

AES 블록 암호 규격

FDL 김동현 (wlsitudpdf31@kookmin.ac.kr)

- 정의
 - 상태 배열 정의
 - 바이트 간 연산 정의: 덧셈, 곱셈
- AES 규격 및 코드 소개
 - Cipher
 - InvCipher
 - KeyExpansion
- AES Instruction Set
 - AESNI
 - NEON

상태 배열 정의

- AES 블록 암호는 상태(state) 배열이라고 하는 4×4 크기의 2차원 바이트 배열을 다룬다.
- 여기서는 상태 배열을 S 로 표현한다.
 - $S_{r,c}$ 는 S 의 r 번째 행, c 번째 열 바이트를 의미한다.
 - S_i 는 S 의 i 번째 열을 워드(4 바이트)로 표현한 것으로, $S = S_{[0:4]}$ 로 표현할 수 있다.

$$S = \begin{array}{|c|c|c|c|} \hline S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ \hline S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ \hline S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ \hline S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline S_0 & S_1 & S_2 & S_3 \\ \hline \end{array} = S_{[0:4]}$$

연산 정의

- 바이트 b 를 다음과 같이 비트 또는 다항식으로 표현한다.
 - 비트 표현: $b = \{b_7b_6b_5b_4b_3b_2b_1b_0\}$
 - 다항식 표현: $b = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

연산 정의: 덧셈

- 바이트 간의 덧셈은 계수들을 2로 모듈로 하는 다항식 덧셈으로 정의한다.
- 예시) 0x57과 0x83간의 덧셈은 다음과 같이 표현할 수 있다.
 - $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$
- 이는 두 바이트 간의 XOR 연산과 같다.
 - $\{01010111\} \oplus \{10000011\} = \{11010100\}$

연산 정의: 덧셈



```
1  uint8_t gadd(uint8_t a, uint8_t b) {  
2      return a^b;  
3  }
```

연산 정의: 곱셈

- 바이트 간의 곱셈은 다항식 간의 곱셈에서 다음 다항식을 나눈 나머지로 정의한다.

$$x^8 + x^4 + x^3 + x + 1$$

- 이 때, 다항식의 계수는 덧셈과 마찬가지로 2로 나눈 나머지로 계산한다.

- 바이트 b 와 $0x02$ 간의 곱셈은 $xTimes$ 함수를 사용하여 유용하게 계산할 수 있다.
- $xTimes$ 함수는 다음과 같다.

$$xTimes(b) = \begin{cases} \{b_6b_5b_4b_3b_2b_1b_00\}, & \text{if } b_7 = 0, \\ \{b_6b_5b_4b_3b_2b_1b_00\} \oplus \{00011011\}, & \text{if } b_7 = 1 \end{cases}$$

- 바이트와 2의 거듭 제곱 간의 곱셈은 $xTimes$ 를 반복 적용하는 것으로 구현할 수 있다.

연산 정의: 곱셈





```
1  #define xtimes(state) (((state) << 1) ^ (((state) >> 7) * 0x1b))
```

연산 정의: 곱셈

- xTimes를 사용하여 곱셈을 유용하게 계산할 수 있다.
- 예) 0x57과 0x13간의 곱셈은 다음과 같이 계산 가능하다.

$$\begin{aligned} & 0x57 \cdot 0x13 \\ &= 0x57 \cdot (0x01 \oplus 0x02 \oplus 0x10) \\ &= 0x57 \oplus (0x57 \cdot 0x02) \oplus (0x57 \cdot 0x10) \end{aligned}$$


xTimes 1번


xTimes 4번



```
1 void mul_example(uint8_t b, uint8_t c)
2 {
3     return xtimes(b ^ c) ^ c;
4 }
```

0x02 * b + 0x03 * c 예제

AES 규격

- 세 개의 AES 블록 암호 AES-128, AES-192, AES-256은 다음 함수를 사용한다.
 - Cipher : 라운드(Round)라고 하는 일정한 변환 과정을 상태 배열에 연속적으로 적용한다.
 - InvCipher : Cipher의 변환을 역순으로 실행한다.
 - KeyExpansion : 키를 입력 받고, 이를 확장하여 라운드 키를 생성한다.
- 세 개의 AES 알고리즘의 블록 길이 n_b , 키 길이 n_k , 라운드 수 n_r 는 다음과 같다.

	블록 워드 길이 n_b	키 워드 길이 n_k	라운드 수 n_r
AES-128	4 (128-bit)	4 (128-bit)	10
AES-192	4 (128-bit)	6 (192-bit)	12
AES-256	4 (128-bit)	8 (256-bit)	14

Cipher

- Cipher의 입력은 다음과 같다.
 - 16 바이트 입력 배열 I
 - 라운드 수 n_r
 - $4(n_r + 1)$ 워드 라운드 키 R
- Cipher의 출력은 다음과 같다.
 - 16바이트 출력 배열 O
- Cipher는 다음 네 가지 변환을 사용한다.
 - SubBytes, ShiftRows, Mixcolumns, AddRoundKey

