2021 암호분석경진대회			
2번 문제 풀이			
구분	일반부	팀명	HB

답) K = 0x687a717a7a6c70737567737568637072

풀이)

1. 문제 개요

1.1 블록암호 구조 분석

입력: 64-bit P, 128-bit K

출력: 64-bit C

- 화이트닝 과정: (k_0-add.)
- 1 라운드: (k_1-add.) - (RC_0-add.)
- 2 라운드: R_1: (S-Layer) - (M-Layer) - (RC_1-add.) - (k_1-add.)
- Middle Layer: (S-Layer) - (M'-Layer) - (Inverse S-Layer)
- 3 라운드: R_2^{-1}: (RC_2-add.) - (k_1-add.) - (Inverse M-Layer) - (Inverse S-Layer)
- 4 라운드: (RC_3-add.) - (k_1-add.)
- 화이트닝 과정: (k_0'-add.)

<PRINCE-64/128 라운드키 생성 설명>

입력된 128-bit 비밀키는 64-bit 씩 분할($k = k_0 \| k_1$) 된 후 Key Expansion 함수를 통해 k_0'을 생성하여 192-bit 로 확장된다.

- **Key Expansion** : PRINCE에서 사용하는 라운드키 생성 함수로 다음과 같이 정의된다.

$$(K_0 \| K_1) \rightarrow (k_0 \| k_0' \| k_1) := (k_0 \| (k_0 \ggg 1) \oplus (k_0 \gg 63) \| k_1)$$

<PRINC-64/128 암호화 설명>

입력 평문은 다음과 같이 4-bit 니블로 분할되어 상태배열로 표현된다.

$$Plaintext = (p_0, p_1, \cdots, p_{14}, p_{15})$$

p_0	p_1	p_2	p_3
p_4	p_5	p_6	p_7
p_8	p_9	p_{10}	p_{11}
p_{12}	p_{13}	p_{14}	p_{15}

- k_1-add.($\oplus k_i$) : 상태배열에 대한 64-bit 라운드키 XOR 연산
- S-Layer(S) : 상태배열의 각 니블에 대한 치환 연산 (4-bit S-box 사용)

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	B	F	3	2	A	C	9	1	6	7	8	0	E	5	D	4

- Inverse S-Layer(S^{-1}) : S-Layer의 역연산

- **M/M'-Layer** (M/M') : M'-Layer는 다음과 같이 정의된 64×64 행렬 M' 곱 연산

$$M_0 = \begin{pmatrix} 0000 \\ 0100 \\ 0010 \\ 0001 \end{pmatrix}, M_1 = \begin{pmatrix} 1000 \\ 0000 \\ 0010 \\ 0001 \end{pmatrix}, M_2 = \begin{pmatrix} 1000 \\ 0100 \\ 0000 \\ 0001 \end{pmatrix}, M_3 = \begin{pmatrix} 1000 \\ 0100 \\ 0010 \\ 0000 \end{pmatrix},$$

$$\hat{M}_0 = \begin{pmatrix} M_0 M_1 M_2 M_3 \\ M_1 M_2 M_3 M_0 \\ M_2 M_3 M_0 M_1 \\ M_3 M_0 M_1 M_2 \end{pmatrix}, \hat{M}_1 = \begin{pmatrix} M_1 M_2 M_3 M_0 \\ M_2 M_3 M_0 M_1 \\ M_3 M_0 M_1 M_2 \\ M_0 M_1 M_2 M_3 \end{pmatrix}$$

$$M' = (\hat{M}_0 \bullet \hat{M}_1 \bullet \hat{M}_1 \bullet \hat{M}_0)$$

M-Layer는 $SR \bullet M'$ 연산으로 정의된다.
여기서 SR은 상태배열의 각 열에 대한 순환이동 연산으로 다음과 같이 동작한다.

