AVX2 구현 연습
: Feistel + PIPO

https://youtu.be/BNH748U8bYg

Contents

CryptoCraft LAB

개요

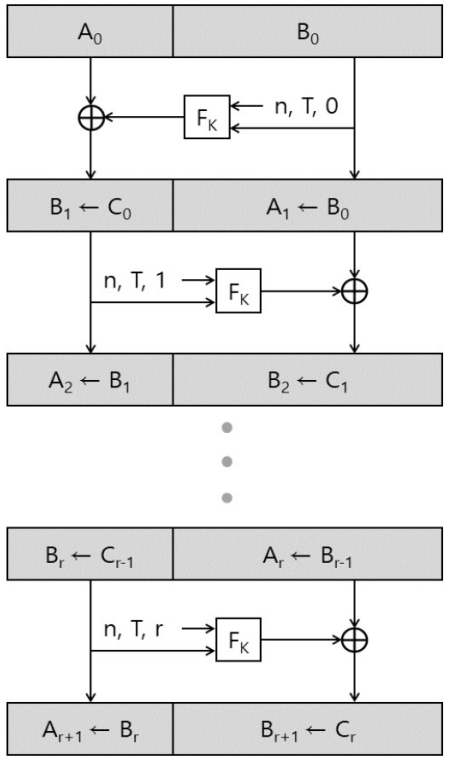
- FF1에 병렬 구현된 AVX-PIPO를 적용하기 위해 FF1 병렬 구현 필요
- Format Preserving Encryption은 Feistel 구조
- FF1을 처음부터 병렬 구현하기 어려워서.. Feistel 구조 아무거나 만들어 보았습니다..

구조

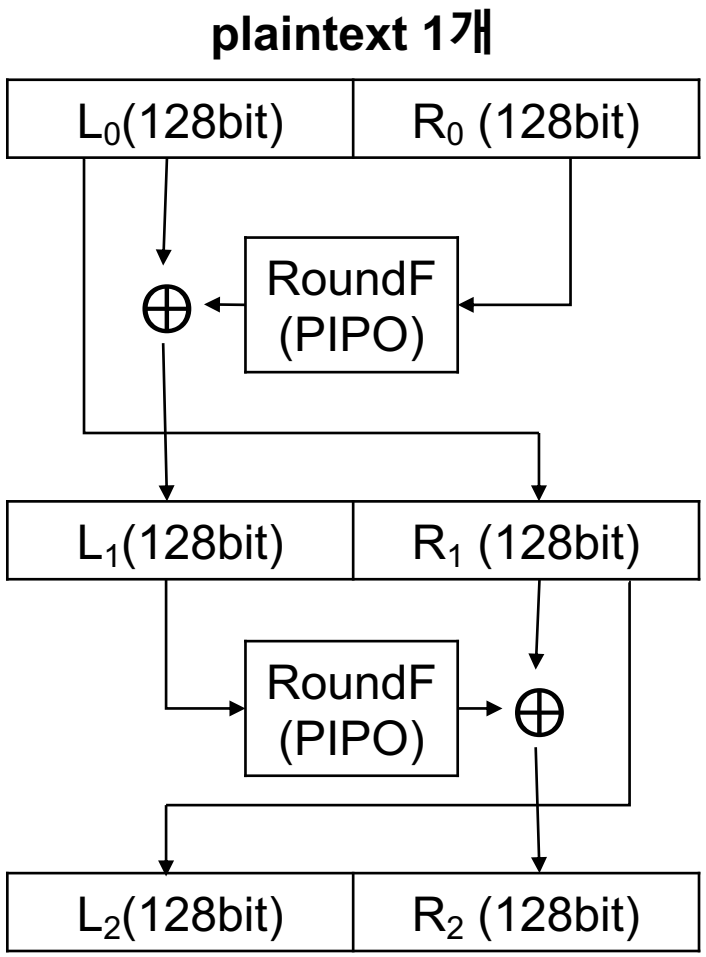
- AVX2-PIPO는 64bit plaintext 32개 처리 가능
- 128bit plaintext (L+R)으로 설정 (32개)

  → L or R이 64bit plaintext 형태로 PIPO에 입력
- 2 round
- 홀수 round는 RoundF(R)

  짝수 round는 RoundF(L)



FF1



구조

- 병렬 구현 포인트는 RoundF…
- RoundF 함수에 사용되는 PIPO를 위해 전체 plaintext의 절반인 L or R을 각각 병렬 처리

