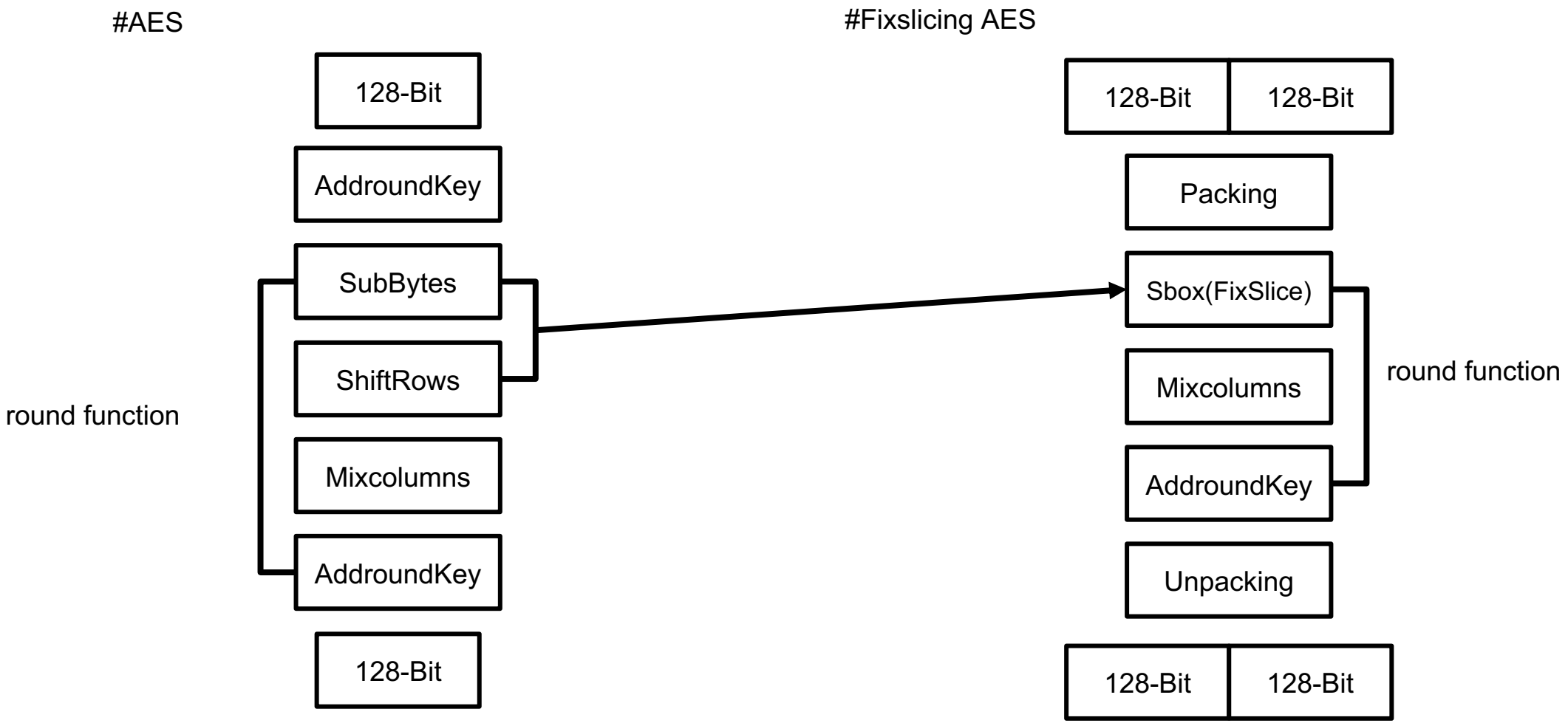
FixSlicing AES

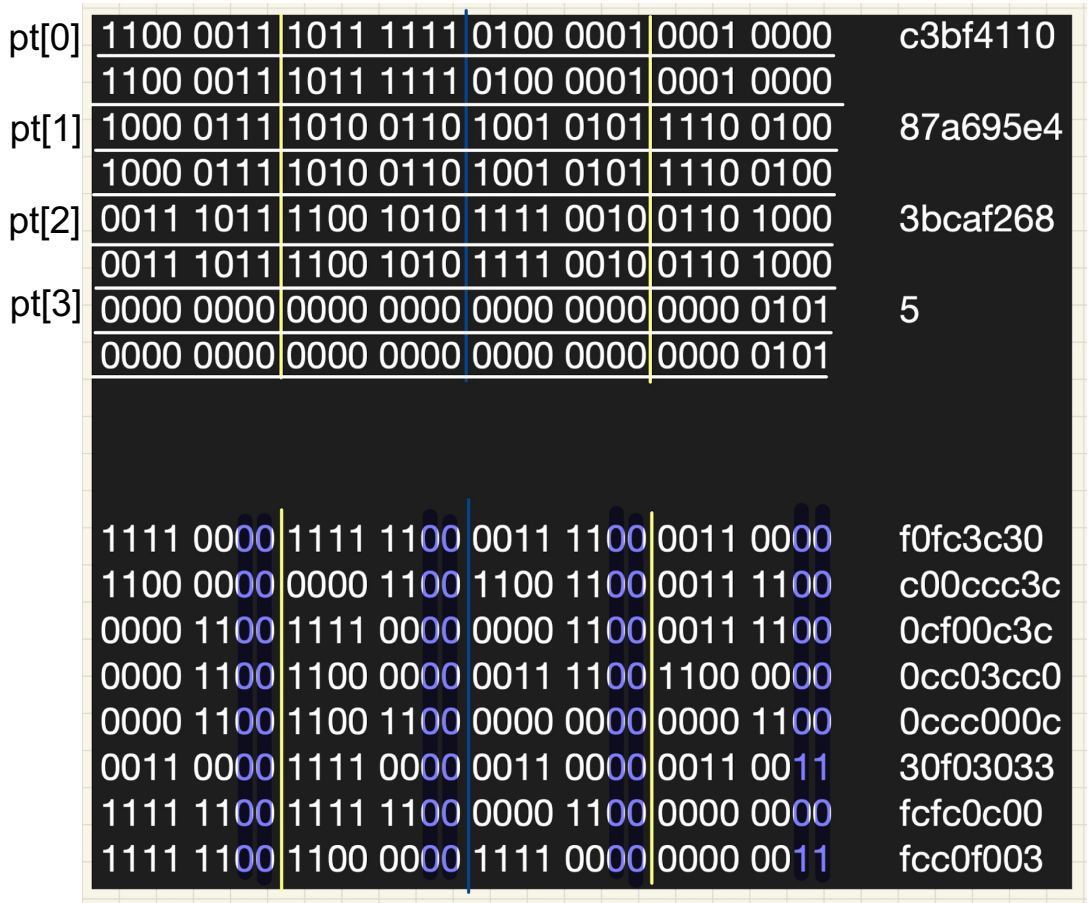
https://youtu.be/E2DKupsDGok

FixSlicing AES



FixSlicing AES – Packing

	row 3								·····	row 0							
	column 0		column 1		column 2		column 3		·····	column 0		column 1		column 2		column 3	
	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1	·····	block 0	block 1	block 0	block 1	block 0	block 1	block 0	block 1
R_0	b_{24}^0	b_{24}^1	b_{56}^0	b_{56}^1	b_{88}^0	b_{88}^1	b_{120}^0	b_{120}^1	·····	b_0^0	b_0^1	b_{32}^0	b_{32}^1	b_{64}^0	b_{64}^1	b_{96}^0	b_{96}^1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	·····	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
R_7	b_{31}^0	b_{31}^1	b_{63}^0	b_{63}^1	b_{95}^0	b_{95}^1	b_{127}^0	b_{127}^1	·····	b_7^0	b_7^1	b_{39}^0	b_{39}^1	b_{71}^0	b_{71}^1	b_{103}^0	b_{103}^1