Cipher

Algorithm 1 CIPHER

Require: 입력 배열 I , 라운드 수 n_r , 라운드 키 R

Ensure: 출력 배열 O

```
1: procedure CIPHER( $I, n_r, R$ )
2:    $S \leftarrow I$ 
3:    $S \leftarrow \text{ADDRoundKey}(S, R_{[0:4]})$ 
4:   for  $i = 1$  to  $n_r - 1$  do
5:      $S \leftarrow \text{SUBBYTES}(S)$ 
6:      $S \leftarrow \text{SHIFTRows}(S)$ 
7:      $S \leftarrow \text{MixColumns}(S)$ 
8:      $S \leftarrow \text{ADDRoundKey}(S, R_{[4i:4(i+1)]})$ 
9:   end for
10:   $S \leftarrow \text{SUBBYTES}(S)$ 
11:   $S \leftarrow \text{SHIFTRows}(S)$ 
12:   $S \leftarrow \text{ADDRoundKey}(S, R_{[4n_r:4(n_r+1)]})$ 
13:   $O \leftarrow S$ 
14:  return  $O$ 
15: end procedure
```

Cipher

```
1 void aes_cipher(uint8_t *in, uint8_t *out, uint8_t *w) {
2
3     uint8_t state[4*Nb];
4     uint8_t r, i, j;
5
6     for (i = 0; i < 4; i++) {
7         for (j = 0; j < Nb; j++) {
8             state[Nb*i+j] = in[i+4*j];
9         }
10    }
11
12    add_round_key(state, w, 0);
13
14    for (r = 1; r < Nr; r++) {
15        sub_bytes(state);
16        shift_rows(state);
17        mix_columns(state);
18        add_round_key(state, w, r);
19    }
20
21    sub_bytes(state);
22    shift_rows(state);
23    add_round_key(state, w, Nr);
24
25    for (i = 0; i < 4; i++) {
26        for (j = 0; j < Nb; j++) {
27            out[i+4*j] = state[Nb*i+j];
28        }
29    }
30 }
```

AddRoundKey

- AddRoundKey는 상태 배열 S 에 라운드 키 R 를 적용하여 XOR 연산을 수행한다.

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

R_{4i}	R_{4i+1}	R_{4i+2}	R_{4i+3}

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$

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

\oplus

R_{4i+1}

$=$

$S'_{0,1}$
$S'_{1,1}$
$S'_{2,1}$
$S'_{3,1}$

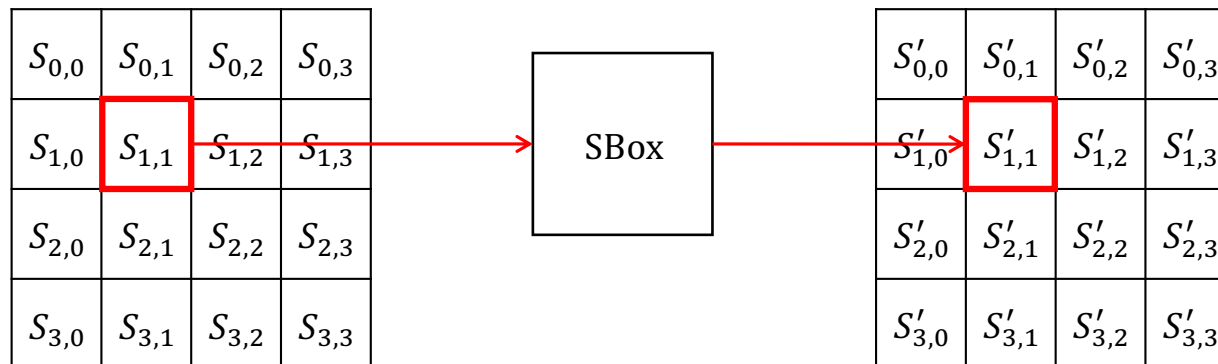
AddRoundKey



```
1 void add_round_key(uint8_t *state, uint8_t *w, uint8_t r) {
2
3     uint8_t c;
4
5     for (c = 0; c < Nb; c++) {
6         state[Nb*0+c] = state[Nb*0+c]^w[4*Nb*r+4*c+0]; //debug, so it works for Nb !=4
7         state[Nb*1+c] = state[Nb*1+c]^w[4*Nb*r+4*c+1];
8         state[Nb*2+c] = state[Nb*2+c]^w[4*Nb*r+4*c+2];
9         state[Nb*3+c] = state[Nb*3+c]^w[4*Nb*r+4*c+3];
10    }
11 }
```

SubBytes

- SubBytes는 상태 행렬 S 각 바이트를 Sbox라고 불리는 치환 테이블에 독립적으로 적용하는 변환이다.



SubBytes

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

위 사진은 Sbox 테이블을 나타낸다. x는 입력 바이트 상위 4비트, y는 하위 4 비트를 나타낸다.

예) Sbox(0x53) = 0xed

SubBytes



```
1  void sub_bytes(uint8_t *state) {  
2  
3      uint8_t i, j;  
4  
5      for (i = 0; i < 4; i++) {  
6          for (j = 0; j < Nb; j++) {  
7              // s_box row: yyyy ----  
8              // s_box col: ---- xxxx  
9              // s_box[16*(yyyy) + xxxx] == s_box[yyyyxxxx]  
10             state[Nb*i+j] = s_box[state[Nb*i+j]];  
11         }  
12     }  
13 }
```

SubBytes

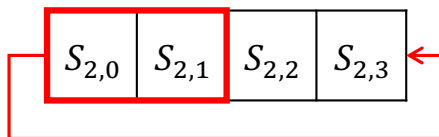
```
1 static uint8_t s_box[256] = {
2     // 0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f
3     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, // 0
4     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, // 1
5     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, // 2
6     0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, // 3
7     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, // 4
8     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, // 5
9     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, // 6
10    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, // 7
11    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, // 8
12    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, // 9
13    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, // a
14    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, // b
15    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, // c
16    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, // d
17    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, // e
18    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16}; // f
```

ShiftRows

- ShiftRows는 상태 배열 S 마지막 세 행에 속한 바이트가 순환 이동하는 변환이다.
- 각 행은 인덱스 r 만큼 왼쪽으로 이동하며, 밀려난 바이트는 행의 오른쪽 끝으로 순환한다.