  → 반쪽이 64-bit 이므로 32개 만들었음 (PIPO에 64비트짜리 32개 딱 들어가게..)
- 즉 128bit plaintext 32개 생성 후, 반으로 나눠 각 plaintext의 L or R만 모아서 병렬 구현

main

- 평문은 결과 확인하려고

  PIPO test vector사용

- 32개의 평문, 각 평문은 16 byte

  0~7 : L // 8~15 : R

plain[32][16]

0,0 0x26	0,1 0x00	…	0,7 0x09	0,8 0x26	0,9 0x00		0,15 0x09
31,0			31,7				31,15

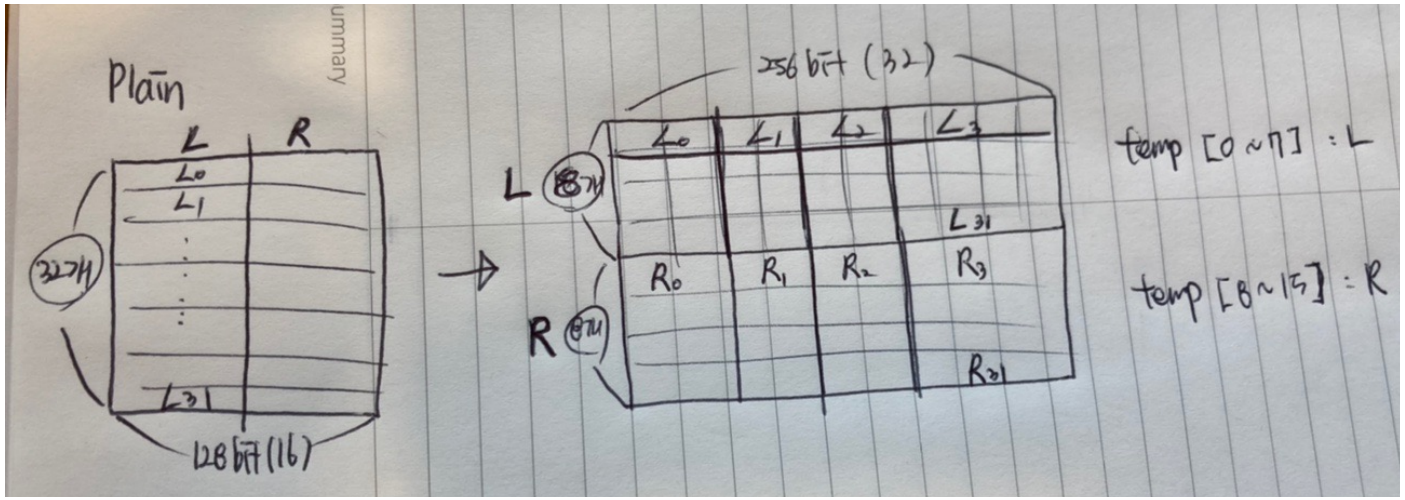
- Enc 함수 호출

```
int main()
{
    uint8_t plain[32][16] ={0x00,};
    uint8_t roundkey[3][8] = {{0x97,0x22,0x15,0x2E,0xAD,0x20,0x1D,0x7E},\
                              {0x97,0x22,0x15,0x2E,0xAD,0x20,0x1D,0x7E},\
                              {0x97,0x22,0x15,0x2E,0xAD,0x20,0x1D,0x7E}};

    for (int i=0; i<32;i++){    //test vector
            plain[i][0]= 0x26;
            plain[i][1]= 0x00;
            plain[i][2]= 0x27;
            plain[i][3]= 0x1E;
            plain[i][4]= 0xF6;
            plain[i][5]= 0x52;
            plain[i][6]= 0x85;
            plain[i][7]= 0x09;  // L

            plain[i][8]= 0x26;
            plain[i][9]= 0x00;
            plain[i][10]= 0x27;
            plain[i][11]= 0x1E;
            plain[i][12]= 0xF6;
            plain[i][13]= 0x52;
            plain[i][14]= 0x85;
            plain[i][15]= 0x09;  // R

    }

    ENC_32_64bit(plain, roundkey);

    return 0;
}
```

