

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

Rijndael:

- Blocklänge: $128 \leq \lambda_b \leq 256 \wedge 32 | \lambda_b \Rightarrow \lambda_b, \lambda_k \in \{128, 160, 192, 224, 256\}$
- Schlüssel-Länge: $128 \leq \lambda_k \leq 256 \wedge 32 | \lambda_k$

AES:

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

Runden 1-9: Mix Columns \circ Shift Rows \circ SubBytes
 Runde 10: Shift Rows \circ SubBytes ("Schlussrunde")

Enc: \downarrow Dec: \uparrow (ISB & ISR) kommutieren: \uparrow $\text{IMC} \circ (+k_i) = (+\text{IMC}(k_i)) \circ \text{IMC}: \uparrow$

	$+k_0$ \downarrow	$+k_0$	$+k_0$	\uparrow $+k_0$	Decrypt = Encrypt, nur mit 2 Abänderungen (1) nehme inverse Funktionen (2) $\text{IMC}(k_i)$ bei Schl.abbl. <i>"Schalter"</i> \Rightarrow Diese Tatsache hat den Freundeskreis von Rijndael enorm erweitert! (da so ungl. schön gemacht)
Verschlü- sselungs- runde #1	SB	ISB	ISR	ISR	
	SR	ISR	ISB	ISB	
	MC	IMC	IMC	$+\text{IMC}(k_1)$	
	$+k_1$	$+k_1$	$+k_1$	IMC	
Verschlü- sselungs- runde #2	SB	ISB	ISR	ISR	
	SR	ISR	ISB	ISB	
	MC	IMC	IMC	$+\text{IMC}(k_2)$	
	$+k_2$	$+k_2$	$+k_2$	IMC	
Schlussrunde (Runde #10)	SB	ISB	ISR	ISR	
	SR	ISR	ISB	ISB	
	$+k_{10}$ \downarrow	$+k_{10}$	$+k_{10}$	\uparrow $+k_{10}$	

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

nicht-linear: SB/SubBytes: = S-Box/monoalph. Verschl.
 (vgl. en. wiki)
 S-Box:
 $\mathbb{F}_{256} \rightarrow \mathbb{F}_{256}: b \mapsto 1\mathbb{F}_{16} \cdot b^{254} \oplus 63_{16}$
 (plus Polynom = XOR Bits)
 wobei: $1\mathbb{F}_{16} \cdot x = x \oplus (x \ll 1) \oplus (x \ll 2) \oplus (x \ll 3) \oplus (x \ll 4)$
 wobei: $b^{254} = \begin{cases} 0 & b=0 \\ 1 & \text{sonst} \end{cases}$
 = Multiplikation im TPR!

Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:	Status vorher:
0x00	0x01	0x00	0x01	0x63	0x7C	0x63	0x7C	0x63	0x7C
0x02	0x03	0x02	0x03	0x77	0x7B	0x77	0x7B	0x77	0x7B
0x00	0x01	0x00	0x01	0x63	0x7C	0x63	0x7C	0x63	0x7C
0x02	0x03	0x02	0x03	0x77	0x7B	0x77	0x7B	0x77	0x7B

linear: SR/ShiftRows:
 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 $\lambda_b = 128$
 aber = const.

0x00	0x01	0x02	0x03	0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07	0x05	0x06	0x07	0x04
0x08	0x09	0x0A	0x0B	0x0A	0x0B	0x08	0x09
0x0C	0x0D	0x0E	0x0F	0x0F	0x0E	0x0D	0x0C

linear: MC/MixColumns: (en. wiki/Rijndael-MixColumns)

Vermische die Daten innerhalb der Spalten.
 MC: $\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \mapsto \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$ EGF(2)⁸
 = rechte mod R(4)
 sind Polynome!

