

AES 블록 암호 규격

FDL 김동현 (wlsitudpdf31@kookmin.ac.kr)

목차

- 정의

- 상태 배열 정의
- 바이트 간 연산 정의: 덧셈, 곱셈

- AES 규격

- Cipher
- InvCipher
- KeyExpansion

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: 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.

- 바이트의 제곱(즉, 바이트 b 와 $0x02$ 간의 곱셈)은 $xTimes$ 함수를 사용하여 유용하게 계산할 수 있다.
- $xTimes$ 함수는 다음과 같다.


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

- 바이트의 거듭 제곱은 $xTimes$ 를 반복 적용하는 것으로 구현할 수 있다.

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

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:      $S \leftarrow \text{SHIFTRROWS}(S)$ 
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:   $S \leftarrow \text{SHIFTRROWS}(S)$ 
12:   $S \leftarrow \text{ADDROUNDKEY}(S, R_{[4n_r:4(n_r+1)]})$ 
13:   $O \leftarrow S$ 
14:  return  $O$ 
15: end procedure
```

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

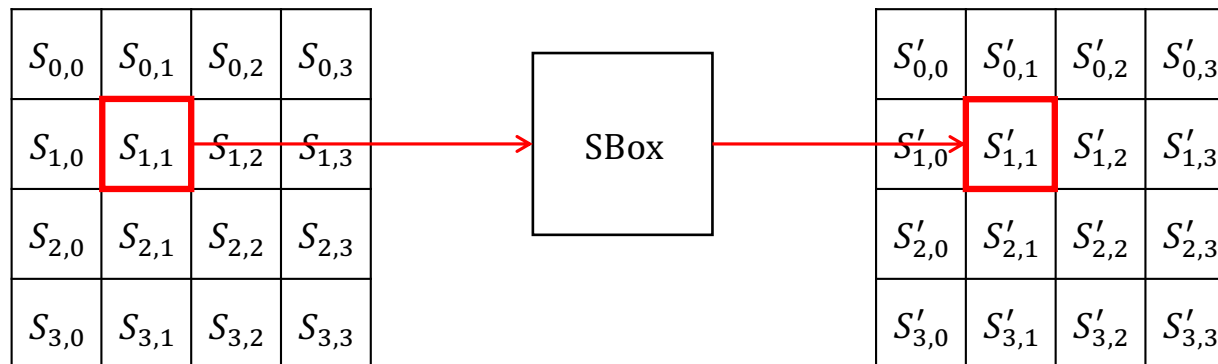
R_{4i+1}

$=$

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

