

## 2019 암호경진대회 : 2번 문제 답안

벤치마크 결과 **811ms**로 sbbox를 대신하여 not 연산을 사용하고, rnd가 생성되는 과정에서 나타나는 성질과 sbbox와 sbbox2를 합친 sbbox3를 이용하는 방법으로 구현하였습니다. 마지막으로 전역변수를 지역변수로 옮겨주는 것이 속도 향상에 큰 영향을 주었습니다.

### 1. 구현

새로운 SPN 기반 블록암호화를 아두이노에서 구현하기 위해 보드 모델은 UNO R3를 사용하였으며 아두이노 1.8.5 에서 구현하였습니다. 속도를 측정한 컴퓨터는 window 환경이며, 프로세서 Intel(R) Core(TM) i5-8265U CPU, RAM 16.0GB, 64비트 운영 체제입니다.

다음 표는 문제에서 주어진 코드에서 각각의 개선 사항과 <속도 측정시 함께 적용한 개선 사항>에 적힌 순번의 개선 사항을 함께 적용하여 코드를 구현해 속도를 측정해 본 결과입니다.

ex. 순번 ④에서 속도 측정은 순번 ①인 기존 코드에 ④번의 개선 사항과 순번 ③번의 개선 사항만 적용하여 속도를 측정한 것입니다. 즉, 순번 ②은 적용되지 않은 속도입니다.

| 순번 | 개선 사항                                                                                                                                                                                               | 속도 측정시 함께 적용한 사항 | 속도     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|--------|
| ①  | 문제에 주어진 코드는 C언어 기반이므로 개선 사항 없이 아두이노 코드로 약간의 수정만 거쳐 구현하였다.                                                                                                                                           |                  | 5416ms |
| ②  | 주어진 sbbox는 not연산과 동일하다. sbbox가 사용되는 부분을 모두 not연산으로 변경하였다.                                                                                                                                           | ①                | 3843ms |
| ③  | key_gen, enc 함수 모두 swap을 위해 짝수/홀수 round를 if문을 통해 구분한다. for문을 100회에서 50회로 줄이고 if문을 제거하였다.                                                                                                            | ①                | 4469ms |
| ④  | key_gen 함수에서 rnd의 인덱스는 0부터 199까지 순서대로 증가하기 때문에 기존에 곱하고 더하는 방식이 아닌 새로운 변수 j를 넣어 j를 0부터 1씩 증가시켰다. 또, sbbox[i] 역시 0xff부터 순서대로 감소하는 것이므로 새로운 변수 k를 넣어주었다.                                               | ①, ③             | 4312ms |
| ⑤  | key_gen 함수에서 rnd가 생성되는 과정에서 나오는 성질을 이용하여 연산량을 줄였다. (성질은 아래 세부 설명에서 자세히 다루겠습니다.)                                                                                                                     | ①, ②, ③, ④       | 2531ms |
| ⑥  | enc 함수에서 한 라운드가 시작되고 끝나는 지점을 변경해, sbbox연산을 없애고 sbbox와 sbbox2를 합친 sbbox3를 만들어 이용하였다. 그리고 $tmp2 = (text1 - (text2 + text1)) \wedge rnd[i * 2 + 1]$ 를 $tmp2 = (-text2) \wedge rnd[i * 2 + 1]$ 로 변경하였다. | ①, ②, ③, ④       | 2852ms |
| ⑦  | ① 부터 ⑥ 까지 개선 사항을 모두 적용하였다.                                                                                                                                                                          | ① ~ ⑥            | 2465ms |
| 최종 | 개선 사항을 모두 적용시키고, 전역변수로 선언되어 있는 text, rnd, key를 모두 메인 함수 안 지역변수로 옮겨주었다.                                                                                                                              | ① ~ ⑦            | 811ms  |

