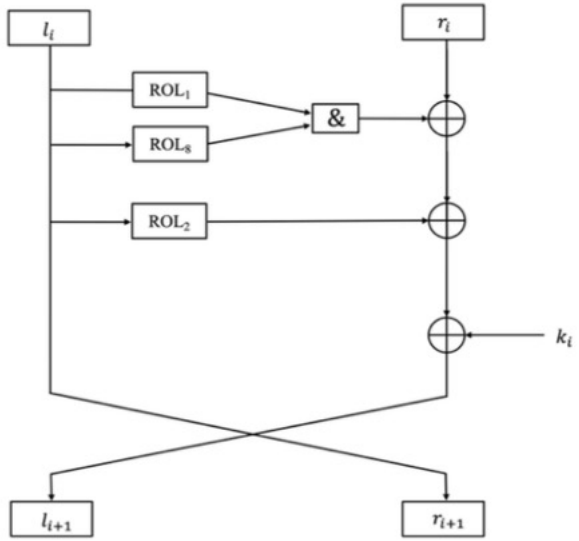
32-bit RISC-V 프로세서 상에서의 블록암호 SIMECK 최적 병렬 구현

https://youtu.be/ezNe1j8GqPs

SIMECK

- CHES'15에서 발표된 Feistel 구조의 경량 블록암호
- SIMON의 라운드 함수를 일부 변형 + SPECK의 키 스케줄링과 유사



The Round Function of Simon

Cipher	n	k	r	w
SIMECK-32/64	32	64	32	16
SIMECK-48/96	48	96	36	24
SIMECK-64/128	64	128	44	32

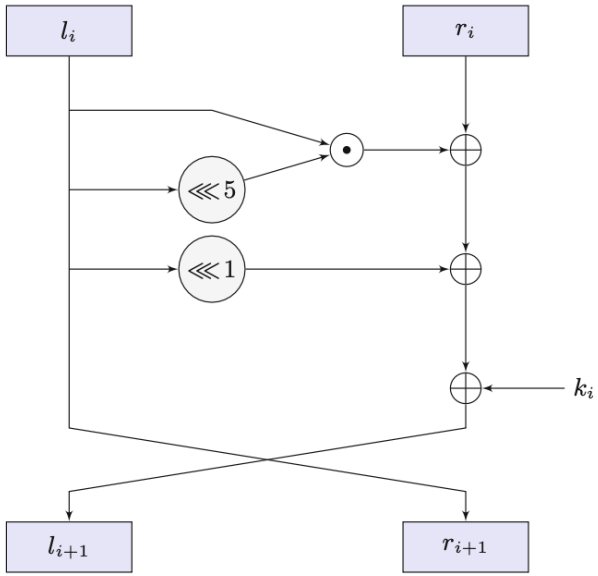


Fig. 1. The Round Function of Simeck

SIMECK

- 키 스케줄링

```
void simeck_64_128(
        const uint32_t master_key[],
        const uint32_t plaintext[],
        uint32_t ciphertext[]
) {
    const int NUM_ROUNDS = 44;
    int idx;

    uint32_t keys[4] = {
        master_key[0],
        master_key[1],
        master_key[2],
        master_key[3],
    };
    ciphertext[0] = plaintext[0];
    ciphertext[1] = plaintext[1];
    uint32_t temp;

    uint32_t constant = 0xFFFFFFFC;
    uint64_t sequence = 0x938BCA3083F;
//NUM_ROUNDS
    for (idx = 0; idx < NUM_ROUNDS; idx++) {
        ROUND64(
                keys[0],
                ciphertext[1],
                ciphertext[0],
                temp
        );

        constant &= 0xFFFFFFFC;
        constant |= sequence & 1;
        sequence >>= 1;
        ROUND64(
                constant,
                keys[1],
                keys[0],
                temp
        );

        // rotate the LFSR of keys
        temp = keys[1];
        keys[1] = keys[2];
        keys[2] = keys[3];
        keys[3] = temp;
    }

}
```

```
#define ROUND64(key, lft, rgt, tmp) do { \
    tmp = (lft); \
    lft = ((lft) & LROT32((lft), 5)) ^ LROT32((lft), 1) ^ (rgt) ^ (key); \
    rgt = (tmp); \
} while (0)
```

