MD5 충돌 공격(MD5 Collision Attack) 구현 보고서

Sepaper

이론적 배경

해시함수 (Hash Function)

해시함수란, 임의의 길이를 갖는 메시지가 입력으로 들어가면, 고정된 길이의 해시 값을 출력하는 함수이다. 해시함수는 보안을 위해 사용될 수 있는데, 이런 해시함수를 암호학적 해시함수(Cryptographic Hash Function)이라 부른다. 최근에 흔하게 쓰이는 RSA, AES는 허가 받지 않은 사람이 암호문으로부터 원래 내용을 알아내지 못하도록 하기 위해 사용되는 암호화 기법이다. 이와 반대로, 암호학적 해시함수는 내용의 변조, 위조 등과 같은 Active attack을 방어하기 위해 사용될 수 있다.

암호학적 해시 함수 (Cryptographic Hash Function)

암호학적 해시 함수는 One-way property와 Collision-free property, 두 조건을 충족해야 한다. One-way property란, 메시지가 주어졌을 때, 대응되는 해시 값을 구하는 것은 쉽지만, 반대로 해시 값이 주어졌을 때, 입력으로 사용된 메시지를 알기 어려워야 한다는 성질이다. Collision-free property란, 동일한 해시 값을 갖는 메시지들을 찾기 어려워야 한다는 성질이다. 임의의 길이를 갖는 메시지를 입력으로 받는 해시함수의 특성 상, 입력 집합의 원소의 개수가 해시 값 집합의 원소의 개수보다 훨씬 크므로, 충돌(Collision)은 필연적이다. 하지만, Collision-free property는 충돌을 쉽게 찾기가 어려워야 함을 의미한다.

Message-Digest algorithm 5(MD5)

MD5는 임의의 길이를 갖는 메시지를 입력으로 받아, 128비트 길이의 해시 값을 출력하는 해시함수이다. MD5의 구조는 다음과 같다.

1. 메시지에 패딩(Padding) 작업을 한다.
2. 패딩 처리된 메시지를 512비트(64바이트) 단위의 메시지 블록들로 나눈다.
3. MD5의 Compress 함수가 메시지 블록을 차례대로 입력 받아 결과를 계산한다. 이 때, 이전 메시지 블록의 연산 결과가 다음 메시지 블록의 연산에서의 Initial state가 된다.
4. 마지막 메시지 블록의 결과가 최종적인 128비트의 해시 값이 된다.

MD5의 Compress 함수의 구조는 다음과 같다.

이 함수의 입력은 두가지로, 하나는 이전 메시지 블록의 연산 결과이고, 하나는, 512비트의 메시지 블록이다.

1. 입력으로 들어온 512비트의 메시지 블록을 16개의 32비트 Word로 나눈다. 이를 각각 M[0], M[1], …, M[15]라 한다.

2. 연산은 총 4라운드로 구성되는데, 각 라운드는 16번의 Step으로 구성 되어있다. 각 step에서는 비선형함수 f_t을 사용한다. 각 라운드에서는 서로 다른 f_t을 사용하는데, 1라운드는 F, 2라운드는 G, 3라운드는 H, 4라운드는 I라는 함수를 사용한다.

$$F(X,Y,Z) = ((X\ \&Y)\ |\ ((\sim X)\ \&\ Z))$$

$$G(X,Y,Z) = \big((X\ \&\ Z)\big|\ (Y\ \&\ Z)\big)$$

$$H(X,Y,Z) = (X \oplus \mathrm{Y} \oplus \mathrm{Z})$$

$$I(X,Y,Z) = (Y\ \oplus (\mathrm{X}\ |(\sim \mathrm{Z})))$$

3. Step t에서, f_t의 입력은 Q_t, Q_{t-1}, Q_{t-2}이 된다. 여기서 Q는 각 Step의 결과로, step t에서는 Q_{t+1}을 계산한다. Step 0에서는 Q_1을 계산하게 된다. Step 0에서 사용되는 $Q_{-3}, Q_0, Q_{-1}, Q_{-2}$의 경우, 이전 메시지 블록의 연산 결과가 사용되게 된다.