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

→

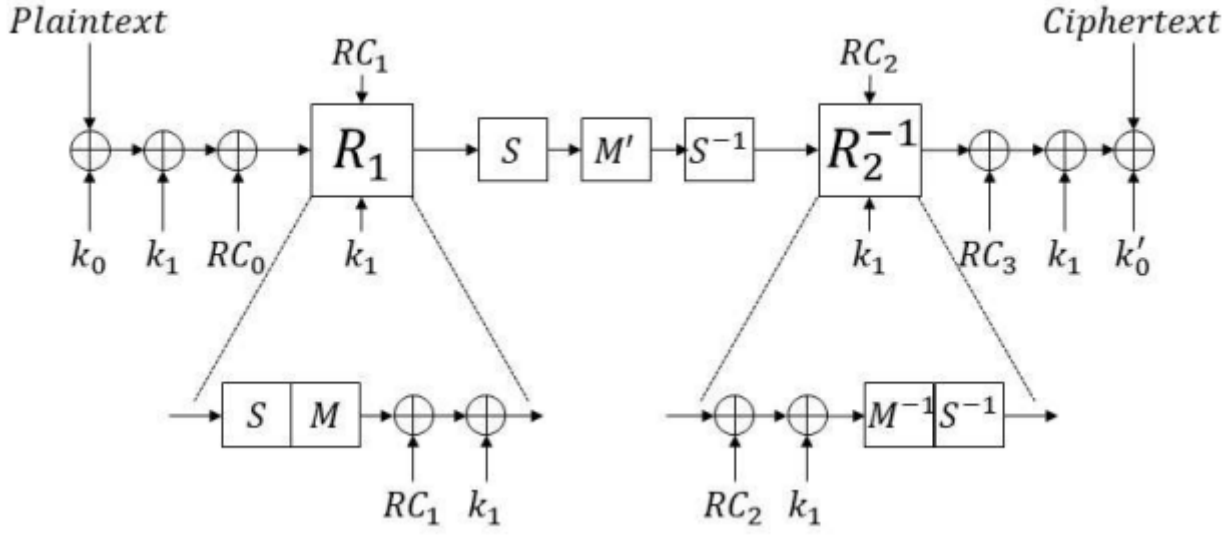
0	5	A	F
4	9	E	3
8	D	2	7
C	1	6	B

- **Inverse M-Layer**(M^{-1}) : M-Layer의 역연산($M' \bullet SR^{-1}$)

- RC_i**-add.**($\oplus RC_i$) : 상태배열에 대한 64-bit 라운드 상수 XOR 연산

RC_0	0000000000000000
RC_1	13198a2e03707344
RC_2	a4093822299f31d0
RC_3	082efa98ec4e6c89

아래 그림은 4-round PRINCE-64 동작 과정이다.



1.2 구현
테스트 벡터를 통해 확인하였다.

```
Testvector 1의 암호화 결과 : 0xe35168f91283502c
Testvector 1의 복호화 결과 : 0x0000000000000000
Testvector 2의 암호화 결과 : 0x96775187fc6a9943
Testvector 2의 복호화 결과 : 0xffffffffffffffff
Testvector 3의 암호화 결과 : 0x6988ae78039566bd
Testvector 3의 복호화 결과 : 0x0000000000000000
Testvector 4의 암호화 결과 : 0xc611a0ec10c574e4
Testvector 4의 복호화 결과 : 0x0000000000000000
Testvector 5의 암호화 결과 : 0x2579f2f660306f5e
Testvector 5의 복호화 결과 : 0x0123456789abcdef
Testvector 6의 암호화 결과 : 0xe31cf8a9ae6a50c7
Testvector 6의 복호화 결과 : 0x0123456789abcdef
```

2. 취약점 분석

평문쌍들을 분석한 결과 평문쌍들은 아래와 같이 16개 단위로, 나머지 nibble은 동일할 때 하나의 nibble이 0x0~0xf까지 나타났다. 또한 256개 단위로 고정되는 평문들이 변화하였다.

```
0xaed66ce184be2309
0xaed66ce184be2319
0xaed66ce184be2329
0xaed66ce184be2339
0xaed66ce184be2349
0xaed66ce184be2359
0xaed66ce184be2369
0xaed66ce184be2379
0xaed66ce184be2389
0xaed66ce184be2399
0xaed66ce184be23a9
0xaed66ce184be23b9
0xaed66ce184be23c9
0xaed66ce184be23d9
0xaed66ce184be23e9
0xaed66ce184be23f9
```

< $pt_0, \ldots, pt_{15}$ >

```
0xaed66ce184be2320
0xaed66ce184be2321
0xaed66ce184be2322
0xaed66ce184be2323
0xaed66ce184be2324
0xaed66ce184be2325
0xaed66ce184be2326
0xaed66ce184be2327
0xaed66ce184be2328
0xaed66ce184be2329
0xaed66ce184be232a
0xaed66ce184be232b
0xaed66ce184be232c
0xaed66ce184be232d
0xaed66ce184be232e
0xaed66ce184be232f
```

< $pt_{16}, \ldots, pt_{31}$ >

…

```
0xebe9bbf1f1499002
0xebe9bbf1f1499012
0xebe9bbf1f1499022
0xebe9bbf1f1499032
0xebe9bbf1f1499042
0xebe9bbf1f1499052
0xebe9bbf1f1499062
0xebe9bbf1f1499072
0xebe9bbf1f1499082
0xebe9bbf1f1499092
0xebe9bbf1f14990a2
0xebe9bbf1f14990b2
0xebe9bbf1f14990c2
0xebe9bbf1f14990d2
0xebe9bbf1f14990e2
0xebe9bbf1f14990f2
```

