlüsselungsrunde #2	SB	ISB	ISR
	SR	ISR	ISB
	MC	IMC	IMC
	$+k_2$	$+k_2$	$+k_2$
	...		
Schlussrunde (Runde #10)	SB	ISB	ISR
	SR	ISR	ISB
	$+k_{10}$ ↓	$+k_{10}$	$+k_{10}$

$IMC \circ (+k_i) = (+IMC(k_i)) \circ IMC$: ↑

↑ $+k_0$
ISR
ISB
$+IMC(k_1)$
IMC
ISR
ISB
$+IMC(k_2)$
IMC
ISR
ISB
↑ $+k_{10}$

Decrypt == Encrypt, nur mit 2 Abänderungen
(1) nehme inverse Funktionen
(2) $IMC(k_i)$ bei Schl.abl. „schalten"

⇒ Diese Tatsache hat den Freundeskreis von Rijndael enorm erweitert! (da so ungl. schön gemacht)

(k_i = i-ter Rundenschlüssel; erhalte aus der Schlüsselexpansion)

nicht-linear: **SB/SubBytes:** = S-Box/monoalph. Verschl. byte-/komponentenweise (vgl. en.wiki/Rijndael_S-box)

S-Box:
$\mathbb{F}_{256} \to \mathbb{F}_{256}: b \mapsto 1F_{16} \cdot b^{254} \oplus 63_{16}$
(Plus Polynom = XOR Bits; b^{254}: invertieren im $\mathbb{F}_{256}$!; $\cdot$: Polynom-Multiplikation im TPR!)

wobei: $b^{254} = \begin{cases} 0 & b=0 \\ 1/b & \text{sonst.} \end{cases}$

wobei: $1F_{16} \cdot x = x \oplus (x \ll 1) \oplus (x \ll 2) \oplus (x \ll 3) \oplus (x \ll 4)$ (= Multiplikation im TPR!)

Status vorher / Status nachher:

	0x00	0x01	0x00	0x01		0x63	0x7C	0x63	0x7C
SB →	0x02	0x03	0x02	0x03		0x77	0x7B	0x77	0x7B
← ISB	0x00	0x01	0x00	0x01		0x63	0x7C	0x63	0x7C
	0x02	0x03	0x02	0x03		0x77	0x7B	0x77	0x7B

linear: **SR/ShiftRows:** (= Diffusion)

Verschiebe Zeilen um eine best. Anzahl von Spalten nach links. Überlaufende Zellen werden von rechts fortgesetzt. Die Anzahl der Verschiebungen ist theor. abh. v. Zeilen- & Blocklänge, AES hat aber $\lambda_b = 128 = \text{const.}$ (AES)

← 0	0x00	0x01	0x02	0x03		0x00	0x01	0x02	0x03
← 1	0x04	0x05	0x06	0x07	SR →	0x05	0x06	0x07	0x04
← 2	0x08	0x09	0x0A	0x0B	← ISR	0x0A	0x0B	0x08	0x09
← 3	0x0C	0x0D	0x0E	0x0F		0x0F	0x0C	0x0D	0x0E

linear: **MC/MixColumns:** (en.wiki/Rijndael_MixColumns)

Vermische die Daten innerh. der Spalten.

$$MC: \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \in GF(2^8)^4$$

(↳ sind Polynome! ↳ rechne mod R(t)!)

	0x01	0xDB	0xDB	0xF2		0x01	0x8E	0x8E	0x9F
MC →	0x01	0x13	0x13	0x0A		0x01	0x4D	0x4D	0xDC
← IMC	0x01	0x53	0x53	0x22		0x01	0xA1	0xA1	0x58
	0x01	0x45	0x45	0x5C		0x01	0xBC	0xBC	0x9D

Rechenbeispiele für SubBytes & MixColumns:

- **SB/SubBytes:** Bsp.: 0x03 $\overset{SB}{\mapsto}$ 0x7B

  - **Mit Sage:** → oder alternativ meine .py

```
var('t')
F256 = PolynomialRing(GF(2),t].quotient_ring(t**8+t**4+t**3+t+1)
TPR = ZZ['x'].quotient(x**8 + 1)
x1F = list(reversed([0, 0, 0, 1, 1, 1, 1, 1]))   # 1F_16
x63 = list(reversed([0, 1, 1, 0, 0, 0, 1, 1]))   # 63_16
inpt = list(reversed([0, 0, 0, 0, 0, 0, 1, 1]))  # 03_16 im Bsp.
F256((  TPR(x1F) * TPR((F256(inpt)**254).list()) + TPR(x63)).lift())
```

Print: ⇒ tbar^6 + tbar^5 + tbar^4 + tbar^3 + tbar + 1

Interpr. als: → [0, 1, 1, 1, 1, 0, 1, 1] ∇ Achtung: Sage nutzt LE beim Erstellen (s.o.) & BE beim Printen ∇ (↑ = Big-Endian)