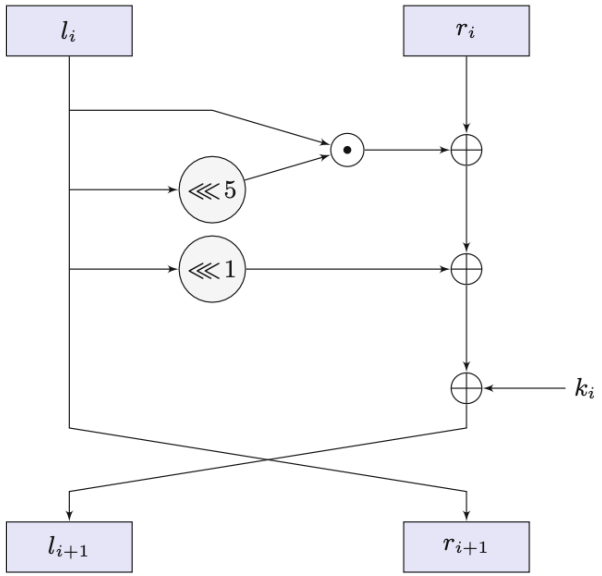


Fig. 1. The Round Function of Simeck

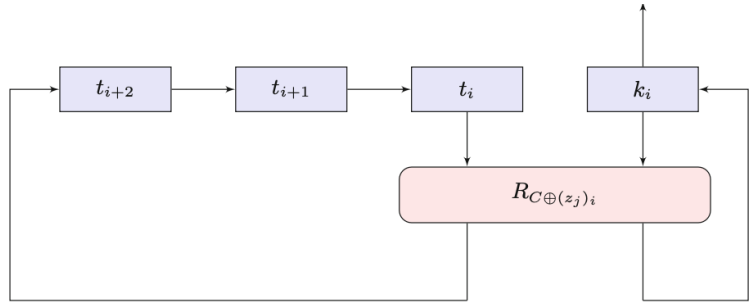


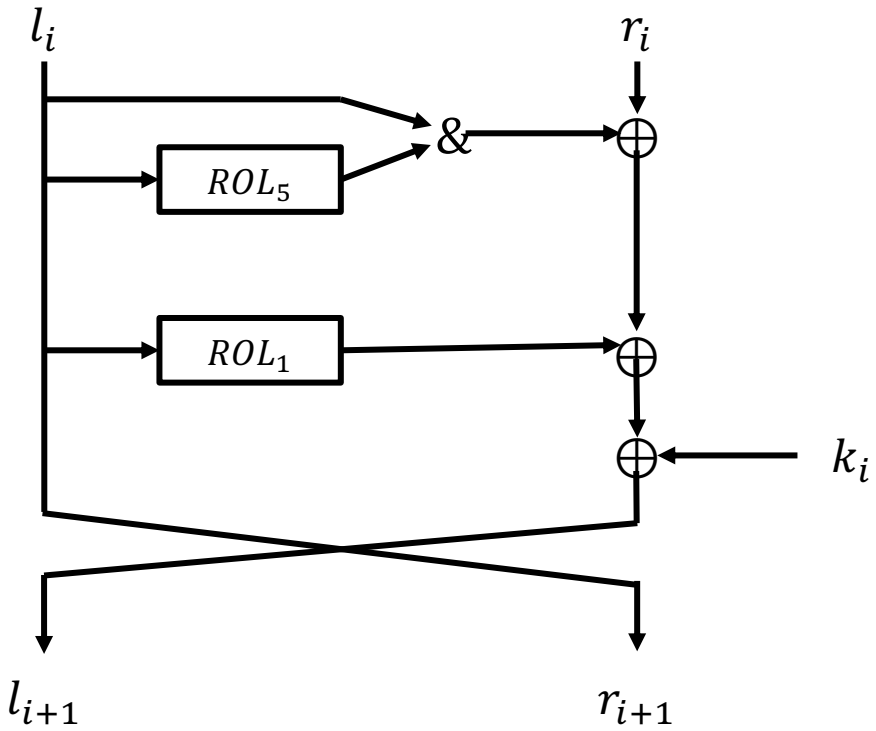
Fig. 2. The Key Expansion of Simeck, where $R_{C\oplus(z_j)_i}$ is the Simeck Round Function with $C \oplus (z_j)_i$ Acting as the Round Key

