,

2020 가을학기 정보보호

프로젝트#2 보고서

: hash function

제출일자: 2020.11.03

교수님: 허준범 교수님

제출인: 2017320233 김채령

[목차]

1. 과제 설명

- 과제 개요

: hash 함수의 취약점을 알아내어, 가능한 시간 내에 공격하기

- 과제 목표

: 256-bit의 output Y가 주어졌을 때, 2^253 hash 연산 내에 768-bit의 input X를 특정하기 위한 최대 라운드 수 구하기, 이를 해결하기 위한 알고리즘 기술

(이때, 코드는 2^32 초과의 hash 연산을 요구하지 않도록 구현해야 하고, 매 라운드의 X를 출력해야 함. 그러나 보고서에는 이론적인 방법론을 모두 기재해야 함.)

- 주어진 hash 함수

input : X = $(A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0, W_0, W_1, \ldots\ldots\ldots\ldots W_{15})$.
Where $A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0, W_0, W_1, \ldots\ldots\ldots\ldots W_{15}$ are each 32-bit.

Intermediate variable : $A_i, B_i, C_i, D_i, E_i, F_i, G_i, H_i, W_i$ each are 32-bit.

number of Round: R

#output : Y = $(A_0 \oplus A_i, B_0 \oplus B_i, C_0 \oplus C_i, D_0 \oplus D_i, E_0 \oplus E_i, F_0 \oplus F_i, G_0 \oplus G_i, H_0 \oplus H_i)$

For (i = 16; i<R; i++):
$W_i = (W_{i-3} <<< 1) \oplus (W_{i-8} <<< 6) \oplus (W_{i-14} <<< 11) \oplus W_{i-16}$

For (i = 0; i<R; i++):
$(A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}, E_{i+1}, F_{i+1}, G_{i+1}, H_{i+1})$ = Round$(A_i, B_i, C_i, D_i, E_i, F_i, G_i, H_i, W_i)$

-> output 은 각 input 과 라운드번째 업데이트된 input 의 XOR 값이며,

각 라운드에서 input 은 input 과 각 라운드번째의 Word(W)를 활용해 연산함.

-> 라운드가 16 이상이 되면(즉, 17 번째 라운드부터), 주어진 Word 로 해당 라운드의 W 를 계산함.

i-th *Round()* function structure

input : $(A_i, B_i, C_i, D_i, E_i, F_i, G_i, H_i, W_i)$

Intermediate variable : T which is 32-bit

output : $(A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}, E_{i+1}, F_{i+1}, G_{i+1}, H_{i+1})$

T = $(W_i[0], W_i[1], W_i[2], W_i[3])$
T = MDS(S(T[0] $\oplus$ $A_i[0]$), S(T[1] $\oplus$ $A_i[1]$), S(T[2] $\oplus$ $A_i[2]$), S(T[3] $\oplus$ $A_i[3]$))
T = MDS(S(T[0] $\oplus$ $B_i[0]$), S(T[1] $\oplus$ $B_i[1]$), S(T[2] $\oplus$ $B_i[2]$), S(T[3] $\oplus$ $B_i[3]$))
T = MDS(S(T[0] $\oplus$ $C_i[0]$), S(T[1] $\oplus$ $C_i[1]$), S(T[2] $\oplus$ $C_i[2]$), S(T[3] $\oplus$ $C_i[3]$))
T = MDS(S(T[0] $\oplus$ $D_i[0]$), S(T[1] $\oplus$ $D_i[1]$), S(T[2] $\oplus$ $D_i[2]$), S(T[3] $\oplus$ $D_i[3]$))
T = MDS(S(T[0] $\oplus$ $E_i[0]$), S(T[1] $\oplus$ $E_i[1]$), S(T[2] $\oplus$ $E_i[2]$), S(T[3] $\oplus$ $E_i[3]$))
T = MDS(S(T[0] $\oplus$ $F_i[0]$), S(T[1] $\oplus$ $F_i[1]$), S(T[2] $\oplus$ $F_i[2]$), S(T[3] $\oplus$ $F_i[3]$))
T = MDS(S(T[0] $\oplus$ $G_i[0]$), S(T[1] $\oplus$ $G_i[1]$), S(T[2] $\oplus$ $G_i[2]$), S(T[3] $\oplus$ $G_i[3]$))

$(A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}, E_{i+1}, F_{i+1}, G_{i+1}, H_{i+1}) = (H_i \oplus T, A_i, B_i, C_i, D_i, E_i, F_i, G_i)$

-> 해당 함수에서 사용되는 연산자는 아래에 정의되어 있음.

- $\oplus$: Bitwise exclusive OR
- <<< n: n-bit left rotation
- S(): S-box of AES (reference: FIPS 197)
- MDS(): Matrix multiplication of AES (reference: FIPS 197)
- T = (T[0],T[1],T[2],T[3]): Concatenates four 8-bit T[0], T[1], T[2], T[3], and assigns the 32-bit result to T
- (T[0],T[1],T[2],T[3]) = T: Separates 32-bit T into four 8-bit T[0], T[1], T[2], T[3]

=> 주어진 hash 함수에서, 256-bit의 image(Y, output)가 주어졌을 때, 2^253 hash 연산 내에 768-bit의 pre-image(X, input)을 결정할 수 있는 최대 라운드 수를 찾고, 매 라운드의 알고리즘을 기술하는 것이 목표이다. 이때, 코드에서는 2^23 hash 연산 이내의 알고리즘을 구현한다.

2. 방법론

- 매 라운드 알고리즘 설명

1) 라운드 1번 (0번째 라운드)

- 0-th round

A1	B1	C1	D1	E1	F1	G1	H1
H0⊕T0	A0	B0	C0	D0	E0	F0	G0

- output(given)

A0⊕A1	B0⊕B1	C0⊕C1	D0⊕D1	E0⊕E1	F0⊕F1	G0⊕G1	H0⊕H1
A0⊕ H0⊕T0	B0⊕A0	C0⊕B0	D0⊕C0	E0⊕D0	F0⊕E0	G0⊕F0	H0⊕G0

- 결론

: output이 주어졌을 때, A0(32-bit)를 추리하면 XOR 연산을 통해 최종 T0와B0~H0까지 구할 수 있다. 또한, A0~G0까지를 모두 알면, Round 연산을 역으로 수행하여 32-bit의 W0도 구할 수 있게 된다. 따라서, 2^32번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0)을 특정할 수 있게 된다.

2) 라운드 2번 (1번째 라운드)

- 1-th round

A2	B2	C2	D2	E2	F2	G2	H2
H1⊕T1	A1	B1	C1	D1	E1	F1	G1
G0⊕T1	H0⊕T0	A0	B0	C0	D0	E0	F0

- output(given)

A0⊕A2	B0⊕B2	C0⊕C2	D0⊕D2	E0⊕E2	F0⊕F2	G0⊕G2	H0⊕H2
A0⊕H1⊕T1	B0⊕A1	C0⊕B1	D0⊕C1	E0⊕D1	F0⊕E1	G0⊕F1	H0⊕G1
A0⊕ G0⊕T1	B0⊕H0⊕T0	C0⊕A0	D0⊕B0	E0⊕C0	F0⊕D0	G0⊕E0	H0⊕F0

- 결론

: output이 주어졌을 때, A0와 B0 총 64-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0와 T1도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로 A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 따라서, 2^64번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1)을 특정할 수 있게 된다.

