ARIA 구현

SeoulTech Cryptography & Information Security Lab.

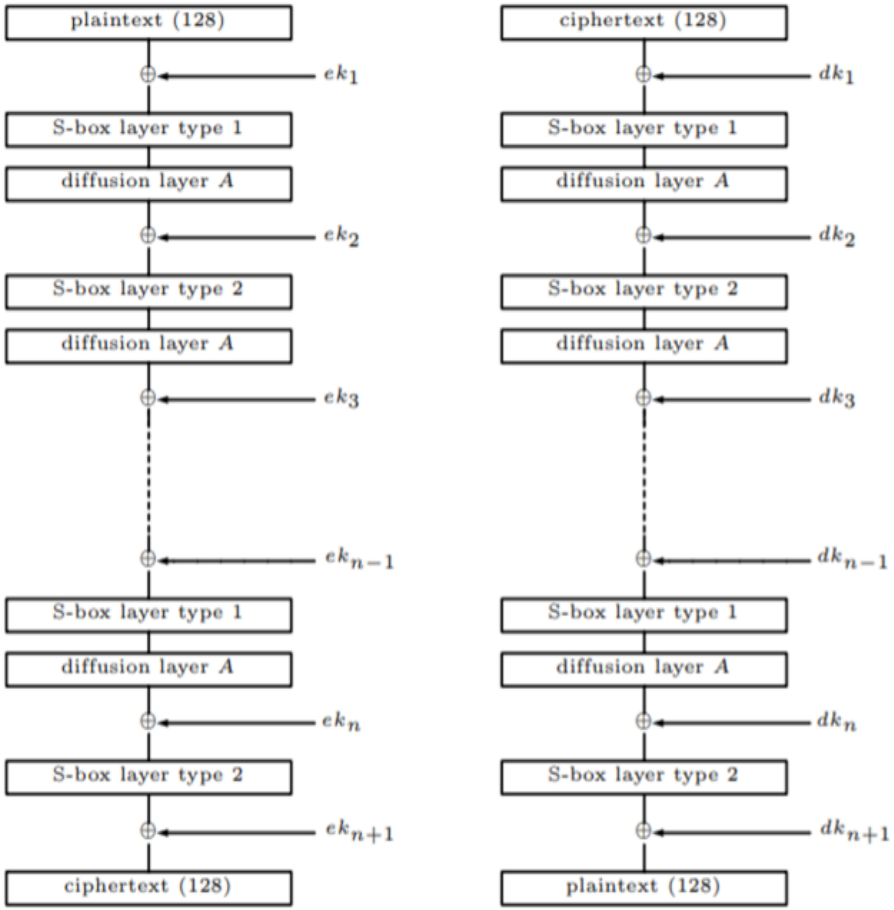
조서연

2019-07-19

Contents

ARIA 알고리즘

ARIA 특징

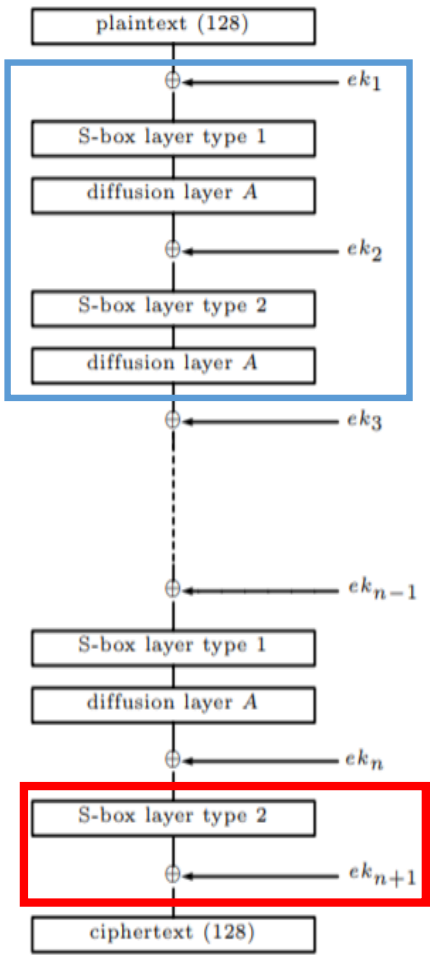


- 블록크기 : 128 bit
- 키 크기 : 128bit
- 라운드 수 : 12 라운드

- 암호화 = 복호화 알고리즘
- 단, 복호화 키는 역순으로 넣어야 함
- 첫번째 마지막 라운드 키 제외
  모든 복호화 키에 DL 연산 적용.

