

C | SecureAES

- High-Performance AES Encryption in C -

Ji Yong-Hyeon



Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

December 16, 2023

Contents

1	Block Cipher AES-128	1
1.1	Overview of Advanced Encryption Standard	1
1.2	Functions and Constants used in AES	2
1.2.1	Key Expansion	2
1.2.2	AddRoundKey	4
1.2.3	SubBytes	5
1.2.4	ShiftRows	5
1.2.5	MixColumns	6
1.3	Code Structure	7
1.4	Detailed Analysis	7
1.4.1	Rcon Array Declaration	7
1.4.2	Function Definition	7
1.4.3	Variable Declarations and Initial Checks	7
1.4.4	Key Expansion Logic	7
A	Additional Data A	8
A.1	Substitution-BOX	8

Chapter 1

Block Cipher AES-128

1.1 Overview of Advanced Encryption Standard

- KeyExpansion : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{1408}$.
- AddRoundKey : $\{0, 1\}^{128} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$.
- SubBytes : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$.
- ShiftRows : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$.
- MixColumns : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$.

Algorithm 1: Encryption of AES-128

Input: block $\text{src} \in \{0, 1\}^{128}$, round-keys $\{rk_i\}_{i=0}^{11}$ ($rk_i \in \{0, 1\}^{128}$)

Output: block $\text{dst} \in \{0, 1\}^{128}$

```
1  $t \leftarrow \text{src};$ 
2  $t \leftarrow \text{AddRoundKey}(t, rk_0);$ 
3 for  $i \leftarrow 1$  to 9 do
4    $t \leftarrow \text{SubBytes}(t);$ 
5    $t \leftarrow \text{ShiftRows}(t);$ 
6    $t \leftarrow \text{MixColumns}(t);$ 
7    $t \leftarrow \text{AddRoundKey}(t, rk_i);$ 
8 end
9  $t \leftarrow \text{SubBytes}(t);$ 
10  $t \leftarrow \text{ShiftRows}(t);$ 
11  $t \leftarrow \text{AddRoundKey}(t, rk_{10});$ 
12  $\text{dst} \leftarrow t;$ 
13 return  $\text{dst};$ 
```

1.2 Functions and Constants used in AES

1.2.1 Key Expansion

- **RotWord** : $\{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$ is defined by

$$\text{RotWord}(X_0 \parallel X_1 \parallel X_2 \parallel X_3) := X_1 \parallel X_2 \parallel X_3 \parallel X_0 \quad \text{for } X_i \in \{0, 1\}^8.$$

Code 1.1: RotWord rotates the input word left by one byte

```
1 u32 RotWord(u32 word) {
2     return (word << 0x08) | (word >> 0x18);
3 }
```

- **SubWord** : $\{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$ is defined by

$$\text{SubWord}(X_0 \parallel X_1 \parallel X_2 \parallel X_3) := s(X_0) \parallel s(X_1) \parallel s(X_2) \parallel s(X_3) \quad \text{for } X_i \in \{0, 1\}^8.$$

Here, $s : \{0, 1\}^8 \rightarrow \{0, 1\}^8$ is the **S-box**.

Code 1.2: SubWord applies the S-box to each byte of the input word

```
1 u32 SubWord(u32 word) {
2     return (u32)s_box[word >> 0x18] << 0x18 |
3         (u32)s_box[(word >> 0x10) & 0xFF] << 0x10 |
4         (u32)s_box[(word >> 0x08) & 0xFF] << 0x08 |
5         (u32)s_box[word & 0xFF];
6 }
```

- **Round Constant rCon**:

The constant $\text{rCon}_i \in \mathbb{F}_{2^8}$ used in generating the i -th round key corresponds to the value of x^{i-1} in the binary finite field \mathbb{F}_{2^8} and is as follows:

i	1	2	3	4	5	6	7	8	9	10
Rcon_i	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36

Code 1.3: rCon Array Declaration

```
1 static const u32 rCon[10] = {
2     0x01000000, 0x02000000, 0x04000000, 0x08000000,
3     0x10000000, 0x20000000, 0x40000000, 0x80000000,
4     0x1b000000, 0x36000000
5 };
```

Algorithm 2: Key Schedule (AES-128)