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

$r = 2$ 이므로, 왼쪽으로 두 칸 순환



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

ShiftRows

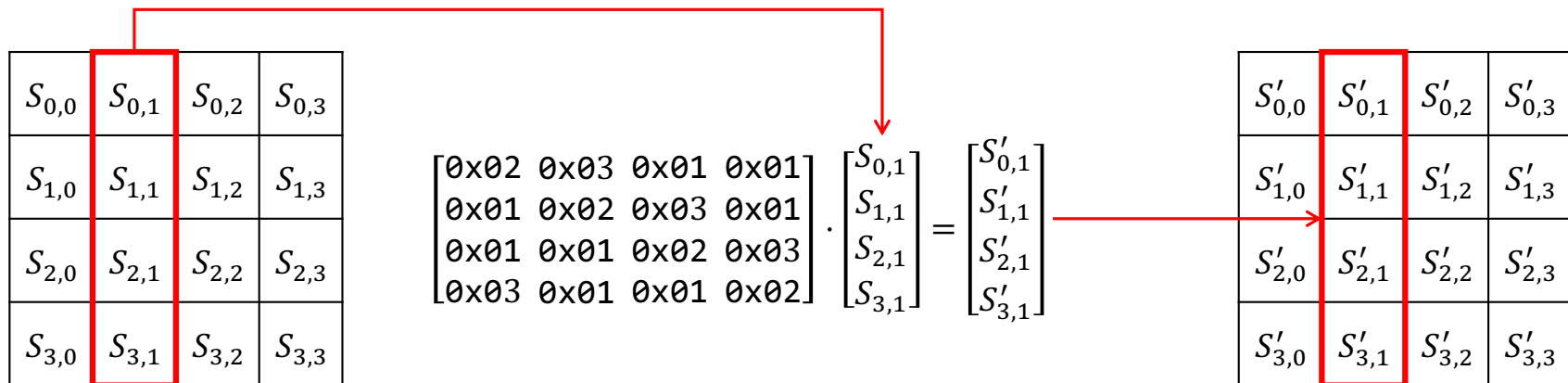


```
1 void shift_rows(uint8_t *state) {
2
3     uint8_t i, k, s, tmp;
4
5     for (i = 1; i < 4; i++) {
6         // shift(1,4)=1; shift(2,4)=2; shift(3,4)=3
7         // shift(r, 4) = r;
8         s = 0;
9         while (s < i) {
10             tmp = state[Nb*i+0];
11
12             for (k = 1; k < Nb; k++) {
13                 state[Nb*i+k-1] = state[Nb*i+k];
14             }
15
16             state[Nb*i+Nb-1] = tmp;
17             s++;
18         }
19     }
20 }
```

MixColumns

- MixColumns는 상태 행렬 S 각 열을 고정된 행렬과 곱하는 변환이다.

- 고정 행렬은
$$\begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix}$$
이다.



MixColumns



```
1  void mix_columns(uint8_t *state) {
2
3      uint8_t a[] = {0x02, 0x01, 0x01, 0x03}; // a(x) = {02} + {01}x + {01}x2 + {03}x3
4      uint8_t i, j, col[4], res[4];
5
6      for (j = 0; j < Nb; j++) {
7          for (i = 0; i < 4; i++) {
8              col[i] = state[Nb*i+j];
9          }
10
11         coef_mult(a, col, res);
12
13         for (i = 0; i < 4; i++) {
14             state[Nb*i+j] = res[i];
15         }
16     }
17 }
```

MixColumns



```
1 void coef_mult(uint8_t *a, uint8_t *b, uint8_t *d) {  
2  
3     d[0] = gmult(a[0],b[0])^gmult(a[3],b[1])^gmult(a[2],b[2])^gmult(a[1],b[3]);  
4     d[1] = gmult(a[1],b[0])^gmult(a[0],b[1])^gmult(a[3],b[2])^gmult(a[2],b[3]);  
5     d[2] = gmult(a[2],b[0])^gmult(a[1],b[1])^gmult(a[0],b[2])^gmult(a[3],b[3]);  
6     d[3] = gmult(a[3],b[0])^gmult(a[2],b[1])^gmult(a[1],b[2])^gmult(a[0],b[3]);  
7 }
```

MixColumns



```
1  /* first column */
2  state[0] = xtimes(t[0] ^ t[1]) ^ t[1] ^ t[2] ^ t[3];
3  state[1] = t[0] ^ xtimes(t[1] ^ t[2]) ^ t[2] ^ t[3];
4  state[2] = t[0] ^ t[1] ^ xtimes(t[2] ^ t[3]) ^ t[3];
5  state[3] = t[0] ^ t[1] ^ t[2] ^ xtimes(t[3] ^ t[0]);
6
```

InvCipher

- InvCipher는 Cipher 변환을 역순으로 실행한다.
- InvCipher는 다음 함수를 포함한다.
 - InvSubBytes
 - InvShiftrows
 - InvMixColumns
 - AddRoundKey