FixSlicing AES – Packing

COLUMN

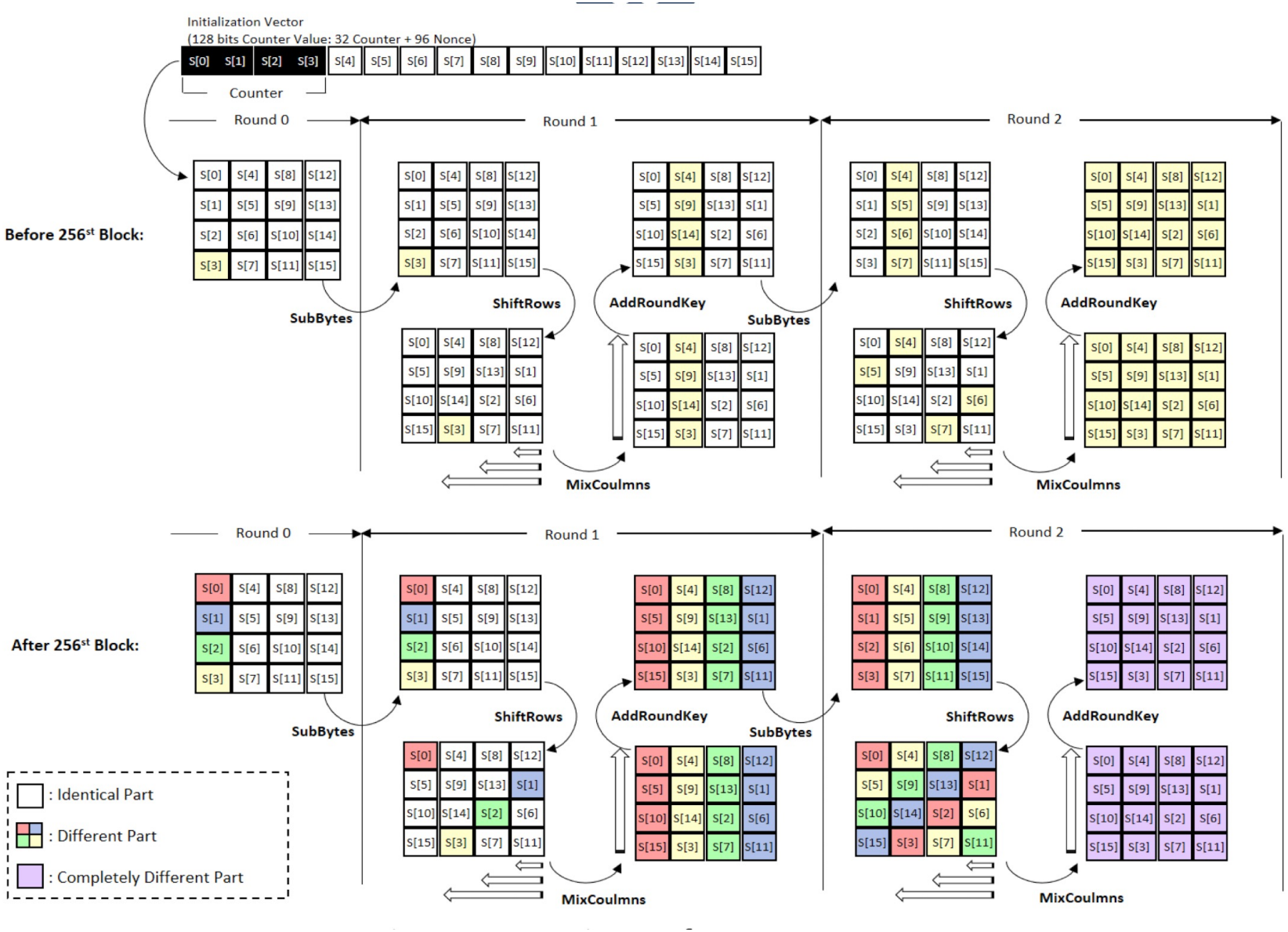
ROW	0	1	2	3
0	A0	A4	A8	A12
1	A1	A5	A9	A13
2	A2	A6	A10	A14
3	A3	A7	A11	A15

ROW	0	1	2	3
0	B0	B4	B8	B12
1	B1	B5	B9	B13
2	B2	B6	B10	B14
3	B3	B7	B11	B15

state[0]