ARIA 알고리즘

암호화&복호화 알고리즘



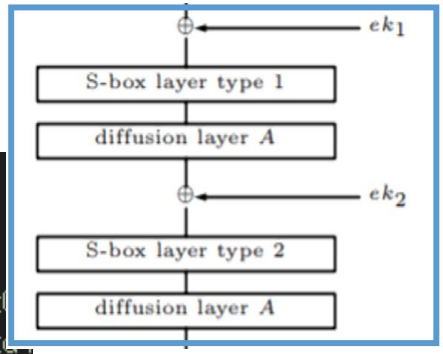
- 128bit 키를 사용하여 12 라운드
- 짝/홀수 묶어서 6번 돌림
- DL의 역 함수 = DL

```
static void ARIA(const uint8_t *p, const uint8_t *e, uint8_t *c)
{
    const uint8_t round = 12;
    int i, j;
    uint8_t t[16];

        memcpy(c,p,16);// 일단 복사해 두고
        for (i = 0; i < 6; i++) // 짝/홀수 나누어서 라운드를 하도록 돌린다.
        {
            for (j = 0; j < 16; j++) t[j] = S[j%4][e[j] ^ c[j]]; //키랑 add 해주고 s 박스
            DL(t, c); e += 16; // DL 연산 해주고 그다음 키로 가도록 포인터 연산을 한다.

            for (j = 0; j < 16; j++) t[j] = S[(2+j) % 4][e[j] ^ c[j]];
            DL(t, c); e += 16;
        }
        DL(c, t); // 마지막에 DL 연산을 해줘서 바꿔서 다시 넣어 준다.
        for (j = 0; j < 16; j++) c[j] = e[j] ^ t[j];
}
```

ARIA 알고리즘

Substitution(치환) :S-box layer type 1 & S-box layer type2

- 키 XOR 해주고 s 박스에 넣기
- Sbox는 8비트 input, output
- [S-box layer type 1] 역행렬= [S-box layer type 2]

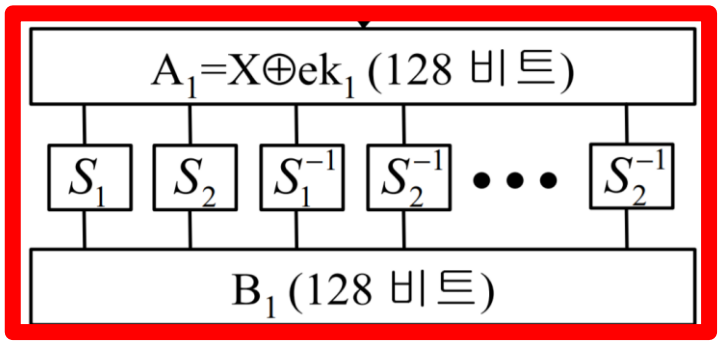


S1, S2, $S1^{-1}, S2^{-1}$

```
static const uint8_t S[4][256] = {
    // S-box type 1
{
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0x
```

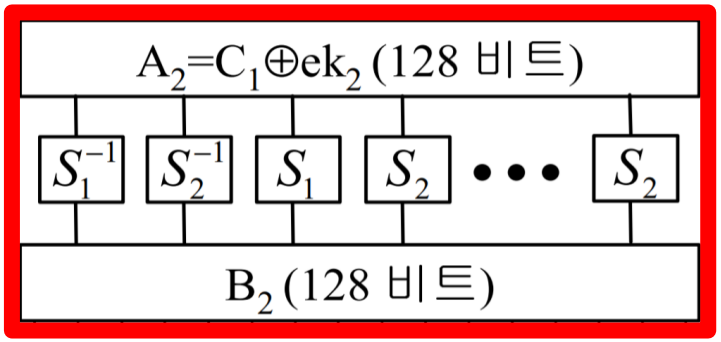
짝수 라운드

S[j%4][e[j] ^ c[j]]



