ARIA

발표자: 양유진

링크: https://youtu.be/239rydN23PY

1. ARIA 개요

Academy, Research Institute, Agency의 약자

= **대학, 연구소, 정부기관**이 공동으로 개발한 정보보호의 핵심 기술이라는 의미를 함축하고 있음

- 128-bit의 데이터를 처리하는 블록암호 알고리즘
  - 블록길이는 128-bit로 고정되어 있으며, 암호키는 128/192/256-bit가 있음.

<표 3> ARIA 사양 (단위: Bytes)

구 분	Nb	Nk	Nr
ARIA-128	16	16	12
ARIA-192	16	24	14
ARIA-256	16	32	16

- ***ISPN 구조**를 가지고 있어 별도의 복호화기를 필요로 하지 않음.

*Involution SPN
- 암호화, 복호화 과정이 같은 구조를 Involution 구조라고 함.
- ARIA의 경우 암호화에 사용되는 계층들이 복호화의 계층들로 사용됨.
  - DiffLayer의 경우, matrix^{-1}가 자기 자신이 되고,
  - SubstLayer의 경우, 짝수 라운드의 inverse 가 홀수 라운드에 사용됨.

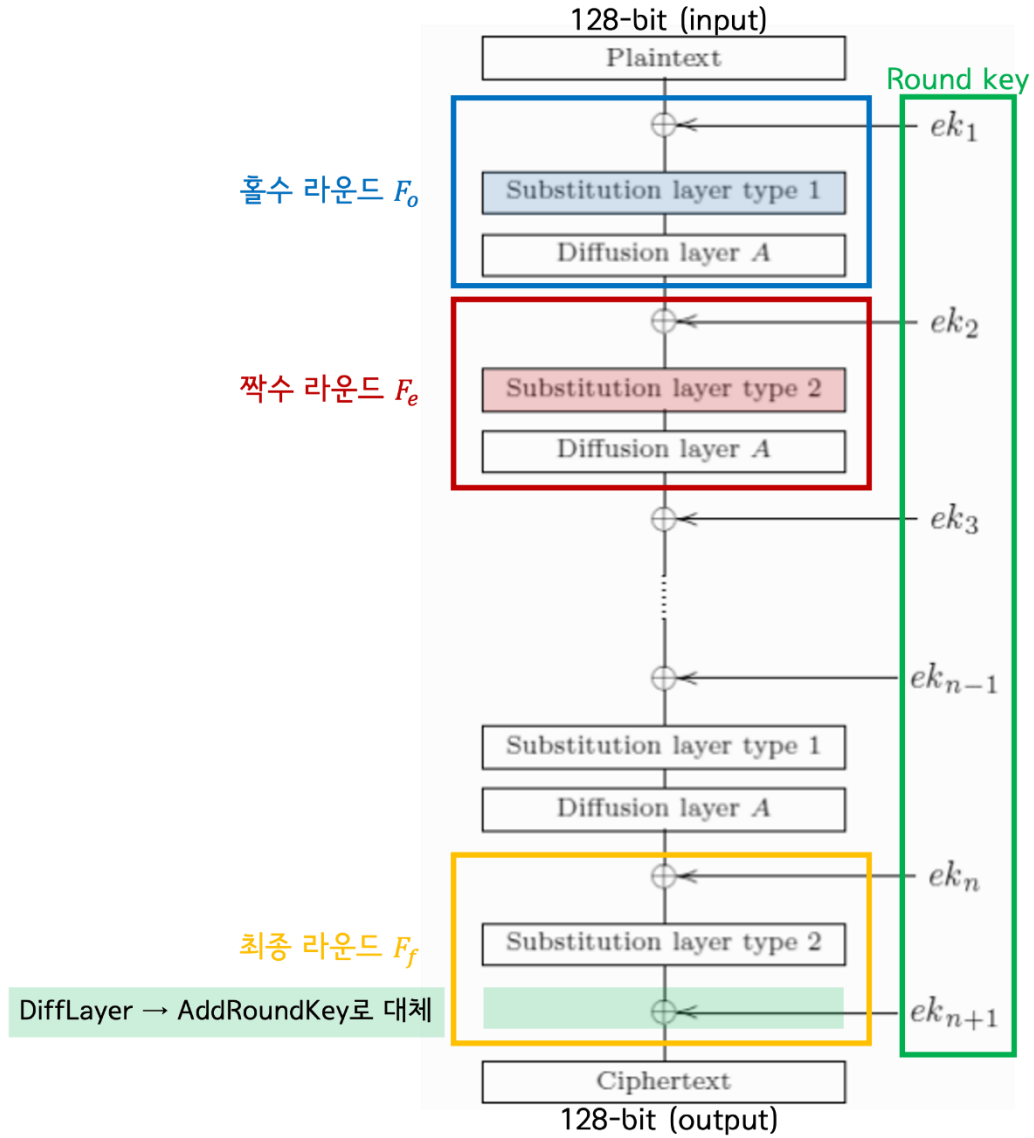
일반적인 SPN 구조

암호화: [평문 → 치환 → 확산 → 암호문]