3) 라운드 3번 (2번째 라운드)

- 2-th round

A3	B3	C3	D3	E3	F3	G3	H3
H2⊕T2	A2	B2	C2	D2	E2	F2	G2
G1⊕T2	H1⊕T1	A1	B1	C1	D1	E1	F1
F0⊕T2	G0⊕T1	H0⊕T0	A0	B0	C0	D0	E0

- output(given)

A0⊕A3	B0⊕B3	C0⊕C3	D0⊕D3	E0⊕E3	F0⊕F3	G0⊕G3	H0⊕H3
A0⊕H2⊕T2	B0⊕A2	C0⊕B2	D0⊕C2	E0⊕D2	F0⊕E2	G0⊕F2	H0⊕G2
A0⊕G1⊕T2	B0⊕H1⊕T1	C0⊕A1	D0⊕B1	E0⊕C1	F0⊕D1	G0⊕E1	H0⊕F1
A0⊕F0⊕T2	B0⊕G0⊕T1	C0⊕H0⊕T0	D0⊕A0	E0⊕B0	F0⊕C0	G0⊕D0	H0⊕E0

- 결론

: output이 주어졌을 때, A0, B0, C0 총 96-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0, T1, T2도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로 A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 마찬가지로 W2도 계산할 수 있다. 따라서, 2^96번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2)을 특정할 수 있게 된다.

4) 라운드 4번 (3번째 라운드)

- 3-th round

A4	B4	C4	D4	E4	F4	G4	H4

H3⊕T3	A3	B3	C3	D3	E3	F3	G3
G2⊕T3	H2⊕T2	A2	B2	C2	D2	E2	F2
F1⊕T3	G1⊕T2	H1⊕T1	A1	B1	C1	D1	E1
E0⊕T3	F0⊕T2	G0⊕T1	H0⊕T0	A0	B0	C0	D0

- output(given)

A0⊕A4	B0⊕B4	C0⊕C4	D0⊕D4	E0⊕E4	F0⊕F4	G0⊕G4	H0⊕H4
A0⊕H3⊕T3	B0⊕A3	C0⊕B3	D0⊕C3	E0⊕D3	F0⊕E3	G0⊕F3	H0⊕G3
A0⊕G2⊕T3	B0⊕H2⊕T2	C0⊕A2	D0⊕B2	E0⊕C2	F0⊕D2	G0⊕E2	H0⊕F2
A0⊕F1⊕T3	B0⊕G1⊕T2	C0⊕H1⊕T1	D0⊕A1	E0⊕B1	F0⊕C1	G0⊕D1	H0⊕E1
A0⊕E0⊕T3	B0⊕F0⊕T2	C0⊕G0⊕T1	D0⊕H0⊕T0	E0⊕A0	F0⊕B0	G0⊕C0	H0⊕D0

- 결론

: output이 주어졌을 때, A0, B0, C0, D0 총 128-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0, T1, T2, T3도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로 A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 마찬가지로 W2와 W3도 계산할 수 있다. 따라서, 2^128번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3)을 특정할 수 있게 된다.

5) 라운드 5번 (4번째 라운드)

- 4-th round

A5	B5	C5	D5	E5	F5	G5	H5
H4⊕T4	A4	B4	C4	D4	E4	F4	G4
G3⊕T4	H3⊕T3	A3	B3	C3	D3	E3	F3
F2⊕T4	G2⊕T3	H2⊕T2	A2	B2	C2	D2	E2
E1⊕T4	F1⊕T3	G1⊕T2	H1⊕T1	A1	B1	C1	D1
D0⊕T4	E0⊕T3	F0⊕T2	G0⊕T1	H0⊕T0	A0	B0	C0

- output(given)

A0⊕A5	B0⊕B5	C0⊕C5	D0⊕D5	E0⊕E5	F0⊕F5	G0⊕G5	H0⊕H5
A0⊕H4⊕T4	B0⊕A4	C0⊕B4	D0⊕C4	E0⊕D4	F0⊕E4	G0⊕F4	H0⊕G4
A0⊕G3⊕T4	B0⊕H3⊕T3	C0⊕A3	D0⊕B3	E0⊕C3	F0⊕D3	G0⊕E3	H0⊕F3
A0⊕F2⊕T4	B0⊕G2⊕T3	C0⊕H2⊕T2	D0⊕A2	E0⊕B2	F0⊕C2	G0⊕D2	H0⊕E2

A0⊕E1⊕T4	B0⊕F1⊕T3	C0⊕G1⊕T2	D0⊕H1⊕T1	E0⊕A1	F0⊕B1	G0⊕C1	H0⊕D1
A0⊕D0⊕T4	B0⊕E0⊕T3	C0⊕F0⊕T2	D0⊕G0⊕T1	E0⊕H0⊕T0	F0⊕A0	G0⊕B0	H0⊕C0

- 결론

: output이 주어졌을 때, A0, B0, C0, D0, E0 총 160-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0, T1, T2, T3, T4도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로 A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 마찬가지로 W2와 W3, W4도 계산할 수 있다. 따라서, 2^160번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4)을 특정할 수 있게 된다.

6) 라운드 6번 (5번째 라운드)

- 5-th round

A6	B6	C6	D6	E6	F6	G6	H6
H5⊕T5	A5	B5	C5	D5	E5	F5	G5
G4⊕T5	H4⊕T4	A4	B4	C4	D4	E4	F4
F3⊕T5	G3⊕T4	H3⊕T3	A3	B3	C3	D3	E3
E2⊕T5	F2⊕T4	G2⊕T3	H2⊕T2	A2	B2	C2	D2
D1⊕T5	E1⊕T4	F1⊕T3	G1⊕T2	H1⊕T1	A1	B1	C1
C0⊕T5	D0⊕T4	E0⊕T3	F0⊕T2	G0⊕T1	H0⊕T0	A0	B0

- output(given)

A0⊕A6	B0⊕B6	C0⊕C6	D0⊕D6	E0⊕E6	F0⊕F6	G0⊕G6	H0⊕H6
A0⊕H5⊕T5	B0⊕A5	C0⊕B5	D0⊕C5	E0⊕D5	F0⊕E5	G0⊕F5	H0⊕G5
A0⊕G4⊕T5	B0⊕H4⊕T4	C0⊕A4	D0⊕B4	E0⊕C4	F0⊕D4	G0⊕E4	H0⊕F4
A0⊕F3⊕T5	B0⊕G3⊕T4	C0⊕H3⊕T3	D0⊕A3	E0⊕B3	F0⊕C3	G0⊕D3	H0⊕E3
A0⊕E2⊕T5	B0⊕F2⊕T4	C0⊕G2⊕T3	D0⊕H2⊕T2	E0⊕A2	F0⊕B2	G0⊕C2	H0⊕D2
A0⊕D1⊕T5	B0⊕E1⊕T4	C0⊕F1⊕T3	D0⊕G1⊕T1	E0⊕H1⊕T1	F0⊕A1	G0⊕B1	H0⊕C1
A0⊕C0⊕T5	B0⊕D0⊕T4	C0⊕E0⊕T3	D0⊕F0⊕T2	E0⊕G0⊕T1	F0⊕H0⊕T0	G0⊕A0	H0⊕B0

- 결론

: output이 주어졌을 때, A0, B0, C0, D0, E0, F0 총 192-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0, T1, T2, T3, T4, T5도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로

A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 마찬가지로 W2와 W3, W4, W5도 계산할 수 있다. 따라서, 2^192번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5)을 특정할 수 있게 된다.