Encryption

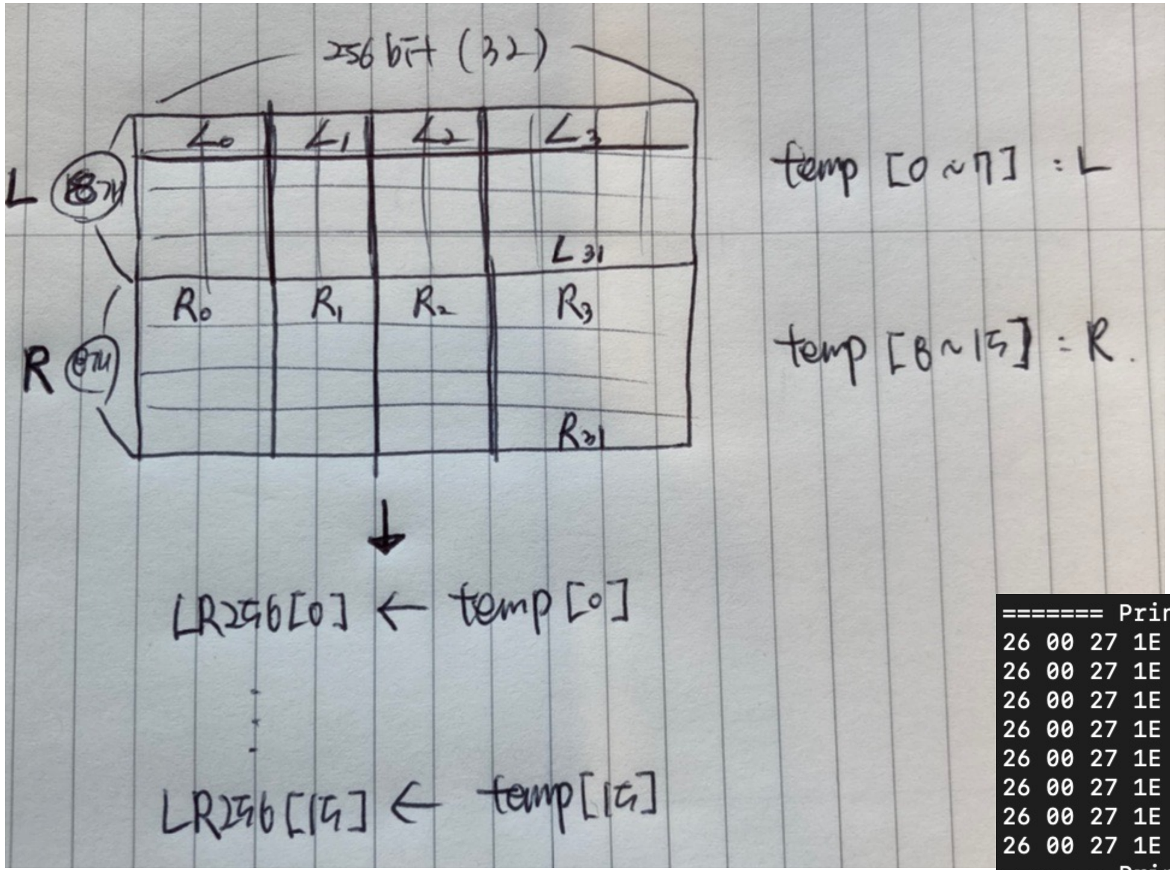
- 순서
  1. plaintext에서 L끼리 R끼리 모으기
  2. 256bit register에 load
  3. RoundF (PIPO S-layer만 사용해봄)
  4. L or R $\oplus$ RoundF(R or L)
  5. SWAP

plaintext에서 L끼리 R끼리 모으기




```
u8 temp[16][32];
int i = 0;
//평문 32개   temp 7까지 L, 15까지 R
for(int row = 0 ; row < 32 ; row++){
        temp[row/4][(row*8)%32] = plain[row][0];
        temp[row/4][((row*8)+1)%32] = plain[row][1];
        temp[row/4][((row*8)+2)%32] = plain[row][2];
        temp[row/4][((row*8)+3)%32] = plain[row][3];
        temp[row/4][((row*8)+4)%32] = plain[row][4];
        temp[row/4][((row*8)+5)%32] = plain[row][5];
        temp[row/4][((row*8)+6)%32] = plain[row][6];
        temp[row/4][((row*8)+7)%32] = plain[row][7];

        temp[(row/4)+8][(row*8)%32] = plain[row][8];
        temp[(row/4)+8][((row*8)+1)%32] = plain[row][9];
        temp[(row/4)+8][((row*8)+2)%32] = plain[row][10];
        temp[(row/4)+8][((row*8)+3)%32] = plain[row][11];
        temp[(row/4)+8][((row*8)+4)%32] = plain[row][12];
        temp[(row/4)+8][((row*8)+5)%32] = plain[row][13];
        temp[(row/4)+8][((row*8)+6)%32] = plain[row][14];
        temp[(row/4)+8][((row*8)+7)%32] = plain[row][15];
    }
```