→ 0x7B ✓

  - **Mit Hand:**

Invertiere zunächst 03_{16} im F256; mithilfe des Erweiterten Euklidischen Algo. auf $1+t$ (03_{16}) und dem Rijndael-Polynom $t^8+t^4+t^3+t+1$:

(! Polynomdivision ! (s.u.) → Pfeil zu $[t^7+t^6+t^5+t^4+t^2+t]$)

Kontrolle: F256(1+t)**254

nutze xgcd_for_polynomials.py

$$t+1 = 0\cdot[t^8+t^4+t^3+t+1]+[t+1]$$
$$t^8+t^4+t^3+t+1 = [t^7+t^6+t^5+t^4+t^2+t]\cdot[t+1]+1$$
$$t+1 = [t+1]\cdot 1+0$$

$$t+1 = 1\cdot[t+1]+0\cdot[t^8+t^4+t^3+t+1]$$
$$1 = [t^7+t^6+t^5+t^4+t^2+t]\cdot[t+1]+\ldots$$
$\checkmark$ (↑ dies ist das Inverse von $(t+1)$)

D.h.: $(1+t)^{-1}$ im $\mathbb{F}_{256}$ ist $\boxed{t^7+t^6+t^5+t^4+t^2+t}$

Rechne nun im TPR:

(fix: $1F_{16}$ →)
$$(t^4+t^3+t^2+t+1)\cdot(t^7+t^6+t^5+t^4+t^2+t)$$
$$=(t^{11}+t^{10}+t^9+t^8+t^6+t^5)+(t^{10}+t^9+t^8+t^7+t^5+t^4)+(t^9+t^8+t^7+t^6+t^4+t^3)+(t^8+t^7+t^6+t^5+t^3+t^2)+(t^7+t^6+t^5+t^4+t^2+t)$$
$$=t^{11}+t^9+t^4+t$$
$$=t^3+t+t^4+t = t^4+t^3 \quad \text{im TPR ist } t^8=1!$$

Addiere nun im TPR $63_{16} = 1+t+t^5+t^6$: $\boxed{t^6+t^5+t^4+t^3+t+1}$ ∎ $=$ 0x7B ✓

- **MC/MixColumns:**

  - **Mit Sage:** → oder alternativ meine .py

```
var('t')
F256 = PolynomialRing(GF(2), t).quotient_ring(t**8+t**4+t**3+t+1)
x01 = F256(1)
x02 = F256(t)
x03 = F256(t+1)
mix_col = Matrix(F256,4,4,[[x02,x03,x01,x01],
                           [x01,x02,x03,x01],
                           [x01,x01,x02,x03],
                           [x03,x01,x01,x02]])
xDB = F256(list(reversed([1,1,0,1, 1,0,1,1])))
x13 = F256(list(reversed([0,0,0,1, 0,0,1,1])))
x53 = F256(list(reversed([0,1,0,1, 0,0,1,1])))
x45 = F256(list(reversed([0,1,0,0, 0,1,0,1])))
mix_col *vector([xDB, x13, x53, x45])
```

⇒ (tbar^7 + tbar^3 + tbar^2 + tbar, tbar^6 + tbar^3 + ...

→ [0x8E, 0x4D, 0xA1, 0xBC]

Polynomdivision in Sage:

```
S.<t> = PolynomialRing(GF(2),'t')
(t**8 + t**4 + t**3 + t + 1)
// (t+1)
```

⇒ t^7 + t^6 + t^5 + t^4 + t^2 + t

```
  100011011 ÷ 11 = 1111 0110 (t^7+t^6+t^5+t^4...)
⊕ 11
   10011011
⊕  11
    1011011...
```

unten bleibt 1 als Rest

- **Mit Hand:**

$$\begin{pmatrix} t & t+1 & 1 & 1 \\ 1 & t & t+1 & 1 \\ 1 & 1 & t & t+1 \\ t+1 & 1 & 1 & t \end{pmatrix}\cdot\underbrace{\begin{pmatrix} t^7+t^6+t^4+t^3+t+1 \\ t^4+t+1 \\ t^6+t^4+t+1 \\ t^6+t^2+1 \end{pmatrix}}_{\text{input vector}} = \begin{pmatrix} [t^8+t^7+t^5+t^4+t^2+t]+[t^8+t^7+t^5+\ldots \\ \ldots \end{pmatrix}$$

! Multiplikation von Elementen im $\mathbb{F}_{256}$ erfordert anschließendes Rechnen mod Rijndael-Polynom $R(t)$ via Polynomdivision; hat man Glück gibt's aber kein t^8 oder größer!