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.

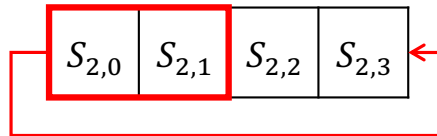
예) Sbox(0x53) = 0xed

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

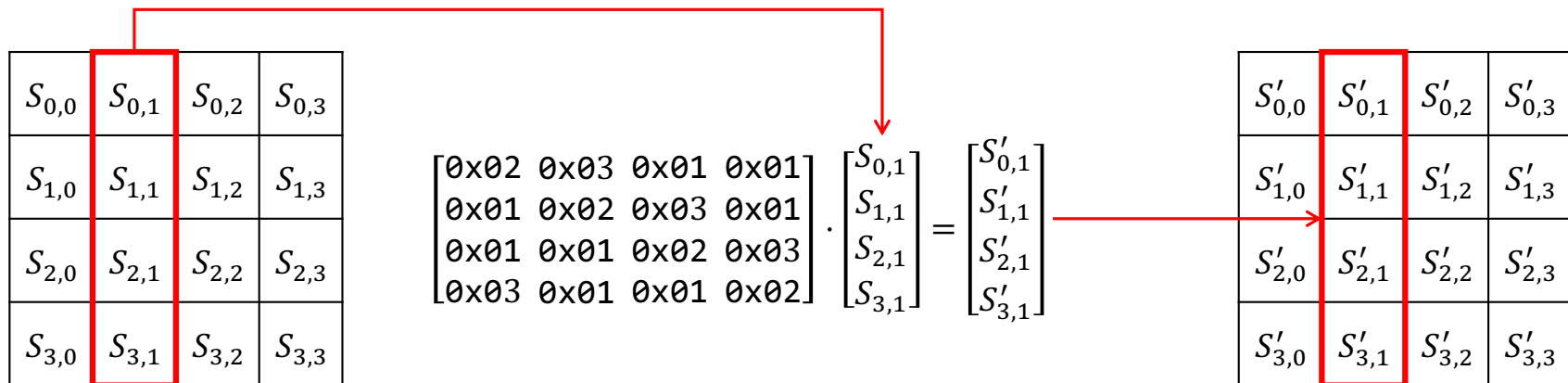


$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

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.



InvCipher

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

InvCipher

Algorithm 2 INV CIPHER

Require: I, n_r, R

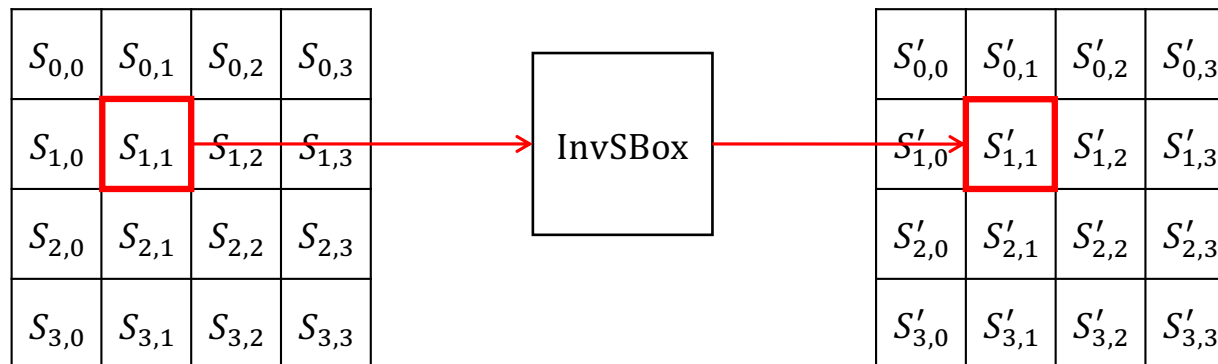
$\triangleright R = \text{KEYEXPANSION}(key)$

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:      $S \leftarrow \text{INV SUBBYTES}(S)$ 
6:      $S \leftarrow \text{INV SHIFTROWS}(S)$ 
7:      $S \leftarrow \text{INV MIXCOLUMNS}(S)$ 
8:      $S \leftarrow \text{ADDROUNDKEY}(S, R_{[4i:4(i+1)]})$ 
9:   end for
10:   $S \leftarrow \text{INV SUBBYTES}(S)$ 
11:   $S \leftarrow \text{INV SHIFTROWS}(S)$ 
12:   $S \leftarrow \text{ADDROUNDKEY}(S, R_{[0:4]})$ 
13:   $O \leftarrow S$ 
14:  return  $O$ 
15: end procedure
```