홀수 라운드

S[(2+j) % 4][e[j] ^ c[j]]

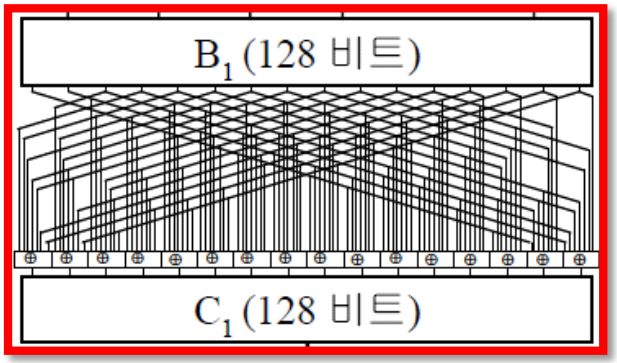


ARIA 알고리즘

ARIA – Diffustion Layer

* involution 행렬 DL = DL^{-1}

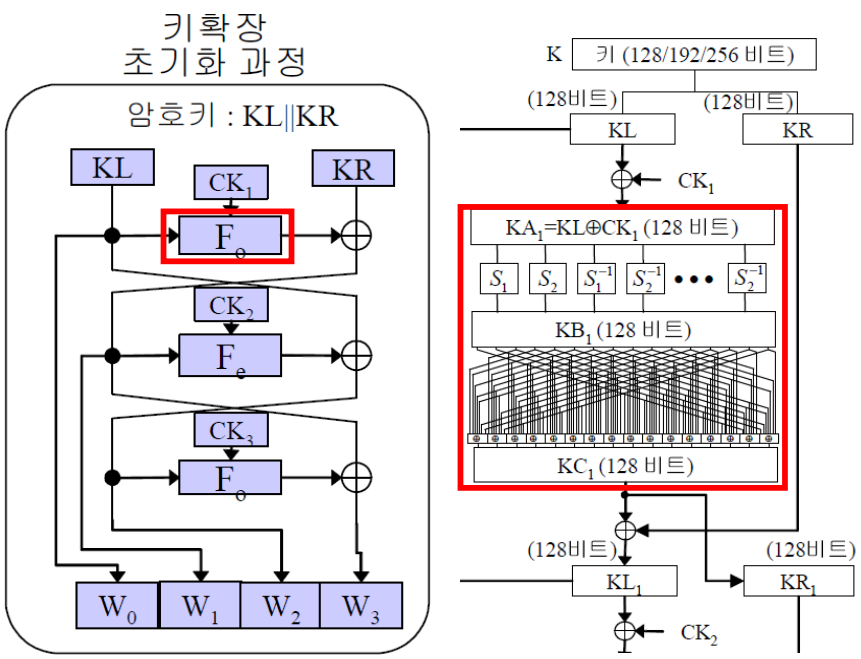
$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \end{pmatrix} = \left(\begin{array}{cccc|cccc|cccc|cccc} 0&0&0&1&1&0&1&0&1&1&0&0&0&1&1&0 \\ 0&0&1&0&0&1&0&1&1&1&0&0&1&0&0&1 \\ 0&1&0&0&1&0&1&0&0&0&1&1&1&0&0&1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




```
    // Diffusion Layer
static void DL (const uint8_t *i, uint8_t *o)
{
    uint8_t 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;
}
```