InvCipher

Algorithm 2 INV CIPHER

Require: I, n_r, R

$\triangleright R = \text{KEYEXPANSION}(key)$

Ensure: O

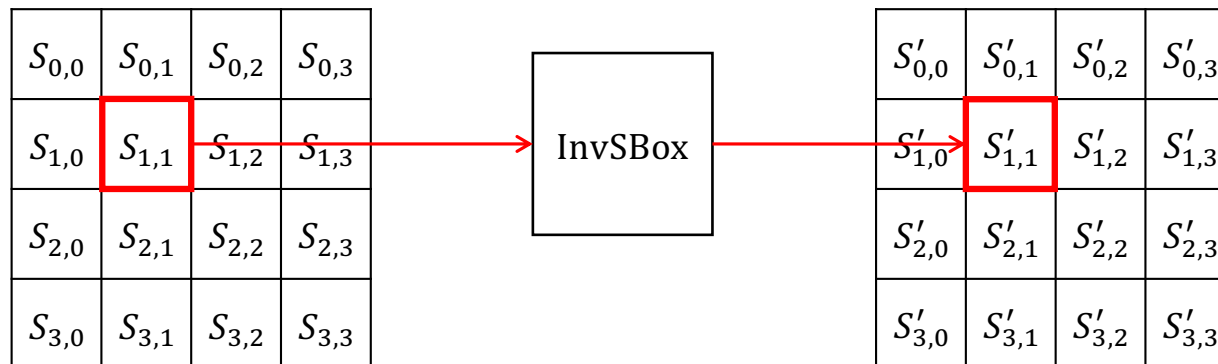
```
1: procedure INV CIPHER( $I, n_r, R$ )
2:    $S \leftarrow I$ 
3:    $S \leftarrow \text{ADDROUNDKEY}(S, R_{[4n_r:4(n_r+1)]})$ 
4:   for  $i = 1$  to  $n_r - 1$  do
5:      $S \leftarrow \text{INVSHIFTROWS}(S)$ 
6:      $S \leftarrow \text{INVSUBBYTES}(S)$ 
7:      $S \leftarrow \text{ADDROUNDKEY}(S, R_{[4i:4(i+1)]})$ 
8:      $S \leftarrow \text{INVMIXCOLUMNS}(S)$ 
9:   end for
10:   $S \leftarrow \text{INVSHIFTROWS}(S)$ 
11:   $S \leftarrow \text{INVSUBBYTES}(S)$ 
12:   $S \leftarrow \text{ADDROUNDKEY}(S, R_{[0:4]})$ 
13:   $O \leftarrow S$ 
14:  return  $O$ 
15: end procedure
```

InvCipher

```
1 void aes_inv_cipher(uint8_t *in, uint8_t *out, uint8_t *w) {
2
3     uint8_t state[4*Nb];
4     uint8_t r, i, j;
5
6     for (i = 0; i < 4; i++) {
7         for (j = 0; j < Nb; j++) {
8             state[Nb*i+j] = in[i+4*j];
9         }
10    }
11
12    add_round_key(state, w, Nr);
13
14    for (r = Nr-1; r >= 1; r--) {
15        inv_shift_rows(state);
16        inv_sub_bytes(state);
17        add_round_key(state, w, r);
18        inv_mix_columns(state);
19    }
20
21    inv_shift_rows(state);
22    inv_sub_bytes(state);
23    add_round_key(state, w, 0);
24
25    for (i = 0; i < 4; i++) {
26        for (j = 0; j < Nb; j++) {
27            out[i+4*j] = state[Nb*i+j];
28        }
29    }
30 }
31
```

InvSubBytes

- InvSubBytes는 SubBytes의 역연산이다.
- 상태 배열 S 각 바이트에 InvSbox 치환 테이블을 적용한다.




InvSubBytes

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	e9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

위 사진은 InvSbox 테이블을 나타낸다. 예) $\text{InvSbox}(0xed) = 0x53$

InvSubBytes



```
1  void inv_sub_bytes(uint8_t *state) {  
2  
3      uint8_t i, j;  
4  
5      for (i = 0; i < 4; i++) {  
6          for (j = 0; j < Nb; j++) {  
7              state[Nb*i+j] = inv_s_box[state[Nb*i+j]];  
8          }  
9      }  
10 }
```

InvSubBytes



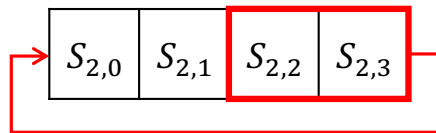
```
1  static uint8_t inv_s_box[256] = {
2      // 0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f
3      0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, // 0
4      0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, // 1
5      0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e, // 2
6      0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, // 3
7      0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, // 4
8      0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84, // 5
9      0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, // 6
10     0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, // 7
11     0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, // 8
12     0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, // 9
13     0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, // a
14     0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4, // b
15     0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, // c
16     0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, // d
17     0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61, // e
18     0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d}; // f
19
```

InvShiftRows

- InvShiftRows는 ShiftRows의 역연산이다.
- 바이트를 해당 행에서 r 칸 씩 오른쪽으로 이동하고, 밀려난 r 개의 바이트를 왼쪽 끝으로 순환 이동한다.