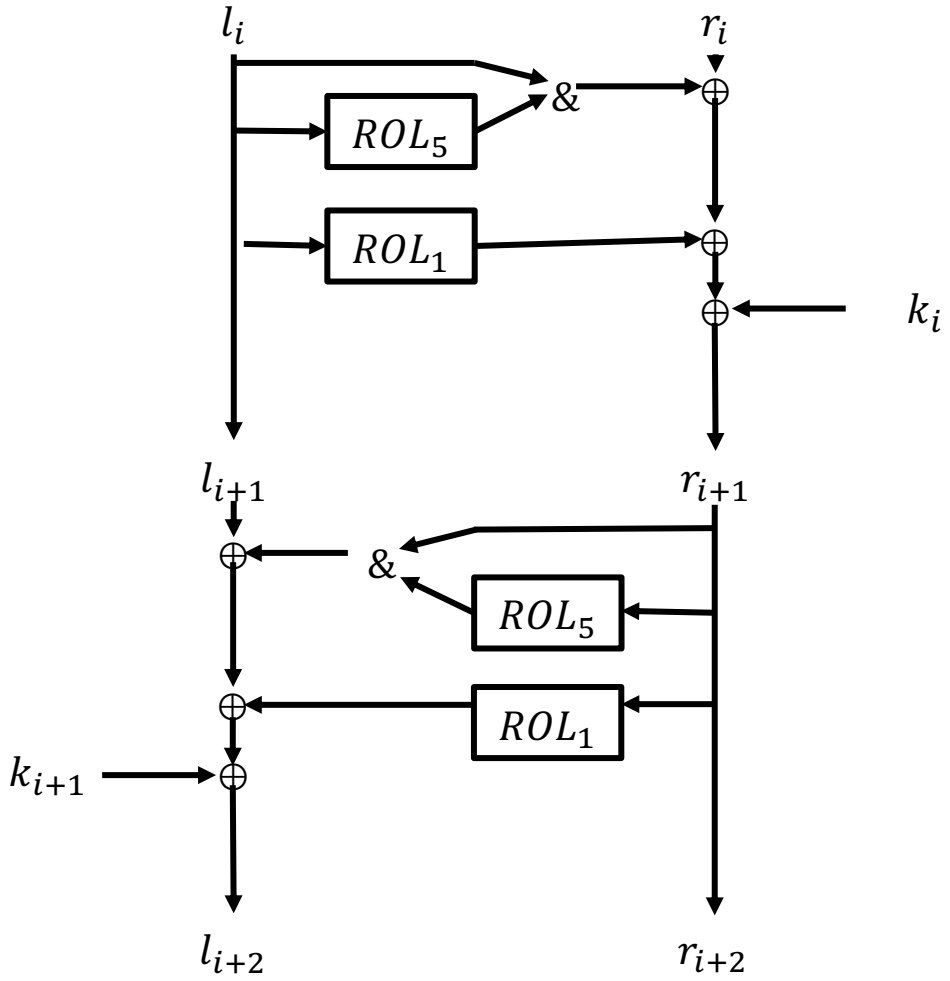
```
==0 round==
0x03020100, 0x0b0a0908, 0x13121110, 0x1b1a1918,
==1 round==
0x0b0a0908, 0x13121110, 0x1b1a1918, 0xebe9eded,
==2 round==
0x13121110, 0x1b1a1918, 0xebe9eded, 0xd0d3d4d5,
==3 round==
0x1b1a1918, 0xebe9eded, 0xd0d3d4d5, 0xd9dbdddd,
==4 round==
0xebe9eded, 0xd0d3d4d5, 0xd9dbdddd, 0x5a1f9093,
==5 round==
0xd0d3d4d5, 0xd9dbdddd, 0x5a1f9093, 0xa5e32b2b,
==6 round==
0xd9dbdddd, 0x5a1f9093, 0xa5e32b2b, 0x85c0090a,
==7 round==
0x5a1f9093, 0xa5e32b2b, 0x85c0090a, 0xd0091304,
==8 round==
0xa5e32b2b, 0x85c0090a, 0xd0091304, 0x4a471818,
==9 round==
0x85c0090a, 0xd0091304, 0x4a471818, 0xd19cc7c2,
==10 round==
0xd0091304, 0x4a471818, 0xd19cc7c2, 0xda2dd0ff,
==11 round==
0x4a471818, 0xd19cc7c2, 0xda2dd0ff, 0xf33bdcc0,
==12 round==
0xd19cc7c2, 0xda2dd0ff, 0xf33bdcc0, 0x0719a822,
```