4. f_t에 Q_{t-3}, AC_t, W_t을 더해 T_t을 계산한다. 여기서 W는 1번에서 구한 16개의 Word 중 하나를 사용하게 된다. 각 step에서 0~15 중 몇번의 Word를 사용할 지는 정해져 있다. AC는 Addition Constant로, 다음과 같이 계산될 수 있다. 각 step에서 어떤 AC를 사용할지는 항상 동일하므로, AC값을 미리 계산해서 표에 넣어서 사용하면 최적화할 수 있다.

$$AC_t = floor(2^{32}|\sin{(t+1)}|)$$

5. 4번에서 구한 T_t을 Circular left shift 연산하여 R_t를 구한다. 각 step에서 몇 번 shift할지 역시 정해져 있기 때문에, 표에 넣어서 사용하면 된다.

6. $Q_{t+1} = R_t + Q_t$, 5번에서 구한 R에 Q_t를 더해 step t의 결과인 Q_{t+1}을 구한다.

7. 위 과정 1~6을 표현한 식은 다음과 같다.

$$F_t = f_t(Q_t, Q_{t-1}, Q_{t-2})$$

$$T_t = F_t + Q_{t-3} + AC_t + W_t$$

$$R_t = RL(T_t, RC_t)$$

$$Q_{t+1} = Q_t + R_t$$

MD5의 취약점

MD5가 암호학적 해시 함수로서 충분한 역할을 하기 위해서는 위에서 소개한 One-way property와 Collision-free property 조건을 만족해야 한다. 하지만, [1]에서 Wang이 MD5의 결함을 찾아 같은 해시 값을 갖는 메시지 쌍을 찾는데 성공하였고, 그 이후 논문에서 [1]에서의 연구를 바탕으로 1대의 개인용 컴퓨터(PC)만을 가지고 길게는 몇 분, 짧게는 몇 초 안에 충돌을 찾아낼 수 있게 만드는 기법들이 등장하게 되었다. 따라서, 사실상 MD5 현재 Collision-free property를 만족시키지 못한다.

공격 시나리오 및 구현 목표

최종적인 공격 목표는 서로 다르게 작동하지만, 해시 값이 동일한 프로그램 두 개를 생성하는 것이다. 하나는 정상적인 작동을 하지만, 다른 하나는, 시스템의 중요 파일을 삭제하는 등 악의적인 작동을 하도록 만들 수 있다. 두 개의 프로그램이 어떤 구조를 가져야할 지에 대한 구체적인 내용은 [4]에 서술되어 있다. [4]에서 소개한 pseudo code는 다음과 같다.

```
Array X;
Array Y

Main (void) {
If(X의 내용 == Y의 내용) 정상적인 코드 실행;
Else 악의적인 코드 실행;
}
```

정상적인 작동을 하는 프로그램을 1번, 악의적인 프로그램을 2번이라고 하자. 1번 프로그램에서의 배열 X와 배열 Y는 A라는 같은 내용을 갖는다고 하고, A와 같은 해시 값을 갖지만, 다른 내용인 B가 존재한다고 했을 때, 2번 프로그램에서는 배열 X는 A라는 내용을 갖지만, 배열 Y는 B라는 내용을 갖도록 한다.

두 개의 프로그램을 hex editor로 열어보게 되면, 프로그램은 다음과 같은 형태를 갖는다.

Prefix- Array X 내용(A) – Array Y 내용(A or B) – Suffix

A와 B의 해시 값이 동일하기 때문에, 프로그램 전체의 해시 값 역시 동일하다. 하지만, 프로그램 1에서는 배열 X와 Y의 내용이 같기 때문에 정상적인 코드가 실행되지만, 프로그램 2에서는 배열 X와 Y의 내용이 다르기 때문에 악의적인 코드가 실행되게 된다.

많은 사람들이 신뢰하는 기관이 1번 프로그램에 대해 검증을 마친 후, MD5 해시 값을 공개하여, 사람들이 정상적인 프로그램인지 확인할 수 있도록 한다면, 공격자는 1번 프로그램에 대해 검증을 받은 후, 사람들에게는 2번 프로그램을 공개하여 사람들이 악의적인 작동을 하는 프로그램을 신뢰할 수 있는 프로그램으로 믿고, 악의적인 프로그램을 실행하도록 만들 수 있다.

