CHAM 256 encrypt,decrypt 구현

유튜브: https://www.youtube.com/watch?v=DZ8y-Bjwl0M

CHAM 256

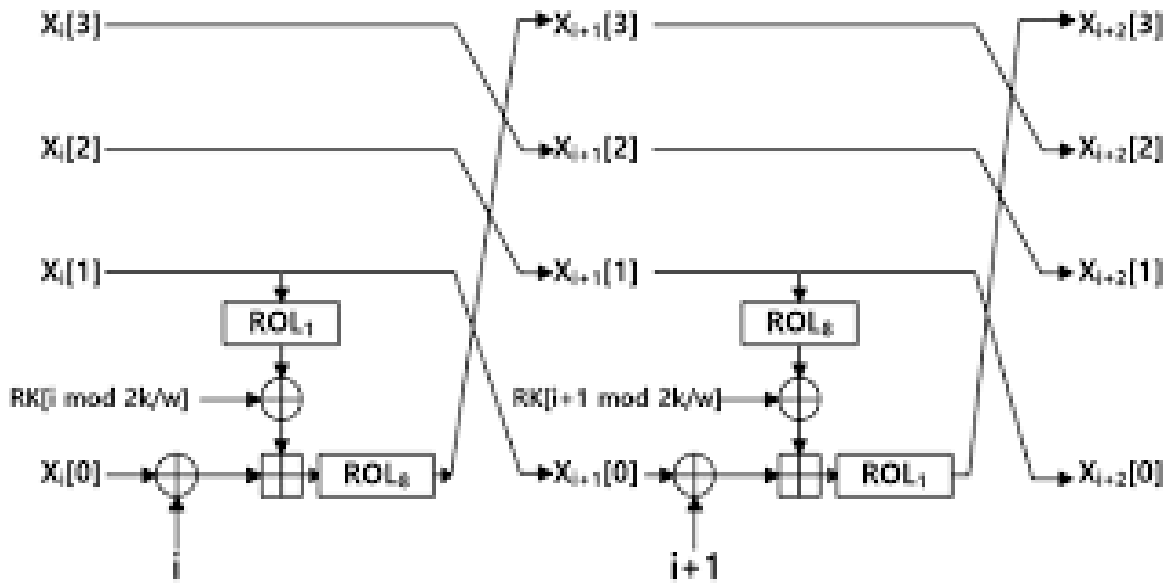
CHAM 256 c언어 분석

CHAM 256 구현

CHAM 256

- CHAM 64/128, CHAM128/128, CHAM 128/256

- CHAM 128/256 => 평문(PT) 128bit , 키(RK) 256bit
  ROUND = 120 rounds



CHAM 256 c언어 분석

```
void test_cham256()
{
    uint8_t mk[] = {
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
        0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xF5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
    };

    uint8_t pt[] = {
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff,
    };

    uint8_t ct[] = {
        0xdc, 0x77, 0x73, 0x02, 0x51, 0x56, 0x0b, 0x12, 0x95, 0x9b, 0x83, 0x8f, 0x75, 0xc0, 0x5e, 0x5e
    };

    uint8_t encrypted[16] = {0,};
    uint8_t decrypted[16] = {0,};

    uint8_t rks[4 * 16] = {0};

    cham256_keygen(rks, mk);
    cham256_encrypt(encrypted, pt, rks);
    cham256_decrypt(decrypted, ct, rks);
```

```
void cham256_keygen(uint8_t* rks, const uint8_t* mk)
{
    const uint32_t* key = (uint32_t*) mk;
    uint32_t* rk = (uint32_t*) rks;

    for (size_t i = 0; i < 8; ++i) {
        rk[i] = key[i] ^ rol32(key[i], 1);
        rk[(i+8)^(0x1)] = rk[i] ^ rol32(key[i], 11);
        rk[i] ^= rol32(key[i], 8);
    }
}
```

CHAM 256 c언어 분석

```c
void cham256_encrypt(uint8_t* dst, const uint8_t* src, const uint8_t* rks)
{
    uint32_t blk[4] = {0};
    memcpy(blk, src, BLOCKSIZE_128);

    const uint32_t* rk = (const uint32_t*) rks;
    uint32_t rc = 0;

    for (size_t round = 0; round < CHAM_128_256_ROUNDS; round += 8) {
        blk[0] = rol32((blk[0] ^ (rc++)) + (rol32(blk[1], 1) ^ rk[0]), 8);
        blk[1] = rol32((blk[1] ^ (rc++)) + (rol32(blk[2], 8) ^ rk[1]), 1);
        blk[2] = rol32((blk[2] ^ (rc++)) + (rol32(blk[3], 1) ^ rk[2]), 8);
        blk[3] = rol32((blk[3] ^ (rc++)) + (rol32(blk[0], 8) ^ rk[3]), 1);

        blk[0] = rol32((blk[0] ^ (rc++)) + (rol32(blk[1], 1) ^ rk[4]), 8);
        blk[1] = rol32((blk[1] ^ (rc++)) + (rol32(blk[2], 8) ^ rk[5]), 1);
        blk[2] = rol32((blk[2] ^ (rc++)) + (rol32(blk[3], 1) ^ rk[6]), 8);
        blk[3] = rol32((blk[3] ^ (rc++)) + (rol32(blk[0], 8) ^ rk[7]), 1);

        rk = (rk == (const uint32_t*) rks) ? rk + 8 : rk - 8;
    }

    memcpy(dst, blk, BLOCKSIZE_128);
}
```

```c
void cham256_decrypt(uint8_t* dst, const uint8_t* src, const uint8_t* rks)
{
    uint32_t blk[4] = {0};
    memcpy(blk, src, BLOCKSIZE_128);

    const uint32_t* rk = (const uint32_t*) rks;
    uint32_t rc = CHAM_128_256_ROUNDS;

    for (size_t round = 0; round < CHAM_128_256_ROUNDS; round += 8) {
        blk[3] = (ror32(blk[3], 1) - (rol32(blk[0], 8) ^ rk[7])) ^ (--rc);
        blk[2] = (ror32(blk[2], 8) - (rol32(blk[3], 1) ^ rk[6])) ^ (--rc);
        blk[1] = (ror32(blk[1], 1) - (rol32(blk[2], 8) ^ rk[5])) ^ (--rc);
        blk[0] = (ror32(blk[0], 8) - (rol32(blk[1], 1) ^ rk[4])) ^ (--rc);

        blk[3] = (ror32(blk[3], 1) - (rol32(blk[0], 8) ^ rk[3])) ^ (--rc);
        blk[2] = (ror32(blk[2], 8) - (rol32(blk[3], 1) ^ rk[2])) ^ (--rc);
        blk[1] = (ror32(blk[1], 1) - (rol32(blk[2], 8) ^ rk[1])) ^ (--rc);
        blk[0] = (ror32(blk[0], 8) - (rol32(blk[1], 1) ^ rk[0])) ^ (--rc);

        rk = (rk == (const uint32_t*) rks) ? rk + 8 : rk - 8;
    }

    memcpy(dst, blk, BLOCKSIZE_128);
}
```

Q & A