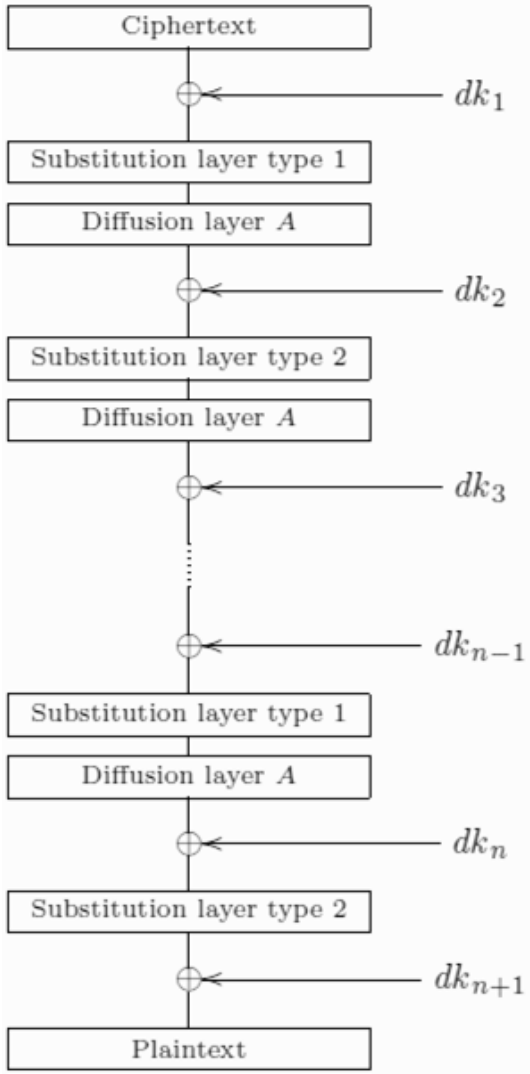
복호화: [암호문 → 확산$^{-1}$ → 치환$^{-1}$ → 평문]

2. ARIA 구조 0) overall

<암호화 알고리즘 구조>



<복호화 알고리즘 구조>



2. ARIA 구조 1) SubstLayer (치환 계층)

- Substitution layer는 **8-bit** 입/출력 S-box와 S-box^{-1} ($S_1, S_2, S_1^{-1}, S_2^{-1}$)로 구성됨.

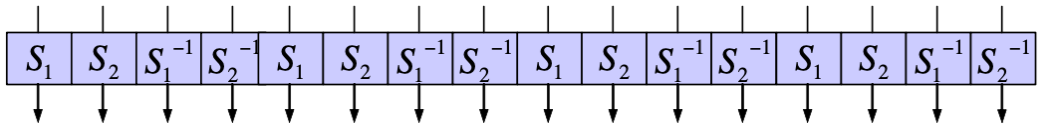
$$S_1(x) = Bx^{-1} \oplus b \qquad S_2(x) = Cx^{247} \oplus c$$

  - B, C 는 8×8 *정칙행렬(non-singular matrix) : 역행렬을 가질 수 있는 행렬 (=invertible matrix)
  - b, c 는 8×1 행렬

- 치환 계층은 **라운드의 홀/짝수 여부**에 따라 두 가지 유형으로 나뉨.

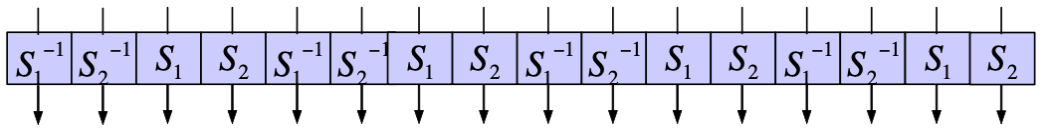
- 두 유형은 서로 **역의 관계**임.