ROW3								ROW2								ROW1								ROW0							
COL 0		COL 1		COL 2		COL 3		COL 0		COL 1		COL 2		COL 3		COL 0		COL 1		COL 2		COL 3		COL 0		COL 1		COL 2		COL 3	
A03	B03	A07	B07	A11	B11	A15	B15	A02	B02	A06	B06	A10	B10	A14	B14	A01	B01	A05	B05	A09	B09	A13	B13	A00	B00	A04	B04	A08	B08	A12	B12

FixSlicing AES – CTR



FixSlicing AES – CTR

```
void aes128_encrypt_ffs(unsigned char *ctext0, unsigned char *ctext1,
                        const unsigned char* ptext0, const unsigned char* ptext1,
                        const uint32_t* rkeys_ffs) {
    uint32_t state[8];                      // 256-bit internal state
    packing(state, ptext0, ptext1);         // packs into bitsliced representation
    ark(state, rkeys_ffs);                  // key whitening
    sbox(state);                           // 1st round
    mixcolumns_0(state);                   // 1st round
    ark(state, rkeys_ffs + 8);              // 1st round
    sbox(state);                           // 2nd round
    mixcolumns_1(state);                   // 2nd round
    ark(state, rkeys_ffs + 16);           // 2nd round
    sbox(state);                            // 3rd round
    mixcolumns_2(state);                    // 3rd round
    ark(state, rkeys_ffs + 24);           // 3rd round
    sbox(state);                            // 4th round
    mixcolumns_3(state);                    // 4th round
    ark(state, rkeys_ffs + 32);           // 4th round
    sbox(state);                            // 5th round
    mixcolumns_0(state);                    // 5th round
    ark(state, rkeys_ffs + 40);           // 5th round
    sbox(state);                           // 6th round
    mixcolumns_1(state);                    // 6th round
    ark(state, rkeys_ffs + 48);           // 6th round
    sbox(state);                            // 7th round
    mixcolumns_2(state);                    // 7th round
    ark(state, rkeys_ffs + 56);           // 7th round
    sbox(state);                            // 8th round
    mixcolumns_3(state);                    // 8th round
    ark(state, rkeys_ffs + 64);           // 8th round
    sbox(state);                            // 9th round
    mixcolumns_0(state);                    // 9th round
    ark(state, rkeys_ffs + 72);           // 9th round
    sbox(state);                            // 10th round
    double_shiftrows(state);                // 10th round (resynchronization)
    ark(state, rkeys_ffs + 80);           // 10th round
    unpacking(ctext0, ctext1, state);     // unpacks the state to the output
}
```

```
void aes128_encrypt_ffs(unsigned char *ctext0, unsigned char *ctext1,
                        const unsigned char* ptext0, const unsigned char* ptext1,
                        const uint32_t* rkeys_ffs) {
    uint32_t state[8];                      // 256-bit internal state
    packing(state, ptext0, ptext1);         // packs into bitsliced representation
    ark(state, rkeys_ffs);                  // key whitening
    sbox(state);                           // 1st round
    mixcolumns_0(state);                   // 1st round
    ark(state, rkeys_ffs + 8);              // 1st round
    sbox(state);                           // 2nd round
    unpacking(ctext0, ctext1, state);     // unpacks the state to the output

//    mixcolumns_1(state);                    // 2nd round
//    ark(state, rkeys_ffs + 16);           // 2nd round
//    sbox(state);                            // 3rd round
//    mixcolumns_2(state);                    // 3rd round
//    ark(state, rkeys_ffs + 24);           // 3rd round
//    sbox(state);                            // 4th round
//    mixcolumns_3(state);                    // 4th round
//    ark(state, rkeys_ffs + 32);           // 4th round
//    sbox(state);                            // 5th round
//    mixcolumns_0(state);                    // 5th round
//    ark(state, rkeys_ffs + 40);           // 5th round
//    sbox(state);                           // 6th round
//    mixcolumns_1(state);                    // 6th round
//    ark(state, rkeys_ffs + 48);           // 6th round
//    sbox(state);                            // 7th round
//    mixcolumns_2(state);                    // 7th round
//    ark(state, rkeys_ffs + 56);           // 7th round
//    sbox(state);                            // 8th round
//    mixcolumns_3(state);                    // 8th round
//    ark(state, rkeys_ffs + 64);           // 8th round
```

