5 Data und Advanced Encryption Standard

5.1 Der Data Encryption Standard (DES)

Der DES wurde von IBM im Zuge einer im Mai 1973 veröffentlichten Ausschreibung des NBS (National Bureau of Standards; heute National Institute of Standards and Technology, NIST) als ein Nachfolger von Lucifer entwickelt, im März 1975 veröffentlicht, und im Januar 1977 als Verschlüsselungsstandard der US-Regierung für nicht geheime Nachrichten genormt. Obwohl der DES ursprünglich nur für einen Zeitraum von 10 bis 15 Jahren als Standard dienen sollte, wurde er circa alle 5 Jahre (zuletzt im Januar 1999) überprüft und als Standard fortgeschrieben.

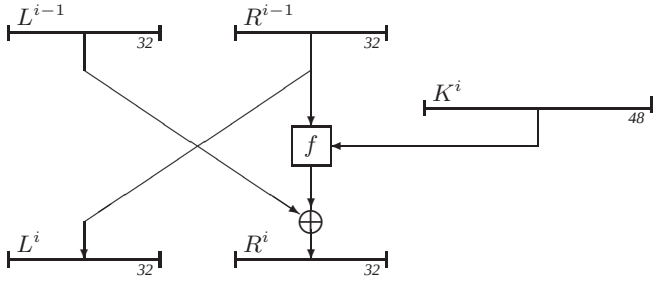
Bereits im September 1997 veröffentlichte das NIST eine Ausschreibung für den AES (Advanced Encryption Standard) genannten Nachfolger des DES. Nach einer mehrjährigen Auswahlprozedur wurde im November 2001 der Rijndael-Algorithmus als AES genormt.

Der DES ist eine Feistel-Chiffre mit 16 Runden. Die Rundenfunktion g einer Feistel-Chiffre berechnet das Zwischenergebnis w^i aus den beiden Hälften L^{i-1} und R^{i-1} von w^{i-1} gemäß der Vorschrift

$$g(K^i, L^{i-1}R^{i-1}) = L^iR^i,$$

wobei sich $w^i = L^iR^i$ zusammensetzt aus

$$\begin{aligned} L^i &= R^{i-1} \text{ und} \\ R^i &= L^{i-1} \oplus f(R^{i-1}, K^i). \end{aligned}$$



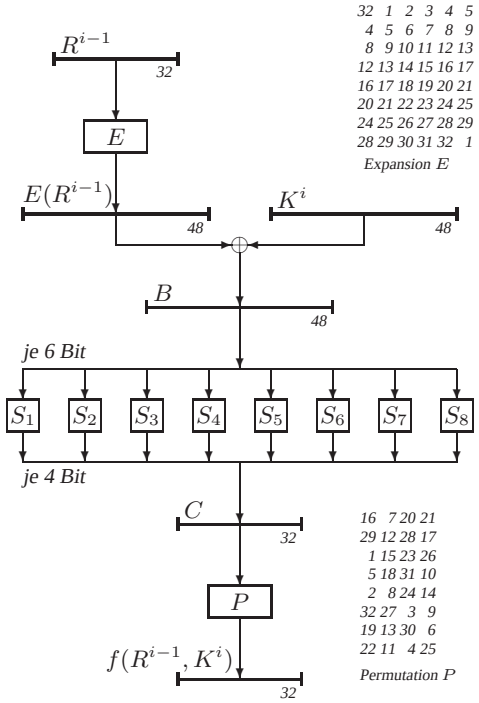
Der DES chiffriert Binärblöcke der Länge 64 und benutzt hierzu einen Schlüssel mit 56 Bit. Der Schlüssel ergibt zusammen mit 8 Paritätsbits (die Bits 8, 16,..., 64) einen ebenfalls 64 Bit langen Schlüsselblock K. Es gibt somit $2^{56} \approx 7{,}2 \cdot 10^{16}$ verschiedene Schlüssel.