(유형1)$^{-1}$ = 유형2



치환 계층 (유형 1)

F_o 에 사용됨



치환 계층 (유형 2)

F_e 에 사용됨

2. ARIA 구조 1) SubstLayer (치환 계층)

- 입력으로 들어온 8-bit를 16진법으로 나타낸 후(**0x(행)(열)**) 이를 S-box의 값으로 치환할 수 있음

ex) 0x05 = 0 행 5 열 = 0x6b

<표 4> S-box S_1

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

- S-box 생성에는 바이트들의 곱셈이 사용되었음 → $GF(2^8) = \mathbb{Z}_2[x]/m(x)$
- ARIA의 8차 기약다항식 $m(x)$ 는 $x^8 + x^4 + x^3 + x + 1$ 가 선택되었음. (AES와 같음)

2. ARIA 구조 2) DiffLayer (확산 계층)

- 치환 계층을 거쳐 입력으로 들어온 16 byte(=128-bit)에 대하여 byte 단위의 행렬 곱을 수행함.
- 행렬 곱에는 16×16 involution 이진 행렬이 사용됨.
- 입력/출력이 각각 $x_{0\sim15}, y_{0\sim15}$인 확산함수 $A : GF(2^8)^{16} \rightarrow GF(2^8)^{16}$ 를 이진행렬 곱으로 표현하면 아래와 같음.

byte단위이기 때문에 2^8임 (1-byte=8-bit)

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix}$$

2. ARIA 구조 3) Key Expansion (키 확장)

(1) Initialization phase

- 4개의 128-bit (W_0, W_1, W_2, W_3)를 생성하는 단계임.
- 이는 암호키 MK와 256-bit의 입/출력을 가지는 3 라운드 Feistel 암호를 이용하여 생성할 수 있음.

$$KL||KR = MK||0\cdots0$$

MK의 상위 128-bit MK 의 나머지 bit + 0…0 (128-bit)

<표 8> 암호키 길이에 따른 초기화 상수

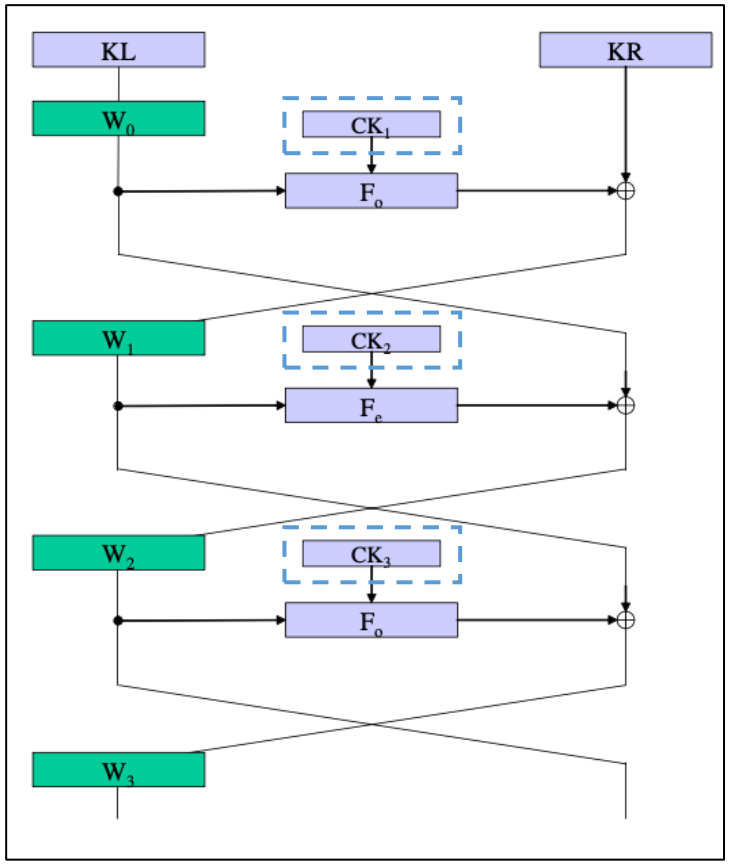
암호키 길이	CK_1	CK_2	CK_3
128-비트	$C1$	$C2$	$C3$
192-비트	$C2$	$C3$	$C1$
256-비트	$C3$	$C1$	$C2$

