CHAPTER 6 AES

In 1997 the National Institute of Standards and Technology (NIST) initiated the selection process for the successor of DES. One of the submissions was the Rijndael cipher. It is named after its inventors Rijmen and Daemen. On November 26, 2001 this encryption scheme has been standardized as the Advanced Encryption Standard (AES) [1].

AES is a block cipher with alphabet $\mathbb{Z}_2$. It is a special case of the Rijndael cipher. In the Rijndael cipher more different block lengths and ciphertext spaces are possible than in AES. Here we describe the Rijndael cipher and AES as a special case.

6.1 Notation

In the description of the the Rijndael cipher we use the following notation.

`Nb` The plaintext and ciphertext blocks consist of `Nb` 32-bit words, $4 \leq$ `Nb` ≤ 8
So the Rijndael block length is $32 *$ `Nb`.
For AES we have `Nb` $= 4$. So the AES block length is 128.

Nk The key consists of Nk 32-bit words, $4 \leq \text{Nk} \leq 8$
So the Rijndael key space is $\mathbb{Z}_2^{32*\text{Nk}}$.
For AES we have Nk = 4,6, or 8.
So the AES key space is $\mathbb{Z}_2^{128}$, $\mathbb{Z}_2^{192}$, or $\mathbb{Z}_2^{256}$.

Nr Number of rounds.

$$\text{For AES we have Nr} = \begin{cases} 10 & \text{for Nk} = 4, \\ 12 & \text{for Nk} = 6, \\ 14 & \text{for Nk} = 8. \end{cases}$$

In the following description the data types `byte` and `word` are used. A `byte` is a bit-vector of length 8. A `word` is a bit-vector of length 32. Plaintext and ciphertext are represented as two-dimensional arrays. Those arrays have for rows and Nb columns. So in the AES algorithm a plaintext and a ciphertext look like this:

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

The Rijndael keys are `word` arrays of length Nk. The Rijndael cipher expands a key `key` using the function `KeyExpansion` to an expanded key `w`. Then a plaintext block `in` is encrypted using the expanded key `w`. The resulting ciphertext is `out`. The encryption function is `Cipher`. In the following sections we first describe the algorithm `Cipher` and then the algorithm `KeyExpansion`.

6.2 Cipher

We describe the function `Cipher`. The input is the plaintext block `byte in[4,Nb]` and the expanded key `word w[Nb*(Nr+1)]`. The output is the ciphertext block `byte out[4,Nb]`. First, the plaintext `in` is copied into the `byte` array `state`. After an initial transformation `state` is transformed using Nr rounds and is then returned as the cipher text. In the first Nr-1 rounds, the transformations `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` are used. In the last round only the transformations `SubBytes`, `ShiftRows` and `AddRoundKey` are applied. The function `AddRoundKey` is also the initial transformation.


```
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
```

FIGURE 6.1 The AES function `Cipher`

In the following sections we describe the transformations in detail.

6.2.1 Identification of Bytes with the Elements of GF(2^8)

Bytes play a crucial role in the Rijndael cipher. They can be written as a pair of hexadecimal numbers.

Example 6.2.1
The pair $\{2F\}$ of hexadecimal numbers corresponds to the pair 0010 1111 of bit-vectors of length four. So that pair represents the byte 00101111. The pair $\{A1\}$ of hexadecimal numbers corresponds to the pair 1010 0001 of bit-vectors. So that pair represents the byte 10100001.

In the Rijndael cipher, bytes are identified with elements of the finite field GF(2^8). As generating polynomial (see section 2.20) the polynomial

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

is used. That polynomial is irreducible over GF(2). So we can write

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

where α satisfies the equation.

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

Hence, the byte

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

corresponds to the element

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

of $\mathrm{GF}(2^8)$. So bytes can be added and multiplied. If a byte is different from zero, it can also be inverted. For the inverse of a byte b we write b^{-1}. For completeness, we set $0^{-1} = 0$.