7) 라운드 7번 (6번째 라운드)

- 6-th round

A7	B7	C7	D7	E7	F7	G7	H7
H6⊕T6	A6	B6	C6	D6	E6	F6	G6
G5⊕T6	H5⊕T5	A5	B5	C5	D5	E5	F5
F4⊕T6	G4⊕T5	H4⊕T4	A4	B4	C4	D4	E4
E3⊕T6	F3⊕T5	G3⊕T4	H3⊕T3	A3	B3	C3	D3
D2⊕T6	E2⊕T5	F2⊕T4	G2⊕T3	H2⊕T2	A2	B2	C2
C1⊕T6	D1⊕T5	E1⊕T4	F1⊕T3	G1⊕T2	H1⊕T1	A1	B1
B0⊕T6	C0⊕T5	D0⊕T4	E0⊕T3	F0⊕T2	G0⊕T1	H0⊕T0	A0

- output(given)

A0⊕A7	B0⊕B7	C0⊕C7	D0⊕D7	E0⊕E7	F0⊕F7	G0⊕G7	H0⊕H7
A0⊕H6⊕T6	B0⊕A6	C0⊕B6	D0⊕C6	E0⊕D6	F0⊕E6	G0⊕F6	H0⊕G6
A0⊕G5⊕T6	B0⊕H5⊕T5	C0⊕A5	D0⊕B5	E0⊕C5	F0⊕D5	G0⊕E5	H0⊕F5
A0⊕F4⊕T6	B0⊕G4⊕T5	C0⊕H4⊕T4	D0⊕A4	E0⊕B4	F0⊕C4	G0⊕D4	H0⊕E4
A0⊕E3⊕T6	B0⊕F3⊕T5	C0⊕G3⊕T4	D0⊕H3⊕T3	E0⊕A3	F0⊕B3	G0⊕C3	H0⊕D3
A0⊕D2⊕T6	B0⊕E2⊕T5	C0⊕F2⊕T4	D0⊕G2⊕T3	E0⊕H2⊕T2	F0⊕A2	G0⊕B2	H0⊕C2
A0⊕C1⊕T6	B0⊕D1⊕T5	C0⊕E1⊕T4	D0⊕F1⊕T3	E0⊕G1⊕T2	F0⊕H1⊕T1	G0⊕A1	H0⊕B1
A0⊕B0⊕T6	B0⊕C0⊕T5	C0⊕D0⊕T4	D0⊕E0⊕T3	E0⊕F0⊕T2	F0⊕G0⊕T1	G0⊕H0⊕T0	H0⊕A0

- 결론

: output이 주어졌을 때, A0, B0, C0, D0, E0, F0, G0 총 224-bit를 추리하면 A0~H0까지 알 수 있게 된다. 또한 최종 T0, T1, T2, T3, T4, T5, T6도 알게 되는데, A0~G0까지 알기 때문에 T0와 이들을 이용하여 라운드의 역 연산으로 W0도 구할 수 있다. 또 이미 아는 값으로 A1~G1도 알아낼 수 있고, 같은 방식으로 T1과 이들을 사용하여 W1도 구할 수 있다. 마찬가지로 W2와 W3, W4, W5, W6도 계산할 수 있다. 따라서, 2^224번의 hash 연산으로 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5, W6)을 특정할 수 있게 된다.

➔ 이후의 라운드에 대해서는, 예를 들어 라운드 8 번(7번째 라운드)의 경우,

- 7-th round

A8	B8	C8	D8	E8	F8	G8	H8
H7⊕T7	A7	B7	C7	D7	E7	F7	G7
G6⊕T7	H6⊕T6	A6	B6	C6	D6	E6	F6
F5⊕T7	G5⊕T6	H5⊕T5	A5	B5	C5	D5	E5
E4⊕T7	F4⊕T6	G4⊕T5	H4⊕T4	A4	B4	C4	D4
D3⊕T7	E3⊕T6	F3⊕T5	G3⊕T4	H3⊕T3	A3	B3	C3
C2⊕T7	D2⊕T6	E2⊕T5	F2⊕T4	G2⊕T3	H2⊕T2	A2	B2
B1⊕T7	C1⊕T6	D1⊕T5	E1⊕T4	F1⊕T3	G1⊕T2	H1⊕T1	A1
A0⊕T7	B0⊕T6	C0⊕T5	D0⊕T4	E0⊕T3	F0⊕T2	G0⊕T1	H0⊕T0

- output(given)

A0⊕A8	B0⊕B8	C0⊕C8	D0⊕D8	E0⊕E8	F0⊕F8	G0⊕G8	H0⊕H8
A0⊕ H7⊕T7	B0⊕A7	C0⊕B7	D0⊕C7	E0⊕D7	F0⊕E7	G0⊕F7	H0⊕G7
A0⊕ G6⊕T7	B0⊕H6⊕T6	C0⊕A6	D0⊕B6	E0⊕C6	F0⊕D6	G0⊕E6	H0⊕F6
A0⊕ F5⊕T7	B0⊕G5⊕T6	C0⊕H5⊕T5	D0⊕A5	E0⊕B5	F0⊕C5	G0⊕D5	H0⊕E5
A0⊕ E4⊕T7	B0⊕F4⊕T6	C0⊕G4⊕T5	D0⊕H4⊕T4	E0⊕A4	F0⊕B4	G0⊕C4	H0⊕D4
A0⊕ D3⊕T7	B0⊕E3⊕T6	C0⊕F4⊕T5	D0⊕G3⊕T4	E0⊕H3⊕T3	F0⊕A3	G0⊕B3	H0⊕C3
A0⊕ C2⊕T7	B0⊕D2⊕T6	C0⊕E2⊕T5	D0⊕F2⊕T4	E0⊕G2⊕T3	F0⊕H2⊕T2	G0⊕A2	H0⊕B2
A0⊕ B1⊕T7	B0⊕C1⊕T6	C0⊕D1⊕T5	D0⊕E1⊕T4	E0⊕F1⊕T3	F0 ⊕G1⊕T2	G0⊕H1⊕T1	H0⊕A1
A0⊕A0⊕T7	B0⊕B0⊕T6	C0⊕C0⊕T5	D0⊕D0⊕T4	E0⊕E0⊕T3	F0⊕F0⊕T2	G0⊕G0⊕T1	H0⊕H0⊕T0

=> output이 T7~T0와 동일해지는데, 이때 A0~H0까지 모두 추측해야하므로 총 256-bit에 대하여 2^256번의 hash 연산을 시도해야 한다. 따라서, 문제에서 주어진 2^253이라는 시간 복잡도에 어긋난다. 그러나, 2^256번의 hash 연산을 할 경우 이후 모든 라운드에 대해서도 input(A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5, W6, W7, ... W_i)을 특정할 수 있게 된다.