SIMECK-64/128 라운드별 키 값

단일 평문 구현

- 블록 이동 생략

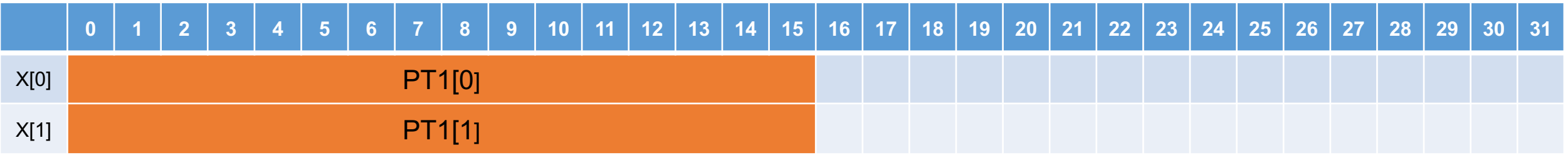




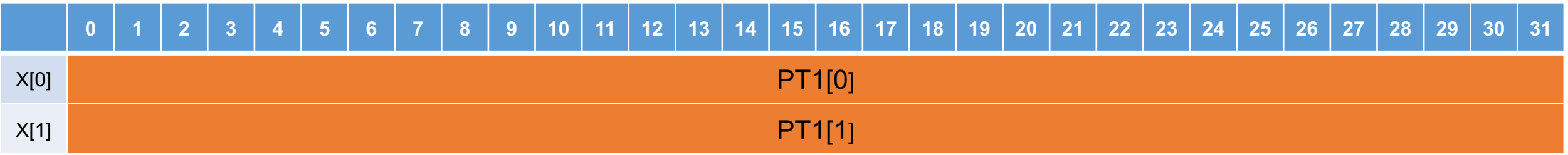
단일 평문 구현

Register	a2	a3
	PT1[0]	PT1[1]