Im Einzelnen werden folgende Chiffrierschritte ausgeführt:

1. Zuerst wird der Klartextblock x einer Initialpermutation IP unterzogen:

$$x_1 x_2 \cdots x_{64} \mapsto IP(x) = x_{58} x_{50} \cdots x_7.$$

2. Danach wird 16 Mal die Rundenfunktion g mit den Rundenschlüsseln $K^1, \ldots, K^{16}$ angewendet, wobei die Funktion $f : \{0,1\}^{32} \times \{0,1\}^{48} \to \{0,1\}^{32}$ wie folgt berechnet wird:



Expansion E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Permutation P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Substitutionsboxen

S_1 :	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2 :	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3 :	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4 :	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5 :	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6 :	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7 :	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8 :	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Bei Eingabe (R^{i-1}, K^i) wird R^{i-1} durch die Expansionsabbildung E auf einen 48-Bit Block $E(R^{i-1})$ erweitert. Dieser wird mit K^i bitweise addiert (x-or); als Ergebnis erhält man den Vektor $B = E(R^{i-1}) \oplus K^i$. Danach wird B in acht 6-Bit Blöcke $B_1, \ldots, B_8$ aufgeteilt, die mittels 8 S-Boxen $S_1, \ldots, S_8$ auf 4-Bit Blöcke $C_i = S_i(B_i)$ reduziert werden. Die S-Boxen sind in Form einer Tabelle dargestellt, die wie folgt ausgewertet wird:

> Ist $B_i = b_1 \cdots b_6$, so findet man $S_i(B_i)$ in Zeile $b_1 b_6$ und Spalte $b_2 b_3 b_4 b_5$ (jeweils aufgefasst als Binärzahl) der Tabelle für S_i. Zum Beispiel ist $S_1(011010) = 1001$, da in Zeile $(00)_2 = 0$ und Spalte $(1101)_2 = 13$ der S_1-Tabelle die Zahl $9 = (1001)_2$ eingetragen ist.

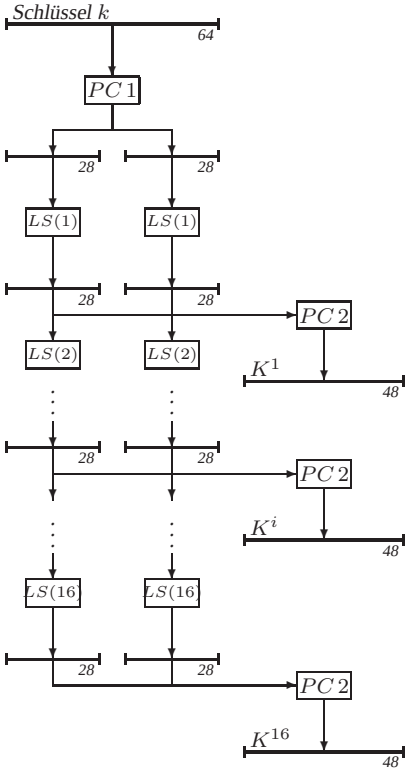
Die Konkatenation der von den acht S-Boxen gelieferten Bitblöcke $C_1 \ldots C_8$ ergibt einen 32-Bit Vektor C, welcher noch der Permutation P unterworfen wird.

3. Aus dem nach der 16. Iteration erhaltenen Bitvektor $w^{16} = L^{16} R^{16}$ wird durch Vertauschen der beiden Hälften und Anwendung der inversen Initialpermutation

der Kryptotext y gebildet:

$$L^{16}R^{16} \mapsto y = IP^{-1}(R^{16}L^{16}).$$

Die Schlüsselgenerierung. Zuerst wählt die Funktion $PC\,1$ (permuted choice 1) aus dem Schlüssel k die kryptographisch relevanten Bits aus und permutiert sie. Das erhaltene Ergebnis wird in zwei 28-Bit Blöcke unterteilt. Diese beiden Blöcke werden dann in 16 Runden jeweils zyklisch um ein oder zwei Bit verschoben (siehe dazu Tabelle $LS(i)$).