## 2. 세부 설명

- ① 문제에 주어진 코드는 C언어 기반이므로 아두이노 코드로 약간의 수정만 거쳐 구현하였습니다.
- ② 주어진 sbbox는 not연산과 동일하다는 것을 알 수 있습니다. sbbox가 사용되는 부분을 모두 not연산으로 변경하였습니다.  
ex.  $\text{sbbox}[i] \Rightarrow \sim i$
- ③ 기존 코드는 swap을 위해 짝수/홀수 round를 if문을 통해 구분합니다. 기존 for문 i의 증가를  $i++$ 에서  $i=i+2$ 로 변경하여 for문이 진행되는 횟수를 100회에서 50회로 줄이고 if문을 제거하였습니다.
- ④ key\_gen, enc 함수에서 rnd의 인덱스는 0부터 199까지 순서대로 증가합니다. 기존에 곱하고 더하는 방식이 아닌 새로운 변수 j를 넣어 j를 0부터 1씩 증가시키는 방식을 사용하여 곱하기 연산을 줄일 수 있었습니다. 같은 방식으로 key\_gen 함수에서 sbbox[i]는 0xff부터 차례로 감소하는 것도 같기 때문에 새로운 변수 k를 넣어 k를 0xff부터 1씩 감소시키는 방식을 사용하였습니다.

```
void key_gen(u8* rnd, u8* key) {
    u8 key1 = key[0];
    u8 key2 = key[1];
    u8 tmp1, tmp2;

    int i;
    for (i = 0; i < ROUND_NUM; i=i+2) {
        key1 = sbbox[key1];
        key2 = sbbox[key2];

        tmp1 = (key1 | key2) + sbbox[i];
        tmp2 = (key1 & key2) - sbbox[i];

        key1 = tmp1;
        key2 = tmp2;

        rnd[i * 2 + 0] = key1;
        rnd[i * 2 + 1] = key2;

        key1 = sbbox[key1];
        key2 = sbbox[key2];

        tmp1 = (key1 | key2) + sbbox[i+1];
        tmp2 = (key1 & key2) - sbbox[i+1];

        key1 = tmp2;
        key2 = tmp1;

        rnd[i * 2 + 2] = key1;
        rnd[i * 2 + 3] = key2;
    }
}
```

<변경 전>

```
void key_gen(u8* rnd, u8* key) {
    u8 key1 = key[0];
    u8 key2 = key[1];
    u8 tmp1, tmp2;

    int i, j=0, k=0xff;
    for (i = 0; i < ROUND_NUM; i=i+2) {
        key1 = sbbox[key1];
        key2 = sbbox[key2];

        tmp1 = (key1 | key2) + k;
        tmp2 = (key1 & key2) - (k--);

        key1 = tmp1;
        key2 = tmp2;

        rnd[j++] = key1;
        rnd[j++] = key2;

        key1 = sbbox[key1];
        key2 = sbbox[key2];

        tmp1 = (key1 | key2) + k;
        tmp2 = (key1 & key2) - (k--);

        key1 = tmp2;
        key2 = tmp1;

        rnd[j++] = key1;
        rnd[j++] = key2;
    }
}
```

<변경 후>

⑤ key\_gen 함수에서 rnd가 생성되는 과정에서 아래 Lemma 발견하였습니다.

Lemma. key\_gen 함수에서 rnd는 다음과 같은 성질을 가진다.

$$\begin{aligned} \text{rnd}[0] + \text{rnd}[1] &= \text{rnd}[4] + \text{rnd}[5] = \text{rnd}[4*i] + \text{rnd}[4*i+1] = \dots = \text{rnd}[196] + \text{rnd}[197] \\ \text{rnd}[2] + \text{rnd}[3] &= \text{rnd}[6] + \text{rnd}[7] = \text{rnd}[4*i+2] + \text{rnd}[4*i+3] = \dots = \text{rnd}[198] + \text{rnd}[199] \end{aligned}$$

pf. 주어진 key\_gen 코드 for문에서 i = 0, 1일 때를 tmp1 + tmp2를 살펴보자.

i = 0일 때, tmp1 + tmp2의 값은

$$\begin{aligned} \text{tmp1} + \text{tmp2} &= (\text{key1} \mid \text{key2}) + (\text{key1} \& \text{key2}) \\ &= \text{key1} + \text{key2} \dots (*) \\ &= \text{sbox}[\text{key}[0]] + \text{sbox}[\text{key}[1]] \end{aligned}$$

이다. 그러므로

$$\text{rnd}[0] + \text{rnd}[1] = \text{sbox}[\text{key}[0]] + \text{sbox}[\text{key}[1]]$$

이 된다.

