Der AES-Algorithmus

6

1997 wurde vom National Institute of Standards and Technology (NIST) der Auswahlprozess für einen Nachfolger des DES begonnen. Unter den Einreichungen war auch die Rijndael-Chiffre. Sie ist benannt nach den beiden Erfindern Rijmen und Daemen. Dieses Verfahren wurde am 26. November 2001 als Advanced Encryption Standard (AES) standardisiert.

AES ist eine Blockchiffre mit Alphabet $\mathbb{Z}_2$. Sie ist ein Spezialfall der Rijndael-Chiffre. Die Rijndael-Chiffre lässt andere Blocklängen und andere Schlüsselräume als AES zu. Wir beschreiben hier die Rijndael-Chiffre und AES als Spezialfall.

6.1 Bezeichnungen

Um die Rijndael-Chiffre zu beschreiben, werden folgende Größen benötigt:

$\mathtt{Nb}$ Die Klartext- und Chiffretextblöcke bestehen aus $\mathtt{Nb}$ vielen 32-Bit Wörtern, $4 \leq \mathtt{Nb} \leq 8$.
Die Rijndael-Blocklänge ist also $32 * \mathtt{Nb}$.
Für AES ist $\mathtt{Nb} = 4$. Die AES-Blocklänge ist also 128.

$\mathtt{Nk}$ Die Schlüssel bestehen aus $\mathtt{Nk}$ vielen 32-Bit Wörtern, $4 \leq \mathtt{Nk} \leq 8$
Der Rijndael-Schlüsselraum ist also $\mathbb{Z}_2^{32 * \mathtt{Nk}}$.
Für AES ist $\mathtt{Nk} = 4$, 6 oder 8.
Der AES-Schlüsselraum ist also $\mathbb{Z}_2^{128}$, $\mathbb{Z}_2^{192}$ oder $\mathbb{Z}_2^{256}$.

$\mathtt{Nr}$ Anzahl der Runden.

$$\text{Für AES ist } \mathtt{Nr} = \begin{cases} 10 & \text{für } \mathtt{Nk} = 4, \\ 12 & \text{für } \mathtt{Nk} = 6, \\ 14 & \text{für } \mathtt{Nk} = 8. \end{cases}$$

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_6

In den folgenden Beschreibungen werden die Datentypen `byte` und `word` benutzt. Ein `byte` ist ein Bitvektor der Länge 8. Ein `word` ist ein Bitvektor der Länge 32. Klartext und Chiffretext werden als zweidimensionale `byte`-Arrays dargestellt. Diese Arrays haben vier Zeilen und `Nb` Spalten. Im AES-Algorithmus sieht ein Klartext oder Chiffretext also so aus:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \tag{6.1}$$

Die Rijndael-Schlüssel sind `word`-Arrays der Länge `Nk`. Die Rijndael-Chiffre expandiert einen Schlüssel `key` mit der Funktion `KeyExpansion` zu dem expandierten Schlüssel `w`. Danach verschlüsselt sie einen Klartextblock `in` mit dem expandierten Schlüssel `w` zu dem Schlüsseltextblock `out`. Hierzu wird `Cipher` verwendet. In den folgenden Abschnitten beschreiben wir zuerst den Algorithmus `Cipher` und dann den Algorithmus `KeyExpansion`.

6.2 Cipher

Wir beschreiben die Funktion `Cipher`, die in Abb. 6.1 dargestellt ist. Eingabe ist der Klartextblock `byte in[4,Nb]` und der expandierte Schlüssel `word w[Nb*(Nr+1)]`. Ausgabe ist der Chiffretextblock `byte out[4,Nb]`. Zuerst wird der Klartext `in` in das `byte`-Array `state` kopiert. Nach einer initialen Transformation durchläuft `state` `Nr` Runden und wird dann als Chiffretext zurückgegeben. In den ersten $\texttt{Nr} - 1$ Runden werden nacheinander die Transformationen `SubBytes`, `ShiftRows`, `MixColumns` und `AddRoundKey` angewendet. In der letzten Runde werden nur noch die Transformationen `SubBytes`, `ShiftRows` und `AddRoundKey` angewendet. `AddRoundKey` ist auch die initiale Transformation.

In den folgenden Abschnitten werden die Transformationen im einzelnen beschrieben.