- 2^253 hash 연산 내에 입력을 특정하기 위한 최대 라운드 수 구하기

정답은 7이다. 즉, 0번째(1번)부터 6번째(7번) 라운드까지는 2^253 hash 연산 내에서 input(A0~H0)와 해당 번째까지의 Word W_i를 특정할 수 있다. 그러나, 그 이후 라운드에

대해서는 2^256번의 hash 연산이 필요하므로 문제 조건과는 맞지 않음을 위의 과정을 통해 확인하였다.

요약하자면, Round를 7번 거칠 때까지는 문제에서 고정한 시간 내에 주어진 output에 대한 input을 Brute-force 접근법으로 결정할 수 있다.

3. 구현

- 과제에서 2^32번을 초과하는 hash 연산을 요하는 알고리즘은 구현할 필요가 없다고 하였다. 따라서, 2^32번의 hash 연산을 이용하여 input을 알아낼 수 있는 첫번째 라운드를 가정하고 input X를 출력해보았다.

(input을 알아내기 위해서 2^32번의 hash 연산을 이용하는 경우는, round 1번(0-th round) 즉, 첫번째 round가 유일하므로, 이를 가정하여 코드를 작성하였다.)

- 즉, 256-bit의 Y가 주어지면, 2^32번의 시도로 A0를 추측하여 A0, B0, C0, D0, E0, F0, G0, H0, 그리고 W0와 A1, B1, C1, D1, E1, F1, G1, H1를 알아낸다. 이때, A0는 Brute force 접근법으로 가정하고, XOR 연산을 통해 B0~H0, T0, A1~H1울 알아낼 수 있다. 구해진 A0~G0와 T0를 이용하여 round 역 연산 구현을 이용해 W0까지 찾아낼 수 있다. 여기에서, 편의상 2^32 번의 A_0 연산 대신, A_0를 임의의 32-bit integer로 정한 뒤, 이것이 올바른 추측이라 가정하였다(A0 값을 int 범위 내에서 임의로 설정.). 2^32가지의 A_0와 그에 따른 출력은 반복만 해주면 되고, 출력물의 가독성을 해치기 때문이다. 따라서, 주어진 Y와 올바른 추측이라 가정한 A_0에 대한 input을 출력하였다.

=> Y와 2^32개의 가능성 중 하나의 A0가 주어질 때,

0. Y를 32-bit의 8개의 문자열로 나눈다.

```
    // split into each of 32-bit
String A_tmp = y.substring(0, 32);
String B_tmp = y.substring(32, 64);
String C_tmp = y.substring(64, 96);
String D_tmp = y.substring(96, 128);
String E_tmp = y.substring(128, 160);
String F_tmp = y.substring(160, 192);
String G_tmp = y.substring(192, 224);
String H_tmp = y.substring(224);
```

1. B0~H0를 XOR 연산으로 구한다

```
int B_0 = A_0^dec_B;

int C_0 = B_0^dec_C;

int D_0 = C_0^dec_D;

int E_0 = D_0^dec_E;
```

```
int F_0 = E_0^dec_F;

int G_0 = F_0^dec_G;

int H_0 = G_0^dec_H;
```

2. T0를 XOR 연산으로 계산한다.

```
int T_0 = dec_T^A_0;
    T_0 = T_0^H_0;
```

3. round 함수의 역을 통해 W0까지 구해낸다.

-> 이는 G_0부터 A_0까지 거슬러 올라가며 아래의 3가지를 반복하면 된다.

해당 모듈은 코드 길이가 상당하므로, 코드에 주석으로 표기했다.

1번) 먼저, 계산한 T_0로 MDS 역 연산을 한다.

2번) 그 결과 값으로, S 역 연산을 한다.

3번) 해당 문자열과 XOR 하여 T_0를 갱신한다.