마찬가지로 i = 1일 때, 알 수 있는 사실은

$$\begin{aligned} \text{tmp1} + \text{tmp2} &= \text{sbox}[\text{rnd}[0]] + \text{sbox}[\text{rnd}[1]], \\ \text{rnd}[2] + \text{rnd}[3] &= \text{sbox}[\text{rnd}[0]] + \text{sbox}[\text{rnd}[1]] \end{aligned}$$

이다. 이 때, sbox는 not 연산과 같고 unsigned char 변수 안에서 계산되므로

$$\text{sbox}[n] = 0\text{xff} - n$$

$$\text{즉, } \text{rnd}[2] + \text{rnd}[3] = (0\text{xff} - \text{rnd}[0]) + (0\text{xff} - \text{rnd}[1]) = 0\text{xfe} - (\text{rnd}[0] + \text{rnd}[1])$$

$$\Rightarrow 0\text{xfe} = \text{rnd}[0] + \text{rnd}[1] + \text{rnd}[2] + \text{rnd}[3]$$

i = 2, 3,...에서도 똑같은 방식으로,

$$\begin{aligned} 0\text{xfe} &= \text{rnd}[0] + \text{rnd}[1] + \text{rnd}[2] + \text{rnd}[3] \\ 0\text{xfe} &= \text{rnd}[2] + \text{rnd}[3] + \text{rnd}[4] + \text{rnd}[5] \\ 0\text{xfe} &= \text{rnd}[4] + \text{rnd}[5] + \text{rnd}[6] + \text{rnd}[7] \\ &\vdots \end{aligned}$$

따라서 rnd[0] + rnd[1] = A, rnd[2] + rnd[3] = B라 두면,

$$A = \text{rnd}[0] + \text{rnd}[1] = \text{rnd}[4] + \text{rnd}[5] = \text{rnd}[8] + \text{rnd}[9] = \dots = \text{rnd}[196] + \text{rnd}[197]$$

$$B = \text{rnd}[2] + \text{rnd}[3] = \text{rnd}[6] + \text{rnd}[7] = \text{rnd}[10] + \text{rnd}[11] = \dots = \text{rnd}[198] + \text{rnd}[199]$$

이 된다. ■

+) 8비트인 key1과 key2에 대해 식 (\*) :  $(key1 \mid key2) + (key1 \& key2) = key1 + key2$ 이 성립한다.

pf. 비트 b1과 b2가 있을 때 or과 and 연산을 algebraic normal form으로 표현하면 다음과 같다.

$$b1 \mid b2 = b1 \oplus b2 \oplus b1 \cdot b2$$

$$b1 \& b2 = b1 \cdot b2$$

위 식이 8비트에서도 성립하는지 알아보기 위해 하위비트 연산 시 carry가 발생했을 때와 발생하지 않을 때로 구분한다.

1) 하위비트 연산 시 carry가 발생하지 않았을 때

| b1 | b2 | $b1 \cdot b2$ | $b1 \mid b2$ | $b1 \& b2$ | $(b1 \mid b2) + (b1 \& b2)$ | $b1 + b2$ |
|----|----|---------------|--------------|------------|-----------------------------|-----------|
| 0  | 0  | 0             | 0            | 0          | 0                           | 0         |
| 0  | 1  | 0             | 1            | 0          | 1                           | 1         |
| 1  | 0  | 0             | 1            | 0          | 1                           | 1         |
| 1  | 1  | 1             | 1            | 1          | 0                           | 0         |

$(b1 \mid b2) + (b1 \& b2)$ 와  $b1 + b2$ 의 결과값이 항상 일치한다.