ARIA 알고리즘

ARIA – 키 확장 & 라운드 키 생성



$$C1 = \texttt{0x517cc1b727220a94fe13abe8fa9a6ee0}$$

$$C2 = \texttt{0x6db14acc9e21c820ff28b1d5ef5de2b0}$$

$$C3 = \texttt{0xdb92371d2126e9700324977504e8c90e}$$

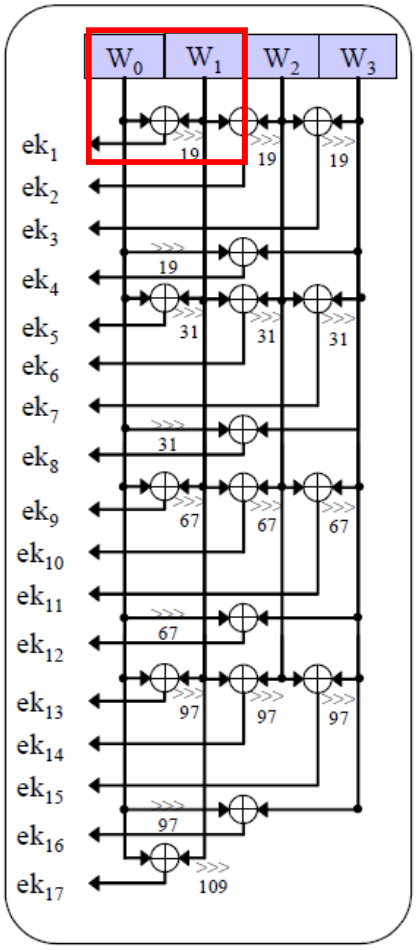
```
void ARIA_Enc_Keyexpansion(uint8_t * W0, uint8_t* e){

    uint8_t  i;
    uint8_t tmp[16], W1[16], W2[16], W3[16];

    for (i = 0; i < 16; i++) tmp[i] = S[i%4][C[0][i] ^ W0[i]];
    DL (tmp, W1);

    for (i = 0; i < 16; i++) tmp[i] = S[(2 + i) % 4][C[1][i] ^ W1[i]];
    DL (tmp, W2);
    for (i = 0; i < 16; i++) W2[i] ^= W0[i];

    for (i = 0; i < 16; i++) tmp[i] = S[i % 4][C[2][i] ^ W2[i]];
    DL (tmp, W3);
    for (i = 0; i < 16; i++) W3[i] ^= W1[i];

    for (i = 0; i < 16*(12+1); i++) e[i] = 0;

    //라운드 키 생성
    RotXOR (W0, 0, e      ); RotXOR (W1,  19, e      );
    RotXOR (W1, 0, e +  16); RotXOR (W2,  19, e +  16);
    RotXOR (W2, 0, e +  32); RotXOR (W3,  19, e +  32);
    RotXOR (W3, 0, e +  48); RotXOR (W0,  19, e +  48);
    RotXOR (W0, 0, e +  64); RotXOR (W1,  31, e +  64);
    RotXOR (W1, 0, e +  80); RotXOR (W2,  31, e +  80);
    RotXOR (W2, 0, e +  96); RotXOR (W3,  31, e +  96);
    RotXOR (W3, 0, e + 112); RotXOR (W0,  31, e + 112);
    RotXOR (W0, 0, e + 128); RotXOR (W1,  67, e + 128);
    RotXOR (W1, 0, e + 144); RotXOR (W2,  67, e + 144);
    RotXOR (W2, 0, e + 160); RotXOR (W3,  67, e + 160);
    RotXOR (W3, 0, e + 176); RotXOR (W0,  67, e + 176);
    RotXOR (W0, 0, e + 192); RotXOR (W1,  97, e + 192);
    }
```

ARIA 알고리즘

ARIA – 키 확장 & 라운드 키 생성

- 모든 로테이션이 >>>(right shift)