$C1 =$ 0x517cc1b727220a94fe13abe8fa9a6ee0

$C2 =$ 0x6db14acc9e21c820ff28b1d5ef5de2b0

$C3 =$ 0xdb92371d2126e9700324977504e8c90e

$$W_0 = KL, \qquad W_2 = F_e(W_1, CK_2) \oplus W_0,$$
$$W_1 = F_o(W_0, CK_1) \oplus KR, \qquad W_3 = F_o(W_2, CK_3) \oplus W_1.$$



2. ARIA 구조 3) Key Expansion (키 확장)

(2) Generation Round Key phase

- 앞서 생성한 4개의 128-bit (W_0, W_1, W_2, W_3)를 조합하여 128-bit 암호화 라운드 키 ek_i를 생성하는 단계임.
- XOR 연산($\oplus$)과 Rotation shift 연산($\ggg, \lll$)이 사용됨.
- 복호화 라운드 키 dk_i 는 암호화 라운드 키로부터 유도됨.

$$
\begin{aligned}
ek_1 &= (W_0) \oplus (W_1^{\ggg 19}), & ek_2 &= (W_1) \oplus (W_2^{\ggg 19}),\\
ek_3 &= (W_2) \oplus (W_3^{\ggg 19}), & ek_4 &= (W_0^{\ggg 19}) \oplus (W_3),\\
ek_5 &= (W_0) \oplus (W_1^{\ggg 31}), & ek_6 &= (W_1) \oplus (W_2^{\ggg 31}),\\
ek_7 &= (W_2) \oplus (W_3^{\ggg 31}), & ek_8 &= (W_0^{\ggg 31}) \oplus (W_3),\\
ek_9 &= (W_0) \oplus (W_1^{\lll 61}), & ek_{10} &= (W_1) \oplus (W_2^{\lll 61}),\\
ek_{11} &= (W_2) \oplus (W_3^{\lll 61}), & ek_{12} &= (W_0^{\lll 61}) \oplus (W_3),\\
ek_{13} &= (W_0) \oplus (W_1^{\lll 31}), & ek_{14} &= (W_1) \oplus (W_2^{\lll 31}),\\
ek_{15} &= (W_2) \oplus (W_3^{\lll 31}), & ek_{16} &= (W_0^{\lll 31}) \oplus (W_3),\\
ek_{17} &= (W_0) \oplus (W_1^{\lll 19}) &&
\end{aligned}
$$

$$
dk_i = \begin{cases} (i = 1 \; or \; n+1) & ek_{n+2-i} \\ (1 < i \le n) & A(ek_{n+2-i}) \end{cases}
$$

3. ARIA 소스코드 0) Pseudo Code(의사코드)

```
        Cipher(byte in[Nb], byte out[Nb], byte w[Nb*(Nr+1)])
begin
   byte state[Nb]

  state = in

  AddRoundKey(state, w[0..Nb-1])          // See Sec.  5.3

  for round = 1 to Nr-1
    SubstLayer(state)                     // See Sec.  5.1
    DiffLayer(state)                      // See Sec.  5.2
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for

  SubstLayer(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  out = state
end
```

Key expansion

Round Function ($1 \sim N_r - 1$)