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

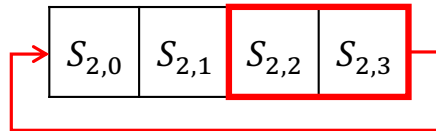
위 사진은 InvSbox 테이블을 나타낸다. 예) $\text{InvSbox}(0xed) = 0x53$

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

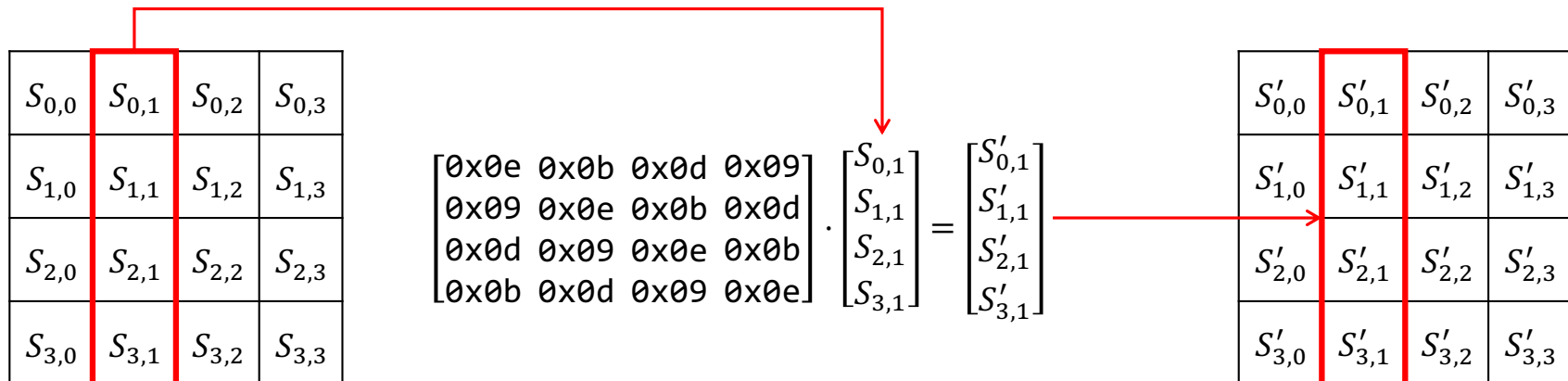


$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,3}$	$S_{1,0}$	$S_{1,1}$	$S_{1,2}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,0}$

InvMixColumns

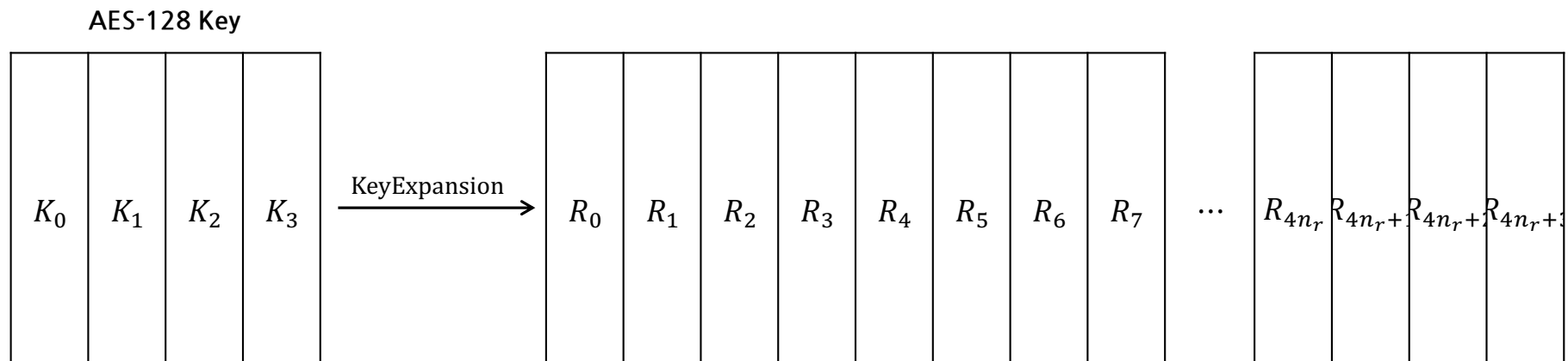
- InvMixColumns는 MixColumns의 역연산이다.

- 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.



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.