2) 하위비트 연산 시 carry가 발생했을 때

| b1 | b2 | $b1 \cdot b2$ | $b1 \mid b2$ | $b1 \& b2$ | carry | $(b1 \mid b2) + (b1 \& b2) + \text{carry}$ | $b1 + b2 + \text{carry}$ |
|----|----|---------------|--------------|------------|-------|--------------------------------------------|--------------------------|
| 0  | 0  | 0             | 0            | 0          | 1     | 1                                          | 1                        |
| 0  | 1  | 0             | 1            | 0          | 1     | 0                                          | 0                        |
| 1  | 0  | 0             | 1            | 0          | 1     | 0                                          | 0                        |
| 1  | 1  | 1             | 1            | 1          | 1     | 1                                          | 1                        |

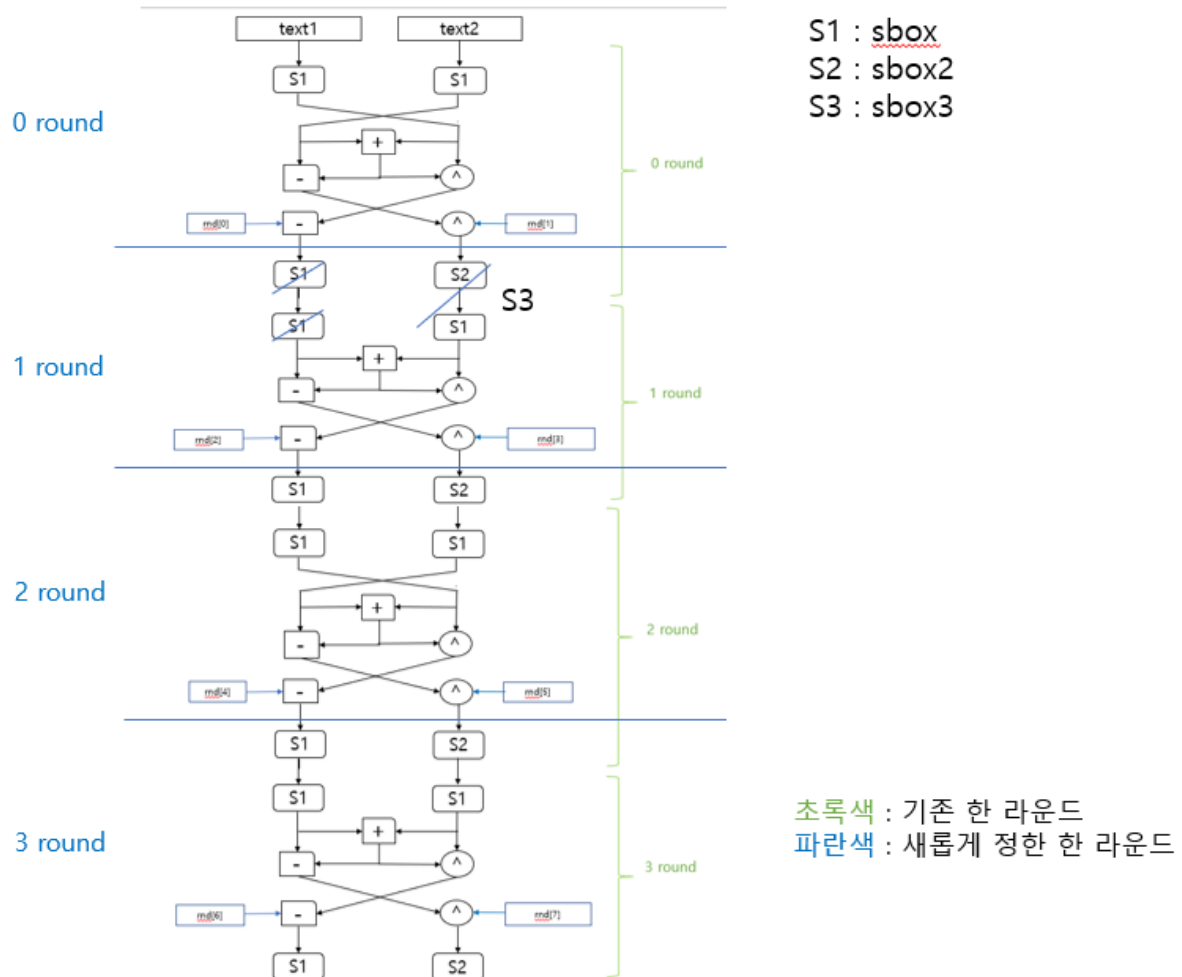
$(b1 \mid b2) + (b1 \& b2) + \text{carry}$ 와  $b1 + b2 + \text{carry}$ 의 결과값이 항상 일치한다.