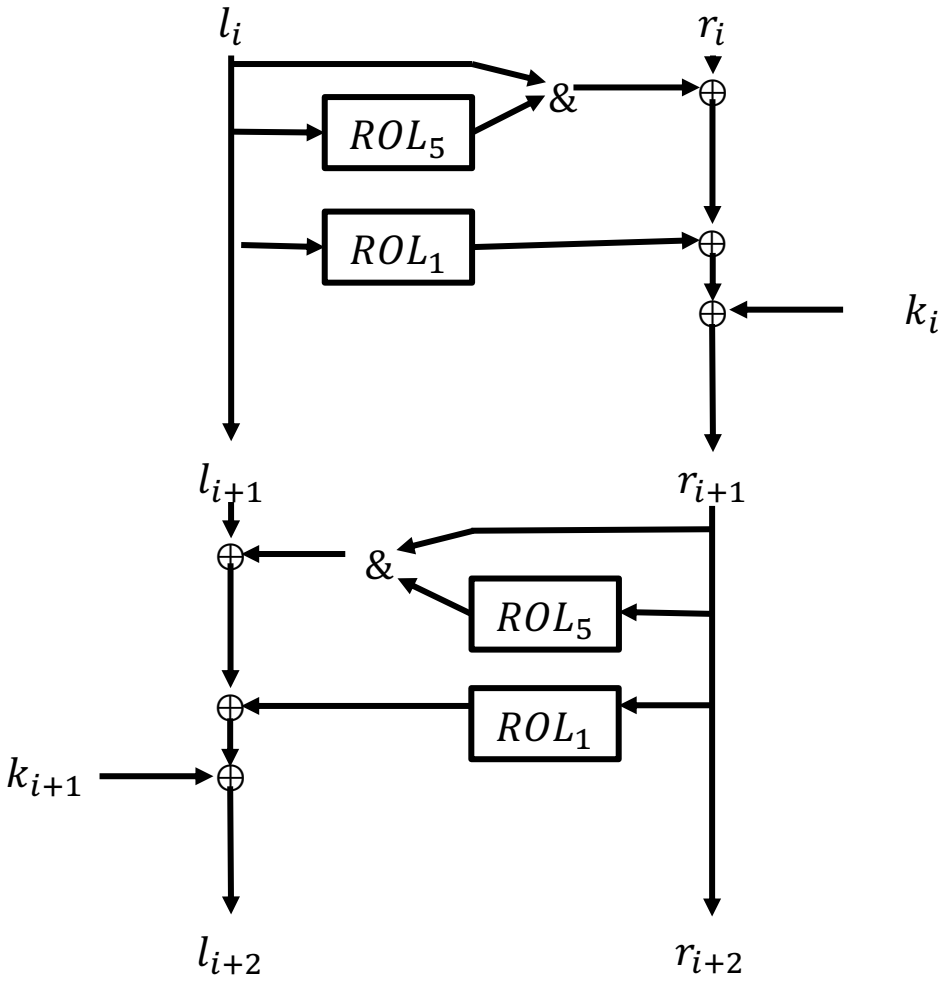
SIMECK-32/64



SIMECK-64/128



단일 평문 구현




```
.macro simeck_round
 //tmp : t0
    lw      a4, 0(a1)

    slli    t1, a3, 5
    srli    t2, a3, 27
    or     t4, t1, t2

    and     t0, t0, t4

    slli    t1, a3, 1
    srli    t2, a3, 31
    or     t4, t1, t2

    xor     t0, t0, t4
    xor     t0, t0, a2
    xor     t0, t0, a4

    mv      a2, t0

    addi    a1, a1, 4
//***********************
    lw      a4, 0(a1)

    slli    t1, a2, 5
    srli    t2, a2, 27
    or     t4, t1, t2

    and     t0, t0, t4

    slli    t1, a2, 1
    srli    t2, a2, 31
    or     t4, t1, t2

    xor     t0, t0, t4
    xor     t0, t0, a3
    xor     t0, t0, a4

    mv      a3, t0

    addi    a1, a1, 4
.endm
```

```
.macro simeck_round_32

    slli    a5, a3, 5
    and     a5, a5, t2
    srli    a6, a3, 11
   and     a6, a6, t3
    xor     a5, a5, a6

    and     t0, t0, a5

    slli    a5, a3, 1
    and     a5, a5, t4
    srli    a6, a3, 15
    and     a6, a6, t5
    xor     a5, a5, a6

    xor     t0, t0, a5

    xor     t0, t0, a2

    lh      a4, 0(a1)
    xor     t0, t0, a4

    mv      a2, t0

    addi    a1, a1, 2
```

평문 2개 병렬

- 평문 2개 암호화 (32/64)

Reg	a2		a3	
	PT1[0]	PT1[1]	PT2[0]	PT2[1]

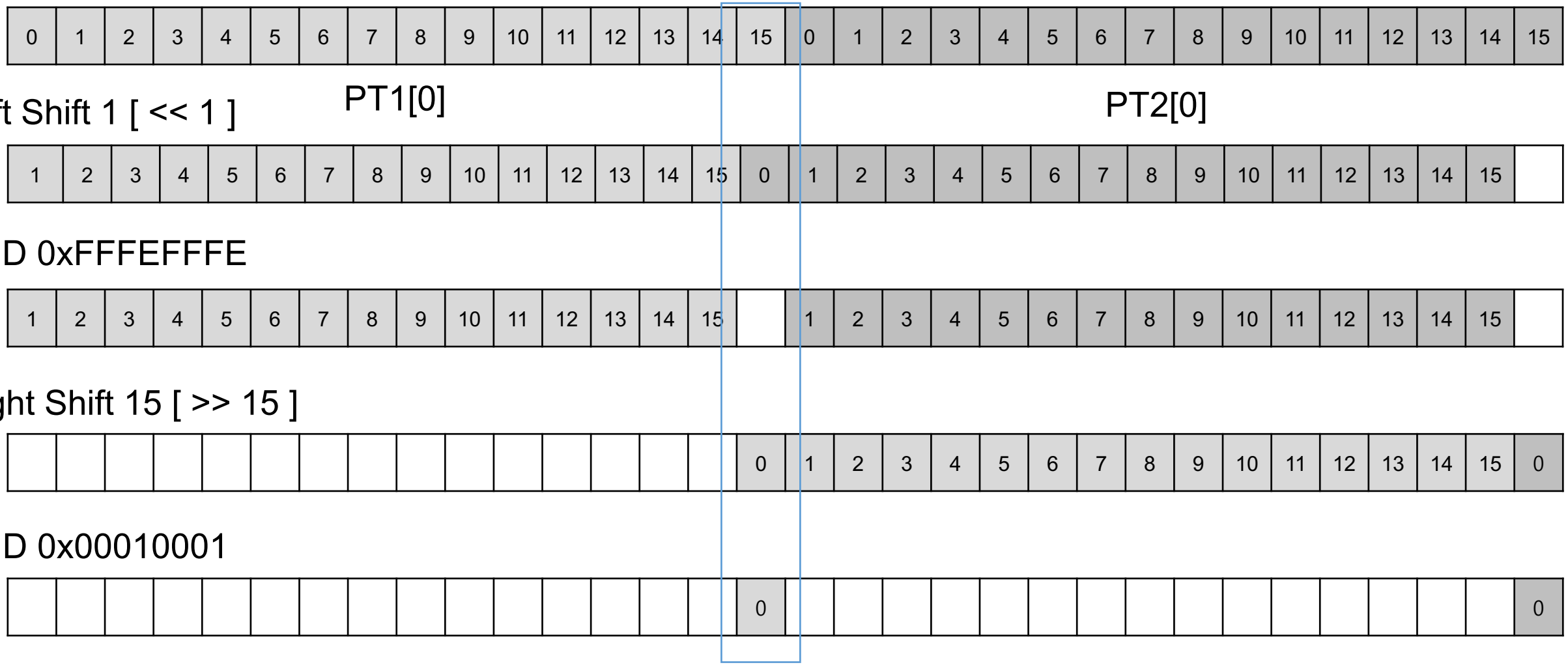
```
lh      a2, 0(a0)
lh      a3, 2(a0)

slli    a2, a2, 16
slli    a3, a3, 16

lh      t2, 4(a0)
lh      t3, 6(a0)

or      a2, a2, t2
or      a3, a3, t3

li      t2, 0xFFE0FFE0
li      t3, 0x001F001F

li      t4, 0xfffefffe
li      t5, 0x00010001
```