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

- 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

Algorithm 3 KEYEXPANSION

Require: $\nexists K$

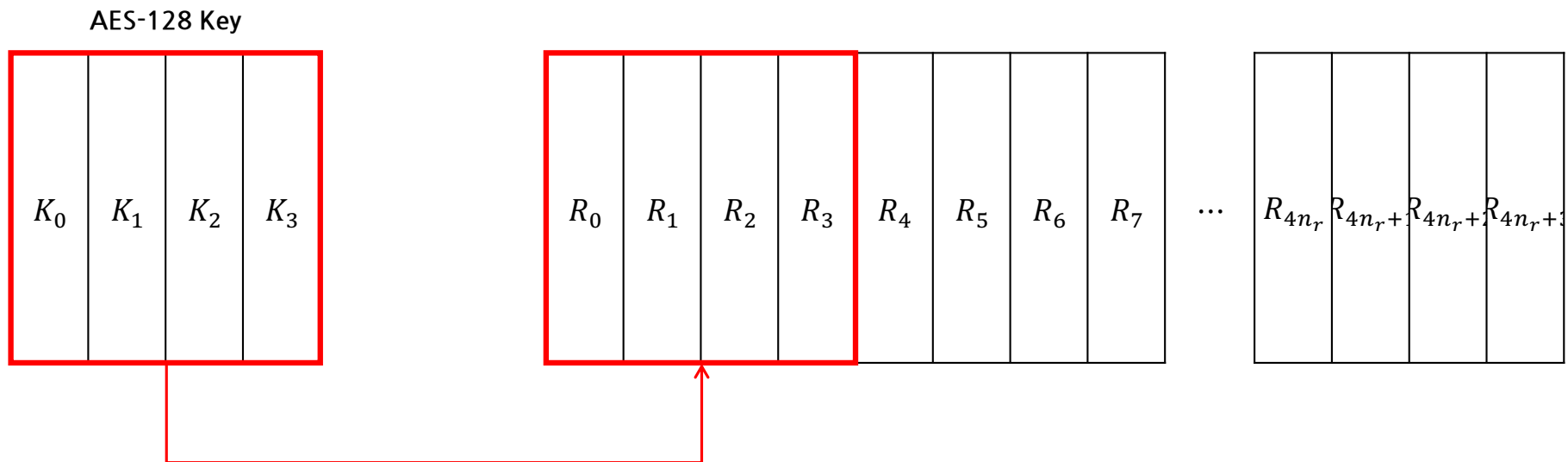
Ensure: 라운드 키 R

```
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.