57	49	41	33	25	17	9		14	17	11	24	1	5
1	58	50	42	34	26	18		3	28	15	6	21	10
10	2	59	51	43	35	27		23	19	12	4	26	8
19	11	3	60	52	44	36		16	7	27	20	13	2
63	55	47	39	31	23	15		41	52	31	37	47	55
7	62	54	46	38	30	22		30	40	51	45	33	48
14	6	61	53	45	37	29		44	49	39	56	34	53
21	13	5	28	20	12	4		46	42	50	36	29	32
permuted choice 1								*permuted choice 2*					

Iteration i	Anzahl der Links-Shifts $LS(i)$
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Aus den beiden Blöcken nach Runde i bestimmt die Funktion $PC\,2$ (permuted choice 2) jeweils den Rundenschlüssel K^i durch Entfernen der 8 Bits an den Stellen 9, 18, 22, 25, 35, 38, 43 und 56 sowie einer Permutation der verbleibenden 48 Bits.

Eigenschaften des DES. Der DES hat sich zwar weitgehend durchgesetzt, jedoch wurde er anfangs von manchen US-Behörden und -Banken nicht verwendet. Der Grund dafür liegt in folgenden Sicherheitsbedenken, die nach seiner Veröffentlichung im Jahre 1975 geäußert wurden:

– Die 56-Bit Schlüssellänge bietet eventuell eine zu geringe Sicherheit gegen Aus-

probieren aller Schlüssel bei einem Angriff mit bekanntem oder gewähltem Klartext.
- Die Entwurfskriterien für die einzelnen Bestandteile, insbesondere für die S-Boxen, sind nicht veröffentlicht worden. Es wurde der Verdacht geäußert, dass der DES mit Hilfe von Falltürinformationen leicht zu brechen sei.
- Kryptoanalytische Untersuchungen, die von IBM und der US National Security Agency (NSA) durchgeführt wurden, sind nicht veröffentlicht worden. Als jedoch Biham und Shamir Anfang der 90er Jahre das Konzept der differentiellen Kryptoanalyse veröffentlichten, gaben die Entwickler von DES bekannt, dass sie diese Angriffsmöglichkeit beim Entwurf von DES bereits kannten und speziell die S-Boxen entsprechend konzipiert hätten.

Im Fall von DES ist die lineare Kryptoanalyse effizienter als die differentielle Kryptoanalyse. Da hierzu jedoch circa 2^{43} Klartext-Kryptotext-Paare notwendig sind (deren Generierung bei einem von Matsui, dem Erfinder der linearen Kryptoanalyse, unternommenen Angriff bereits 40 Tage in Anspruch nahm), stellen diese Angriffe keine realistische Bedrohung dar.

Dagegen wurde im Juli 1998 mit einer von der Electronic Frontier Foundation (EFF) für 250 000 Dollar gebauten Maschine namens "DES Cracker" eine vollständige Schlüsselsuche in circa 56 Stunden durchgeführt (was den Gewinn der von RSA Laboratory ausgeschriebenen "DES Challenge II-2" bedeutete). Und im Januar 1999 gewann `Distributed.Net`, eine weltweite Vereinigung von Computerfans, den mit 10 000 Dollar dotierten "DES Challenge III". Durch den kombinierten Einsatz eines Supercomputer namens "Deep Crack" von EFF und 100 000 PCs, die weltweit über das Internet kommunizierten, wurden nur 22 Stunden und 15 Minuten benötigt, um den Schlüssel für ein Klartext-Kryptotextpaar mit dem Klartext „See you in Rome (second AES Conference, March 22-23, 1999)" zu finden.

Definition 107 (**schwache Schlüssel**)

Ein DES-Schlüssel K heißt **schwach**, falls alle durch ihn erzeugten Rundenschlüssel gleich sind (d.h. es gilt $\|\{K^1,\ldots, K^{16}\}\|$=1).