Abb. 6.1 Die AES-Funktion `Cipher`

```
Cipher(byte in[4,Nb], byte out[4,Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]
   state = in
   AddRoundKey(state, w[0, Nb-1])
   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
   out = state
end
```

6.2.1 Identifikation der Bytes mit Elementen von $\mathrm{GF}(2^8)$

Bytes spielen eine zentrale Rolle in der Rijndael-Chiffre. Sie können auch als ein Paar von Hexadezimalzahlen geschrieben werden.

Beispiel 6.1 Das Paar $\{2F\}$ von Hexadezimalzahlen entspricht dem Paar 0010 1111 von Bitvektoren der Länge vier, also dem Byte 00101111. Das Paar $\{A1\}$ von Hexadezimalzahlen entspricht dem Paar 1010 0001 von Bitvektoren, also dem Byte 10100001.

Bytes werden in der Rijndael-Chiffre mit Elementen des endlichen Körpers $\mathrm{GF}(2^8)$ identifiziert. Als erzeugendes Polynom (siehe Abschn. 2.20) wird das über $\mathrm{GF}(2)$ irreduzible Polynom

$$m(X) = X^8 + X^4 + X^3 + X + 1 \tag{6.2}$$

gewählt. Damit ist

$$\mathrm{GF}(2^8) = \mathrm{GF}(2)(\alpha)$$

wobei α der Gleichung

$$\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$$

genügt. Ein Byte

$$(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$$

entspricht also dem Element

$$\sum_{i=0}^{7} b_i \alpha^i$$

Damit können Bytes multipliziert und addiert werden. Falls sie von Null verschieden sind, können sie auch invertiert werden. Für das Inverse von b wird b^{-1} geschrieben. Wir definieren auch $0^{-1} = 0$.

Beispiel 6.2 Das Byte $b = (0,0,0,0,0,0,1,1)$ entspricht dem Körperelement $\alpha + 1$. Gemäß Beispiel 2.42 ist $(\alpha + 1)^{-1} = \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha$. Daher ist $b^{-1} = (1,1,1,1,0,1,1,0)$.

6.2.2 `SubBytes`

`SubBytes(state)` ist eine nicht-lineare Funktion. Sie transformiert die einzelnen Bytes. Die Transformation wird S-Box genannt. Aus jedem Byte b von `state` macht die S-Box das neue Byte

$$b \leftarrow Ab^{-1} \oplus c \tag{6.3}$$

mit

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0. \end{pmatrix}.$$

Diese S-Box kann tabelliert werden, weil sie nur 2^8 mögliche Argumente hat. Dann kann die Anwendung von `SubBytes` durch Table-Lookups realisiert werden.

Beispiel 6.3 Wir berechnen, welches Byte die S-Box aus dem Vektor $b = (0, 0, 0, 0, 0, 0, 1, 1)$ macht. Nach Beispiel 6.2 ist $b^{-1} = (1, 1, 1, 1, 0, 1, 1, 0)$. Damit gilt $Ab^{-1} + c = (0, 1, 1, 0, 0, 1, 1, 1)$.

Die S-Box garantiert die Nicht-Linearität von AES.

6.2.3 `ShiftRows`

Sei s ein `state`, also ein durch AES teiltransformierter Klartext. Schreibe s als Matrix. Die Einträge sind Bytes. Die Matrix hat 4 Zeilen und `Nb` Spalten. Im Fall von AES ist diese Matrix

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \tag{6.4}$$

`ShiftRows` wendet auf die Zeilen dieser Matrix einen zyklischen Linksshift an. Genauer: `ShiftRows` hat folgende Wirkung:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \to \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix} \tag{6.5}$$

Im Allgemeinen wird die i-te Zeile um c_i Positionen nach links verschoben, wobei c_i in Tab. 6.1 zu finden ist. Diese Transformation sorgt bei Anwendung in mehreren Runden für hohe Diffusion.

Tab. 6.1 Zyklischer Linksshift in ShiftRows

Nb	c_0	c_1	c_2	c_3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

6.2.4 MixColumns

Für $0 \leq j < \mathtt{Nb}$ wird die Spalte

$$s_j = (s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j})$$

von state mit dem Polynom

$$s_{0,j} + s_{1,j}x + s_{2,j}x^2 + s_{3,j}x^3 \in \mathrm{GF}(2^8)[x] \tag{6.6}$$

identifiziert. Die Transformation MixColumns setzt

$$s_j \leftarrow (s_j * a(x)) \bmod (x^4 + 1), \quad 0 \leq j < \mathtt{Nb}, \tag{6.7}$$

wobei

$$a(x) = \{03\} * x^3 + \{01\} * x^2 + \{01\} * x + \{02\}. \tag{6.8}$$

Das kann auch als lineare Transformation in $\mathrm{GF}(2^8)^4$ beschrieben werden. MixColumns setzt nämlich

$$s_j \leftarrow \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix} s_j \quad 0 \leq j < \mathtt{Nb}. \tag{6.9}$$

Diese Transformation sorgt für eine Diffusion innerhalb der Spalten von state.