AND		
0	0	0
0	1	0
1	0	0
1	1	1

```
slli    a5, a3, 5
and     a5, a5, t2
srli    a6, a3, 11
and     a6, a6, t3
xor     a5, a5, a6

and     t0, t0, a5

slli    a5, a3, 1
and     a5, a5, t4
srli    a6, a3, 15
and     a6, a6, t5
xor     a5, a5, a6

xor     t0, t0, a5

xor     t0, t0, a2

lw      a4, 0(a1)
xor     t0, t0, a4

mv      a2, t0

addi    a1, a1, 4
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X[0]	PT1[0]																PT2[0]															
X[1]	PT1[1]																PT2[1]															

평문 2개 병렬 32/64

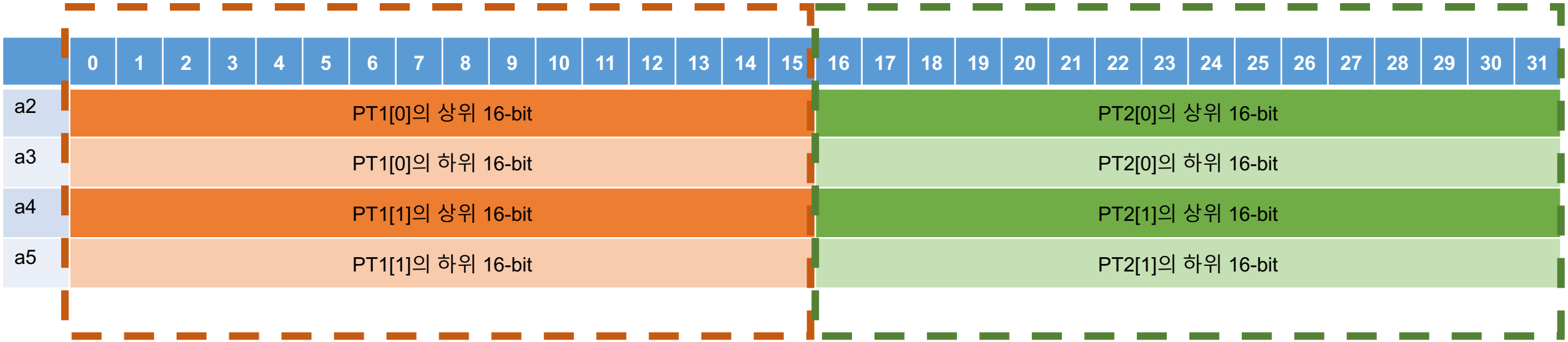

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
PT1[0]
PT2[0]
Left Shift 1 [<< 1]
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
AND 0xFFFEFFFE
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Right Shift 15 [>> 15]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
AND 0x00010001
0 0
Left Rotation 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0

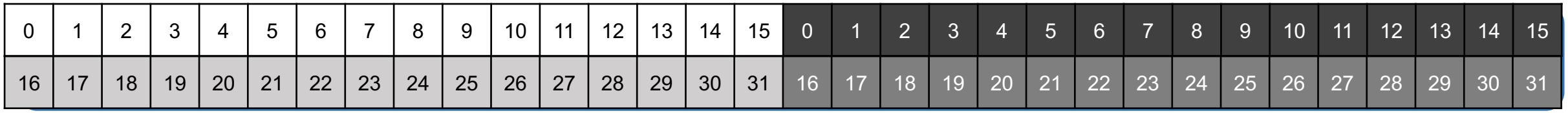
평문 2개 병렬

- 평문 2개 암호화 (64/128)

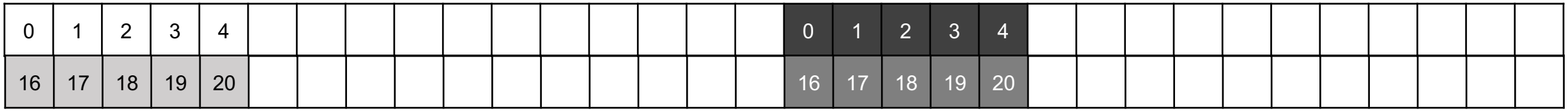
```
lh      a3, 0(a0)
lh      a4, 2(a0)
lh      a5, 4(a0)
lh      a6, 6(a0)

