CHAM

송민호

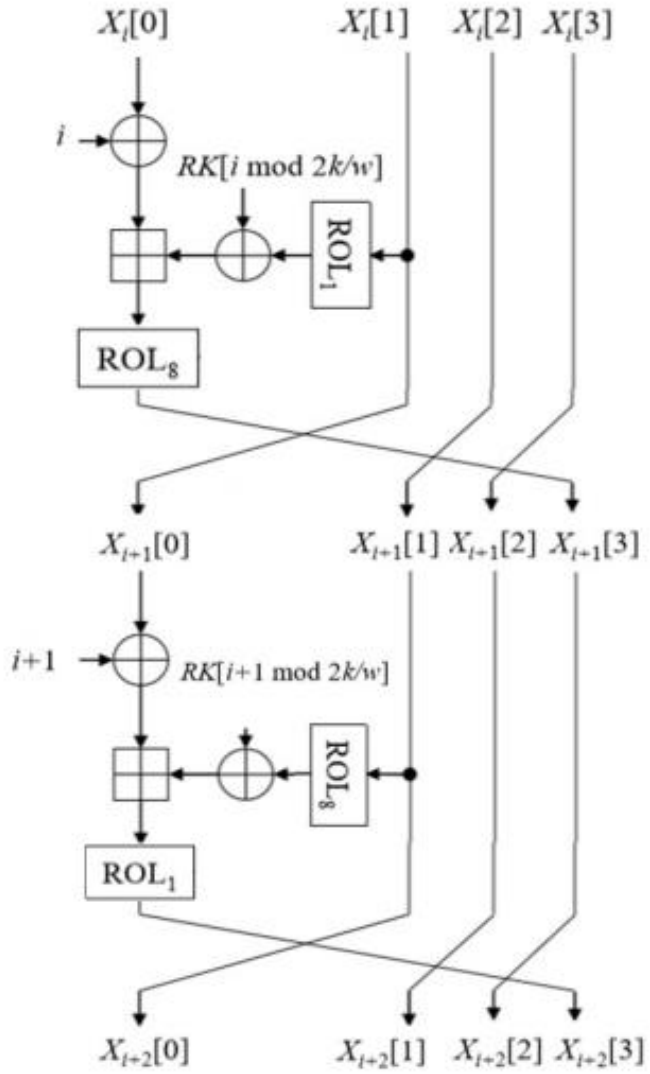
유튜브 주소: https://youtu.be/MizGwEh-5UQ

CHAM

CIPHER	BLOCK	KEY	ROUND
CHAM_64/128	64	128	88
CHAM_128/128	128	128	112
CHAM_128/256	128	256	120

CHAM

- CHAM_64/128
- 경량 암호
- ARX 구조

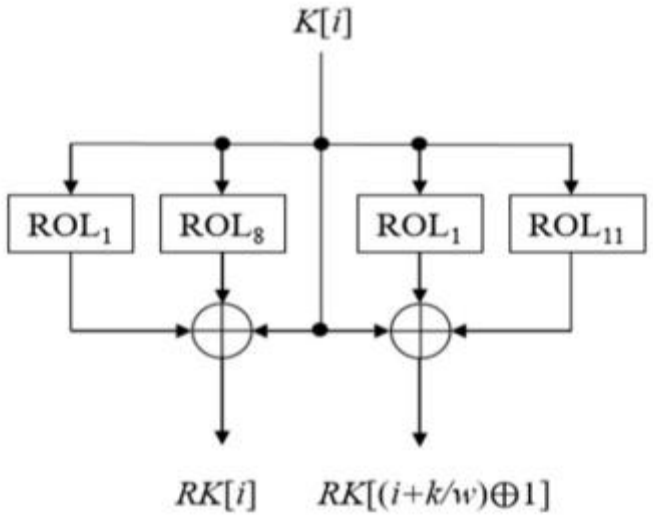


CHAM

```
void test_cham64()
{
    uint8_t mk[] = {
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
    };

    uint8_t pt[] = {
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
    };

    uint8_t ct[] = {
        0x79, 0x65, 0x04, 0x12, 0x3f, 0x12, 0xa9, 0xe5,
    };

    uint8_t encrypted[8] = {0,};
    uint8_t decrypted[8] = {0,};

    cham64_keygen(rks, mk);
    cham64_encrypt(encrypted, pt, rks);
    cham64_decrypt(decrypted, ct, rks);

}
```

CHAM_keygen

rol16 = 16비트 상에서 Rotation Left




```
void cham64_keygen(uint8_t* rks, const uint8_t* mk)
{
    const uint16_t* key = (uint16_t*) mk;
    uint16_t* rk = (uint16_t*) rks;

    for (size_t i = 0; i < 8; ++i) {
        rk[i] = key[i] ^ rol16(key[i], 1);
        rk[(i+8)^(0x1)] = rk[i] ^ rol16(key[i], 11);
        rk[i] ^= rol16(key[i], 8);
    }
}
```