FixSlicing AES – CTR

count = 00 00 00 00

s	0	1	2	3
0	06	d8	61	c2
1	f9	d9	46	b1
2	5c	38	57	e0
3	01	03	59	0f

count = 00 00 00 01

s	0	1	2	3
0	06	ea	61	c2
1	f9	d9	e8	b1
2	5c	38	57	9f
3	0e	03	59	0f

count = 00 00 00 02

s	0	1	2	3
0	06	fa	61	c2
1	f9	d9	c5	b1
2	5c	38	57	31
3	65	03	59	0f

count = 00 00 00 ff

s	0	1	2	3
0	06	f5	61	c2
1	f9	d9	62	b1
2	5c	38	57	a6
3	94	03	59	0f

count = 00 00 00 00

s	0	1	2	3
0	0	4	8	12
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15

count = 00 00 00 00

s	0	1	2	3
0	06	d8	61	c2
1	f9	d9	46	b1
2	5c	38	57	e0
3	01	03	59	0f

count = 00 00 01 00

s	0	1	2	3
0	06	d8	50	c2
1	f9	d9	46	1c
2	44	38	57	e0
3	01	bd	59	0f

count = 00 00 0a 00

s	0	1	2	3
0	06	d8	bb	c2
1	f9	d9	46	2d
2	01	38	57	e0
3	01	6d	59	0f

count = 00 00 00 00

s	0	1	2	3
0	06	d8	58	c2
1	f9	d9	46	c1
2	98	38	57	e0
3	01	0e	59	0f

count = 00 00 00 00

s	0	1	2	3
0	06	d8	61	c2
1	f9	d9	46	b1
2	5c	38	57	e0
3	01	03	59	0f

count = 00 01 00 00

s	0	1	2	3
0	06	d8	61	13
1	62	d9	46	b1
2	5c	31	57	e0
3	01	03	19	0f

count = 00 0a 00 00

s	0	1	2	3
0	06	d8	61	53
1	8d	d9	46	b1
2	5c	00	57	e0
3	01	03	85	0f

count = 00 ff 00 00

s	0	1	2	3
0	06	d8	61	69
1	59	d9	46	b1
2	5c	03	57	e0
3	01	03	2e	0f

count = 00 00 00 00

s	0	1	2	3
0	06	d8	61	c2
1	f9	d9	46	b1
2	5c	38	57	e0
3	01	03	59	0f

count = 01 00 00 00

s	0	1	2	3
0	70	d8	61	c2
1	f9	fd	46	b1
2	5c	38	f9	e0
3	01	03	59	8d

count = 0a 00 00 00

s	0	1	2	3
0	ee	d8	61	c2
1	f9	7f	46	b1
2	5c	38	12	e0
3	01	03	59	ee

count = ff 00 00 00

s	0	1	2	3
0	d2	d8	61	c2
1	f9	b6	46	b1
2	5c	38	2f	e0
3	01	03	59	af

FixSlicing AES – CTR 구현

```c
void Pre_Table(unsigned char* IV, const uint32_t* rkeys_ffs, u8 pretable[4][256][4])

    u8 temp_IV[16];
    for(int i=0; i<16; i++) {
        if(i<4) temp_IV[i] = IV[i]+1;
        else temp_IV[i] = IV[i];
    }

    for(int i=0; i<256; i+=2){
        uint32_t state[8];
        u8 temp_1[16] = {0x0, };
        u8 temp_2[16] = {0x0, };

        packing(state, IV, temp_IV);
        ark(state, rkeys_ffs);                  // key whitening
        sbox(state);                           // 1st round
        mixcolumns_0(state);                   // 1st round
        ark(state, rkeys_ffs + 8);              // 1st round
        sbox(state);                            // 2nd round

        unpacking(temp_1, temp_2, state);

        for(int j=0; j<4; j++){
            if(j==0){
                pretable[j][i][0] = temp_1[3];
                pretable[j][i][1] = temp_1[4];
                pretable[j][i][2] = temp_1[9];
                pretable[j][i][3] = temp_1[14];
                pretable[j][i+1][0] = temp_2[3];
                pretable[j][i+1][1] = temp_2[4];
                pretable[j][i+1][2] = temp_2[9];
                pretable[j][i+1][3] = temp_2[14];
            }
```

```c
void aes128_encrypt_ffs(unsigned char *ctext0, unsigned char *ctext1,
                        const unsigned char* ptext0, const unsigned char* ptext1,
                        const uint32_t* rkeys_ffs) {
    uint32_t state[8];                      // 256-bit internal state
    packing(state, ptext0, ptext1);         // packs into bitsliced representation
    ark(state, rkeys_ffs);                  // key whitening
    sbox(state);                           // 1st round
    mixcolumns_0(state);                   // 1st round
    ark(state, rkeys_ffs + 8);              // 1st round
    sbox(state);                           // 2nd round
    unpacking(ctext0, ctext1, state);      // unpacks the state to the output

//    mixcolumns_1(state);                  // 2nd round
//    ark(state, rkeys_ffs + 16);           // 2nd round
//    sbox(state);                          // 3rd round
//    mixcolumns_2(state);                  // 3rd round
//    ark(state, rkeys_ffs + 24);           // 3rd round
//    sbox(state);                          // 4th round
//    mixcolumns_3(state);                  // 4th round
//    ark(state, rkeys_ffs + 32);           // 4th round
//    sbox(state);                          // 5th round
//    mixcolumns_0(state);                  // 5th round
//    ark(state, rkeys_ffs + 40);           // 5th round
//    sbox(state);                          // 6th round
//    mixcolumns_1(state);                  // 6th round
//    ark(state, rkeys_ffs + 48);           // 6th round
//    sbox(state);                          // 7th round
//    mixcolumns_2(state);                  // 7th round
//    ark(state, rkeys_ffs + 56);           // 7th round
//    sbox(state);                          // 8th round
//    mixcolumns_3(state);                  // 8th round