Es gibt vier schwache Schlüssel (siehe Übungen):

```
0101010101010101
FEFEFEFEFEFEFEFE
1F1F1F1F0E0E0E0E
E0E0E0E0F1F1F1F1
```

und für sie gilt

$$\mathrm{DES}(K, \mathrm{DES}(K, x)) = x.$$

Neben diesen schwachen Schlüsseln existieren noch sechs weitere sogenannte „semischwache" Schlüsselpaare (K, K'), für die gilt (siehe Übungen)

$$\mathrm{DES}(K', \mathrm{DES}(K, x)) = x.$$

5.2 Der Advanced Encryption Standard (AES)

Geschichte des AES:

- Im September 1997 veröffentlichte das NIST eine Ausschreibung für den AES, in der eine Blocklänge von 128 Bit und variable Schlüssellängen von 128, 192 und 256 Bit gefordert wurden. Einreichungsschluss war der 15. Juni 1998.
- Von den 21 Einreichungen erfüllten 15 die geforderten Kriterien. Diese stammten aus den Ländern Australien, Belgien, Costa Rica, Deutschland, Frankreich, Großbritannien, Israel, Japan, Korea, Norwegen sowie den USA und wurden auf der 1. AES-Konferenz am 20. August 1998 als AES-Kandidaten akzeptiert.
- Im August 1999 wählte NIST auf der 2. AES-Konferenz in Rom die Finalisten MARS, RC6, Rijndael, Serpent und Twofish aus.
- Im April 2000 wurde der Rijndael-Algorithmus auf der 3. AES-Konferenz zum Sieger erklärt und im November 2001 als AES genormt.

Die wichtigsten Entscheidungskriterien waren

- Sicherheit,
- Kosten (Effizienz bei Software-, Hardware- und Smartcard-Implementationen) sowie
- Algorithmen- und Implementations-Charakeristika (unter anderem Flexibilität und Einfachheit des Designs).

Die Blocklänge und die Schlüssellänge können beim Rijndael unabhängig voneinander im Bereich 128, 192 oder 256 Bit gewählt werden. Die Rundenzahl N des Rijndael hängt wie folgt von der Blocklänge und der gewählten Schlüssellänge ab:

N	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

Wir beschränken uns hier auf die Beschreibung des 10-Runden AES mit $l = 128$ Bit Blocklänge und $k = 128$ Bit Schlüssellänge.

Die Schlüsselgenerierung. Beim 10-Runden AES mit Block- und Schlüssellänge $l = k = 128$ werden 11 Rundenschlüssel $K^0, \ldots, K^{10}$ der Länge 128 benutzt. Jedes K^i besteht also aus 16 Bytes bzw. 4 Worten mit jeweils 4 Bytes. Reihen wir die 11 Rundenschlüssel aneinander, so entsteht ein Array $w[0], \ldots, w[43]$ von 44 Worten, die gemäß folgendem Algorithmus aus dem 128-Bit Schlüssel K berechnet werden.

Algorithmus 108 KEYEXPANSION(K)

Eingabe: $K = K[0] \cdots K[15]$
  $RCon[1] \leftarrow 01000000$
  $RCon[2] \leftarrow 02000000$
  $RCon[3] \leftarrow 04000000$
  $RCon[4] \leftarrow 08000000$
  $RCon[5] \leftarrow 10000000$
  $RCon[6] \leftarrow 20000000$
  $RCon[7] \leftarrow 40000000$
  $RCon[8] \leftarrow 80000000$
  $RCon[9] \leftarrow 1B000000$
  $RCon[10] \leftarrow 36000000$
  for $i \leftarrow 0$ **to** 3 **do**
    $w[i] \leftarrow (K[4i], K[4i+1], K[4i+2], K[4i+3])$ **end**
  for $i \leftarrow 4$ **to** 43 **do**
    $temp \leftarrow w[i-1]$
    if $i \equiv_4 0$ **then**
      $temp \leftarrow$ SUBWORD(ROTWORD($temp$)) $\oplus$ $RCon[i/4]$
    $w[i] \leftarrow w[i-4] \oplus temp$ **end**
Ausgabe: $w[0] \ldots w[43]$

Die hierbei benutzten Funktionen sind wie folgt definiert:

- ROTWORD : $\mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \rightarrow \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8$ führt eine zyklische Verschiebung der 4 Eingabebytes um ein Byte nach links durch:
$$\text{ROTWORD}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0),$$
- SUBWORD : $\mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \rightarrow \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8 \times \mathbb{F}(2)^8$ ersetzt jedes Eingabebyte B_i durch $\pi_{\text{SUBBYTES}}(B_i)$:
$$\begin{aligned} &\text{SUBWORD}(B_0, B_1, B_2, B_3) \\ &\quad = \text{SUBBYTES}(B_0, B_1, B_2, B_3) \\ &\quad = (\pi_{\text{SUBBYTES}}(B_0), \pi_{\text{SUBBYTES}}(B_1), \pi_{\text{SUBBYTES}}(B_2), \pi_{\text{SUBBYTES}}(B_3)) \end{aligned}$$

Unter Benutzung der 11 Rundenschlüssel $K^0, \ldots, K^{10}$ werden nun folgende Chiffrierschritte ausgeführt:

ADDROUNDKEY(K^0)
```
for i:= 1 downto 9 do
```
  SUBBYTES
  SHIFTROWS
  MIXCOLUMNS

```
    AddRoundKey(K^i)
end
SubBytes
ShiftRows
AddRoundKey(K^10)
```

1. Zuerst wird der Klartextblock x einer Addition mit dem 128-Bit Rundenschlüssel K^0 unterworfen. Diese Operation wird mit AddRoundKey bezeichnet.

2. Danach werden 9 Runden ausgeführt, wobei in jeder Runde i eine Substitution namens SubBytes, eine Permutation namens ShiftRows, eine lineare Substitution namens MixColumns und eine AddRoundKey Operation mit dem Rundenschlüssel K^i durchgeführt werden.

3. Es folgt Runde 10 mit den Operationen SubBytes, ShiftRows und AddRoundKey(K^{10}).

Der Klartext (und alle daraus berechneten Zwischenergebnisse) werden in Form eines Arrays

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

dargestellt, das wie folgt initialisiert wird:

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

Die Substitution SubBytes basiert auf einer 8-Bit S-Box π_{SubBytes}, die durch folgenden Algorithmus berechnet wird (die Indexrechnung in Zeile 8 erfolgt modulo 8).

Algorithmus 109 $\pi_{\text{SubBytes}}(a_7 \cdots a_0)$

1 **Eingabe:** $(a_7 \cdots a_0)$
2 $z \leftarrow \text{BinaryToField}(a_7 \cdots a_0)$
3 **if** $z \neq 0$ **then**
4 $z \leftarrow \text{FieldInv}(z)$
5 $a_7 \cdots a_0 \leftarrow \text{FieldToBinary}(z)$
6 $c_7 \cdots c_0 \leftarrow (01100011)$
7 **for** $i \leftarrow 0$ **to** 7 **do**
8 $b_i \leftarrow a_i \oplus a_{i+4} \oplus a_{i+5} \oplus a_{i+6} \oplus a_{i+7} \oplus c_i$ **end**
9 **Ausgabe:** $(b_7 \cdots b_0)$

Die hierbei benutzten Funktionen sind wie folgt definiert:

- FIELDINV: $\mathbb{F}(2^8) \rightarrow \mathbb{F}(2^8)$ berechnet das multiplikative Inverse im Körper $\mathbb{F}(2^8) = \mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$,
- BINARYTOFIELD: $\{0,1\}^8 \rightarrow \mathbb{F}(2^8)$ berechnet aus der 8-Bit Koeffizienten-Darstellung das zugehörige Körperelement.
- FIELDTOBINARY: $\mathbb{F}(2^8) \rightarrow \{0,1\}^8$ berechnet die Inverse der Funktion BINARYTOFIELD.