이번 구현의 목표는 충돌을 일으키는 A와 B를 직접 찾아보고, 구한 A와 B를 이용해, 같은 해시 값을 갖지만, 서로 다른 동작을 하는 프로그램 두 개를 직접 만들어보는 것이다.

본론

이번 구현에서는 두 개의 메시지 블록에 대한 충돌을 찾는 것을 목표로 했다. (1024비트, 128바이트 길이) 각 메시지 블록을 첫번째 메시지 블록(first block), 두번째 메시지 블록(second block)이라고 하자. [1]에서는 128바이트 길이의 메시지의 충돌을 찾기 위한 차분 경로(Differential Path)를 제시하였고, 메시지들이 해당 차분 경로를 따라가기 위해 만족시켜야 할 Sufficient Condition을 제시하였다. 여기서 사용하는 차분은 Add-difference이고, $\delta X = X' - X \ (mod\ 2\text{^}32)$ 이라 표현할 수 있다. (X로 Q_t, F_t, W_t, T_t 등이 사용될 수 있고, 이는 각각 step t에서의 Q의 차분, F의 차분, W의 차분, T의 차분을 의미한다.) [1]의 내용을 바탕으로, [3]에서는 개선된 Sufficient Condition을 제시함과 동시에, 두번째 메시지 블록에 대한 또 다른 여러 개의 차분 경로를 제시하였다. 이번 구현에서는 [1]에서 제시한 첫번째 메시지 블록 차분 경로([3]에서의 Table B-1)와 [3]에서 제시한 두번째 메시지 블록 3번 차분 경로(nr. 3, [3]에서 Table B-11)를 이용해 충돌을 찾았다.

차분 경로(Differential path)

아래는 [3]에서 Table B-1을 그대로 가져온 것으로, 첫번째 블록의 차분 경로를 나타낸다.

t	ΔQ_t (BSDR of δQ_t)	δF_t	δw_t	δT_t	RC_t
$0-3$	$-$	$-$	$-$	$-$	·
4	$-$	$-$	2^{31}	2^{31}	7
5	$+2^6 \ldots +2^{21}, -2^{22}$	$2^{11}+2^{19}$	$-$	$2^{11}+2^{19}$	12
6	$-2^6+2^{23}+2^{31}$	$-2^{10}-2^{14}$	$-$	$-2^{10}-2^{14}$	17
7	$+2^0 \ldots +2^4, -2^5, +2^6 \ldots +2^{10}$ $-2^{11}, -2^{23} \ldots -2^{25}, +2^{26} \ldots +2^{31}$	$-2^2+2^5+2^{10}$ $+2^{16}-2^{25}-2^{27}$	$-$	$-2^2+2^5+2^{10}$ $+2^{16}-2^{25}-2^{27}$	22
8	$+2^0+2^{15}-2^{16}+2^{17}$ $+2^{18}+2^{19}-2^{20}-2^{23}$	$2^6+2^8+2^{10}$ $+2^{16}-2^{24}+2^{31}$	$-$	$2^8+2^{10}+2^{16}$ $-2^{24}+2^{31}$	7
9	$-2^0+2^1+2^6+2^7-2^8-2^{31}$	$2^0+2^6-2^{20}$ $-2^{23}+2^{26}+2^{31}$	$-$	$2^0-2^{20}+2^{26}$	12
10	$-2^{12}+2^{13}+2^{31}$	$2^0+2^6+2^{13}-2^{23}$	$-$	$2^{13}-2^{27}$	17
11	$+2^{30}+2^{31}$	-2^0-2^8	2^{15}	$-2^8-2^{17}-2^{23}$	22
12	$+2^7-2^8, +2^{13} \ldots +2^{18}, -2^{19}+2^{31}$	$2^7+2^{17}+2^{31}$	$-$	$2^0+2^6+2^{17}$	7
13	$-2^{24}+2^{25}+2^{31}$	$-2^{13}+2^{31}$	$-$	-2^{12}	12
14	$+2^{31}$	$2^{18}+2^{31}$	2^{31}	$2^{18}-2^{30}$	17
15	$+2^3-2^{15}+2^{31}$	$2^{25}+2^{31}$	$-$	$-2^7-2^{13}+2^{25}$	22
16	$-2^{29}+2^{31}$	2^{31}	$-$	2^{24}	5
17	$+2^{31}$	2^{31}	$-$	$-$	9
18	$+2^{31}$	2^{31}	2^{15}	2^3	14
19	$+2^{17}+2^{31}$	2^{31}	$-$	-2^{29}	20
20	$+2^{31}$	2^{31}	$-$	$-$	5
21	$+2^{31}$	2^{31}	$-$	$-$	9
22	$+2^{31}$	2^{31}	$-$	2^{17}	14
23	$-$	$-$	2^{31}	$-$	20
24	$-$	2^{31}	$-$	$-$	5
25	$-$	$-$	2^{31}	$-$	9
$26-33$	$-$	$-$	$-$	$-$	·
34	$-$	$-$	2^{15}	2^{15}	16
35	$\delta Q_{35} = 2^{31}$	2^{31}	2^{31}	$-$	23
36	$\delta Q_{36} = 2^{31}$	$-$	$-$	$-$	4
37	$\delta Q_{37} = 2^{31}$	2^{31}	2^{31}	$-$	11
$38-49$	$\delta Q_t = 2^{31}$	2^{31}	$-$	$-$	·
50	$\delta Q_{50} = 2^{31}$	$-$	2^{31}	$-$	15
$51-59$	$\delta Q_t = 2^{31}$	2^{31}	$-$	$-$	·
60	$\delta Q_{60} = 2^{31}$	$-$	2^{31}	$-$	6
61	$\delta Q_{61} = 2^{31}$	2^{31}	2^{15}	2^{15}	10
62	$\delta Q_{62} = 2^{31}+2^{25}$	2^{31}	$-$	$-$	15
63	$\delta Q_{63} = 2^{31}+2^{25}$	2^{31}	$-$	$-$	21
64	$\delta Q_{64} = 2^{31}+2^{25}$	×	×	×	×