따라서 각 비트에 대해 연산 결과가 동일하므로

8비트 key1, key2에 대해  $(key1 \mid key2) + (key1 \& key2) = key1 + key2$ 이다. ■

Lemma를 이용하여, 첫 라운드는 for문 밖으로 옮겨 A, B를 먼저 계산해 놓은 후,  $\text{rnd}[4*i]$ 와  $\text{rnd}[4*i+3]$  값을 먼저 계산하고  $\text{rnd}[4*i+1]$ 와  $\text{rnd}[4*i+2]$ 는 A, B,  $\text{rnd}[4*i]$ ,  $\text{rnd}[4*i+3]$ 을 이용하여 계산 하는 방법으로 구현했습니다. 그리고 그 과정에서 불필요한 대입연산을 제거할 수 있었고, for문 밖 연산들은 상수를 직접 대입하였습니다.

⑥ enc 함수에서 기존 한 라운드가 시작되고 끝나는 지점을 새롭게 변경해보았습니다.



파란색 기준으로 라운드를 새로 결정하면, sbox가 not 연산이기 때문에 sbox와 sbox가 만나 연산이 상쇄되고 sbox2, sbox 순서대로 연산 하는 것을 합쳐 sbox3로 만들 수 있었습니다. 이렇게 라운드를 바꾸면 0, 99라운드는 1~98라운드와 연산이 조금 다르기 때문에 0, 99라운드는 for문 밖으로 빼내었습니다.

마지막으로  $\text{tmp2} = (\text{text1} - (\text{text2} + \text{text1})) \wedge \text{rnd}[i * 2 + 1]$ 를  $\text{tmp2} = (-\text{text2}) \wedge \text{rnd}[i * 2 + 1]$ 로 변경 하였습니다.

⑦ ①부터 ⑥까지 모든 개선 사항을 적용하여 속도를 측정하였습니다.

최종. ⑦의 상태에서 마지막으로 전역변수인 key, rnd, text를 main함수의 지역변수로 바꾸었습니다.

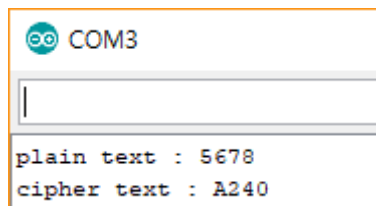
### 3. 테스트 벡터 및 결과

개선 사항을 모두 적용한 코드를 구현한 후, 아래 코드를 이용하여 테스트 벡터를 확인하였습니다.

```
void testVector(){
    u8 key[2] = { 0x12, 0x34 };
    u8 rnd[ROUND_NUM * 2] = { 0, };
    u8 text[2] = { 0x56, 0x78 };

    Serial.print("plain text : ");Serial.print(text[0],HEX);Serial.print(text[1],HEX);
    key_gen(rnd, key);
    enc(text, rnd);
    Serial.print("\ncipher text : ");Serial.print(text[0],HEX);Serial.println(text[1],HEX);
}
```

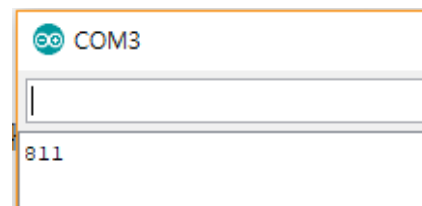
testVector(): 호출시 출력되는 값이 문제에 주어진 값과 동일하였습니다.



```
COM3
plain text : 5678
cipher text : A240
```

### 4. 벤치마크 결과

개선 사항을 모두 적용한 코드를 이용하여 시간을 계산하였습니다.



```
COM3
811
```

캡처된 화면과 같이 **811ms**이 나왔고, 기존 코드보다 약 85% 향상된 속도의 코드를 얻을 수 있었습니다.

▶ 벤치마크 결과를 출력하는 과정에서 testVector()를 함께 실행시키면 testVector()가 속도에 영향을 주게 되어 답안 작성시 속도는 모두 제출한 코드의 testVector()를 주석처리한 후 측정하였습니다.