//    ark(state, rkeys_ffs + 64);           // 8th round
```

FixSlicing AES – CTR 구현

```
void aes128_encrypt_ffs(unsigned char *ctext0, unsigned char *ctext1,
                        const unsigned char* ptext0, const unsigned char* ptext1,
                        const uint32_t* rkeys_ffs) {
    uint32_t state[8];                      // 256-bit internal state
    packing(state, ptext0, ptext1);         // packs into bitsliced representation
    ark(state, rkeys_ffs);                  // key whitening
    sbox(state);                           // 1st round
    mixcolumns_0(state);                   // 1st round
    ark(state, rkeys_ffs + 8);              // 1st round
    sbox(state);                           // 2nd round
    mixcolumns_1(state);                   // 2nd round
    ark(state, rkeys_ffs + 16);           // 2nd round
    sbox(state);                           // 3rd round
    mixcolumns_2(state);                   // 3rd round
    ark(state, rkeys_ffs + 24);           // 3rd round
    sbox(state);                           // 4th round
    mixcolumns_3(state);                   // 4th round
    ark(state, rkeys_ffs + 32);           // 4th round
    sbox(state);                           // 5th round
    mixcolumns_0(state);                   // 5th round
    ark(state, rkeys_ffs + 40);           // 5th round
    sbox(state);                           // 6th round
    mixcolumns_1(state);                   // 6th round
    ark(state, rkeys_ffs + 48);           // 6th round
    sbox(state);                           // 7th round
    mixcolumns_2(state);                   // 7th round
    ark(state, rkeys_ffs + 56);           // 7th round
    sbox(state);                           // 8th round
    mixcolumns_3(state);                   // 8th round
    ark(state, rkeys_ffs + 64);           // 8th round
    sbox(state);                           // 9th round
    mixcolumns_0(state);                   // 9th round
    ark(state, rkeys_ffs + 72);           // 9th round
    sbox(state);                           // 10th round
    double_shiftrows(state);               // 10th round (resynchronization)
    ark(state, rkeys_ffs + 80);           // 10th round
    unpacking(ctext0, ctext1, state);      // unpacks the state to the output
}
```

```
void ctr_aes128_encrypt_ffs(unsigned char *ctext0, unsigned char *ctext1,
                        const unsigned char* ptext0, const unsigned char* ptext1,
                        const uint32_t* rkeys_ffs) {
    uint32_t state[8];                      // 256-bit internal state
    packing(state, ptext0, ptext1);         // packs into bitsliced representation

//    ark(state, rkeys_ffs);                  // key whitening
//    sbox(state);                           // 1st round
//    mixcolumns_0(state);                   // 1st round
//    ark(state, rkeys_ffs + 8);              // 1st round
//    sbox(state);                           // 2nd round
//    unpacking(ctext0, ctext1, state);

    mixcolumns_1(state);                   // 2nd round
    ark(state, rkeys_ffs + 16);           // 2nd round
    sbox(state);                           // 3rd round
    mixcolumns_2(state);                   // 3rd round
    ark(state, rkeys_ffs + 24);           // 3rd round
    sbox(state);                           // 4th round
    mixcolumns_3(state);                   // 4th round
    ark(state, rkeys_ffs + 32);           // 4th round
    sbox(state);                           // 5th round
    mixcolumns_0(state);                   // 5th round
    ark(state, rkeys_ffs + 40);           // 5th round
    sbox(state);                           // 6th round
    mixcolumns_1(state);                   // 6th round
    ark(state, rkeys_ffs + 48);           // 6th round
    sbox(state);                           // 7th round
    mixcolumns_2(state);                   // 7th round
    ark(state, rkeys_ffs + 56);           // 7th round
    sbox(state);                           // 8th round
    mixcolumns_3(state);                   // 8th round
    ark(state, rkeys_ffs + 64);           // 8th round
    sbox(state);                           // 9th round
    mixcolumns_0(state);                   // 9th round
    ark(state, rkeys_ffs + 72);           // 9th round
    sbox(state);                           // 10th round
    double_shiftrows(state);               // 10th round (resynchronization)
    ark(state, rkeys_ffs + 80);           // 10th round
    unpacking(ctext0, ctext1, state);      // unpacks the state to the output
}
```