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

$r = 2$ 이므로, 오른쪽으로 두 칸 순환



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

InvShiftRows

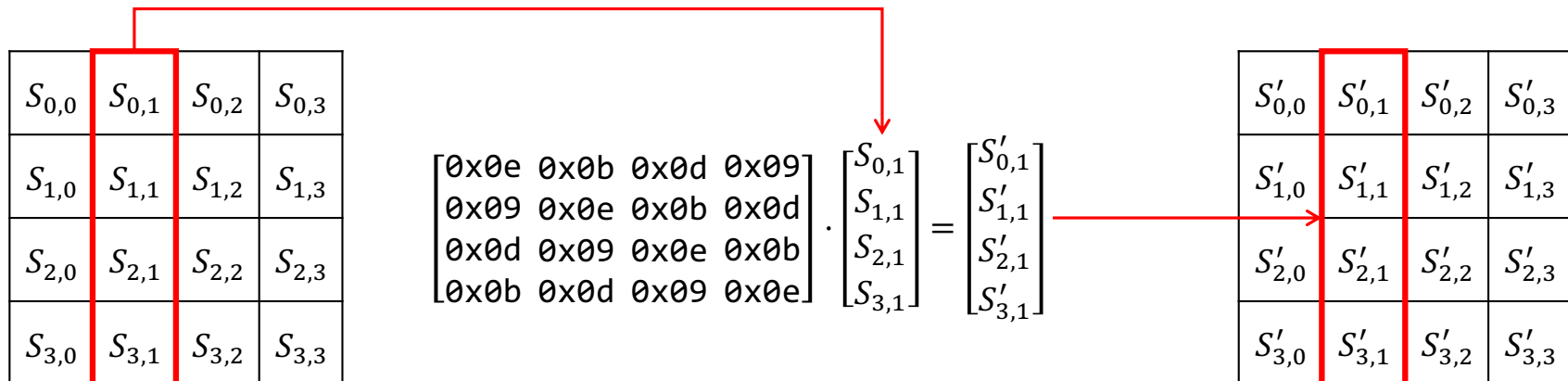


```
1 void inv_shift_rows(uint8_t *state) {
2
3     uint8_t i, k, s, tmp;
4
5     for (i = 1; i < 4; i++) {
6         s = 0;
7         while (s < i) {
8             tmp = state[Nb*i+Nb-1];
9
10            for (k = Nb-1; k > 0; k--) {
11                state[Nb*i+k] = state[Nb*i+k-1];
12            }
13
14            state[Nb*i+0] = tmp;
15            s++;
16        }
17    }
18 }
```

InvMixColumns

- InvMixColumns는 MixColumns의 역연산이다.

- 상태 배열 S 각 열에 고정 행렬 $\begin{bmatrix} 0x0e & 0x0b & 0x0d & 0x09 \\ 0x09 & 0x0e & 0x0b & 0x0d \\ 0x0d & 0x09 & 0x0e & 0x0b \\ 0x0b & 0x0d & 0x09 & 0x0e \end{bmatrix}$ 을 곱한다.



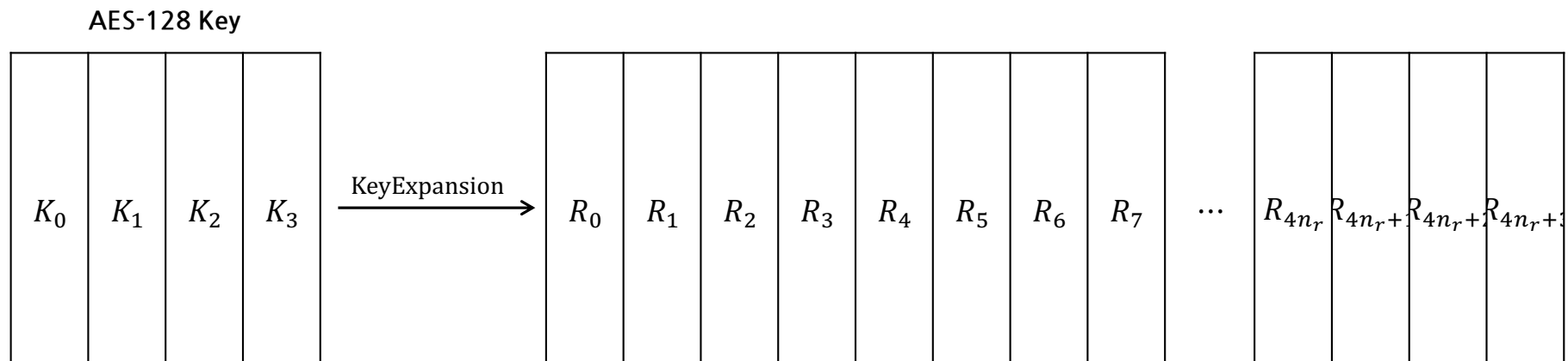
InvMixColumns



```
1 void inv_mix_columns(uint8_t *state) {
2
3     uint8_t a[] = {0x0e, 0x09, 0x0d, 0x0b}; // a(x) = {0e} + {09}x + {0d}x2 + {0b}x3
4     uint8_t i, j, col[4], res[4];
5
6     for (j = 0; j < Nb; j++) {
7         for (i = 0; i < 4; i++) {
8             col[i] = state[Nb*i+j];
9         }
10
11         coef_mult(a, col, res);
12
13         for (i = 0; i < 4; i++) {
14             state[Nb*i+j] = res[i];
15         }
16     }
17 }
```

KeyExpansion

- KeyExpansion은 키 K 를 사용하여 $4(n_r + 1)$ 개의 워드를 생성하는 함수이다.
- 이렇게 생성한 $4(n_r + 1)$ 개의 워드를 라운드 키 R 이라고 한다.




KeyExpansion

- KeyExpansion은 $j = 0, 1, \dots, 10$ 에 대해 $Rcon[j]$ 로 나타내는 고정 워드를 사용한다.
 - 이를 라운드 상수(Round Constant)라고 하며, $Rcon[j]$ 값은 다음과 같다.