라운드 키 생성



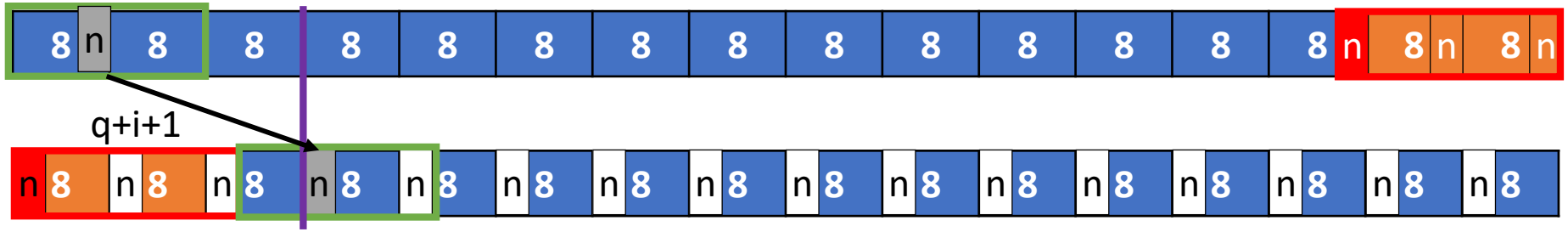

```
void ARIA_Enc_Keyexpansion(uint8_t * W0, uint8_t* e){

    uint8_t  i;
    uint8_t tmp[16], W1[16], W2[16], W3[16];

    for (i = 0; i < 16; i++) tmp[i] = S[i%4][C[0][i] ^ W0[i]];
    DL (tmp, W1);

    for (i = 0; i < 16; i++) tmp[i] = S[(2 + i) % 4][C[1][i] ^ W1[i]];
    DL (tmp, W2);
    for (i = 0; i < 16; i++) W2[i] ^= W0[i];

    for (i = 0; i < 16; i++) tmp[i] = S[i % 4][C[2][i] ^ W2[i]];
    DL (tmp, W3);
    for (i = 0; i < 16; i++) W3[i] ^= W1[i];

    for (i = 0; i < 16*(12+1); i++) e[i] = 0;

    //라운드 키 생성
    RotXOR (W0, 0, e      ); RotXOR (W1,  19, e      );
    RotXOR (W1, 0, e +  16); RotXOR (W2,  19, e +  16);
    RotXOR (W2, 0, e +  32); RotXOR (W3,  19, e +  32);
    RotXOR (W3, 0, e +  48); RotXOR (W0,  19, e +  48);
    RotXOR (W0, 0, e +  64); RotXOR (W1,  31, e +  64);
    RotXOR (W1, 0, e +  80); RotXOR (W2,  31, e +  80);
    RotXOR (W2, 0, e +  96); RotXOR (W3,  31, e +  96);
    RotXOR (W3, 0, e + 112); RotXOR (W0,  31, e + 112);
    RotXOR (W0, 0, e + 128); RotXOR (W1,  67, e + 128);
    RotXOR (W1, 0, e + 144); RotXOR (W2,  67, e + 144);
    RotXOR (W2, 0, e + 160); RotXOR (W3,  67, e + 160);
    RotXOR (W3, 0, e + 176); RotXOR (W0,  67, e + 176);
    RotXOR (W0, 0, e + 192); RotXOR (W1,  97, e + 192);
}
```

ARIA 알고리즘

ARIA – 키 확장 & 라운드 키 생성 RotXOR 함수

- RotXOR 알고리즘

```
//RotXOR(W2,  19, e +  16);
static void RotXOR (const uint8_t *s, uint8_t n, uint8_t *t)
{
    uint32_t i, q;

    q = n/8; n %= 8; //BYTE는 8비트 128비트는 16바이트 8비트가 넘어가면 배열의 인덱스를 바꾸어야 한다.
    for (i = 0; i < 16; i++) {
        t[(q+i) % 16] ^= (s[i] >> n); // 안의 n 비트 공간을 남기고 우선 배열의 인덱스 로테이션
        if (n != 0) t[(q+i+1) % 16] ^= (s[i] << (8-n)); // n 비트 채움
    }
}
```