위 표를 설명하면, "-"로 표시된 부분은, 차분이 0이라는 의미이다. $W_4 = M_4$로, step 4에서 사용되는 W는 M[4]인데, (M'[4]-M[4] mod 2^32)의 결과가 2^31이라는 의미다. Step 11, step 14에서도 이처럼 해석하면 된다. 이를 이용해서, M[4], M[11], M[14]을 먼저 구한 다음에, 이 3개에 각자의 차분을 더해줘서 M'[4], M'[11], M'[14]을 구할 수 있다. (4,11,14를 제외한 나머지 0~15에서는 M=M'이다.) 아래에서 살펴볼 Sufficient Condition을 통해 구할 수 있는 것은 M들뿐인데, M을 찾고 난 뒤, 위 표에서의 M의 차분을 이용해 M'들을 구할 수 있다. 그럼, 첫번째 메시지 블록 쌍(M과 M')을

찾게 되는 것이다.

아래는 [3]에서 Table B-11, Table B-2을 그대로 가져온 것으로, 두번째 블록의 차분 경로를 나타낸다.

t	ΔQ_t (BSDR of δQ_t)	δF_t	δw_t	δT_t	RC_t
-3	$+2^{31}$	$\times$	$\times$	$\times$	$\times$
-2	$+2^{25}+2^{31}$	$\times$	$\times$	$\times$	$\times$
-1	$+2^{25}+2^{31}$	$\times$	$\times$	$\times$	$\times$
0	$+2^{25}+2^{31}$	$2^{25}+2^{31}$	$-$	2^{25}	7
1	$+2^{0}+2^{25}+2^{31}$	$2^{25}+2^{31}$	$-$	2^{26}	12
2	$+2^{0}+2^{6}+2^{25}+2^{31}$	$2^{25}+2^{31}$	$-$	2^{26}	17
3	$+2^{0}+2^{6}+2^{11}+2^{25}+2^{31}$	$2^{0}+2^{6}+2^{25}+2^{31}$	$-$	$2^{0}+2^{6}+2^{26}$	22
4	$-2^{0}+2^{1}, -2^{6}\ldots-2^{10}$ $+2^{12}-2^{16}-2^{17}-2^{18}$ $+2^{19}-2^{22}+2^{23}+2^{25}$ $-2^{28}+2^{29}-2^{31}$	$2^{0}-2^{6}+2^{9}-2^{11}$ $+2^{13}+2^{17}+2^{19}+2^{22}$ $+2^{25}-2^{29}+2^{31}$	2^{31}	$2^{1}-2^{6}+2^{9}-2^{11}$ $+2^{13}+2^{17}+2^{19}+2^{22}$ $+2^{26}-2^{29}+2^{31}$	7
5	$-2^{0}+2^{2}+2^{4}-2^{5}+2^{8}+2^{11}$ $+2^{13}+2^{14}+2^{15}-2^{16}+2^{17}$ $+2^{18}-2^{19}-2^{20}+2^{21}+2^{22}$ $-2^{24}+2^{27}-2^{28}+2^{30}+2^{31}$	$2^{1}-2^{5}+2^{11}-2^{14}$ $+2^{18}-2^{21}+2^{24}+2^{30}$	$-$	$-2^{0}+2^{2}+2^{5}+2^{11}$ $-2^{14}+2^{18}-2^{21}$ $-2^{24}+2^{26}-2^{30}$	12