Input: User key $uk = (uk_0, \dots, uk_{15})$ ($uk_i \in \{0, 1\}^8$); // $uk \in \{0, 1\}^{128}$ is 16-byte

Output: round-keys $\{rk_i\}_{i=0}^{43}$ ($rk_i \in \{0, 1\}^{32}$); // $\{rk_i\}_{i=0}^{43} \in \{0, 1\}^{1408}$ is 176-byte

```

1  $rk_0 \leftarrow uk_0 \parallel uk_1 \parallel uk_2 \parallel uk_3;$ 
2  $rk_1 \leftarrow uk_4 \parallel uk_5 \parallel uk_6 \parallel uk_7;$ 
3  $rk_2 \leftarrow uk_8 \parallel uk_9 \parallel uk_{10} \parallel uk_{11};$ 
4  $rk_3 \leftarrow uk_{12} \parallel uk_{13} \parallel uk_{14} \parallel uk_{15};$ 
5 for  $i = 4$  to 43 do
6    $t \leftarrow rk_{i-1};$ 
7   if  $i \bmod 4 = 0$  then
8     /* SubWord  $\circ$  RotWord :  $\{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$  */
9      $t \leftarrow \text{RotWord}(t);$ 
10     $t \leftarrow \text{SubWord}(t);$ 
11     $t \leftarrow t \oplus (rCon_{i/4} \parallel 0x00 \parallel 0x00 \parallel 0x00);$ 
12  end
13   $rk_i \leftarrow rk_{i-4} \oplus_{32} t;$ 
14 end

```

Code 1.4: AES Key Expansion

```

1 void KeyExpansion(const u8* uKey, u32* rKey) {
2   u32 temp;
3   int i = 0;
4
5   // Copy the input key to the first round key
6   while (i < 4) {
7     rKey[i] = (u32)uKey[4*i] << 0x18 |
8     (u32)uKey[4*i+1] << 0x10 |
9     (u32)uKey[4*i+2] << 0x08 |
10    (u32)uKey[4*i+3];
11    i++;
12  }
13
14  i = 4;
15
16  // Generate the remaining round keys
17  while (i < 44) {
18    temp = rKey[i-1];
19    if (i % 4 == 0) {
20      temp = SubWord(RotWord(temp)) ^ rCon[i/4-1];
21    }
22    rKey[i] = rKey[i-4] ^ temp;
23    i++;
24  }
25 }

```

1.2.2 AddRoundKey

- $\text{AddRoundKey} : \{0, 1\}^{128} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ is defined by

$$\text{AddRoundKey}(\{X_i\}_{i=0}^{15}, \{rk_i\}_{i=0}^3) := \{X_i \oplus_8 uk_i\}_{i=0}^{15}.$$

Code 1.5: AES AddRoundKey

```

1 void AddRoundKey(u8* state, const u32* rKey) {
2     for (int i = 0; i < AES_KEY_SIZE; i++) {
3         // i = 0, 1, 2, 3 => wordIndex = 0
4         // i = 4, 5, 6, 7 => wordIndex = 1
5         // i = 8, 9, 10, 11 => wordIndex = 2
6         // i = 12, 13, 14, 15 => wordIndex = 3
7         int wordIndex = i / 4;
8
9         // i = 0, 1, 2, 3 => bytePosition = 0, 1, 2, 3
10        // i = 4, 5, 6, 7 => bytePosition = 0, 1, 2, 3
11        // i = 8, 9, 10, 11 => bytePosition = 0, 1, 2, 3
12        // i = 12, 13, 14, 15 => bytePosition = 0, 1, 2, 3
13        int bytePosition = i % 4;
14        /*
15         * +-----+-----+-----+-----+
16         * | i      | wordIndex | bytePosition | shiftedWord |
17         * +-----+-----+-----+-----+
18         * | 0-3    | 0        | 0          | rKey[0] >> 0x18 |
19         * |        |        | 1          | rKey[0] >> 0x10 |
20         * |        |        | 2          | rKey[0] >> 0x08 |
21         * |        |        | 3          | rKey[0]         |
22         * +-----+-----+-----+-----+
23         * | 4-7    | 1        | 0          | rKey[1] >> 24   |
24         * |        |        | 1          | rKey[1] >> 16   |
25         * |        |        | 2          | rKey[1] >> 8    |
26         * |        |        | 3          | rKey[1]         |
27         * +-----+-----+-----+-----+
28         * | ...    | ...      | ...        | ...             |
29         * +-----+-----+-----+-----+
30         * | 15     | 3        | 3          | rKey[3]         |
31         * +-----+-----+-----+-----+
32        */
33        u32 shiftedWord =
34            rKey[wordIndex] >> (8 * (3 - bytePosition));
35
36        u8 keyByte = shiftedWord & 0xFF;
37        state[i] ^= keyByte;
38
39        /* Extract the corresponding byte from the round key word */
40        // state[i] ^= (rKey[i / 4] >> (8 * (3 - (i % 4)))) & 0xFF;
41    }
42 }