slli    a3, a3, 16
slli    a4, a4, 16
slli    a5, a5, 16
slli    a6, a6, 16

lh      s0, 8(a0)
lh      s1, 10(a0)
lh      s2, 12(a0)
lh      s3, 14(a0)

or      a3, a3, s0
or      a4, a4, s1
or      a5, a5, s2
or      a6, a6, s3
```

Reg	a2		a3		a4		a5	
	PT1[0] 상위 16bit	PT1[0] 하위 16bit	PT1[1] 상위 16bit	PT1[0] 하위 16bit	PT2[0] 상위 16bit	PT2[0] 하위 16bit	PT2[1] 상위 16bit	PT2[1] 하위 16bit

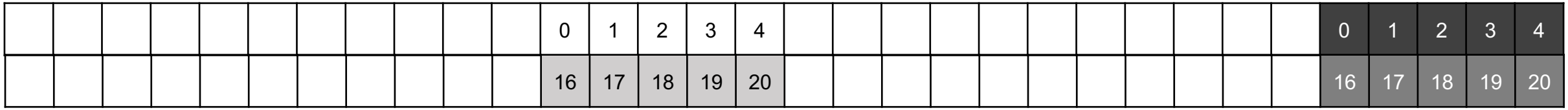




AND 0xF800F800

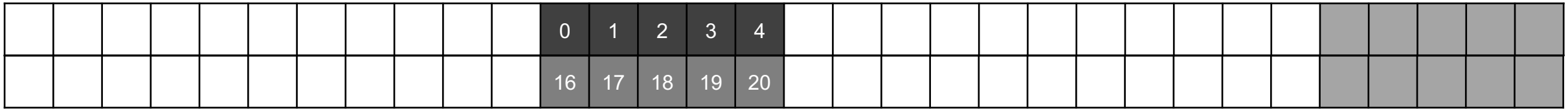


Shift right 11[>>16]

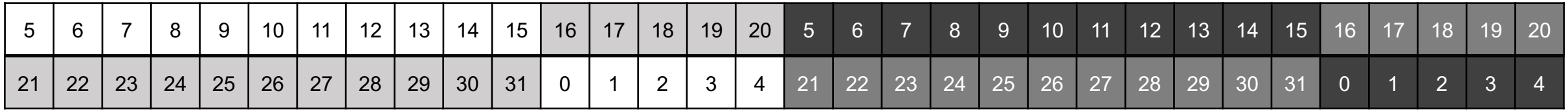


Shift left 5 [<<5]

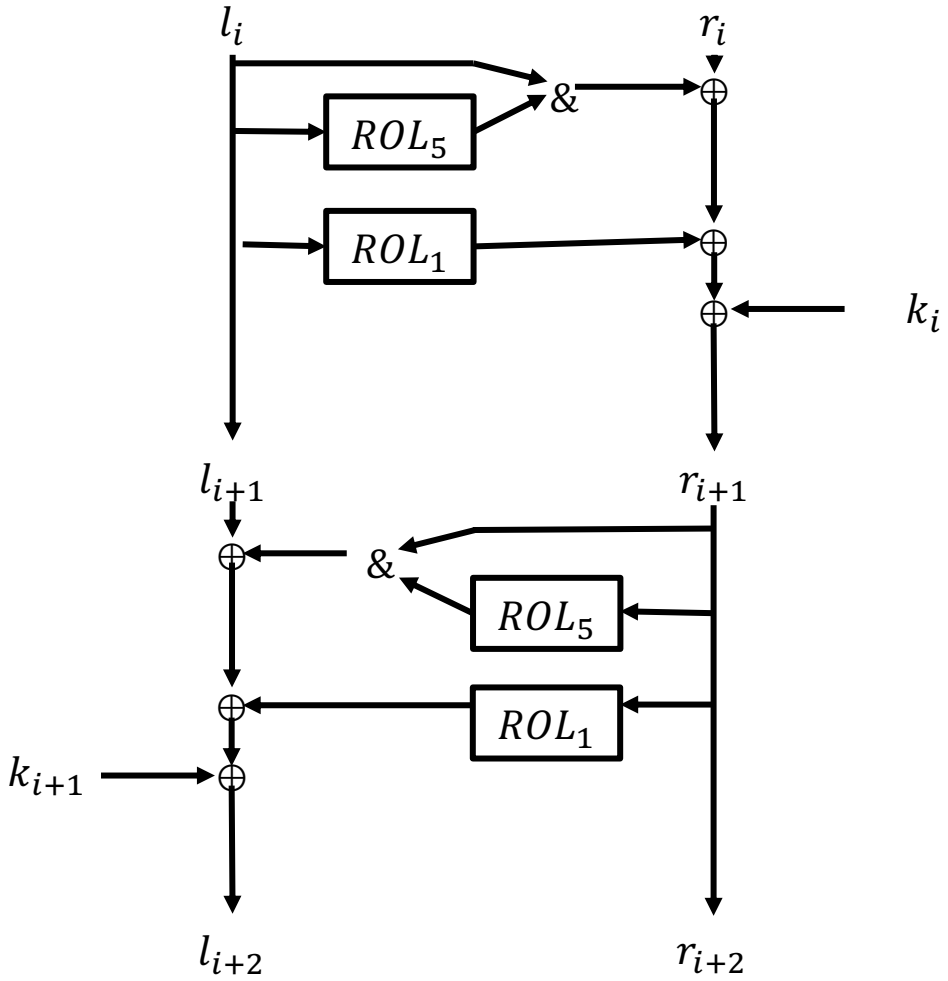
AND 0xFFE0FFE0



Rotation left 5 [<<5]



평문 2개 병렬 64/128