Beispiel 110 *Wir berechnen* $\pi_{\text{SUBBYTES}}(01010011)$. *Die Funktion* BINARYTOFIELD *liefert das zugehörige Polynom*

$$z = \text{BINARYTOFIELD}(01010011) = x^6 + x^4 + x + 1.$$

Das multiplikative Inverse von z *in* $\mathbb{F}(2^8)$ *ist*

$$x^7 + x^6 + x^3 + x$$

(siehe Beispiel 115). Die Funktion FIELDTOBINARY *liefert die zugehörige Koeffizienten-Darstellung*

$$\text{FIELDTOBINARY}(x^7 + x^6 + x^3 + x) = (11001010).$$

Es folgt die Berechnung der Ausgabe $(b_7 \cdots b_0) = (11101101)$ *mittels*

$$\begin{aligned} b_0 &= a_0 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus c_0 \\ &= 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \\ &= 1 \\ b_1 &= a_1 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_0 \oplus c_1 \\ &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \\ &= 0 \end{aligned}$$

usw. Somit ist $\pi_{\text{SUBBYTES}}(01010011) = (11001010)$ *oder hexadezimal dargestellt:* $\pi_{\text{SUBBYTES}}(53) = ED$.

Wir können die AES S-Box in Form einer 16×16-Matrix angeben, wobei in Zeile X und Spalte Y der Wert $\pi_{\text{SUBBYTES}}(XY)$ eingetragen ist:

	Y															
X	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
⋮								⋮								
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

SHIFTROWS ist eine 128-Bit Permutation, die wie folgt definiert ist:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \quad \mapsto \quad \begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

MIXCOLUMNS ist eine lineare 32-Bit Substitution, die auf den Spalten der Zwischenergebnisse operiert. Zu ihrer Berechnung wird folgende Funktion benutzt:

– FIELDMULT: $\mathbb{F}(2^8) \times \mathbb{F}(2^8) \to \mathbb{F}(2^8)$ führt die Multiplikation im Körper $\mathbb{F}(2^8)$ aus.

Algorithmus 111 MIXCOLUMNS(c_0, c_1, c_2, c_3)

1 **Eingabe:** (c_0, c_1, c_2, c_3)
2 **for** $i \leftarrow 0$ **to** 3 **do**
3 $t_i \leftarrow$ BINARYTOFIELD(c_i) **end**
4 $u_0 \leftarrow$ FIELDMULT$(x, t_0) +$ FIELDMULT$(x+1, t_1) + t_2 + t_3$
5 $u_1 \leftarrow$ FIELDMULT$(x, t_1) +$ FIELDMULT$(x+1, t_2) + t_3 + t_0$
6 $u_2 \leftarrow$ FIELDMULT$(x, t_2) +$ FIELDMULT$(x+1, t_3) + t_0 + t_1$
7 $u_3 \leftarrow$ FIELDMULT$(x, t_3) +$ FIELDMULT$(x+1, t_0) + t_1 + t_2$
8 **for** $i \leftarrow 0$ **to** 3 **do**
9 $c_i \leftarrow$ FIELDTOBINARY(u_i) **end**
10 **Ausgabe:** (c_0, c_1, c_2, c_3)

MIXCOLUMNS führt eine lineare Transformation sowohl in $(\mathbb{F}_{2^8})^4$ als auch in $(\mathbb{F}_2)^{32}$ aus, die sich auch wie folgt beschreiben lässt (hierbei benutzen wir die Koeffizientendarstellung der Elemente (Polynome) in $\mathbb{F}_{2^8}$, etwa 03 für $x+1$):

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \quad \mapsto \quad \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Besonders elegant lässt sich die Operation MIXCOLUMNS im Polynom-Restklassenring $\mathbb{F}_{2^8}[z]/(z^4+1)$ beschreiben. Sei $c(z) = \sum_{i=0}^{3} c_i z^i$ das durch die Spalte (c_0, c_1, c_2, c_3) repräsentierte Polynom in diesem Ring und sei $a(z)$ das Polynom $a(z) = 03z^3 + 01z^2 + 01z + 02$. Dann ist

$$\text{MIXCOLUMNS}(c(z)) = a(z)c(z).$$

D.h., MIXCOLUMNS ist eine multiplikative Chiffre mit festem Schlüssel $a(z)$ im Ring $\mathbb{F}_{2^8}[z]/(z^4+1)$. Da das Polynom z^4+1 nicht irreduzibel in $\mathbb{F}_{2^8}[z]$ ist, ist $\mathbb{F}_{2^8}[z]/(z^4+1)$ zwar kein Körper. Da jedoch $a(z)$ invertierbar im Ring $\mathbb{F}_{2^8}[z]/(z^4+1)$ ist, kann die Inverse zu MIXCOLUMNS mittels

