2021 암호분석경진대회			
5번 문제 풀이			
구분	일반부	팀명	HB

답)

M_0 =
0x88888888888888889999999999999999888888888888888899999999999999998899898999988888998898988
889999988998989999888889988989888899

M_1 = 0x9998899898999988888998898988889999988998989999888889988989888899

해시 값: 0x9999999998989889888888888889898998

풀이)

1. 문제 개요

1.1 해시함수 구조 분석

Midori64 기반 해시함수 정의

입력: 256-bit의 배수 길이의 메시지 $M = M_0 \| M_1 \| \cdots \| M_{m-1}$

출력: 128-bit 해시 값 CV_m

$$IV = CV_0 = CV_0^{0,1,2,3} = 0x88888888888888889999999999999999;$$
$$For\ i = 0 \sim m-1:$$
$$CV_{i+1} = Compression(CV_i, M_i);$$
$$\text{return } CV_m$$

$Compression$ 함수 정의

입력: 128-bit 길이의 연쇄변수 CV_{n-1}, 256-bit의 길이의 메시지 블록 M_{n-1}

출력: 128-bit 연쇄변수 CV_n

$$C^{0,1} = Midori64(CV_{n-1}^{0,1}, M_{n-1}^{0,1,2,3} \oplus CV_{n-1});$$
$$C^{2,3} = Midori64(CV_{n-1}^{2,3}, M_{n-1}^{4,5,6,7} \oplus CV_{n-1});$$
$$C^{0,1,2,3} = C^{0,1} \| C^{2,3};$$
$$C^{0,1,2,3} = C^{0,1,2,3} \oplus M_{n-1}^{0,1,2,3} \oplus M_{n-1}^{4,5,6,7};$$
$$CV_n = C^0 \| C^3 \| C^2 \| C^1 = MIX(C^{0,1,2,3});$$
$$\text{return } CV_n$$

Midori64 함수 정의

입력: 64-bit Data, 128-bit Key
출력: 64-bit 암호문 C
(WK, RK) = KeyGen(Key)
C = Midoricore(Data, WK, RK)
return C

KeyGen 함수 정의
입력: 128-bit 길이의 Key
출력: 64-bit 길이의 화이트닝 키 WK, 64-bit 길이의 라운드 키가 15개 있는 배열 RK
$WK = Key^0 \oplus Key^1$
β는 주어진 배열

Table 5: The Round Constants β_i

i	β_i	i	β_i	i	β_i	i	β_i	i	β_i	i	β_i	i	β_i
0	0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1	1	0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0	2	1 0 0 0 0 1 0 1 1 0 1 0 0 0 1 1	3	0 0 0 0 1 0 0 0 1 1 0 1 0 0 1 1	4	0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1	5	1 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0	6	0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0
7	0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0	8	1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1	9	0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0	10	0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 1	11	0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 0	12	0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0	13	1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0
14	1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0	15	0 1 1 0 1 1 0 0 1 0 0 0 1 0 0 1	16	0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0	17	0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 0	18	0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 0				

$RK[i] = Key^{i\%2} \& \ \beta_i$($i$는 라운드)
return WK, RK

Midoricore 함수 정의
입력: 64-bit X, 64-bit 길이의 화이트닝 키 WK, 64-bit 길이의 라운드 키가 있는 배열 RK
출력: 64-bit Y
$R[i] = R_i$라 하자

Algorithm MidoriCore$_{(R)}(X, WK, RK_0, ..., RK_{R-2})$ **:**
$S \leftarrow$ KeyAdd(X, WK)
for $i = 0$ to $R-2$ do
    $S \leftarrow$ SubCell(S)
    $S \leftarrow$ ShuffleCell(S)
    $S \leftarrow$ MixColumn(S)
    $S \leftarrow$ KeyAdd(S, RK_i)
$S \leftarrow$ SubCell(S)
$Y \leftarrow$ KeyAdd(S, WK)

(R = 16)

SubCell 함수 정의
입력: 64-bit S
출력: 64-bit S
Sbox = ['0xc', '0xa', '0xd', '0x3', '0xe', '0xb', '0xf', '0x7', '0x8', '0x9', '0x1', '0x5', '0x0', '0x2', '0x4', '0x6']
for i in range(16):
    S^i = Sbox(S^i)
return S

ShuffleCell 함수 정의
입력: 64-bit S
출력: 64-bit S

$(s_0, s_1, ..., s_{15}) \leftarrow (s_0, s_{10}, s_5, s_{15}, s_{14}, s_4, s_{11}, s_1, s_9, s_3, s_{12}, s_6, s_7, s_{13}, s_2, s_8)$.