6	$-2^{0}\ldots-2^{3}+2^{4}, -2^{5}-2^{6}$ $+2^{7}+2^{8}+2^{10}, -2^{12}\ldots-2^{18}$ $+2^{19}+2^{20}-2^{22}+2^{26}-2^{28}$	$-2^{0}-2^{3}-2^{5}+2^{9}$ $+2^{11}-2^{13}-2^{20}-2^{23}$ $+2^{26}-2^{28}+2^{31}$	$-$	$-2^{3}+2^{5}+2^{9}-2^{12}$ $-2^{20}-2^{23}-2^{25}-2^{27}$	17
7	$+2^{0}-2^{27}-2^{29}$	$2^{0}+2^{3}+2^{5}+2^{7}+2^{10}$ $+2^{12}-2^{14}+2^{16}-2^{19}$ $-2^{22}-2^{27}-2^{29}+2^{31}$	$-$	$2^{1}+2^{3}-2^{5}+2^{8}$ $-2^{10}-2^{13}+2^{17}-2^{19}$ $+2^{25}+2^{27}-2^{29}$	22
8	$-2^{3}+2^{7}-2^{9}+2^{15}+2^{17}$ $-2^{19}+2^{23}+2^{25}+2^{28}$	$-2^{0}+2^{4}+2^{9}+2^{13}$ $+2^{15}+2^{17}+2^{19}$ $+2^{23}-2^{25}-2^{29}$	$-$	$2^{1}-2^{8}-2^{10}+2^{12}$ $+2^{15}-2^{19}-2^{21}-2^{25}$ $-2^{27}+2^{29}+2^{31}$	7
9	$-2^{0}+2^{2}-2^{6}-2^{22}-2^{24}$	$2^{7}-2^{9}+2^{28}$	$-$	$2^{0}+2^{5}-2^{7}+2^{10}$ $+2^{12}+2^{20}-2^{22}+2^{26}$	12
10	$+2^{12}+2^{17}-2^{19}$	$2^{7}-2^{12}$	$-$	$2^{0}+2^{7}-2^{12}-2^{27}-2^{29}$	17
11	$-2^{14}-2^{18}+2^{24}-2^{29}$	-2^{24}	-2^{15}	$-2^{3}+2^{7}-2^{9}+2^{17}-2^{19}$ $-2^{23}+2^{25}+2^{28}$	22

12	$+2^{7}-2^{9}+2^{13}-2^{24}-2^{31}$	–	–	$-2^{0}+2^{2}-2^{6}-2^{22}-2^{24}$	7
13	$-2^{24}-2^{29}$	–	–	$2^{12}+2^{17}-2^{19}$	12
14	-2^{31}	$-2^{24}-2^{29}$	2^{31}	$-2^{14}-2^{18}+2^{30}$	17
15	$-2^{3}+2^{15}$	$-2^{24}+2^{31}$	–	$2^{7}-2^{9}+2^{13}-2^{25}$	22
16	$+2^{29}-2^{31}$	2^{29}	–	-2^{24}	5
17	$+2^{31}$	2^{31}	–	–	9
18	-2^{31}	–	-2^{15}	-2^{3}	14
19	$-2^{17}-2^{31}$	2^{31}	–	2^{29}	20
20	-2^{31}	2^{31}	–	–	5
21	-2^{31}	2^{31}	–	–	9
22	-2^{31}	2^{31}	–	-2^{17}	14
23	–	–	2^{31}	–	20
24	–	2^{31}	–	–	5
25	–	–	2^{31}	–	9