```


1.2.3 SubBytes

- SubBytes : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ is defined by

$$\text{SubBytes}(\{X_i\}_{i=0}^{15}) = \{s(X_i)\}_{i=0}^{15}.$$

Table 1.1: Substitution Box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82
30
40
50
60
70
80
90
a0
b0
c0
d0	c1
e0	28	...
f0	16

Code 1.6: Byte Substitution

```

1 void SubBytes(u8* state) {
2     for (int i = 0; i < AES_KEY_SIZE; i++) {
3         state[i] = s_box[state[i]];
4     }
5 }

```

1.2.4 ShiftRows

- ShiftRows : $\{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ is defined by

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

 \Rightarrow

X ₀	X ₄	X ₈	X ₁₂
X ₅	X ₉	X ₁₃	X ₁
X ₁₀	X ₁₄	X ₂	X ₆
X ₁₅	X ₃	X ₇	X ₁₁

Code 1.7: ShiftRows

```
1 void ShiftRows(u8* state) {  
2     u8 temp;  
3  
4     // Row 1: shift left by 1  
5     temp = state[1];  
6     state[1] = state[5];  
7     state[5] = state[9];  
8     state[9] = state[13];  
9     state[13] = temp;  
10  
11    // Row 2: shift left by 2  
12    temp = state[2];  
13    state[2] = state[10];  
14    state[10] = temp;  
15    temp = state[6];  
16    state[6] = state[14];  
17    state[14] = temp;  
18  
19    // Row 3: shift left by 3 (or right by 1)  
20    temp = state[15];  
21    state[15] = state[11];  
22    state[11] = state[7];  
23    state[7] = state[3];  
24    state[3] = temp;  
25 }
```

1.2.5 MixColumns

1.3 Code Structure

1. Rcon Array Declaration
2. Function Definition
3. Variable Declarations and Initial Checks
4. Key Expansion Logic

1.4 Detailed Analysis

1.4.1 Rcon Array Declaration

1.4.2 Function Definition

```
int AES_set_encrypt_key(const unsigned char *userKey, const int bits
```

1.4.3 Variable Declarations and Initial Checks

```
u32 *rk;  
int i = 0;  
u32 temp;  
if (!userKey || !key)  
    return -1;  
if (bits != 128 && bits != 192 && bits != 256)  
    return -2;
```

1.4.4 Key Expansion Logic

1. Initial Key Setup
2. Key Expansion based on key size

Appendix A

Additional Data A

A.1 Substitution-BOX

Code A.1: S-Box

```
1  static const u8 s_box[256] = {
2      0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
3      0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
4      0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
5      0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
6      0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
7      0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
8      0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
9      0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
10     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
11     0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
12     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
13     0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
14     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
15     0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
16     0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
17     0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
18     0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
19     0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
20     0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
21     0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
22     0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
23     0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
24     0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
25     0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
26     0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
27     0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
28     0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
29     0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
30     0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
31     0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
32     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
33     0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
```

34

};

Code A.2: Inverse S-Box

```

1  static const u8 inv_s_box[256] = {
2      0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
3      0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
4      0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
5      0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
6      0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
7      0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
8      0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
9      0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
10     0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
11     0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
12     0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
13     0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
14     0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
15     0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
16     0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
17     0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
18     0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
19     0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
20     0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
21     0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
22     0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
23     0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
24     0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
25     0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
26     0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
27     0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
28     0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
29     0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
30     0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
31     0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
32     0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
33     0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
34 };

```