256bit register에 load




```
__m256i LR256[16], T256[8];

LR256[0] = _mm256_loadu_si256((__m256i*)&temp[0]);
LR256[1] = _mm256_loadu_si256((__m256i*)&temp[1]);
LR256[2] = _mm256_loadu_si256((__m256i*)&temp[2]);
LR256[3] = _mm256_loadu_si256((__m256i*)&temp[3]);
LR256[4] = _mm256_loadu_si256((__m256i*)&temp[4]);
LR256[5] = _mm256_loadu_si256((__m256i*)&temp[5]);
LR256[6] = _mm256_loadu_si256((__m256i*)&temp[6]);
LR256[7] = _mm256_loadu_si256((__m256i*)&temp[7]);
LR256[8] = _mm256_loadu_si256((__m256i*)&temp[8]);
LR256[9] = _mm256_loadu_si256((__m256i*)&temp[9]);
LR256[10] = _mm256_loadu_si256((__m256i*)&temp[10]);
LR256[11] = _mm256_loadu_si256((__m256i*)&temp[11]);
LR256[12] = _mm256_loadu_si256((__m256i*)&temp[12]);
LR256[13] = _mm256_loadu_si256((__m256i*)&temp[13]);
LR256[14] = _mm256_loadu_si256((__m256i*)&temp[14]);
LR256[15] = _mm256_loadu_si256((__m256i*)&temp[15]);
```

```
======= Print L =======
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
======= Print R =======
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
```

RoundF (S-layer from PIPO)

- round에 따라 L, R 중 뭐 쓸지 고름

  → 홀수 라운드 : R, 짝수 라운드 : L 입력

- 홀수 라운드면 R

  → R은 LR256[8~15]

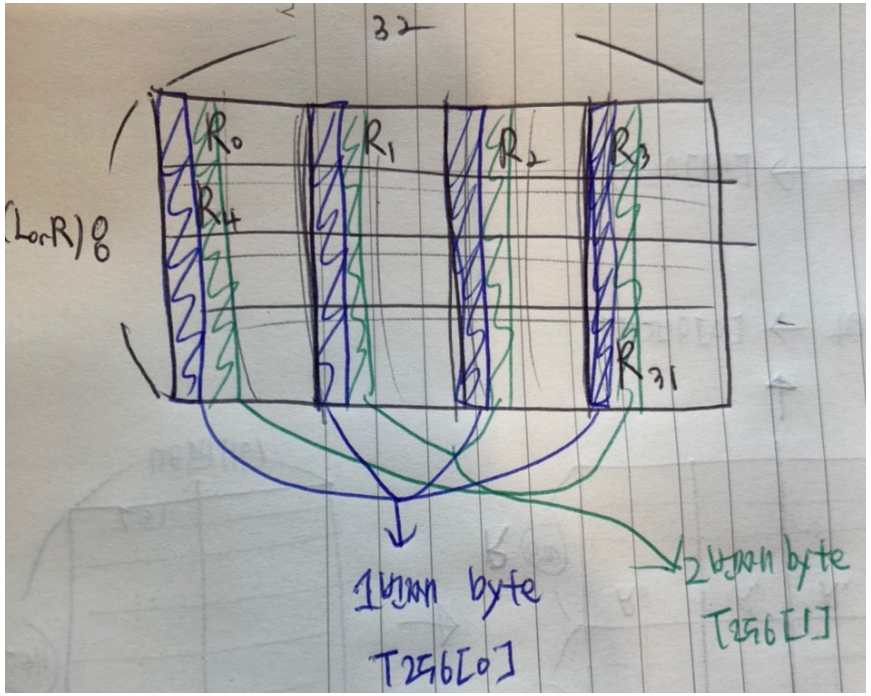
  → 256*8 bit 건너뛴 지점에서부터 시작

- 각 평문의 L or R (64bit)들에서

  동일 byte들끼리만 모아서 T256[n]에 set

- S-layer는 동일

```
u8* setpt = (u8*)LR256;
int idx = (round%2)*256;
```




```
T256[3]
    =_mm256_setr_epi8(setpt[idx+3],setpt[idx+11],setpt[idx+19],setpt[idx+27],setpt[idx+35],setpt[idx+43],setpt[idx+51],setpt[idx+59],setpt
    [idx+67],setpt[idx+75],setpt[idx+83],setpt[idx+91],setpt[idx+99],setpt[idx+107],setpt[idx+115],setpt[idx+123],setpt[idx+131],setpt
    [idx+139],setpt[idx+147],setpt[idx+155],setpt[idx+163],setpt[idx+171],setpt[idx+179],setpt[idx+187],setpt[idx+195],setpt[idx+203],setpt
    [idx+211],setpt[idx+219],setpt[idx+227],setpt[idx+235],setpt[idx+243],setpt[idx+251]);
```