Rcon[1]:	[0x01, 0x00, 0x00, 0x00]	Rcon[6]:	[0x20, 0x00, 0x00, 0x00]
Rcon[2]:	[0x02, 0x00, 0x00, 0x00]	Rcon[7]:	[0x40, 0x00, 0x00, 0x00]
Rcon[3]:	[0x04, 0x00, 0x00, 0x00]	Rcon[8]:	[0x80, 0x00, 0x00, 0x00]
Rcon[4]:	[0x08, 0x00, 0x00, 0x00]	Rcon[9]:	[0x1b, 0x00, 0x00, 0x00]
Rcon[5]:	[0x10, 0x00, 0x00, 0x00]	Rcon[10]:	[0x36, 0x00, 0x00, 0x00]


- KeyExpansion은 다음 두 변환을 사용한다. (입력 워드 a 를 $a = [a_0, a_1, a_2, a_3]$ 라고 하자.)
 - $\text{SubWord}([a_0, a_1, a_2, a_3]) = [\text{Sbox}(a_0), \text{Sbox}(a_1), \text{Sbox}(a_2), \text{Sbox}(a_3)]$.
 - $\text{RotWord}([a_0, a_1, a_2, a_3]) = [a_1, a_2, a_3, a_0]$.

KeyExpansion



```
1  void sub_word(uint8_t *w) {  
2  
3      uint8_t i;  
4  
5      for (i = 0; i < 4; i++) {  
6          w[i] = s_box[w[i]];  
7      }  
8  }
```

KeyExpansion



```
1  void rot_word(uint8_t *w) {  
2  
3      uint8_t tmp;  
4      uint8_t i;  
5  
6      tmp = w[0];  
7  
8      for (i = 0; i < 3; i++) {  
9          w[i] = w[i+1];  
10     }  
11  
12     w[3] = tmp;  
13 }
```

KeyExpansion



```
1  static const uint8_t rcon[11] = {  
2      0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36};  
3
```

KeyExpansion

Algorithm 3 KEYEXPANSION

Require: $\nexists K$

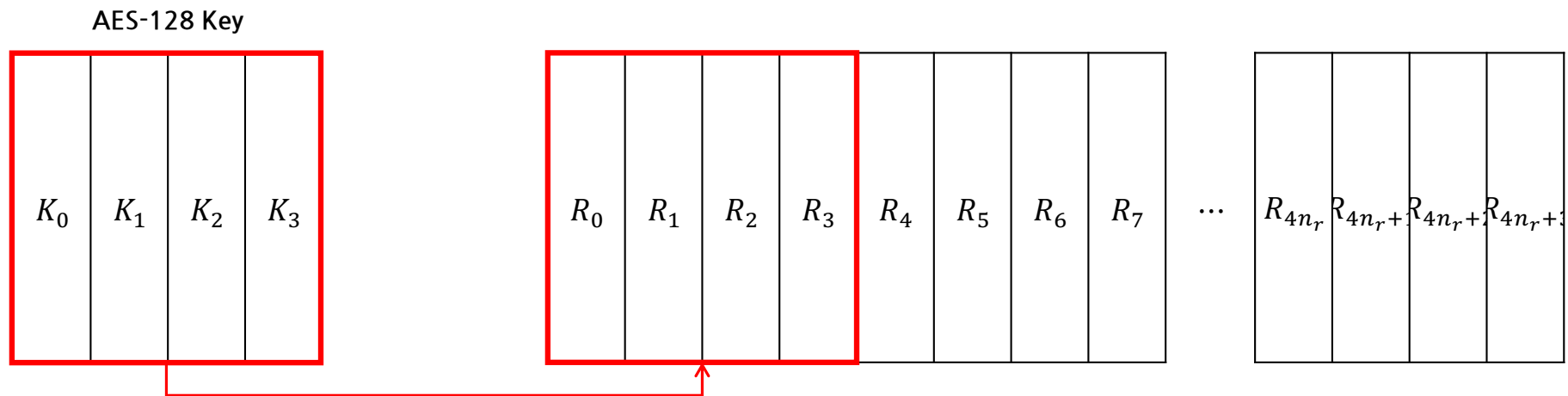
Ensure: 라운드 키 R

```
1: procedure KEYEXPANSION( $K$ )
2:   for  $i = 0$  to  $n_k - 1$  do
3:      $R_i \leftarrow K_i$ 
4:   end for
5:   for  $i = n_k$  to  $4n_r + 3$  do
6:      $t \leftarrow R_{i-1}$ 
7:     if  $i \bmod n_k = 0$  then
8:        $t \leftarrow \text{SUBWORD}(\text{ROTWORD}(t)) \oplus \text{Rcon}[i/n_k]$ 
9:     else if  $n_k > 6$  and  $i \bmod n_k = 4$  then
10:       $t \leftarrow \text{SUBWORD}(t)$ 
11:    end if
12:     $R_i \leftarrow R_{i-n_k} \oplus t$ 
13:  end for
14:  return  $R$ 
15: end procedure
```

▷ AES-256에서만 동작

KeyExpansion

- 먼저, 라운드 키 R 의 처음 n_k 워드는 키 K 로 결정한다.