CHAM_encrypt

```
for (size_t round = 0; round < CHAM_64_128_ROUNDS; round += 8) {
    blk[0] = rol16((blk[0] ^ (rc++)) + (rol16(blk[1], 1) ^ rk[0]), 8);
    blk[1] = rol16((blk[1] ^ (rc++)) + (rol16(blk[2], 8) ^ rk[1]), 1);
    blk[2] = rol16((blk[2] ^ (rc++)) + (rol16(blk[3], 1) ^ rk[2]), 8);
    blk[3] = rol16((blk[3] ^ (rc++)) + (rol16(blk[0], 8) ^ rk[3]), 1);

    blk[0] = rol16((blk[0] ^ (rc++)) + (rol16(blk[1], 1) ^ rk[4]), 8);
    blk[1] = rol16((blk[1] ^ (rc++)) + (rol16(blk[2], 8) ^ rk[5]), 1);
    blk[2] = rol16((blk[2] ^ (rc++)) + (rol16(blk[3], 1) ^ rk[6]), 8);
    blk[3] = rol16((blk[3] ^ (rc++)) + (rol16(blk[0], 8) ^ rk[7]), 1);
}
```

CHAM_encrypt

```
#define X00 R18     // blk 0
#define X01 R19
#define X10 R20     // blk 1
#define X11 R21
#define X20 R22     // blk 2
#define X21 R23
#define X30 R24     // blk 3
#define X31 R25

#define TM0 R26     // temp
#define TM1 R27

#define RC R16  // ROUND COUNTER
#define RK R0   // ROUND KEY

#define CNT R17
```

```
.macro ENC64_ODD_ROUND
    MOVW TM0, X10

    LSL TM0         // ROL 1
    ROL TM1
    ADC TM0, R1

    LD RK, Z+       // XOR RK
    EOR TM0, RK
    LD RK, Z+
    EOR TM1, RK

    EOR X00, RC     // XOR RC

    ADD X00, TM0    // MODULA
    ADC X01, TM1

    MOV TM0, X00    // ROL 8
    MOV X00, X01
    MOV X01, TM0

    MOVW TM0, X00   // BLOCK ROT
    MOVW X00, X10
    MOVW X10, X20
    MOVW X20, X30
    MOVW X30, TM0

    INC RC          // rc++
.endm
```

```
.macro ENC64_EVEN_ROUND
    MOV TM1, X10    // ROL 8
    MOV TM0, X11

    LD RK, Z+       // XOR RK
    EOR TM0, RK
    LD RK, Z+
    EOR TM1, RK

    EOR X00, RC     // XOR RC

    ADD X00, TM0    // MODULA
    ADC X01, TM1

    LSL X00         // ROL 1
    ROL X01
    ADC X00, R1

    MOVW TM0, X00
    MOVW X00, X10
    MOVW X10, X20
    MOVW X20, X30
    MOVW X30, TM0   // BLOCK ROT

    INC RC          // rc++
.endm
```

CHAM_decrypt

- ror16 = 16비트 상에서 Rotation Right

```
for (size_t round = 0; round < CHAM_64_128_ROUNDS; round += 8) {
    blk[3] = (ror16(blk[3], 1) - (rol16(blk[0], 8) ^ rk[7])) ^ (--rc);
    blk[2] = (ror16(blk[2], 8) - (rol16(blk[3], 1) ^ rk[6])) ^ (--rc);
    blk[1] = (ror16(blk[1], 1) - (rol16(blk[2], 8) ^ rk[5])) ^ (--rc);
    blk[0] = (ror16(blk[0], 8) - (rol16(blk[1], 1) ^ rk[4])) ^ (--rc);

    blk[3] = (ror16(blk[3], 1) - (rol16(blk[0], 8) ^ rk[3])) ^ (--rc);
    blk[2] = (ror16(blk[2], 8) - (rol16(blk[3], 1) ^ rk[2])) ^ (--rc);
    blk[1] = (ror16(blk[1], 1) - (rol16(blk[2], 8) ^ rk[1])) ^ (--rc);
    blk[0] = (ror16(blk[0], 8) - (rol16(blk[1], 1) ^ rk[0])) ^ (--rc);

}
```

CHAM_decrypt

```
#define X00 R18     // blk 0
#define X01 R19
#define X10 R20     // blk 1
#define X11 R21
#define X20 R22     // blk 2
#define X21 R23
#define X30 R24     // blk 3
#define X31 R25

#define TM0 R26     // temp
#define TM1 R27

#define RC R16  // ROUND COUNTER
#define RK R0   // ROUND KEY

#define CNT R17
```

```
.macro DEC64_ODD_ROUND
    BST X30, 0      // ROR 1
    LSR X31
    ROR X30
    BLD X31, 7

    MOV TM1, X00    // ROL 8
    MOV TM0, X01

    LD RK, -Z       // XOR RK
    EOR TM1, RK
    LD RK, -Z
    EOR TM0, RK

    SUB X30, TM0    // MINUS
    SBC X31, TM1

    DEC RC          // --rc

    EOR X30, RC     // XOR RC

    MOVW TM0, X30   // BLOCK ROT
    MOVW X30, X20
    MOVW X20, X10
    MOVW X10, X00
    MOVW X00, TM0
.endm
```

```
.macro DEC64_EVEN_ROUND
    MOV TM0, X30    // ROR 8
    MOV X30, X31
    MOV X31, TM0

    MOVW TM0, X00

    LSL TM0         // ROL 1
    ROL TM1
    ADC TM0, R1

    LD RK, -Z       // XOR RK
    EOR TM1, RK
    LD RK, -Z
    EOR TM0, RK

    SUB X30, TM0    // MINUS
    SBC X31, TM1

    DEC RC          // --rc

    EOR X30, RC     // XOR RC

    MOVW TM0, X30   // BLOCK ROT
    MOVW X30, X20
    MOVW X20, X10
    MOVW X10, X00
    MOVW X00, TM0
.endm
```

Q & A