FixSlicing AES – CTR 성능

- 성능 테스트
  - 1MB 암호화 1000번 반복

  - ECB : 약 43초
  - CTR : 약 36초

  - 약 7초 정도의 차이 발생

```
clock_t start, end;
start = clock();

aes128_keyschedule_ffs(rk32, key8, key8);
for(int loop=0; loop<1000; loop++){
    for(int i=0; i<LOOP_LEN; i++){
        aes128_encrypt_ffs(ct8_1, ct8_2, IV_0, IV_0, rk32);
        for(int i=0; i<16; i++) ct8_1[i] ^= pt8_1[i];
        for(int i=0; i<16; i++) ct8_2[i] ^= pt8_1[i];
    }
}
end = clock();
printf("타이머 : %.2f",((float)(end - start) / CLOCKS_PER_SEC));
```

```
aes128_keyschedule_ffs(rk, key, key);
Pre_Table(IV_CNT1, rk, pretable);
for(int loop=0; loop<1000; loop++){
    for(int i=0; i<LOOP_LEN; i++){
        u8 temp_1[16], temp_2[16];
        pre_state(pretable, IV_CNT1, IV_CNT2, temp_1, temp_2);
        ctr_aes128_encrypt_ffs(&ct_1[i*16], &ct_2[i*16], temp_1, temp_2, rk);
        IV32_1[0] += 2;
        IV32_2[0] += 2;
    }
    for(int i=0; i<PT_LEN; i++) ct_1[i] ^= pt_1[i];
    for(int i=0; i<PT_LEN; i++) ct_2[i] ^= pt_2[i];
}
end = clock();
printf("타이머 : %.2f",((float)(end - start) / CLOCKS_PER_SEC));
```

질문…

count = 0	count = 1
PT_1 / 128-bit	PT_2 / 128-bit

count = 0	
PT_1 / 128-bit	PT_2 / 128-bit

count = 2	count = 3
PT_3 / 128-bit	PT_4 / 128-bit

count = 1	
PT_3 / 128-bit	PT_4 / 128-bit

Q & A