$26-33$	–	–	–	–	·
34	–	–	-2^{15}	-2^{15}	16
35	$\delta Q_{35}=2^{31}$	2^{31}	2^{31}	–	23
36	$\delta Q_{36}=2^{31}$	–	–	–	4
37	$\delta Q_{37}=2^{31}$	2^{31}	2^{31}	–	11
$38-49$	$\delta Q_{t}=2^{31}$	2^{31}	–	–	·
50	$\delta Q_{50}=2^{31}$	–	2^{31}	–	15
$51-59$	$\delta Q_{t}=2^{31}$	2^{31}	–	–	·
60	$\delta Q_{60}=2^{31}$	–	2^{31}	–	6
61	$\delta Q_{61}=2^{31}$	2^{31}	-2^{15}	-2^{15}	10
62	$\delta Q_{62}=2^{31}-2^{25}$	2^{31}	–	–	15
63	$\delta Q_{63}=2^{31}-2^{25}$	2^{31}	–	–	21
64	$\delta Q_{64}=2^{31}-2^{25}$	×	×	×	×

충분 조건 (Sufficient condition)

충분 조건은 $Q_t(t=-3\sim64)$에 대한 조건으로, Q_t가 조건을 만족할 경우, 차분 경로에서 제시된 Q_t의 차분을 가질 확률이 높아진다. 충분 조건을 만족하는 Q_t을 구한 다음, 아래 식을 이용해 역으로 M_t를 구할 수 있다. MD5 Compress 함수에서 step 0~15에서는 W로 M[0]~M[15]를 사용하기 때문에, step 0~15에서의 항들을 이용해 M[t] (t = 0~15)를 역으로 계산할 수 있다.

$$M_t = RR(Q_{t+1} - Q_t, RC_t) - f_t(Q_t, Q_{t-1}, Q_{t-2}) - Q_{t-3} - AC_t$$

M[0], M[1], ..., M[15]를 구한 다음, 위의 차분 경로 표에 제시된 δW을 이용해 M'[0], M'[1], ..., M'[15]를 구할 수 있다. 이 과정은 첫번째, 두번째 메시지 블록 둘 다에 대해 적용될 수 있다. 충분 조건을 만족하게 되면 차분 경로를 만족할 확률이 높은 것일 뿐, 위 차분 경로를 반드시 따라가는 것은 아니기 때문에, 충분 조건을 만족하는 Q를 찾은 후에는 Q와 Q'의 차분이 차분 경로를 따라가는지 확인해야한다.

Table 4-1: Sufficient bitconditions.

Symbol	condition on $Q_t[i]$	direct/indirect
.	none	direct
0	$Q_t[i] = 0$	direct
1	$Q_t[i] = 1$	direct
^	$Q_t[i] = Q_{t-1}[i]$	indirect
!	$Q_t[i] = \overline{Q_{t-1}[i]}$	indirect

충분 조건 표에서의 "."는 해당 비트가 0 또는 1 둘 중 아무거나 되어도 상관 없다는 뜻이고, "^"은 Q[t]의 i번째 비트가 이전 Q인 Q[t-1]의 i번째 비트와 같아야 한다는 것이고, "!"은 Q[t]의 i번째 비트가 Q[t-1]의 i번째 비트를 NOT(~, 비트 반전) 연산한 값과 같아야 한다는 것이다.

아래는 [3]에서의 Table B-3로, 첫번째 블록 연산에서의 충분 조건이다.

t	Conditions on Q_t: $b_{31} \dots b_0$	#
3	....0... ....0... .0......	3
4	1....... 0^^^1^^^ ^^^^1^^^ ^011....	19 + 2
5	1000100. 01..0000 00000000 0010.1.1	22 + 5
6	0000001^ 01111111 10111100 0100^0^1	32
7	00000011 11111110 11111000 00100000	32
8	00000001 1..10001 0.0.0101 01000000	28
9	11111011 ...10000 0.1^1111 00111101	28
10	0111.... 0..11111 1101...0 01....00	17 + 2
11	0010.... ....0001 1100...0 11....10	15 + 2
12	000...^^ ....1000 0001...1 0.......	14 + 1
13	01....01 ....1111 111....0 0...1...	14
14	0.0...00 ....1011 111....1 1...1...	14
15	0.1...01 ....0 1....... ....0...	6 + 1
16	0!1..... ......!.	2 + 2
17	0!..... ......0. ^....... ....^...	4 + 1
18	0.^..... ......1.	3
19	0....... ......0.	2
20	0....... .....!..	1 + 1
21	0....... ......^.	2
22	0.......	1
23	0.......	1
24	1.......	1