Final Round Function

3. ARIA 소스코드 1) EncKeySetup(): Key Expansion

(1) Initialization phase

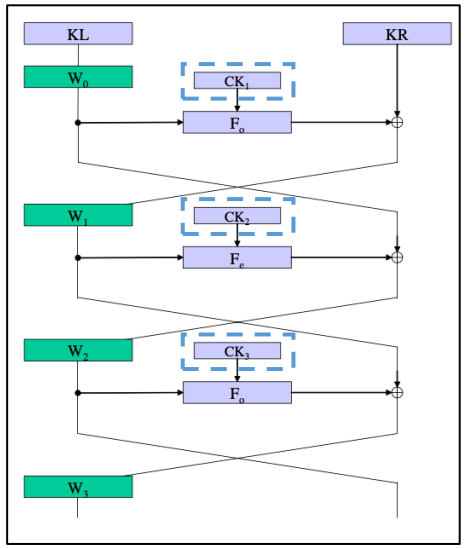
(암호키 MK, 암호화 라운드 키 ek_i, 키 길이)

```
int EncKeySetup(const Byte *w0, Byte *e, int keyBits) {
    int  i, R=(keyBits+256)/32, q;
    Byte t[16], w1[16], w2[16], w3[16];

    q = (keyBits - 128) / 64; //길이에 따라 초기화 상수 적용하기 위함 초기화 상수
F_o for (i = 0; i < 16; i++) t[i] = S[     i  % 4][KRK[q][i    ] ^ w0[i]];
    DL (t, w1);
    if (R==14)//키길이가 192-bit인 경우
        for (i = 0; i <  8; i++) w1[i] ^= w0[16+i];
    else if (R==16)//키길이가 256-bit인 경우
        for (i = 0; i < 16; i++) w1[i] ^= w0[16+i];

    q = (q==2)? 0 : (q+1);
F_e for (i = 0; i < 16; i++) t[i] = S[(2 + i) % 4][KRK[q][i] ^ w1[i]];
    DL (t, w2);
    for (i = 0; i < 16; i++) w2[i] ^= w0[i];

    q = (q==2)? 0 : (q+1);
F_o for (i = 0; i < 16; i++) t[i] = S[     i  % 4][KRK[q][i] ^ w2[i]];
    DL (t, w3);
    for (i = 0; i < 16; i++) w3[i] ^= w1[i];
```

```
const Byte KRK[3][16] = {
  {0x51, 0x7c, 0xc1, 0xb7, 0x27, 0x22,
      0x0a, 0x94, 0xfe, 0x13, 0xab,
      0xe8, 0xfa, 0x9a, 0x6e, 0xe0},
  {0x6d, 0xb1, 0x4a, 0xcc, 0x9e, 0x21,
      0xc8, 0x20, 0xff, 0x28, 0xb1,
      0xd5, 0xef, 0x5d, 0xe2, 0xb0},
  {0xdb, 0x92, 0x37, 0x1d, 0x21, 0x26,
      0xe9, 0x70, 0x03, 0x24, 0x97,
      0x75, 0x04, 0xe8, 0xc9, 0x0e}
};
```

<표 8> 암호키 길이에 따른 초기화 상수

암호키 길이	CK_1	CK_2	CK_3
128-비트	$C1$	$C2$	$C3$
192-비트	$C2$	$C3$	$C1$
256-비트	$C3$	$C1$	$C2$



3. ARIA 소스코드 1) EncKeySetup(): Key Expansion

(2) Generation Round Key phase

```
int EncKeySetup(const Byte *w0, Byte *e, int keyBits) {
```
(암호키 MK, 암호화 라운드 키 ek_i, 키 길이)

```
    ⋮
    for (i = 0; i < 16*(R+1); i++) e[i] = 0; //초기화
```

암호화 라운드 키 ek_i 생성

e에 w값 넣음

```
RotXOR (w0,  0, e      );
RotXOR (w1,  0, e +  16);
RotXOR (w2,  0, e +  32);
RotXOR (w3,  0, e +  48);
RotXOR (w0,  0, e +  64);
RotXOR (w1,  0, e +  80);
RotXOR (w2,  0, e +  96);
RotXOR (w3,  0, e + 112);
RotXOR (w0,  0, e + 128);
RotXOR (w1,  0, e + 144);
RotXOR (w2,  0, e + 160);
RotXOR (w3,  0, e + 176);
RotXOR (w0,  0, e + 192);
```

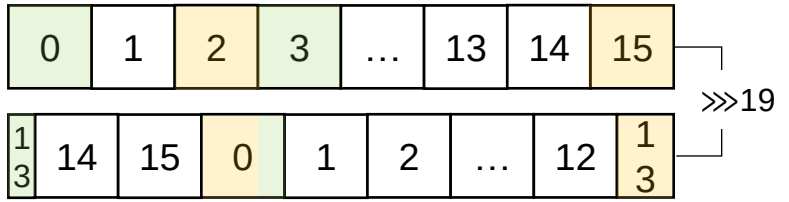
shift rotation 한 값을 XOR함.

```
RotXOR (w1,  19, e      );
RotXOR (w2,  19, e +  16);
RotXOR (w3,  19, e +  32);
RotXOR (w0,  19, e +  48);
RotXOR (w1,  31, e +  64);
RotXOR (w2,  31, e +  80);
RotXOR (w3,  31, e +  96);
RotXOR (w0,  31, e + 112);
RotXOR (w1,  67, e + 128);
RotXOR (w2,  67, e + 144);
RotXOR (w3,  67, e + 160);
RotXOR (w0,  67, e + 176);
RotXOR (w1,  97, e + 192);
```

```
    if (R > 12) { // 키 길이 192-bit인 경우
        RotXOR (w1, 0, e + 208); RotXOR (w2,  97, e + 208);
        RotXOR (w2, 0, e + 224); RotXOR (w3,  97, e + 224);
    }
    if (R > 14) { // 키 길이 256-bit인 경우
        RotXOR (w3, 0, e + 240); RotXOR (w0,  97, e + 240);
        RotXOR (w0, 0, e + 256); RotXOR (w1, 109, e + 256);
    }
    return R;//R = 라운드 수
}
```

```
void RotXOR (const Byte *s, int n, Byte *t)
{
    int i, q;

    q = n/8; n %= 8; //Byte 단위
    for (i = 0; i < 16; i++) {
        t[(q+i) % 16] ^= (s[i] >> n);
        if (n != 0) t[(q+i+1) % 16] ^= (s[i] << (8-n));
    }
}
```