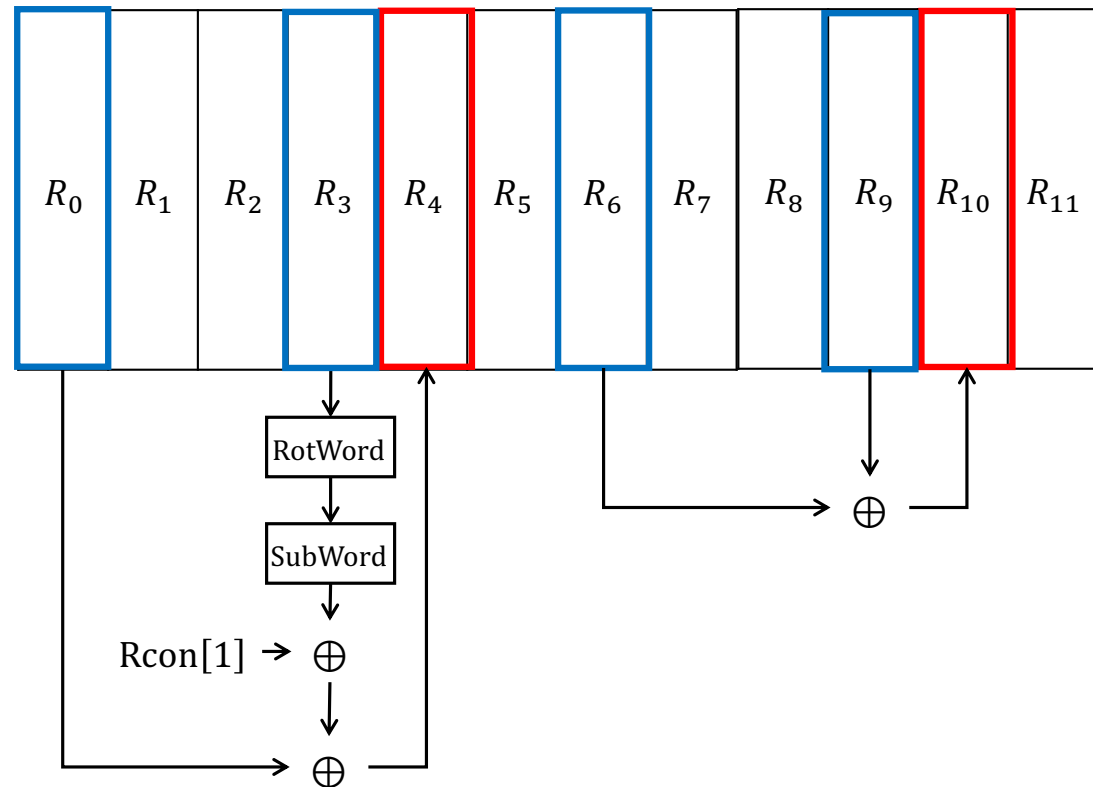
KeyExpansion

- 라운드 키 R 의 인덱스 i 가 n_k 의 배수이면 다음을 계산한다.

- $R_i = R_{i-n_k} \oplus \text{SubWord}(\text{RotWord}(R_{i-1})) \oplus \text{Rcon}[i/n_k].$

- 아니라면, 다음을 계산한다.

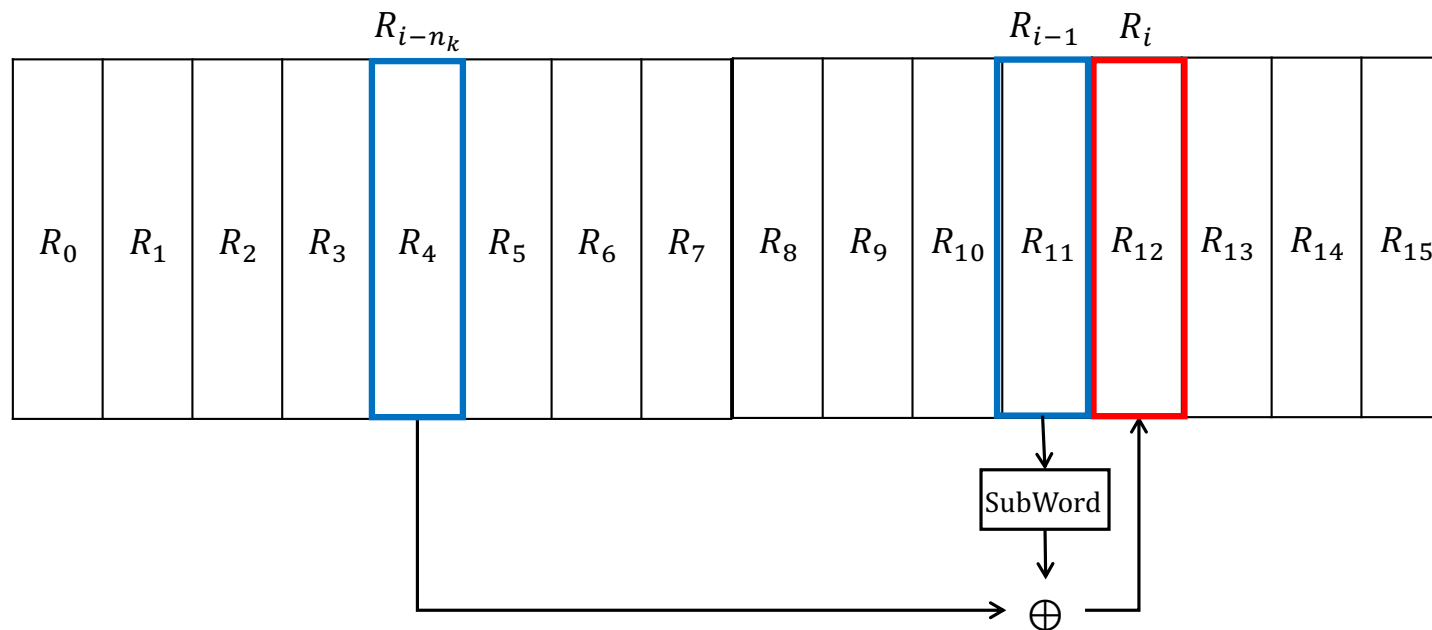
- $R_i = R_{i-n_k} \oplus R_{i-1}.$



KeyExpansion

• 단, AES-256에서 $i + 4$ 가 8의 배수일 때는 다음과 같이 계산한다.