return S

MixColumn 함수 정의

입력: 64-bit S

출력: 64-bit S

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

${}^t(s_i, s_{i+1}, s_{i+2}, s_{i+3}) \leftarrow M\,{}^t(s_i, s_{i+1}, s_{i+2}, s_{i+3})$ and $i = 0, 4, 8, 12$.

return S

KeyAdd 함수 정의

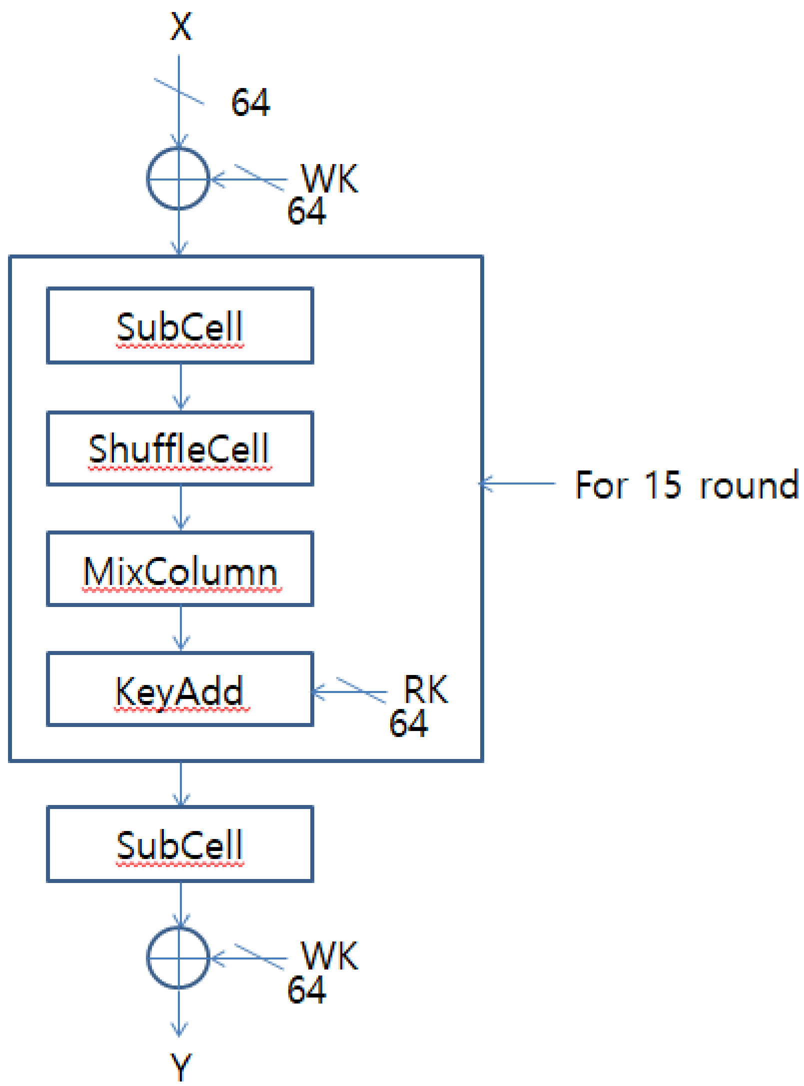
입력: 64-bit S, 64-bit K

출력: 64-bit S

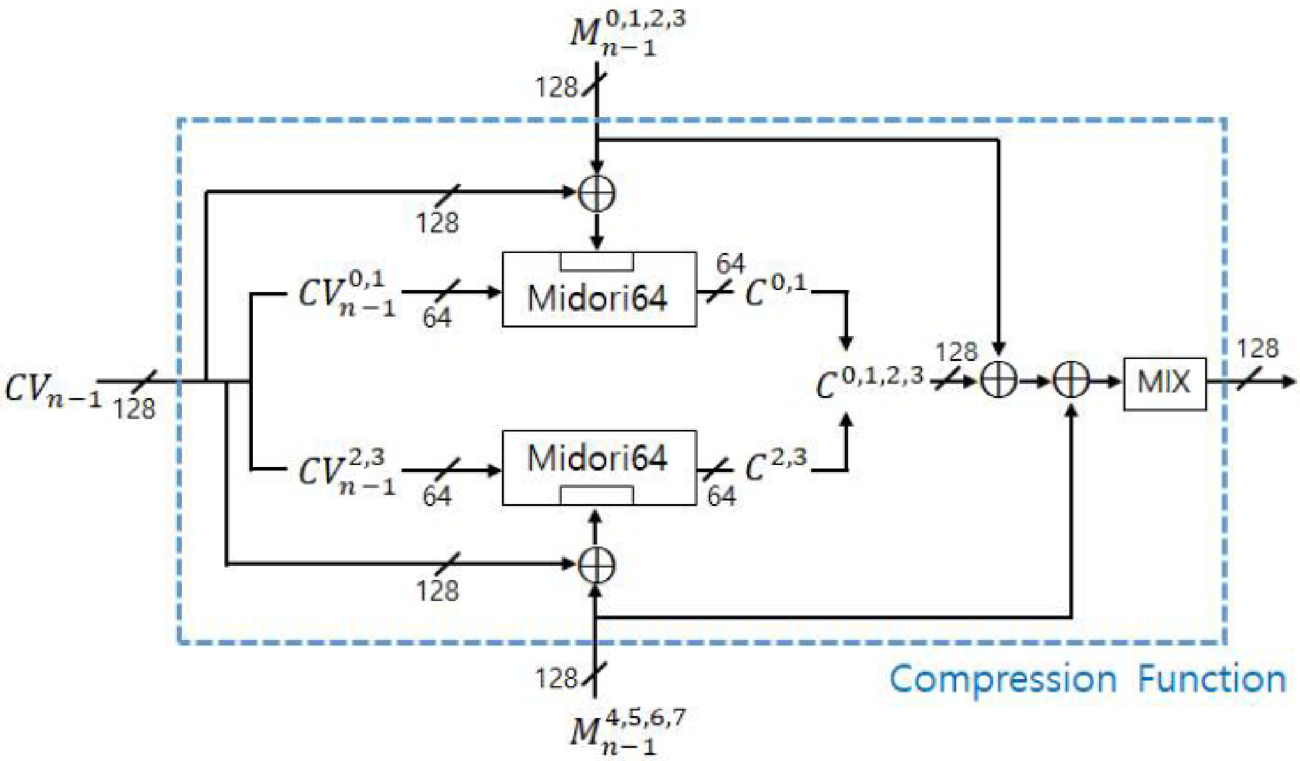
S = S ⊕ K

return S

아래 그림은 Midori64 함수의 도식도이다.



아래 그림은 Midori64 기반으로 설계된 블록암호 기반 해시함수의 n번째 compression 함수의 도식도이다.



1.2 구현

테스트 벡터를 통해 확인하였다.

```
Test Vector 1 해시 값 : 0xca712515a15b5dbf5357d91d7b6bbd14
Test Vector 2 해시 값 : 0x823e590925d042b24d74e29d61bb207d
Test Vector 3 해시 값 : 0xe4cfa3b8a55c123d449ec93801c3095e
```

2. 취약점 분석