< $pt_{256}, \ldots, pt_{271}$ >

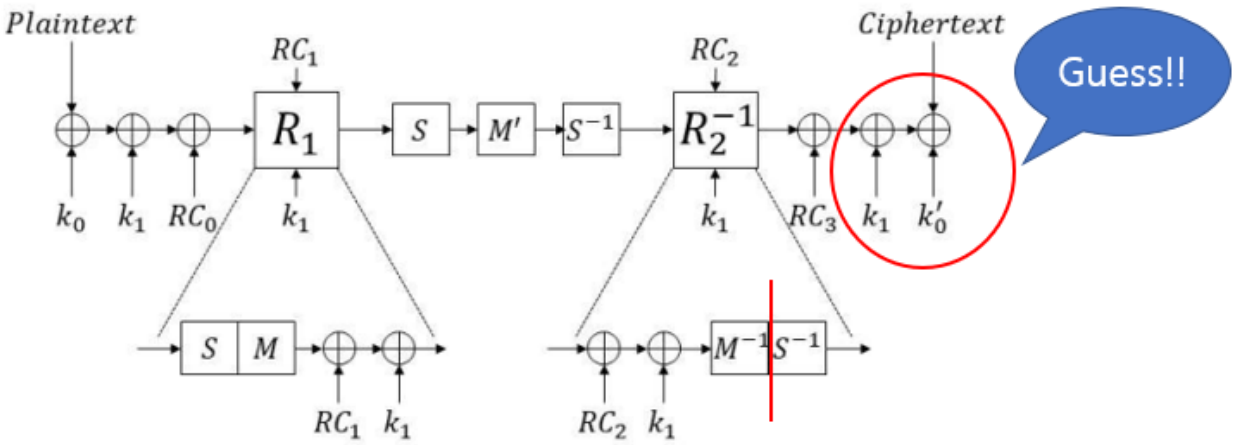
3. 공격

3.1 $k_1 \oplus k_0'$ 찾기

2에 의해 다음 공격법을 이용하였다.

1. Encrypt 5 sets of 2^4 plaintexts with one active nibble
2. **for all** 16 nibbles
   (a) **for all** values of $k_1 \oplus k_0'$ nibble
      i. **for all** 5 sets of plaintexts
         A. Decrypt all ciphertexts for a given set through the Sbox
         B. Sum the nibbles calculated in Step A. **If** the sum is zero, **then** the guess is a candidate for correct $k_1 \oplus k_0'$
   (b) Identify a candidate of $k_1 \oplus k_0'$ which appears in all 5 sets for a given nibble

즉 아래의 그림을 guess하여 빨간색 부분의 S^{-1}까지 복구한 후 balanced한지를 확인한다.



balanced란 다음과 같다.

$$\forall n \in \{0, \ldots, 15\}, \bigoplus_{c \in \mathcal{C}} c[n] = 0.$$

위를 이용하여 $k_1 \oplus k_0'$ = 0xc15a4bc85555484b임을 알 수 있다.

k_0와 k_1은 모두 소문자로만 구성되어있기 때문에 k_0와 k_1은 둘 다 0x6? 또는 0x7?이다. 따라서 $k_1 \oplus k_0'$은 홀수 자리에 0x4, 0x5, 0xc, 0xd만 나타날 수 있고 위해서 구한 $k_1 \oplus k_0'$은 이를 만족한다.

3.2 k_0, k_1 찾기

$k_1 \oplus k_0'$을 알고 있기 때문에 k_1만 알면 k_0'을 알아낼 수 있고 k_0'을 알아내면 방정식을 품으로써 k_0을 알아낼 수 있다. k_1에 대해 그냥 전수조사를 한다면 복잡도가 2^{64}이겠지만, 문제에서 키는 알파벳 소문자로만 구성되어있다고 하였으므로 (26^8)만 전수조사하면 알 수 있고, 더 나아가 $k_1 \oplus k_0'$의 홀수 자리에 0x4, 0x5, 0xc, 0xd을 보고 k_1이 0x6?인지 0x7?인지도 알아낼 수 있다. 따라서 $(15^3)*(11^5) \approx 2^{29}$의 복잡도로 k_1을 구할 수 있고 따라서 수식을 이용하여 k_0도 구할 수 있다.

4. 복구된 비밀키

복구된 비밀키 K = 0x687a717a7a6c70737567737568637072 이다.

참고문헌

1. Pawel Morawiecki : Practical Attacks on the Round-reduced PRINCE (2015)