부록(Source code와 주석): java 사용

```
import java.util.*;
import java.lang.Math.*;


public class kcr_2017320233_prj2 {

	public static void main(String[] args) {

		// 1. get 256-bit output Y
		System.out.printf("\n *** Give an 256-bit binary Y in binary:  \n");

		//Scanner sc = new Scanner(System.in);
		//String y = sc.nextLine();

		String y =
"00110010001100000011000100110010001100010011000000110110001100000011000000110010010010110100101101010111010010110100101101010111001100
00110000001100010011001000110001001100000011011000110000001100000011001001001011010010110101011101001011010010110101011101";
		System.out.printf(" - given Y: "+y+"\n");

		// 2. 2^32 trial to guess A_0 and compute others
			// split into each of 32-bit
		String A_tmp = y.substring(0, 32);
		String B_tmp = y.substring(32, 64);
		String C_tmp = y.substring(64, 96);
		String D_tmp = y.substring(96, 128);
		String E_tmp = y.substring(128, 160);
		String F_tmp = y.substring(160, 192);
		String G_tmp = y.substring(192, 224);
		String H_tmp = y.substring(224);
			// for each guess of A_0 -> B0~H0, T0, W0, A1~B1
		/*for(int A_0=0; A_0<(int)Math.pow(2.0, 32); A_0++) { */
			// ** type: long으로 한 후, 2^32 가지 경우의 수에 대해 다 출력해볼 수도 있음.
			// convert long to 32-bit binary A_0
				// 편의를 위해, A_0를 임의 int 가정(2^32 경우의 수 중에 하나로)
			int A_0 = 25767; // decimal, if longer => type long
			String tmp_a = Integer.toBinaryString(A_0);
			String bin_A_0 = String.format("%32s", tmp_a).replaceAll(" ", "0");
			System.out.printf("\n * an arbitrary guessed 32-bit decimal A_0: ");
			System.out.println(A_0+"\n");

			// compute B0~H0, T0 through XOR
			int dec_B = Integer.parseInt(B_tmp, 2);
			int B_0 = A_0^dec_B;
			String tmp_b = Integer.toBinaryString(B_0);
			String bin_B_0 = String.format("%32s", tmp_b).replaceAll(" ", "0");
			int dec_C = Integer.parseInt(C_tmp, 2);
			int C_0 = B_0^dec_C;
			String tmp_c = Integer.toBinaryString(C_0);
```

```
String bin_C_0 = String.format("%32s", tmp_c).replaceAll(" ", "0");
int dec_D = Integer.parseInt(D_tmp, 2);
int D_0 = C_0^dec_D;
String tmp_d = Integer.toBinaryString(D_0);
String bin_D_0 = String.format("%32s", tmp_d).replaceAll(" ", "0");
int dec_E = Integer.parseInt(E_tmp, 2);
int E_0 = D_0^dec_E;
String tmp_e = Integer.toBinaryString(E_0);
String bin_E_0 = String.format("%32s", tmp_e).replaceAll(" ", "0");
int dec_F = Integer.parseInt(F_tmp, 2);
int F_0 = E_0^dec_F;
String tmp_f = Integer.toBinaryString(F_0);
String bin_F_0 = String.format("%32s", tmp_f).replaceAll(" ", "0");
int dec_G = Integer.parseInt(G_tmp, 2);
int G_0 = F_0^dec_G;
String tmp_g = Integer.toBinaryString(G_0);
String bin_G_0 = String.format("%32s", tmp_g).replaceAll(" ", "0");
int dec_H = Integer.parseInt(H_tmp, 2);
int H_0 = G_0^dec_H;
String tmp_h = Integer.toBinaryString(H_0);
String bin_H_0 = String.format("%32s", tmp_h).replaceAll(" ", "0");

int dec_T = Integer.parseInt(A_tmp, 2);
int T_0 = dec_T^A_0;
T_0 = T_0^H_0;
String tmp_t = Integer.toBinaryString(T_0);
String bin_T_0 = String.format("%32s", tmp_t).replaceAll(" ", "0");

// print A0~H0, T0
System.out.printf("  CASE of A_0 to be "+bin_A_0+" in binary: \n");
System.out.printf("      => A_0: "+ bin_A_0 +" \n");
System.out.printf("      => B_0: "+ bin_B_0+" \n");
System.out.printf("      => C_0: "+ bin_C_0+" \n");
System.out.printf("      => D_0: "+ bin_D_0+" \n");
System.out.printf("      => E_0: "+ bin_E_0+" \n");
System.out.printf("      => F_0: "+ bin_F_0+" \n");
System.out.printf("      => G_0: "+ bin_G_0+" \n");
System.out.printf("      => H_0: "+ bin_H_0+" \n");
System.out.printf("      => T_0: "+ bin_T_0+" \n\n");

// compute A1~H1 through XOR
int A_1 = H_0^T_0;
String tmp_a_1 = Integer.toBinaryString(A_1);
String bin_A_1 = String.format("%32s", tmp_a_1).replaceAll(" ", "0");
int B_1 = A_0;
String tmp_b_1 = Integer.toBinaryString(B_1);
String bin_B_1 = String.format("%32s", tmp_b_1).replaceAll(" ", "0");
int C_1 = B_0;
String tmp_c_1 = Integer.toBinaryString(C_1);
String bin_C_1 = String.format("%32s", tmp_c_1).replaceAll(" ", "0");
int D_1 = C_0;
String tmp_d_1 = Integer.toBinaryString(D_1);
String bin_D_1 = String.format("%32s", tmp_d_1).replaceAll(" ", "0");
int E_1 = D_0;
String tmp_e_1 = Integer.toBinaryString(E_1);
String bin_E_1 = String.format("%32s", tmp_e_1).replaceAll(" ", "0");
int F_1 = E_0;
String tmp_f_1 = Integer.toBinaryString(F_1);
String bin_F_1 = String.format("%32s", tmp_f_1).replaceAll(" ", "0");
int G_1 = F_0;
String tmp_g_1 = Integer.toBinaryString(G_1);
String bin_G_1 = String.format("%32s", tmp_g_1).replaceAll(" ", "0");
int H_1 = G_0;
String tmp_h_1 = Integer.toBinaryString(H_1);
String bin_H_1 = String.format("%32s", tmp_h_1).replaceAll(" ", "0");

// print A1~H1 (intermediate values)
System.out.printf("      => A_1: "+ bin_A_1+" \n");
System.out.printf("      => B_1: "+ bin_B_1+" \n");
System.out.printf("      => C_1: "+ bin_C_1+" \n");
System.out.printf("      => D_1: "+ bin_D_1+" \n");
System.out.printf("      => E_1: "+ bin_E_1+" \n");
System.out.printf("      => F_1: "+ bin_F_1+" \n");
System.out.printf("      => G_1: "+ bin_G_1+" \n");
System.out.printf("      => H_1: "+ bin_H_1+" \n\n");

// compute W0 through inverse of round function
String t_0 = bin_T_0.substring(0, 8);
String t_1 = bin_T_0.substring(8, 16);
String t_2 = bin_T_0.substring(16, 24);
String t_3 = bin_T_0.substring(24);

int[][] M_inv = {{0x0e, 0x0b, 0x0d, 0x09},
                 {0x09, 0x0e, 0x0b, 0x0d},
                 {0x0d, 0x09, 0x0e, 0x0b},
                 {0x0b, 0x0d, 0x09, 0x0e}};

int[][] S_inv = {{0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81,
0xf3, 0xd7, 0xfb},
                 {0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb},
                 {0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e},
                 {0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25},
                 {0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92},
```

```
{0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84},
{0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06},
{0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b},
{0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73},
{0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e},
{0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b},
{0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4},
{0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f},
{0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef},
{0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61},
{0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d}};
                                    // 1. G
                                    // inverse MDS
                                            // if>256 -> return %256
                                    String t_0_0 = t_0.substring(0, 2);
                                    String t_0_1 = t_0.substring(2, 4);
                                    String t_0_2 = t_0.substring(4, 6);
                                    String t_0_3 = t_0.substring(6);
                                    int tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
                                    tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
                                    tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
                                    tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
                                    tmp_t_0 = tmp_t_0%256;