문제에서 IV = 0x88888888888888889999999999999999이기 때문에 첫 Compression 함수에서 Midori64(0x8888888888888888, ?)와 Midori64(0x9999999999999999, ?)는 첫 Compression 함수에서 수행된다.

SubCell의 Sbox는 Sbox(0x8) = 0x8, Sbox(0x9)= 0x9이다.

ShuffleCell은 연산없이 순서만 섞인다.

MixColumn에서는 0x8 ⊕ 0x8 ⊕ 0x8, 0x8 ⊕ 0x8 ⊕ 0x9, 0x8 ⊕ 0x9 ⊕ 0x9, 0x9 ⊕ 0x9 ⊕ 0x9이므로 0x8 또는 0x9이다.

RK에서 β의 값은 0x0과 0x1이 조합되어 있으므로 0x8은 0x9로 0x9는 0x8로 된다.

즉 key의 값이 0x0과 0x1로 조합되어있다면 Midori64(0x8888888888888888, key)와 Midori64(0x9999999999999999, key)의 값은 항상 0x8과 0x9로 이루어질 것이다.

3. 공격

3.1 IV를 이용한 Midori64 충돌쌍 공격

문제에서 IV = 0x88888888888888889999999999999999이기 때문에 첫 Compression 함수에서 Midori64(0x8888888888888888, ?)와 Midori64(0x9999999999999999, ?)는 첫 Compression 함수에서 수행된다. 2를 이용하여 이 두 값에 대한 충돌쌍을 먼저 조사해보았다.