6.2.5 AddRoundKey

Sind $s_0, \ldots, s_{\mathtt{Nb}-1}$ die Spalten von state, dann setzt der Aufruf der Funktion AddRoundKey(state, w[l*Nb, (l+1)*Nb-1])

$$s_j \leftarrow s_j \oplus w[l * \mathtt{Nb} + j], \quad 0 \leq j < \mathtt{Nb}, \tag{6.10}$$

wobei $\oplus$ das bitweise $\oplus$ ist. Die Wörter des Rundenschlüssels werden also mod 2 zu den Spalten von state addiert. Dies ist eine sehr einfache und effiziente Transformation, die die Transformation einer Runde schlüsselabhängig macht.

6.3 KeyExpansion

Der Algorithmus KeyExpansion, der in Abb. 6.2 gezeigt wird, macht aus dem Rijndael-Schlüssel key, der ein byte-array der Länge 4*Nk ist, einen expandierten Schlüssel w, der ein word-array der Länge Nb*(Nr+1) ist. Die Verwendung des expandierten Schlüssels wurde in Abschn. 6.2 erklärt. Zuerst werden die ersten Nk Wörter im expandierten Schlüssel w mit den Bytes des Schlüssels key gefüllt. Die folgenden Wörter in w werden erzeugt, wie es im Pseudocode von KeyExpansion beschrieben ist. Die Funktion word schreibt die Bytes einfach hintereinander.

Wir beschreiben nun die einzelnen Prozeduren.

SubWord bekommt als Eingabe ein Wort. Dieses Wort kann als Folge (b_0, b_1, b_2, b_3) von Bytes dargestellt werden. Auf jedes dieser Bytes wird die Funktion SubBytes angewendet. Jedes dieser Bytes wird gemäß (6.3) transformiert. Die Folge

$$(b_0, b_1, b_2, b_3) \leftarrow (Ab_0^{-1} + c, Ab_1^{-1} + c, Ab_2^{-1} + c, Ab_3^{-1} + c) \tag{6.11}$$

der transformierten Bytes wird zurückgegeben.

Die Funktion RotWord erhält als Eingabe ebenfalls ein Wort (b_0, b_1, b_2, b_3). Die Ausgabe ist

$$(b_0, b_1, b_2, b_3) \leftarrow (b_1, b_2, b_3, b_0). \tag{6.12}$$

Außerdem ist

$$\texttt{Rcon}[\texttt{n}] = (\{\texttt{02}\}^{\texttt{n}}, \{\texttt{00}\}, \{\texttt{00}\}, \{\texttt{00}\}). \tag{6.13}$$

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word temp
   i = 0
   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while
   i = Nk
   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Abb. 6.2 Die AES-Funktion KeyExpansion

6.4 Ein Beispiel

Wir präsentieren ein Beispiel für die Anwendung der AES-Chiffre. Das Beispiel stammt von Brian Gladman (brg@gladman.uk.net).

Die Bezeichnungen haben folgende Bedeutung:

`input`	Klartext
`k_sch`	Rundenschlüssel für Runde `r`
`start`	state zu Beginn von Runde `r`
`s_box`	state nach Anwendung der S-Box `SubBytes`
`s_row`	state nach Anwendung von `ShiftRows`
`m_col`	state nach Anwendung von `MixColumns`
`output`	Schlüsseltext

```
PLAINTEXT:    3243f6a8885a308d313198a2e0370734
KEY:          2b7e151628aed2a6abf7158809cf4f3c
ENCRYPT       16 byte block, 16 byte key
R[00].input   3243f6a8885a308d313198a2e0370734
R[00].k_sch   2b7e151628aed2a6abf7158809cf4f3c
R[01].start   193de3bea0f4e22b9ac68d2ae9f84808
R[01].s_box   d42711aee0bf98f1b8b45de51e415230
R[01].s_row   d4bf5d30e0b452aeb84111f11e2798e5
R[01].m_col   046681e5e0cb199a48f8d37a2806264c
R[01].k_sch   a0fafe1788542cb123a339392a6c7605
R[02].start   a49c7ff2689f352b6b5bea43026a5049
R[02].s_box   49ded28945db96f17f39871a7702533b
R[02].s_row   49db873b453953897f02d2f177de961a
R[02].m_col   584dcaf11b4b5aacdbe7caa81b6bb0e5
R[02].k_sch   f2c295f27a96b9435935807a7359f67f
R[03].start   aa8f5f0361dde3ef82d24ad26832469a
R[03].s_box   ac73cf7befc111df13b5d6b545235ab8
R[03].s_row   acc1d6b8efb55a7b1323cfdf457311b5
R[03].m_col   75ec0993200b633353c0cf7cbb25d0dc
R[03].k_sch   3d80477d4716fe3e1e237e446d7a883b
R[04].start   486c4eee671d9d0d4de3b138d65f58e7
R[04].s_box   52502f2885a45ed7e311c807f6cf6a94
R[04].s_row   52a4c89485116a28e3cf2fd7f6505e07
R[04].m_col   0fd6daa9603138bf6fc0106b5eb31301
R[04].k_sch   ef44a541a8525b7fb671253bdb0bad00
R[05].start   e0927fe8c86363c0d9b1355085b8be01
R[05].s_box   e14fd29be8fbfbba35c89653976cae7c
R[05].s_row   e1fb967ce8c8ae9b356cd2ba974ffb53
```

```
R[05].m_col  25d1a9adbd11d168b63a338e4c4cc0b0
R[05].k_sch  d4d1c6f87c839d87caf2b8bc11f915bc
R[06].start  f1006f55c1924cef7cc88b325db5d50c
R[06].s_box  a163a8fc784f29df10e83d234cd503fe
R[06].s_row  a14f3dfe78e803fc10d5a8df4c632923
R[06].m_col  4b868d6d2c4a8980339df4e837d218d8
R[06].k_sch  6d88a37a110b3efddbf98641ca0093fd
R[07].start  260e2e173d41b77de86472a9fdd28b25
R[07].s_box  f7ab31f02783a9ff9b4340d354b53d3f
R[07].s_row  f783403f27433df09bb531ff54aba9d3
R[07].m_col  1415b5bf461615ec274656d7342ad843
R[07].k_sch  4e54f70e5f5fc9f384a64fb24ea6dc4f
R[08].start  5a4142b11949dc1fa3e019657a8c040c
R[08].s_box  be832cc8d43b86c00ae1d44dda64f2fe
R[08].s_row  be3bd4fed4e1f2c80a642cc0da83864d
R[08].m_col  00512fd1b1c889ff54766dcdfa1b99ea
R[08].k_sch  ead27321b58dbad2312bf5607f8d292f
R[09].start  ea835cf00445332d655d98ad8596b0c5
R[09].s_box  87ec4a8cf26ec3d84d4c46959790e7a6
R[09].s_row  876e46a6f24ce78c4d904ad897ecc395
R[09].m_col  473794ed40d4e4a5a3703aa64c9f42bc
R[09].k_sch  ac7766f319fadc2128d12941575c006e
R[10].start  eb40f21e592e38848ba113e71bc342d2
R[10].s_box  e9098972cb31075f3d327d94af2e2cb5
R[10].s_row  e9317db5cb322c723d2e895faf090794
R[10].k_sch  d014f9a8c9ee2589e13f0cc8b6630ca6
R[10].output 3925841d02dc09fbdc118597196a0b32
```

6.5 `InvCipher`

Die Entschlüsselung der Rijndael-Chiffre wird von der Funktion `InvCipher` besorgt, die in Abb. 6.3 dargestellt ist. Die Spezifikation der Funktionen `InvShiftRows` und `InvSubBytes` ergibt sich aus der Spezifikation von `ShiftRows` und `SubBytes`.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]
   state = in
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])
   out = state
end
```

Abb. 6.3 Die AES-Funktion `InvCipher`

6.6 Übungen

Übung 6.1 Stellen Sie die AES-S-Box wie in Tab. 17.1 dar.

Übung 6.2 Beschreiben Sie die Funktionen `InvShiftRows`, `InvSubBytes` und `InvMixColumns`.

Übung 6.3 Entschlüsseln Sie den Schlüsseltext aus Abschn. 6.4 mit `InvCipher`.