$25-45$	....	0
46	I.....	0
47	J.....	0
48	I.....	1
49	J.....	1
50	K.....	1
51	J.....	1
52	K.....	1
53	J.....	1
54	K.....	1
55	J.....	1
56	K.....	1
57	J.....	1
58	K.....	1
59	J.....	1
60	I.....	1
61	J.....	1
62	I.....	1
63	J.....	1
64	....	0

Note: $I, J, K \in \{0, 1\}$ and $K = \overline{I}$.

I로 표시된 부분은 I로 표시된 비트끼리 같은 값을 가진다는 의미이다. K는 I를 반전한 값을 가져야한다.

아래는 [3]에서의 Table B-3로, 첫번째 블록 연산에서의 충분 조건이다.

t	Conditions on Q_t: $b_{31} \dots b_0$	#
−2	......0.	(1)
−1	^.....0. 1	(3)
0	^.....0. .... .0.....1	(4)
	Total # IV conditions for 1st block	(8)
1	^.....0. ....00.. .0!....0	7
2	^.11..0. .1..0110 ...00100 00.....0	17
3	^001..00 ^0111100 00110010 1011.0^0	29
4	!101^001 01010111 11101111 11010001	32
5	.00101.1 00011001 00000000 10101011	31
6	.0110011 01000111 11110000 01101111	31
7	01111001 1110101. 1001111. 01..0110	28
8	1.100101 01.01000 0001.011 01101101	29
9	..111.01 01..0.0. 0..0..1. 111100.1	19
10	1011..10 10..1^0. 1^.0..1. 00..10.0	20
11	111....0 .1..010! .1^0..1. 01...1.1	17
12	100....1 ....101. .001..1. 0.......	12
13	011....1 .....1.. 110...0. 0...1...	11
14	111..... .... 1.1...1. 1...1...	8
15	101....0 ....! 0....... ....1...	7
16	100..... ......!.	4
17	0....... ......0. ^....... ....^...	4
18	1.^..... ......1.	3
19	1....... ......1.	2
20	1....... .....!..	2
21	1....... ......^.	2
22	1.......	1
23	0.......	1
24	1.......	1

$25-45$	....	
46	I.....	0
47	J.....	0
48	I.....	1
49	J.....	1
50	K.....	1
51	J.....	1
52	K.....	1
53	J.....	1
54	K.....	1
55	J.....	1
56	K.....	1
57	J.....	1
58	K.....	1
59	J.....	1
60	I.....	1
61	J.....	1
62	I.....	1
63	J.....	1
64	....	0

Tunneling

MD5 구조 상, 공격자가 위에 제시된 Q에 대한 충분 조건(Sufficient Condition)을 만족시키기 위해서 건드릴 수 있는 부분은 M[0], M[1], ..., M[15]이다. MD5에서는 step 0일 때, W로 M[0], step 1일 때, W로 M[1], ..., step 15일 때, W로 M[15]를 사용하기 때문에, step 0~15에서의 W인 M[0], ..., M[15]들을 잘 선택하여, step 15 이후, step 16~63에서의 Q가 충분 조건을 만족하도록 해야 한다.