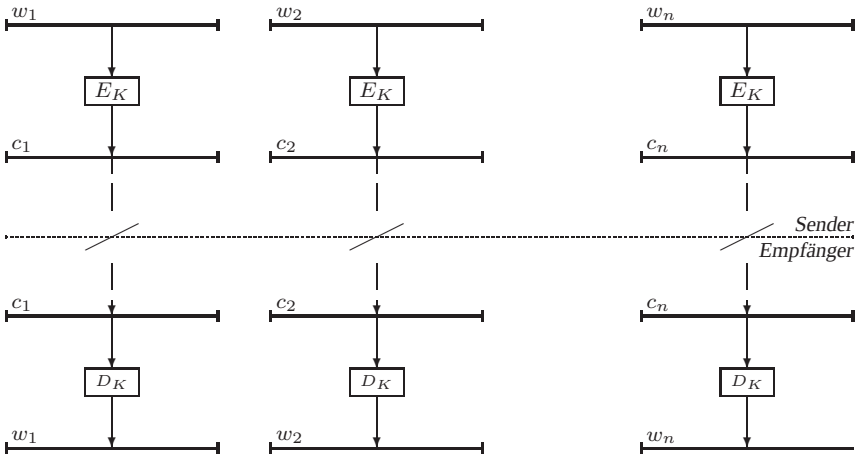
$$\text{MIXCOLUMNS}^{-1}(c(z)) = a^{-1}(z)c(z)$$

berechnet werden, wobei $a^{-1}(z) = 0Bz^3 + 0Dz^2 + 09z + 0E$ ist.

5.3 Betriebsarten von Blockchiffren

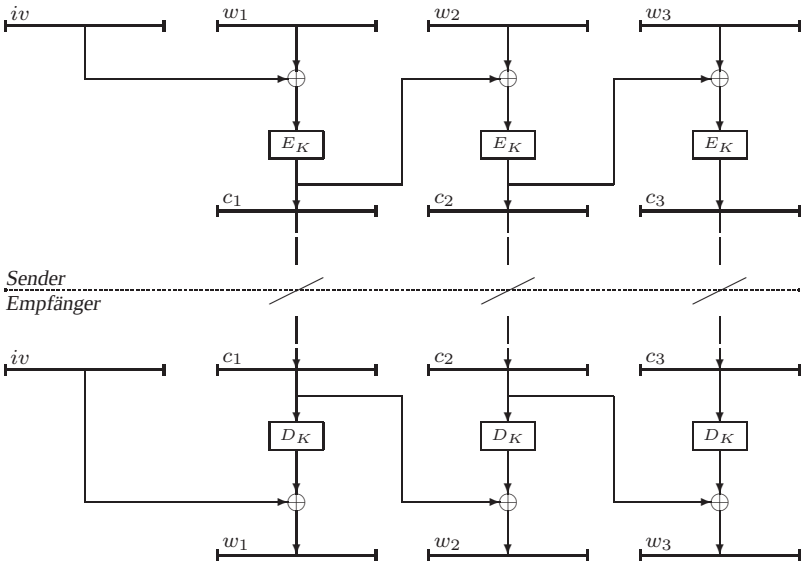
Für den DES wurden vier verschiedene Betriebsarten vorgeschlagen, in denen grundsätzlich jede Blockchiffre E mit beliebiger Blocklänge l betrieben werden kann. Bei den ersten beiden Betriebsarten (ECB und CBC) werden Kryptotextblöcke der Länge l übertragen. Mit einer Blockchiffre kann aber auch ein Stromsystem realisiert werden, mit dem sich Kryptotextblöcke einer beliebigen Länge t, $1 \leq t \leq l$, übertragen lassen (OFB und CFB).