- $R_i = R_{i-n_k} \oplus \text{SubWord}(R_{i-1})$



KeyExpansion

Algorithm 3 KEYEXPANSION

Require: $\nexists K$

Ensure: 라운드 키 R

```
1: procedure KEYEXPANSION( $K$ )
2:   for  $i = 0$  to  $n_k - 1$  do
3:      $R_i \leftarrow K_i$ 
4:   end for
5:   for  $i = n_k$  to  $4n_r + 3$  do
6:      $t \leftarrow R_{i-1}$ 
7:     if  $i \bmod n_k = 0$  then
8:        $t \leftarrow \text{SUBWORD}(\text{ROTWORD}(t)) \oplus \text{Rcon}[i/n_k]$ 
9:     else if  $n_k > 6$  and  $i \bmod n_k = 4$  then
10:       $t \leftarrow \text{SUBWORD}(t)$ 
11:    end if
12:     $R_i \leftarrow R_{i-n_k} \oplus t$ 
13:  end for
14:  return  $R$ 
15: end procedure
```

▷ AES-256에서만 동작

KeyExpansion

```
1 void aes_key_expansion(uint8_t *key, uint8_t *w) {
2
3     uint8_t tmp[4];
4     uint8_t i;
5     uint8_t len = Nb*(Nr+1);
6
7     for (i = 0; i < Nk; i++) {
8         w[4*i+0] = key[4*i+0];
9         w[4*i+1] = key[4*i+1];
10        w[4*i+2] = key[4*i+2];
11        w[4*i+3] = key[4*i+3];
12    }
13
14    for (i = Nk; i < len; i++) {
15        tmp[0] = w[4*(i-1)+0];
16        tmp[1] = w[4*(i-1)+1];
17        tmp[2] = w[4*(i-1)+2];
18        tmp[3] = w[4*(i-1)+3];
19
20        if (i%Nk == 0) {
21
22            rot_word(tmp);
23            sub_word(tmp);
24            coef_add(tmp, Rcon(i/Nk), tmp);
25
26        } else if (Nk > 6 && i%Nk == 4) {
27
28            sub_word(tmp);
29
30        }
31
32        w[4*i+0] = w[4*(i-Nk)+0]^tmp[0];
33        w[4*i+1] = w[4*(i-Nk)+1]^tmp[1];
34        w[4*i+2] = w[4*(i-Nk)+2]^tmp[2];
35        w[4*i+3] = w[4*(i-Nk)+3]^tmp[3];
36    }
37 }
```

- AES Instruction Set은 AES 암호화 및 복호화 연산을 효율적으로 수행하도록 설계된 명령어들의 집합이다.
- 현대 프로세서에서는 AES Instruction Set을 일반적으로 제공하고 있다.
 - Window: #include <wmmintrin.h>
 - MacOS: #include <arm_neon.h>

NEON



```
1 void AES256_ECB(unsigned char *key,
2     unsigned char *ctr,
3     unsigned char *buffer)
4 {
5
6     uint8_t    rkey[15][16];
7     uint8x16_t buffer128 = vld1q_u8(ctr);
8
9     aes256_key_schedule(rkey, key);
10
11     for (int i = 0; i < 13; i++)
12     {
13         buffer128 = veorq_u8(buffer128, vld1q_u8(rkey[i]));
14         buffer128 = vaeseq_u8(buffer128, vdupq_n_u8(0));
15         buffer128 = vaesmcq_u8(buffer128);
16     }
17     buffer128 = veorq_u8(buffer128, vld1q_u8(rkey[13]));
18     buffer128 = vaeseq_u8(buffer128, vdupq_n_u8(0));
19     buffer128 = veorq_u8(buffer128, vld1q_u8(rkey[14]));
20
21     vst1q_u8(buffer, buffer128);
22 }
```

AESNI



```
1 void AES256_ECB(unsigned char *key,  
2     unsigned char *ctr,  
3     unsigned char *buffer)  
4 {  
5     uint8_t    rkey[15][16];  
6     aes256_key_schedule(rkey, key);  
7  
8     __m128i state128 = _mm_loadu_si128((__m128i*)ctr);  
9     state128 = _mm_xor_si128(state128, _mm_loadu_si128((__m128i*)rkey[0]));  
10  
11     for (int i = 1; i < 14; i++) {  
12         state128 = _mm_aesenc_si128(state128, _mm_loadu_si128((__m128i*)rkey[i]));  
13     }  
14     state128 = _mm_aesenclast_si128(state128, _mm_loadu_si128((__m128i*)rkey[14]));  
15     _mm_storeu_si128((__m128i*)buffer, state128);  
16 }
```

AES 블록 암호 규격

FDL 김동현 (wlsitudpdf31@kookmin.ac.kr)