                                    String t_1_0 = t_1.substring(0, 2);
                                    String t_1_1 = t_1.substring(2, 4);
                                    String t_1_2 = t_1.substring(4, 6);
                                    String t_1_3 = t_1.substring(6);
                                    int tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
                                    tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
                                    tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
                                    tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
                                    tmp_t_1 = tmp_t_1%256;

                                    String t_2_0 = t_2.substring(0, 2);
                                    String t_2_1 = t_2.substring(2, 4);
                                    String t_2_2 = t_2.substring(4, 6);
                                    String t_2_3 = t_2.substring(6);
                                    int tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
                                    tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
                                    tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
                                    tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
                                    tmp_t_2 = tmp_t_2%256;

                                    String t_3_0 = t_3.substring(0, 2);
                                    String t_3_1 = t_3.substring(2, 4);
                                    String t_3_2 = t_3.substring(4, 6);
                                    String t_3_3 = t_3.substring(6);
                                    int tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
                                    tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
                                    tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
                                    tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
                                    tmp_t_3 = tmp_t_3%256;


                                    // inverse S
                                    String S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
                                    String S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
                                    String S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
                                    String S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

                                               // row: 4-bit msb, col: 4-bit lsb
                                    int row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
                                    int col_0 = Integer.parseInt(S_t_0.substring(4), 2);
                                    int row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
                                    int col_1 = Integer.parseInt(S_t_1.substring(4), 2);
                                    int row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
                                    int col_2 = Integer.parseInt(S_t_2.substring(4), 2);
                                    int row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
                                    int col_3 = Integer.parseInt(S_t_3.substring(4), 2);

                                    int S_inv_t0 = S_inv[row_0][col_0];
                                    int S_inv_t1 = S_inv[row_1][col_1];
                                    int S_inv_t2 = S_inv[row_2][col_2];
                                    int S_inv_t3 = S_inv[row_3][col_3];

                                    // XOR with G_0 & update to new binary string t_0, t_1. t_2. t_3
                                    int G_0_0 = Integer.parseInt(bin_G_0.substring(0, 8), 2);
                                    int G_0_1 = Integer.parseInt(bin_G_0.substring(8, 16), 2);
                                    int G_0_2 = Integer.parseInt(bin_G_0.substring(16, 24), 2);
                                    int G_0_3 = Integer.parseInt(bin_G_0.substring(24), 2);

                                    t_0 = String.format("%8s", Integer.toBinaryString(G_0_0^S_inv_t0)).replaceAll(" ", "0");
                                    t_1 = String.format("%8s", Integer.toBinaryString(G_0_1^S_inv_t1)).replaceAll(" ", "0");
                                    t_2 = String.format("%8s", Integer.toBinaryString(G_0_2^S_inv_t2)).replaceAll(" ", "0");
                                    t_3 = String.format("%8s", Integer.toBinaryString(G_0_3^S_inv_t3)).replaceAll(" ", "0");
```

```
                    // 2. F
                    // inverse MDS
                    // if>256 -> return %256
                t_0_0 = t_0.substring(0, 2);
                t_0_1 = t_0.substring(2, 4);
                t_0_2 = t_0.substring(4, 6);
                t_0_3 = t_0.substring(6);
                tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
                tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
                tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
                tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
                tmp_t_0 = tmp_t_0%256;

                t_1_0 = t_1.substring(0, 2);
                t_1_1 = t_1.substring(2, 4);
                t_1_2 = t_1.substring(4, 6);
                t_1_3 = t_1.substring(6);
                tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
                tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
                tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
                tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
                tmp_t_1 = tmp_t_1%256;

                t_2_0 = t_2.substring(0, 2);
                t_2_1 = t_2.substring(2, 4);
                t_2_2 = t_2.substring(4, 6);
                t_2_3 = t_2.substring(6);
                tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
                tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
                tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
                tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
                tmp_t_2 = tmp_t_2%256;

                t_3_0 = t_3.substring(0, 2);
                t_3_1 = t_3.substring(2, 4);
                t_3_2 = t_3.substring(4, 6);
                t_3_3 = t_3.substring(6);
                tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
                tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
                tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
                tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
                tmp_t_3 = tmp_t_3%256;

                // inverse S
                S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
                S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
                S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
                S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

                    // row: 4-bit msb, col: 4-bit lsb
                row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
                col_0 = Integer.parseInt(S_t_0.substring(4), 2);
                row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
                col_1 = Integer.parseInt(S_t_1.substring(4), 2);
                row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