ECB-Mode (electronic code book; elektronisches Codebuch): Die Binär-Nachricht x wird in 64-Bit Blöcke w_i zerlegt. Der letzte Block w_n wird, falls nötig, mit einer vorher vereinbarten Bitfolge aufgefüllt. Die Blöcke werden nacheinander mit demselben Schlüssel K einzeln verschlüsselt, übertragen und auf Empfängerseite mittels der zu E gehörigen Dechiffrierfunktion D wieder entschlüsselt:

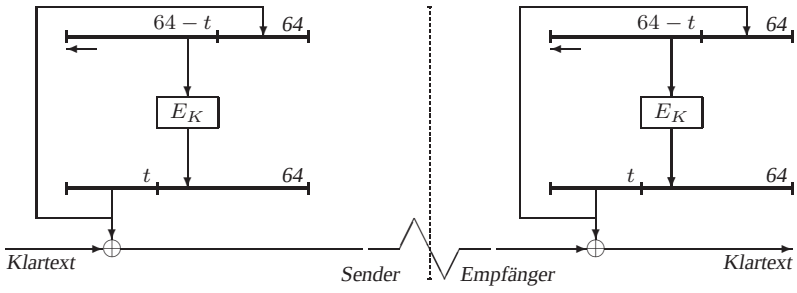


Um zu verhindern, dass ein Eindringling den Kryptotext verändert, ohne dass der Empfänger dies bemerkt, wird jeder Kryptotextblock nicht nur von dem zugehörigen Klartextblock, sondern auch von allen vorausgehenden Blöcken abhängig gemacht.

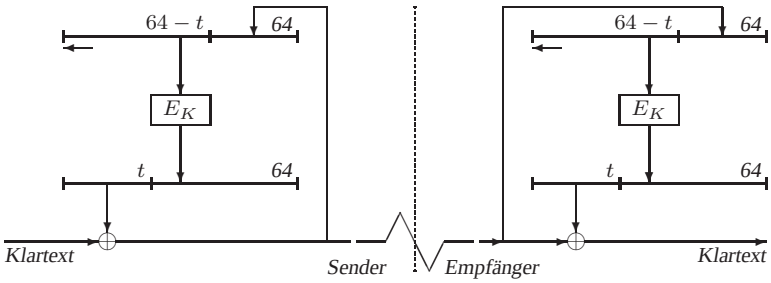
CBC-Mode (cipher block chaining; Blockverkettung des Schlüsseltextes): Jeder Klartextblock w_i wird mit dem Kryptotextblock $E_K(w_{i-1})$ bitweise (modulo 2) addiert, bevor er verschlüsselt wird (zur Verschlüsselung von w_1 wird ein Initialisierungsvektor iv verwendet:



OFB-Mode (output feedback; Rückführung der Ausgabe): Die Binär-Nachricht x wird in t-Bit Blöcke (für festes t: $1 \leq t \leq l$) zerlegt. Die Chiffrierfunktion E_K dient zur Erzeugung einer pseudozufälligen Folge von t-Bit Blöcken, die bitweise (modulo 2) zu den entsprechenden Klartextblöcken addiert werden. Als Eingabe für die Chiffrierfunktion E_K dient ein Schieberegister, das anfangs mit einem Initialisierungsvektor IV geladen ist. Bei jeder Übertragung eines t-Bit Klartextblockes w_i erzeugt die Chiffrierfunktion E_K zunächst einen Ausgabevektor, von dem nur die ersten t Bits verwendet werden. Diese dienen sowohl zur Verschlüsselung von w_i, als auch zur Modifikation des Eingaberegisters, in das sie von rechts geschoben werden:



CFB-Mode (cipher feedback; Rückführung des Kryptotextes): Zur Erneuerung des Eingaberegisters wird nicht die E_K-Ausgabe, sondern der t-Bit Kryptotextblock verwendet:


64 − t
64
E_K
t
64
Klartext
Sender
Empfänger
64 − t
64
E_K
t
64
Klartext