0x01	0x0B	0x0B	0x0F	0x01	0x8E	0x8E	0x9F
0x01	0x13	0x13	0x0A	0x01	0x4D	0x4D	0x3C
0x01	0x53	0x53	0x22	0x01	0xA1	0xA1	0x58
0x01	0x45	0x45	0x5C	0x01	0xBC	0xBC	0x9D

Rechenbeispiele für SubBytes & MixColumns:

- SB/SubBytes: Bsp.: $0x03 \xrightarrow{SB} 0x7B$

- Mit Sage: \rightarrow 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])) # 1F16
x63 = list(reversed([0, 1, 1, 0, 0, 0, 1, 1])) # 6316
inpt = list(reversed([0, 0, 0, 0, 0, 0, 1, 1])) # 0316 im Bsp.
F256((TPR(x1F) * TPR((F256(inpt)**254).list())) + TPR(x63)).list())
    
```

Print: $\Rightarrow tbar^{16} + tbar^{15} + tbar^{14} + tbar^{13} + tbar + 1$
 Interpret: $\Rightarrow [0, 1, 1, 1, 1, 0, 1, 1]$ ∇ Achtung: Sage nutzt LE beim V
 als: $\Rightarrow 0x7B$ \checkmark \rightarrow Big-Endian 0 Erstellen (s.o.) & BE beim Printen

- Mit Hand:

Polynomdivision
(s.u.)

Kontrolle:
F256(1+t)
**254

nutz xgcd
- for poly
nomial spy

Invertiere zunächst 03_{16} im F_{256} ; mithilfe des Erweiterten
 Euklidischen Algo. auf $1+t$ (03_{16}) und dem Rijndael-
 Polynom $t^8 + t^4 + t^3 + t + 1$:

$t+1 = 0 \cdot [t^8 + t^4 + t^3 + t + 1] + [t+1]$ $| t+1 = 1 \cdot [t+1] + 0 \cdot [t^8 + t^4 + t^3 + t + 1]$
 $t^8 + t^4 + t^3 + t + 1 = [t^7 + t^6 + t^5 + t^4 + t^2 + t] \cdot [t+1] + 1$ $| 1 = [t^7 + t^6 + t^5 + t^4 + t^2 + t] \cdot [t+1] + \dots$
 $t+1 = [t+1] \cdot 1 + 0$ $| \therefore$ Dies ist das Inverse von $(t+1)$

D.h.: $(1+t)^{-1}$ im F_{256} ist $t^7 + t^6 + t^5 + t^4 + t^2 + t$

fix: 1F₁₆

Rechne nun im TPR:

$(t^4 + t^3 + t^2 + t + 1) \cdot (t^7 + t^6 + t^5 + t^4 + t^2 + t)$
 $= (t^{11} + t^{10} + t^9 + t^8 + t^6 + t^5) + (t^{10} + t^9 + t^8 + t^7 + t^5 + t^4) + (t^9 + t^8 + t^7 + t^6 + t^4 + t^3) +$
 $(t^8 + t^7 + t^6 + t^5 + t^3 + t^2) + (t^7 + t^6 + t^5 + t^4 + t^2 + t)$
 $= t^{11} + t^9 + t^4 + t$
 $= t^3 + t + t^4 + t = t^4 + t^3$ \downarrow im TPR ist $t^8 = 1$ $\Rightarrow 0x7B$ \checkmark
 Addiere nun im TPR $63_{16} = 1 + t + t^5 + t^6$: $t^6 + t^5 + t^4 + t^3 + t + 1$

- MC/MixColumns:

\rightarrow oder alternativ meine .py

- Mit Sage:

```

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],
    
```

Polynomdivision in Sage:

$S.<t> = \text{PolynomialRing}(GF(2), t)$

$(t**8 + t**4 + t**3 + t + 1)$
 $// (t+1)$

$\Rightarrow t^{17} + t^{16} + t^{15} + t^{14}$
 $+ t^2 + t$

$100011011 \div 11 =$
 $\oplus 11$
 100011011
 $\oplus 11$
 $1011011 \dots$

- Mit Hand:

$\begin{pmatrix} t & t+1 & 1 & 1 \\ 1 & t & t+1 & 1 \\ 1 & 1 & t & t+1 \\ t+1 & 1 & 1 & t \end{pmatrix} \cdot \begin{pmatrix} t^7 + t^6 + t^4 + t^3 + t + 1 \\ t^4 + t + 1 \\ t^6 + t^4 + t + 1 \\ t^6 + t^2 + 1 \end{pmatrix}$
 input vector

$\begin{pmatrix} [t^8 + t^7 + t^5 + t^4 + t^2 + t] + [t^8 + t^7 + t^5 + \dots] \\ 006 \end{pmatrix}$
 Multiplikation von Elementen im F_{256} erfordert anschließendes Rechnen mod
 Rijndael-Polynom $R(t)$ via Polynomdivision;
 hat man Glück gibt's aber kein t^8 oder größer!