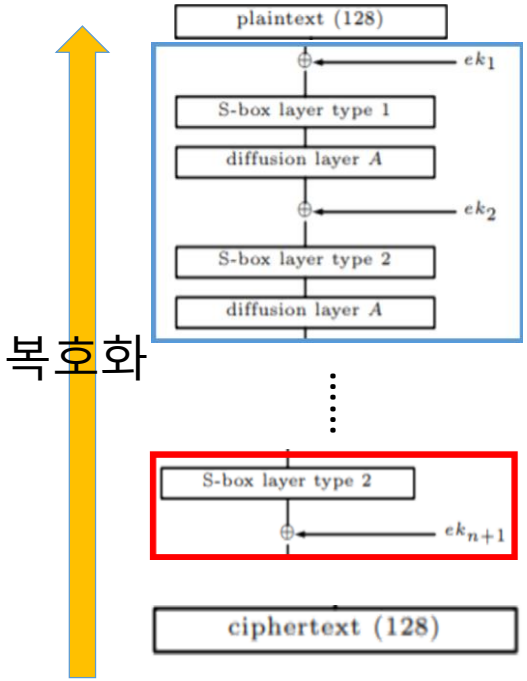
각 블록 마다 >>> n 하며, q 만큼의 블록 단위 로테이션

ARIA 알고리즘

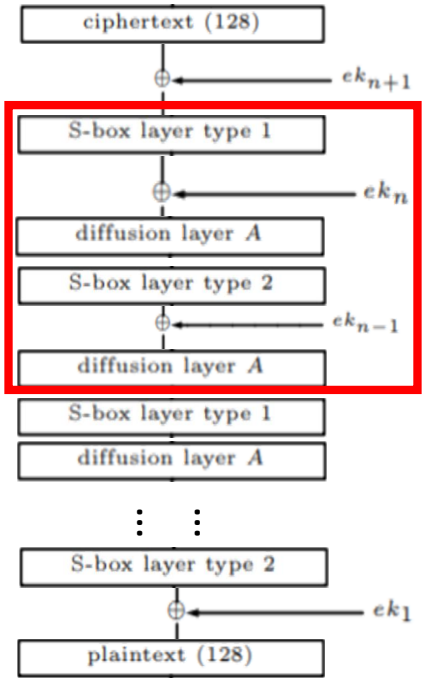
ARIA –복호화 라운드 키 생성

- S-box layer type 2 inverse = S-box layer type 1

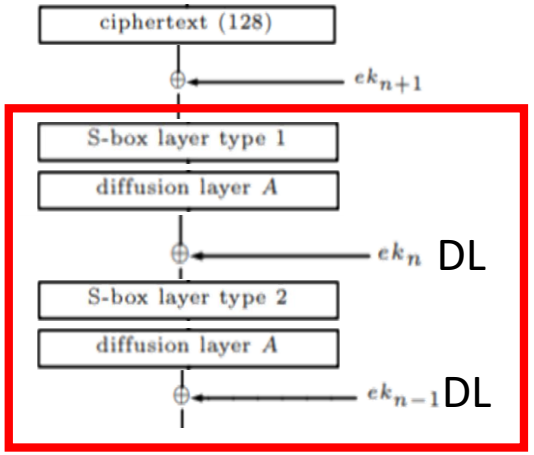
$$dk_1 = ek_{n+1}, dk_2 = A(ek_n), dk_3 = A(ek_{n-1}), \cdots, dk_n = A(ek_2), dk_{n+1} = ek_1.$$



[그림 1]



[그림 2]



[그림 3]

ARIA 알고리즘

ARIA – 키 확장 & 복호화 라운드 키 생성

$$dk_1 = ek_{n+1}, dk_2 = A(ek_n), dk_3 = A(ek_{n-1}), \cdots, dk_n = A(ek_2), dk_{n+1} = ek_1.$$

```
void ARIA_Dec_Keyexpansion(uint8_t * W0, uint8_t* d){
    int  i, j;
    uint8_t t[16];

    ARIA_Enc_Keyexpansion(W0, d);
    for (j = 0; j < 16; j++){
        t[j] = d[j];
        d[j] = d[16*12 + j];
        d[16*12 + j] = t[j];
    }// 0, 12 키를 바꾼다

    for (i = 1; i <= 6; i++){
        DL (d + i*16, t); //확산함수 저장
        DL (d + (12-i)*16, d + i*16); // 1, 11/ 2, 10, /3, 9/ 4, 8/ 5, 7/ 6,6 DL 적용해서 뒤집는다.
        for (j = 0; j < 16; j++) d[(12-i)*16 + j] = t[j];
    }
    return;
    }
```

ARIA 알고리즘

ARIA – TEST

- 128 비트 암호키

  key : 000102030405060708090a0b0c0d0e0f

- 128 비트 평문

  plaintext : 00112233445566778899aabbccddeeff

round[12].output	d718fbd6ab644c739da95f3be6451778

```
enc_Text : d718fbd6ab644c739da95f3be6451778
dec_Text : 00112233445566778899aabbccddeeff
ARIA Cycle : 43560029
```