ex) 19-bit = 2 x 8-bit + 3-bit



$\lll 61 = \ggg 67$

$\lll 31 = \ggg 97$

$\lll 19 = \ggg 109$

3. ARIA 소스코드 2) Crypt(): Round Function

```
void Crypt(const Byte *p, int R, const Byte *e, Byte *c)
{
    int i, j;
    Byte t[16];

    for (j = 0; j < 16; j++) c[j] = p[j]; //암호 블럭에 평문 삽입

    // 라운드 함수
    for (i = 0; i < R/2; i++)
    {
        // 홀수 라운드 함수 F_o
        for (j = 0; j < 16; j++) t[j] = S[     j  % 4][e[j] ^ c[j]];// SubstLayer,
        DL(t, c); e += 16; // DiffLayer                                AddRoundKey
        // 짝수 라운드 함수 F_e
        for (j = 0; j < 16; j++) t[j] = S[(2 + j) % 4][e[j] ^ c[j]];
        DL(t, c); e += 16;
    }                                                  // 마지막 라운드; SubstLayer, AddRoundKey
    DL(c, t);
    for (j = 0; j < 16; j++) c[j] = e[j] ^ t[j];
}
```

```
void DL (const Byte *i, Byte *o)
{
    Byte T;

    T = i[ 3] ^ i[ 4] ^ i[ 9] ^ i[14];
    o[ 0] = i[ 6] ^ i[ 8] ^ i[13] ^ T;
    o[ 5] = i[ 1] ^ i[10] ^ i[15] ^ T;
    o[11] = i[ 2] ^ i[ 7] ^ i[12] ^ T;
    o[14] = i[ 0] ^ i[ 5] ^ i[11] ^ T;
    T = i[ 2] ^ i[ 5] ^ i[ 8] ^ i[15];
    o[ 1] = i[ 7] ^ i[ 9] ^ i[12] ^ T;
    o[ 4] = i[ 0] ^ i[11] ^ i[14] ^ T;
    o[10] = i[ 3] ^ i[ 6] ^ i[13] ^ T;
    o[15] = i[ 1] ^ i[ 4] ^ i[10] ^ T;
    T = i[ 1] ^ i[ 6] ^ i[11] ^ i[12];
    o[ 2] = i[ 4] ^ i[10] ^ i[15] ^ T;
    o[ 7] = i[ 3] ^ i[ 8] ^ i[13] ^ T;
    o[ 9] = i[ 0] ^ i[ 5] ^ i[14] ^ T;
    o[12] = i[ 2] ^ i[ 7] ^ i[ 9] ^ T;
    T = i[ 0] ^ i[ 7] ^ i[10] ^ i[13];
    o[ 3] = i[ 5] ^ i[11] ^ i[14] ^ T;
    o[ 6] = i[ 2] ^ i[ 9] ^ i[12] ^ T;
    o[ 8] = i[ 1] ^ i[ 4] ^ i[15] ^ T;
    o[13] = i[ 3] ^ i[ 6] ^ i[ 8] ^ T;
}
```

3. ARIA 코드 결과

- 192 비트 암호키

```
key        : 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
             00 11 22 33 44 55 66 77
```

- 128 비트 평문

```
plaintext  : 11 11 11 11 aa aa aa aa 11 11 11 11 bb bb bb bb
```

```
ECB mode   : 8d 14 70 62 5f 59 eb ac b0 e5 5b 53 4b 3e 46 2b
```

<테스트 벡터>

```
key      : 00112233 44556677 8899aabb ccddeeff 00112233 44556677
plaintext: 11111111 aaaaaaaa 11111111 bbbbbbbb
result is: 8d147062 5f59ebac b0e55b53 4b3e462b
```

<코드 실행 결과>

감사합니다