RoundF 결과

```
======= S layer =======
d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1 c1
6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e 6e
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e 8e
ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca ca
```

PIPO 첫번째 S-layer 결과

1 round S-layer (RoundF) 결과

```
======= After S layer =======
D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3 D3
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1
6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E 6E
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E
CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA
```

L or R ⊕ RoundF(R or L)

- round에 따라 L, R 결정

  → 홀수 라운드면 LR256이 L, 짝수 라운드면 LR256이 R 되도록

- T256은 RoundF(R or L)의 결과값 저장되어 있음

  → XOR 한 후 갱신

```
i = (round-1)%2;

T256[0] = _mm256_xor_si256(LR256[i*8],T256[0]);
T256[1] = _mm256_xor_si256(LR256[i*8+1],T256[1]);
T256[2] = _mm256_xor_si256(LR256[i*8+2],T256[2]);
T256[3] = _mm256_xor_si256(LR256[i*8+3],T256[3]);
T256[4] = _mm256_xor_si256(LR256[i*8+4],T256[4]);
T256[5] = _mm256_xor_si256(LR256[i*8+5],T256[5]);
T256[6] = _mm256_xor_si256(LR256[i*8+6],T256[6]);
T256[7] = _mm256_xor_si256(LR256[i*8+7],T256[7]);
```

```
======= After (RoundF - XOR) =======
F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA
17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38
59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76
E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8
48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67
25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A
A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87
EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3
```

SWAP

- round에 따라 연산에 사용되지 않은 쪽 먼저 바꾸고 roundF 후 XOR한 결과도 반대쪽으로

```
============================================ Round 2 ============================================
======= Print L =======
F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA F5 D3 F4 CD 25 81 56 DA
17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38 17 31 16 2F C7 63 B4 38
59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76 59 7F 58 61 89 2D FA 76
E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8 E7 C1 E6 DF 37 93 44 C8
48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67 48 6E 49 70 98 3C EB 67
25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A 25 03 24 1D F5 51 86 0A
A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87 A8 8E A9 90 78 DC 0B 87
EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3 EC CA ED D4 3C 98 4F C3
======= Print R =======
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09 26 00 27 1E F6 52 85 09
```

```
LR256[(1-i)*8] = _mm256_loadu_si256((__m256i*)&LR256[i*8]);
LR256[(1-i)*8+1] = _mm256_loadu_si256((__m256i*)&LR256[i*8+1])
LR256[(1-i)*8+2] = _mm256_loadu_si256((__m256i*)&LR256[i*8+2])
LR256[(1-i)*8+3] = _mm256_loadu_si256((__m256i*)&LR256[i*8+3])
LR256[(1-i)*8+4] = _mm256_loadu_si256((__m256i*)&LR256[i*8+4])
LR256[(1-i)*8+5] = _mm256_loadu_si256((__m256i*)&LR256[i*8+5])
LR256[(1-i)*8+6] = _mm256_loadu_si256((__m256i*)&LR256[i*8+6])
LR256[(1-i)*8+7] = _mm256_loadu_si256((__m256i*)&LR256[i*8+7])

LR256[i*8] = _mm256_loadu_si256((__m256i*)&T256[0]);
LR256[i*8+1] = _mm256_loadu_si256((__m256i*)&T256[1]);
LR256[i*8+2] = _mm256_loadu_si256((__m256i*)&T256[2]);
LR256[i*8+3] = _mm256_loadu_si256((__m256i*)&T256[3]);
LR256[i*8+4] = _mm256_loadu_si256((__m256i*)&T256[4]);
LR256[i*8+5] = _mm256_loadu_si256((__m256i*)&T256[5]);
LR256[i*8+6] = _mm256_loadu_si256((__m256i*)&T256[6]);
LR256[i*8+7] = _mm256_loadu_si256((__m256i*)&T256[7]);
```

향후 계획

- swap 안 하고 L,R 반대로 넣을 수 있는지..
- ff1에 적용

Q & A