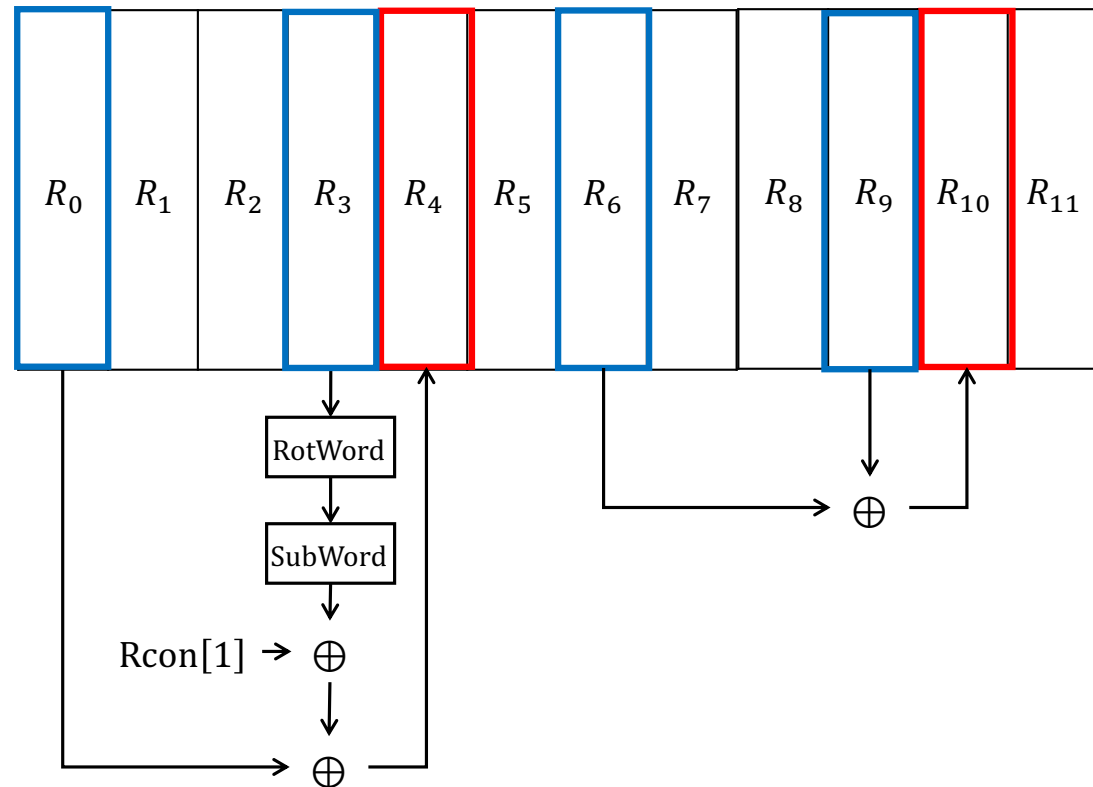
KeyExpansion

- 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.

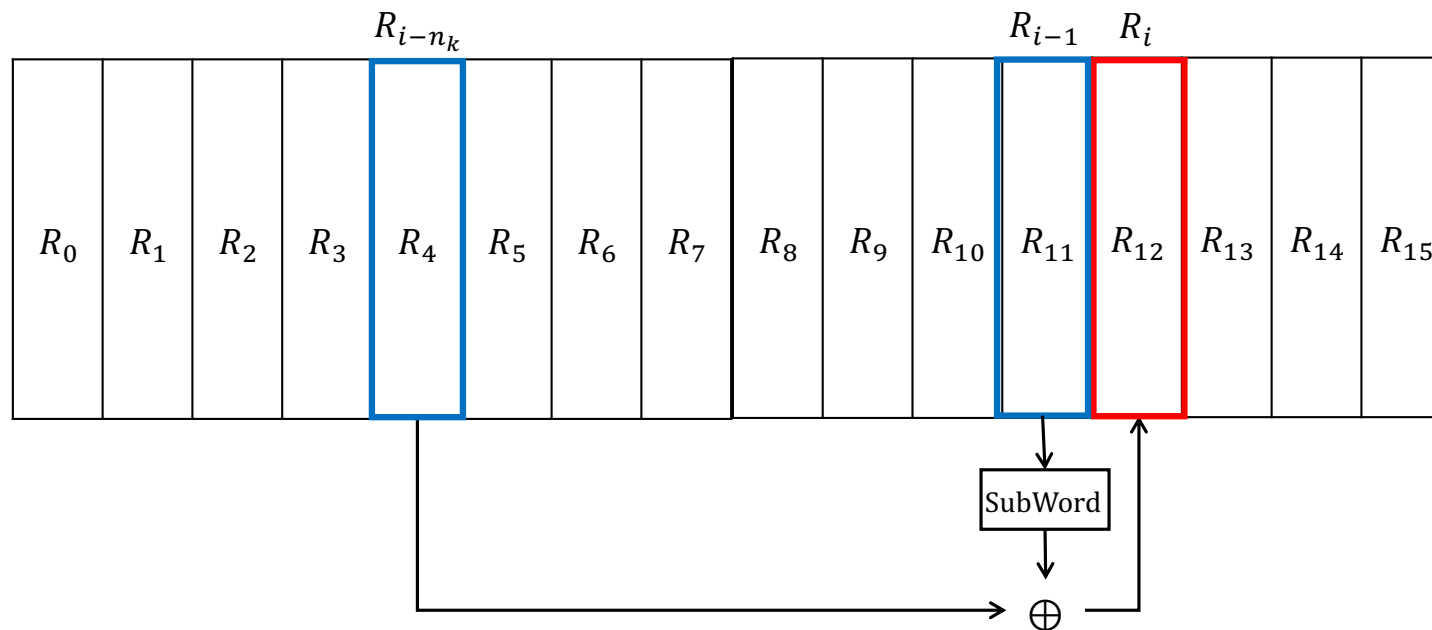
- $R_i = R_{i-n_k} \oplus R_{i-1}.$



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

AES 블록 암호 규격

FDL 김동현 (wlsitudpdf31@kookmin.ac.kr)