Example 6.2.2
The byte $b = (0,0,0,0,0,0,1,1)$ corresponds to the field element $\alpha + 1$. As we have seen in example 2.20.4 we have $(\alpha + 1)^{-1} = \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha$. Therefore, $b^{-1} = (1,1,1,1,0,1,1,0)$.

6.2.2 SubBytes

`SubBytes(state)` a non-linear function. It transforms the individual bytes of `state`. This transformation is called S-Box. Each byte of `state` is mapped to

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

with

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

Here b is considered as a bit-vector. Since there are only 2^8 possible arguments, the S-box can be tabulated. Then `SubBytes` can be implemented by table lookups.

Example 6.2.3
We determine the value of the S-box when applied to $b = (0,0,0,0,0,0,1,1)$. By example 6.2.2 we have $b^{-1} = (1,1,1,1,0,1,1,0)$. So $Ab^{-1} + c = (0,1,1,0,0,1,1,1)$

The S-box guarantees the non-linearity of AES.

6.2.3 ShiftRows

Let s be a `state`, that is, a plaintext that has been subject to a few transformations of AES. Write s as a matrix. The entries are bytes. That matrix has 4 rows and `Nb` columns. In the case of AES this is the matrix

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

The function `ShiftRows` applies cyclic left-shifts to the rows of to this matrix. More precisely, this is what `ShiftRows` does:

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

TABLE 6.1 Cyclic left-shift 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

In general, the a left-shift of c_i positions is applied the ith row with c_i from table 6.1.

The effect of this transformation when applied in several rounds is a high diffusion.

6.2.4 `MixColumns`

For $0 \le j < \mathtt{Nb}$ the column

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

of `state` is identified with the polynomial

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

The transformation `MixColumns` is

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

where

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

This can also be viewed as a linear transformation in $\mathrm{GF}(2^8)^4$. In fact, `MixColumns` is

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

The effect of this transformation is diffusion within the columns of state `state`.

6.2.5 `AddRoundKey`

Let $s_0, \ldots, s_{\mathtt{Nb}-1}$ be the columns of `state`. Then the function `AddRoundKey(state, w[l*Nb, (l+1)*Nb-1])` is

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

where $\oplus$ is applied to the individual bits. So the words of the round key are added mod 2 to the columns of `state`. This is a very simple transformation which makes each round key-dependent.

6.3 `KeyExpansion`

The algorithms `KeyExpansion` expands a Rijndael key `key`, which is a `byte`-array of length `4*Nk`, an expanded key `w`, which is a `word`-array of length `Nb*(Nr+1)`. The application of the expanded keys has been explained in section 6.2. Initially, the first `Nk` words of the expanded key `w` are filled with the bytes of `key`. The following words of `word` are generated as explained in the pseudocode of `KeyExpansion`. The function `word` just concatenates its arguments.

We now describe the individual procedures.

The input to `SubWord` is a word. This word can be written as a sequence (b_0, b_1, b_2, b_3) of bytes. To each byte the function `SubBytes` is applied. Each byte is transformed as in (6.3). The sequence

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

of transformed bytes is returned.

The input to `RotWord` is also a word (b_0, b_1, b_2, b_3). The output is

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

Finally, we have

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

```
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
```

FIGURE 6.2 The AES function KeyExpansion

6.4 An Example

We present an example for the application of the AES cipher. The example is due to Brian Gladman.

Notation:

input	plaintext
k_sch	round key for round r
start	state at the beginning of round r
s_box	state after the application of the S-box SubBytes
s_row	state after the application of ShiftRows
m_col	state after the application of MixColumns
output	ciphertext

```
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
xR[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

The Rijndael cipher is decrypted using the function `InvCipher` (see Figure 6.3). The specification of `InvShiftRows` and `InvSubBytes` can be deduced from `ShiftRows` and `SubBytes`.

6.6 Exercises

Exercise 6.6.1
Describe the AES S-box as in Table 16.1.

Exercise 6.6.2
Describe the functions `InvShiftRows`, `InvSubBytes` and `InvMix-Columns`.

Exercise 6.6.3
Decrypt the ciphertext from Section 6.4 with `InvCipher`.


```
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
```

FIGURE 6.3 The AES function `InvCipher`