                col_2 = Integer.parseInt(S_t_2.substring(4), 2);
                row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
                col_3 = Integer.parseInt(S_t_3.substring(4), 2);

                S_inv_t0 = S_inv[row_0][col_0];
                S_inv_t1 = S_inv[row_1][col_1];
                S_inv_t2 = S_inv[row_2][col_2];
                S_inv_t3 = S_inv[row_3][col_3];

                // XOR with F_0 & update to new binary string t_0, t_1. t_2. t_3
                int F_0_0 = Integer.parseInt(bin_F_0.substring(0, 8), 2);
                int F_0_1 = Integer.parseInt(bin_F_0.substring(8, 16), 2);
                int F_0_2 = Integer.parseInt(bin_F_0.substring(16, 24), 2);
                int F_0_3 = Integer.parseInt(bin_F_0.substring(24), 2);

                t_0 = String.format("%8s", Integer.toBinaryString(F_0_0^S_inv_t0)).replaceAll(" ", "0");
                t_1 = String.format("%8s", Integer.toBinaryString(F_0_1^S_inv_t1)).replaceAll(" ", "0");
                t_2 = String.format("%8s", Integer.toBinaryString(F_0_2^S_inv_t2)).replaceAll(" ", "0");
                t_3 = String.format("%8s", Integer.toBinaryString(F_0_3^S_inv_t3)).replaceAll(" ", "0");


                    // 3. E
                // inverse MDS
                // if>256 -> return %256
            t_0_0 = t_0.substring(0, 2);
            t_0_1 = t_0.substring(2, 4);
            t_0_2 = t_0.substring(4, 6);
            t_0_3 = t_0.substring(6);
            tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
            tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
            tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
            tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
            tmp_t_0 = tmp_t_0%256;

            t_1_0 = t_1.substring(0, 2);
            t_1_1 = t_1.substring(2, 4);
            t_1_2 = t_1.substring(4, 6);
            t_1_3 = t_1.substring(6);
            tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
            tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
```

```
            tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
            tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
            tmp_t_1 = tmp_t_1%256;

            t_2_0 = t_2.substring(0, 2);
            t_2_1 = t_2.substring(2, 4);
            t_2_2 = t_2.substring(4, 6);
            t_2_3 = t_2.substring(6);
            tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
            tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
            tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
            tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
            tmp_t_2 = tmp_t_2%256;

            t_3_0 = t_3.substring(0, 2);
            t_3_1 = t_3.substring(2, 4);
            t_3_2 = t_3.substring(4, 6);
            t_3_3 = t_3.substring(6);
            tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
            tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
            tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
            tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
            tmp_t_3 = tmp_t_3%256;

            // inverse S
            S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
            S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
            S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
            S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

                        // row: 4-bit msb, col: 4-bit lsb
            row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
            col_0 = Integer.parseInt(S_t_0.substring(4), 2);
            row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
            col_1 = Integer.parseInt(S_t_1.substring(4), 2);
            row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
            col_2 = Integer.parseInt(S_t_2.substring(4), 2);
            row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
            col_3 = Integer.parseInt(S_t_3.substring(4), 2);

            S_inv_t0 = S_inv[row_0][col_0];
            S_inv_t1 = S_inv[row_1][col_1];
            S_inv_t2 = S_inv[row_2][col_2];
            S_inv_t3 = S_inv[row_3][col_3];

            // XOR with E_0 & update to new binary string t_0, t_1. t_2. t_3
            int E_0_0 = Integer.parseInt(bin_E_0.substring(0, 8), 2);
            int E_0_1 = Integer.parseInt(bin_E_0.substring(8, 16), 2);
            int E_0_2 = Integer.parseInt(bin_E_0.substring(16, 24), 2);
            int E_0_3 = Integer.parseInt(bin_E_0.substring(24), 2);

            t_0 = String.format("%8s", Integer.toBinaryString(E_0_0^S_inv_t0)).replaceAll(" ", "0");
            t_1 = String.format("%8s", Integer.toBinaryString(E_0_1^S_inv_t1)).replaceAll(" ", "0");
            t_2 = String.format("%8s", Integer.toBinaryString(E_0_2^S_inv_t2)).replaceAll(" ", "0");
            t_3 = String.format("%8s", Integer.toBinaryString(E_0_3^S_inv_t3)).replaceAll(" ", "0");


                                    // 4. D
            // inverse MDS
            // if>256 -> return %256
t_0_0 = t_0.substring(0, 2);
t_0_1 = t_0.substring(2, 4);
t_0_2 = t_0.substring(4, 6);
t_0_3 = t_0.substring(6);
tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
tmp_t_0 = tmp_t_0%256;

t_1_0 = t_1.substring(0, 2);
t_1_1 = t_1.substring(2, 4);
t_1_2 = t_1.substring(4, 6);
t_1_3 = t_1.substring(6);
tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
tmp_t_1 = tmp_t_1%256;

t_2_0 = t_2.substring(0, 2);
t_2_1 = t_2.substring(2, 4);
t_2_2 = t_2.substring(4, 6);
t_2_3 = t_2.substring(6);
tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
tmp_t_2 = tmp_t_2%256;

t_3_0 = t_3.substring(0, 2);
t_3_1 = t_3.substring(2, 4);
t_3_2 = t_3.substring(4, 6);
t_3_3 = t_3.substring(6);
tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
```

```
tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
tmp_t_3 = tmp_t_3%256;

// inverse S
S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

            // row: 4-bit msb, col: 4-bit lsb
row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
col_0 = Integer.parseInt(S_t_0.substring(4), 2);
row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
col_1 = Integer.parseInt(S_t_1.substring(4), 2);
row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
col_2 = Integer.parseInt(S_t_2.substring(4), 2);
row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
col_3 = Integer.parseInt(S_t_3.substring(4), 2);

S_inv_t0 = S_inv[row_0][col_0];
S_inv_t1 = S_inv[row_1][col_1];
S_inv_t2 = S_inv[row_2][col_2];
S_inv_t3 = S_inv[row_3][col_3];

// XOR with D_0 & update to new binary string t_0, t_1. t_2. t_3
int D_0_0 = Integer.parseInt(bin_D_0.substring(0, 8), 2);
int D_0_1 = Integer.parseInt(bin_D_0.substring(8, 16), 2);
int D_0_2 = Integer.parseInt(bin_D_0.substring(16, 24), 2);
int D_0_3 = Integer.parseInt(bin_D_0.substring(24), 2);

t_0 = String.format("%8s", Integer.toBinaryString(D_0_0^S_inv_t0)).replaceAll(" ", "0");
t_1 = String.format("%8s", Integer.toBinaryString(D_0_1^S_inv_t1)).replaceAll(" ", "0");
t_2 = String.format("%8s", Integer.toBinaryString(D_0_2^S_inv_t2)).replaceAll(" ", "0");
t_3 = String.format("%8s", Integer.toBinaryString(D_0_3^S_inv_t3)).replaceAll(" ", "0");


                                    // 5. C
// inverse MDS
// if>256 -> return %256
t_0_0 = t_0.substring(0, 2);
t_0_1 = t_0.substring(2, 4);
t_0_2 = t_0.substring(4, 6);
t_0_3 = t_0.substring(6);
tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