```
li      t4, 0xF800F800
li      t5, 0xFFE0FFE0
li      t6, 0x80008000
li      s3, 0xfffefffe
```

```
.macro simeck_round_parl
//////////////////////////////////////////
    and     s2, a4, t4
    srli    s2, s2, 11 //01234 01234

    and     s4, a5, t4
    srli    s4, s4, 11 //1617181920 1617181920


    slli    s0, a4, 5
    and     s0, s0, t5
    xor     s0, s0, s4


    slli    s1, a5, 5
    and     s1, s1, t5
    xor     s1, s1, s2


    and     t0, t0, s0
    and     t1, t1, s1
//////////////////////////////////////////

    and     s2, a4, t6
    srli    s2, s2, 15 //0 0

    and     s4, a5, t6
    srli    s4, s4, 15 //16 16

    slli    s0, a4, 1
    and     s0, s0, s3
    xor     s0, s0, s4

    slli     s1, a5, 1
    and     s1, s1, s3
    xor     s1, s1, s2

    xor     t0, t0, s0
    xor     t1, t1, s1

//////////////////////////////////////////
    xor     t0, t0, a2
    xor     t1, t1, a3
//////////////////////////////////////////
    lw      a6, 0(a1) //rk1
    lw      a7, 4(a1)

    xor     t0, t0, a6
    xor     t1, t1, a7
//////////////////////////////////////////
    mv      a2, t0
    mv      a3, t1

    addi    a1, a1, 8
//////////////////////////////////////////
```

성능 평가

SIMECK-32/64	clock cycles	cpb
Ref-c	3,298	824.50
1-PT	**655**	163.75
2-PT	**635**	158.75

SIMECK-64/128	clock cycles	cpb
Ref-c	2,734	341.75
1-PT	612	76.50
2-PT	1,560	195

SIMECK-32/64의 구현물의 경우, 병렬 구현물이 명령어 수는 더 많은데 더 적은 Clock cycles이 나옴...?

Q & A