Thank you

ARIA 알고리즘

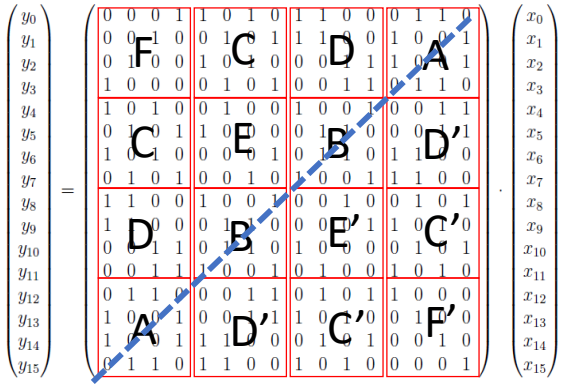
ARIA – 순열(Permutation)

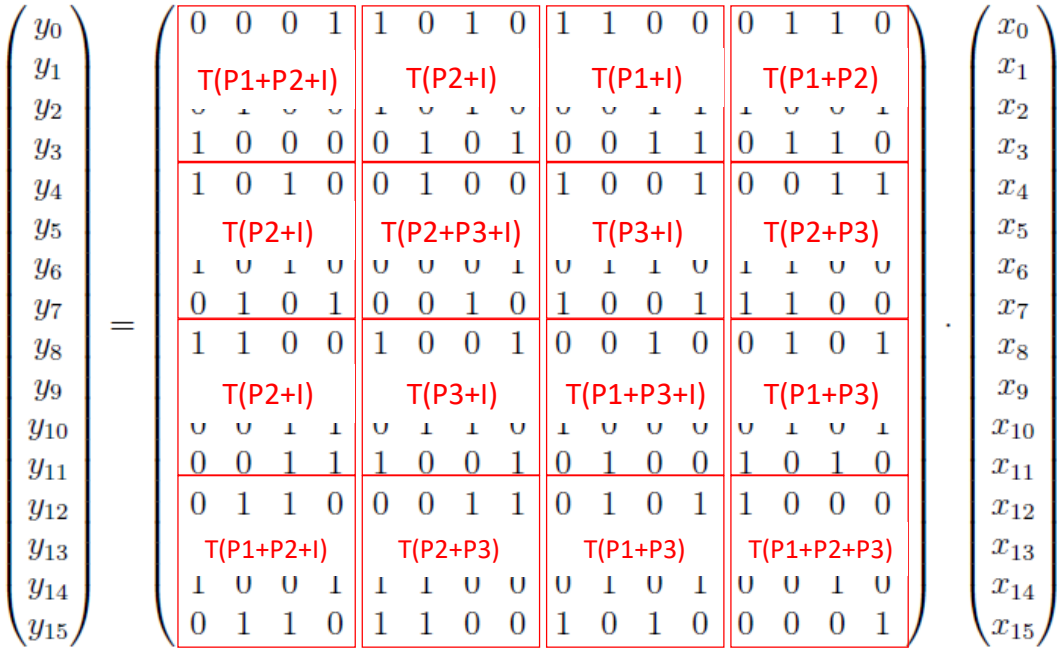
- 16 BY 16 행렬 $A = M_1 \cdot M_2 \cdot M_1$
- 전치 행렬이며 역행렬과 같은 형태, 128비트 단위로 연산

$$T = \begin{pmatrix} 0&1&1&1\\ 1&0&1&1\\ 1&1&0&1\\ 1&1&1&0 \end{pmatrix}, \quad P_1 = \begin{pmatrix} 0&1&0&0\\ 1&0&0&0\\ 0&0&0&1\\ 0&0&1&0 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 0&0&1&0\\ 0&0&0&1\\ 1&0&0&0\\ 0&1&0&0 \end{pmatrix}, \quad P_3 = \begin{pmatrix} 0&0&0&1\\ 0&0&1&0\\ 0&1&0&0\\ 1&0&0&0 \end{pmatrix}.$$

$$M_1 = \begin{pmatrix} I & I & I & 0\\ I & 0 & I & I\\ I & I & 0 & I\\ 0 & I & I & I \end{pmatrix} \qquad M_1^{-1} = M_1$$

DL = DL^{-1}





CIS

참고문헌

[1] https://seed.kisa.or.kr/kisa/Board/19/detailView.do
[2] ARIA 알고리즘 명세서