tmp_t_0 = tmp_t_0%256;

t_1_0 = t_1.substring(0, 2);
t_1_1 = t_1.substring(2, 4);
t_1_2 = t_1.substring(4, 6);
t_1_3 = t_1.substring(6);
tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
tmp_t_1 = tmp_t_1%256;

t_2_0 = t_2.substring(0, 2);
t_2_1 = t_2.substring(2, 4);
t_2_2 = t_2.substring(4, 6);
t_2_3 = t_2.substring(6);
tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
tmp_t_2 = tmp_t_2%256;

t_3_0 = t_3.substring(0, 2);
t_3_1 = t_3.substring(2, 4);
t_3_2 = t_3.substring(4, 6);
t_3_3 = t_3.substring(6);
tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
tmp_t_3 = tmp_t_3%256;

// inverse S
S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

// row: 4-bit msb, col: 4-bit lsb
row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
col_0 = Integer.parseInt(S_t_0.substring(4), 2);
row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
col_1 = Integer.parseInt(S_t_1.substring(4), 2);
row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
col_2 = Integer.parseInt(S_t_2.substring(4), 2);
row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
col_3 = Integer.parseInt(S_t_3.substring(4), 2);

S_inv_t0 = S_inv[row_0][col_0];
S_inv_t1 = S_inv[row_1][col_1];
```

```
S_inv_t2 = S_inv[row_2][col_2];
S_inv_t3 = S_inv[row_3][col_3];

// XOR with C_0 & update to new binary string t_0, t_1. t_2. t_3
int C_0_0 = Integer.parseInt(bin_C_0.substring(0, 8), 2);
int C_0_1 = Integer.parseInt(bin_C_0.substring(8, 16), 2);
int C_0_2 = Integer.parseInt(bin_C_0.substring(16, 24), 2);
int C_0_3 = Integer.parseInt(bin_C_0.substring(24), 2);

t_0 = String.format("%8s", Integer.toBinaryString(C_0_0^S_inv_t0)).replaceAll(" ", "0");
t_1 = String.format("%8s", Integer.toBinaryString(C_0_1^S_inv_t1)).replaceAll(" ", "0");
t_2 = String.format("%8s", Integer.toBinaryString(C_0_2^S_inv_t2)).replaceAll(" ", "0");
t_3 = String.format("%8s", Integer.toBinaryString(C_0_3^S_inv_t3)).replaceAll(" ", "0");


                                    // 6. B
// inverse MDS
// if>256 -> return %256
t_0_0 = t_0.substring(0, 2);
t_0_1 = t_0.substring(2, 4);
t_0_2 = t_0.substring(4, 6);
t_0_3 = t_0.substring(6);
tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
tmp_t_0 = tmp_t_0%256;

t_1_0 = t_1.substring(0, 2);
t_1_1 = t_1.substring(2, 4);
t_1_2 = t_1.substring(4, 6);
t_1_3 = t_1.substring(6);
tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
tmp_t_1 = tmp_t_1%256;

t_2_0 = t_2.substring(0, 2);
t_2_1 = t_2.substring(2, 4);
t_2_2 = t_2.substring(4, 6);
t_2_3 = t_2.substring(6);
tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
tmp_t_2 = tmp_t_2%256;

t_3_0 = t_3.substring(0, 2);
t_3_1 = t_3.substring(2, 4);
t_3_2 = t_3.substring(4, 6);
t_3_3 = t_3.substring(6);
tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
tmp_t_3 = tmp_t_3%256;

// inverse S
S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

// row: 4-bit msb, col: 4-bit lsb
row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
col_0 = Integer.parseInt(S_t_0.substring(4), 2);
row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
col_1 = Integer.parseInt(S_t_1.substring(4), 2);
row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
col_2 = Integer.parseInt(S_t_2.substring(4), 2);
row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
col_3 = Integer.parseInt(S_t_3.substring(4), 2);

S_inv_t0 = S_inv[row_0][col_0];
S_inv_t1 = S_inv[row_1][col_1];
S_inv_t2 = S_inv[row_2][col_2];
S_inv_t3 = S_inv[row_3][col_3];

// XOR with B_0 & update to new binary string t_0, t_1. t_2. t_3
int B_0_0 = Integer.parseInt(bin_B_0.substring(0, 8), 2);
int B_0_1 = Integer.parseInt(bin_B_0.substring(8, 16), 2);
int B_0_2 = Integer.parseInt(bin_B_0.substring(16, 24), 2);
int B_0_3 = Integer.parseInt(bin_B_0.substring(24), 2);

t_0 = String.format("%8s", Integer.toBinaryString(B_0_0^S_inv_t0)).replaceAll(" ", "0");
t_1 = String.format("%8s", Integer.toBinaryString(B_0_1^S_inv_t1)).replaceAll(" ", "0");
t_2 = String.format("%8s", Integer.toBinaryString(B_0_2^S_inv_t2)).replaceAll(" ", "0");
t_3 = String.format("%8s", Integer.toBinaryString(B_0_3^S_inv_t3)).replaceAll(" ", "0");


                                    // 7. A
// inverse MDS
// if>256 -> return %256
t_0_0 = t_0.substring(0, 2);
t_0_1 = t_0.substring(2, 4);
t_0_2 = t_0.substring(4, 6);
```

```
t_0_3 = t_0.substring(6);
tmp_t_0 = M_inv[0][0]*Integer.parseInt(t_0_0,2);
tmp_t_0 = tmp_t_0^(M_inv[0][1]*Integer.parseInt(t_0_1,2));
tmp_t_0 = tmp_t_0^(M_inv[0][2]*Integer.parseInt(t_0_2,2));
tmp_t_0 = tmp_t_0^(M_inv[0][3]*Integer.parseInt(t_0_3,2));
tmp_t_0 = tmp_t_0%256;

t_1_0 = t_1.substring(0, 2);
t_1_1 = t_1.substring(2, 4);
t_1_2 = t_1.substring(4, 6);
t_1_3 = t_1.substring(6);
tmp_t_1 = M_inv[1][0]*Integer.parseInt(t_1_0,2);
tmp_t_1 = tmp_t_1^(M_inv[1][1]*Integer.parseInt(t_1_1,2));
tmp_t_1 = tmp_t_1^(M_inv[1][2]*Integer.parseInt(t_1_2,2));
tmp_t_1 = tmp_t_1^(M_inv[1][3]*Integer.parseInt(t_1_3,2));
tmp_t_1 = tmp_t_1%256;

t_2_0 = t_2.substring(0, 2);
t_2_1 = t_2.substring(2, 4);
t_2_2 = t_2.substring(4, 6);
t_2_3 = t_2.substring(6);
tmp_t_2 = M_inv[2][0]*Integer.parseInt(t_2_0,2);
tmp_t_2 = tmp_t_0^(M_inv[2][1]*Integer.parseInt(t_2_1,2));
tmp_t_2 = tmp_t_0^(M_inv[2][2]*Integer.parseInt(t_2_2,2));
tmp_t_2 = tmp_t_0^(M_inv[2][3]*Integer.parseInt(t_2_3,2));
tmp_t_2 = tmp_t_2%256;

t_3_0 = t_3.substring(0, 2);
t_3_1 = t_3.substring(2, 4);
t_3_2 = t_3.substring(4, 6);
t_3_3 = t_3.substring(6);
tmp_t_3 = M_inv[3][0]*Integer.parseInt(t_3_0,2);
tmp_t_3 = tmp_t_3^(M_inv[3][1]*Integer.parseInt(t_3_1,2));
tmp_t_3 = tmp_t_3^(M_inv[3][2]*Integer.parseInt(t_3_2,2));
tmp_t_3 = tmp_t_3^(M_inv[3][3]*Integer.parseInt(t_3_3,2));
tmp_t_3 = tmp_t_3%256;

// inverse S
S_t_0 = String.format("%8s", Integer.toBinaryString(tmp_t_0)).replaceAll(" ", "0");
S_t_1 = String.format("%8s", Integer.toBinaryString(tmp_t_1)).replaceAll(" ", "0");
S_t_2 = String.format("%8s", Integer.toBinaryString(tmp_t_2)).replaceAll(" ", "0");
S_t_3 = String.format("%8s", Integer.toBinaryString(tmp_t_3)).replaceAll(" ", "0");

// row: 4-bit msb, col: 4-bit lsb
row_0 = Integer.parseInt(S_t_0.substring(0, 4), 2);
col_0 = Integer.parseInt(S_t_0.substring(4), 2);
row_1 = Integer.parseInt(S_t_1.substring(0, 4), 2);
col_1 = Integer.parseInt(S_t_1.substring(4), 2);
row_2 = Integer.parseInt(S_t_2.substring(0, 4), 2);
col_2 = Integer.parseInt(S_t_2.substring(4), 2);
row_3 = Integer.parseInt(S_t_3.substring(0, 4), 2);
col_3 = Integer.parseInt(S_t_3.substring(4), 2);

S_inv_t0 = S_inv[row_0][col_0];
S_inv_t1 = S_inv[row_1][col_1];
S_inv_t2 = S_inv[row_2][col_2];
S_inv_t3 = S_inv[row_3][col_3];

// XOR with A_0 & update to new binary string t_0, t_1. t_2. t_3
int A_0_0 = Integer.parseInt(bin_A_0.substring(0, 8), 2);
int A_0_1 = Integer.parseInt(bin_A_0.substring(8, 16), 2);
int A_0_2 = Integer.parseInt(bin_A_0.substring(16, 24), 2);
int A_0_3 = Integer.parseInt(bin_A_0.substring(24), 2);

String w_0 = String.format("%8s", Integer.toBinaryString(A_0_0^S_inv_t0)).replaceAll(" ", "0");
String w_1 = String.format("%8s", Integer.toBinaryString(A_0_1^S_inv_t1)).replaceAll(" ", "0");
String w_2 = String.format("%8s", Integer.toBinaryString(A_0_2^S_inv_t2)).replaceAll(" ", "0");
String w_3 = String.format("%8s", Integer.toBinaryString(A_0_3^S_inv_t3)).replaceAll(" ", "0");


                                    // print each of it for each A_0
                                    System.out.printf("      => W_0: "+w_0+w_1+w_2+w_3+" \n");
                        //}
            }


}
```

끝입니다. 고맙습니다😊