차분을 이용하여 수행해본 결과,
Midori64(0x8888888888888888, 0x00000000000000000000000000000000)
= Midori64(0x8888888888888889, 0x00000000000000010000000000000001)
= Midori64(0x8888888888888898, 0x00000000000000100000000000000010)
…
= Midori64(0x9888888888888888, 0x10000000000000001000000000000000)

Midori64(0x9999999999999999, 0x00000000000000000000000000000000)
= Midori64(0x9999999999999998, 0x00000000000000010000000000000001)
= Midori64(0x9999999999999989, 0x00000000000000100000000000000010)
…
= Midori64(0x8999999999999999, 0x10000000000000001000000000000000)
임을 알 수 있었다.

위를 응용해본 결과 위에서처럼 한 자리만 1의 차분이 가능한 것이 아니라
Midori64(0x8888888888888888, 0x00000000000000000000000000000000)
= Midori64(0x8989898989898989, 0x01010101010101010101010101010101)
과 같이 여러 자리도 가능함을 알 수 있었다.

뿐만 아니라
Midori64(0x8888888888888888, 0x00000000000000010000000000000001)
= Midori64(0x8888888888888889, 0x00000000000000000000000000000000)
과 같이도 가능하였다.

3.2 Compression 함수 충돌쌍 공격
문제에서 첫 CV는 IV이기 때문에 Compression(IV, ?)의 충돌쌍을 찾아본다.
3.1에서
Midori64(0x8888888888888888, 0x00000000000000000000000000000000)
= Midori64(0x8888888888888889, 0x00000000000000010000000000000001) 이고
Midori64(0x9999999999999999, 0x00000000000000000000000000000000)
= Midori64(0x9999999999999998, 0x00000000000000010000000000000001)
이기 때문에
Compression(IV, IV || IV)
= Compression(IV ⊕ 0x00000000000000010000000000000001, IV || IV)이다.

뿐만 아니라
Compression(IV, IV || IV)
= Compression(IV ⊕ 0x00000000000000100000000000000010, IV || IV)
= Compression(IV ⊕ 0x00000000000001000000000000000100, IV || IV)
…
= Compression(IV ⊕ 0x10000000000000001000000000000000, IV || IV)
= Compression(IV ⊕ 0x11111111111111111111111111111111, IV || IV)

또한 A = IV ⊕ 0x00000000000000010000000000000001에 대해
Compression(IV, A || A) = Compression(A, A || A)도 가능하다.

뿐만 아니라
A = IV ⊕ 0x00000000000000010000000000000010
A = IV ⊕ 0x00000000000000100000000000000100
…
A = IV ⊕ 0x10000000000000001000000000000000
인 위의 모든 A에 대해서도 Compression(IV, A || A) = Compression(A, A || A)이 만족한다.

최종적으로 A = IV ⊕ 0x01010101010101010101010101010101에서도
Compression(IV, A || A) = Compression(A, A || A)이 만족한다.

3.3 Hash 충돌쌍 공격
Hash(IV || IV) = 0x99988998989999888888998898988889 9이고
Hash(IV || IV) ⊕ IV = 0x11100110101111001110011010111100이므로 Hash(IV || IV)를 3.2에서의 A로 사용 가능하다.
A = IV ⊕ 0x11100110101111001110011010111100에 대해 3.2에 의해
Compression(IV, A || A) = Compression(A, A || A)이므로
Hash(A || A) = Hash(IV || IV || A || A)
(∵ Hash(A || A) = Compression(IV, A || A)
Hash(IV || IV || A || A) = Compression(A, A || A) (∵ Hash(IV || IV) = A))

즉
Hash(0x88888888888888889999999999999999888888888888888899999999999999998898989999888889988
989888899999889989899998888899889898888899)
= Hash(0x9998899898999988888899889898888899999889989899998888899889898888899)이다.

구현한 해시를 통해 이를 확인하였다.

4. 공격복잡도
따로 범위를 잡고 조사하지 않았으므로 공격 복잡도는 O(1)이다.

5. pseudo code
1. Hash(IV || IV) = A 계산
2. Hash(A || A), Hash(IV || IV || A || A) 계산

6. 동일한 해시 값을 가지는 서로 다른 메시지 쌍
M_0 =
0x88888888888888889999999999999999888888888888888899999999999999998898989999888889989898
8899999889989899998888899889898888899
M_1 = 0x9998899898999988888899889898888899999889989899998888899889898888899
해시 값: 0x99999999989898898888888889898998

참고문헌
1. Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni : Midori: A Block Cipher for Low Energy(Extended Version) (2015)