[3]에서 제시한 알고리즘을 보면, step 0~15에서 충분 조건을 만족하는 Q를 임의로 선택한다(임의의 Q[1]~Q[16]을 선택). 아래 식을 이용해 M[0]~M[15]를 구한다.

$$M_t = RR(Q_{t+1} - Q_t, RC_t) - f_t(Q_t, Q_{t-1}, Q_{t-2}) - Q_{t-3} - AC_t$$

그리고, 구한 M[0]~M[15]를 이용하여 step 16~63에서의 Q[17]~Q[64]를 계산한다. 이미 M[0]~M[15]는 결정되어 있기 때문에, Q[17]~Q[64]가 충분조건을 만족시키는 것은 확률적인 문제가 된다. 만약 충분 조건을 만족시키지 못한다면, Q[1]~Q[16]을 새로 선택하여 다시 시도해야 한다.

$$M_t = RR(Q_{t+1} - Q_t, RC_t) - f_t(Q_t, Q_{t-1}, Q_{t-2}) - Q_{t-3} - AC_t$$

[2]에서 등장한 Tunneling을 적용하게 된다면, Q[17]~Q[64]에서 충분 조건을 만족하지 못하는 것이 나왔을 때, 다시 새로운 Q[1]~Q[16]을 선택하고, 새로운 Q[17]~Q[64]들이 충분 조건을 만족하는지 확인하는 방법 대신, step 0~15에서의 대부분의 Q는 유지한 상태에서, 일부 Q만 바꿔 새로

운 Q[17]~Q[64]를 구하는 방법을 사용할 수 있다. 만약 Tunneling을 적용하지 않는다면, 이전 Q와 이후 Q는 서로 연관되어 있기 때문에, 이전 단계에서 일부 Q가 변하게 된다면, 이후 Q들 역시 다 영향을 받아 변하기 때문에, 상당히 많은 계산이 필요하다. 하지만 Tunneling을 사용한다면, 변경된 Q 이후의 Q의 변화를 최소화할 수 있게 된다. Round 1, 즉 Q[1]~Q[16] 중 일부를 바꾸더라도, 바뀐 Q를 제외한 나머지 Q[1]~Q[16]은 영향을 받지 않도록 만들 수 있다. 이는 Tunnel이 Round 1에서 사용하는 비선형함수인 f의 특성을 이용하기 때문에 가능한 일이다. 아래는 [3]에서의 Table 5-2로, Tunnel을 위한 비트 조건을 나타낸 것이다.

Table 5-2: Tunnels for collision finding

Tunnel	Required bitconditions	First affected Q_t, $t > 16$
$\mathcal{T}(Q_9, m_9)$	$Q_{10}[i] = 0 \wedge Q_{11}[i] = 1$	Q_{25}
$\mathcal{T}(Q_4, m_4)$	$Q_5[i] = 0 \wedge Q_6[i] = 1$	Q_{24}
$\mathcal{T}(Q_9, m_{10})$	$Q_{10}[i] = 1 \wedge Q_{11}[i] = 1$	Q_{22}
$\mathcal{T}(Q_{10}, m_{10})$	$Q_{11}[i] = 0$	Q_{22}
$\mathcal{T}(Q_4, m_5)$	$Q_5[i] = 1 \wedge Q_6[i] = 1$	Q_{21}
$\mathcal{T}(Q_5, m_5)$	$Q_6[i] = 0$	Q_{21}

T(Q, m)에서 Q는 Tunnel에서 변경할 Q를 의미하고, m은 변경된 Q의 영향을 받는 m을 의미한다. First affected Q는 변경된 Q의 영향을 받은 m이 변함에 따라, 영향을 받기 시작하는 첫번째 Q를 의미한다. T(Q[9], m[9])에서는 step 24~step 63부터 Tunneling의 영향을 받기 때문에, Q[25]~Q[64]만을 다시 계산하여 충분 조건을 만족하는지 확인하면 된다. [3]에서는 Q[25]와 같이 검증하기 시작하는 시점을 POV(Point Of Verification)이라 부른다. Q[9]이 변하더라도, 그 이후인 Q[10]~Q[24]는 영향을 받지 않았고, Q[25] 부터 영향을 받기 시작하여 Q[25]~Q[64]만 다시 계산하면 되기 때문에, Tunneling을 사용하지 않는 경우보다 훨씬 빠르고 효율적으로 충분 조건을 만족하는 Q[1]~Q[64]를 찾아낼 수 있게 된다.

이번 구현에서는 [2]에서 제시한 Tunnel 중 Tunnel Q4, Tunnel Q9,9, Tunnel Q9,10, Tunnel Q10,10을 사용한다.

사용한 알고리즘

아래 두 알고리즘은 각각 [3]에서의 Algorithm 5.1, Algorithm 5.2이다.

Block 1 search 알고리즘
1. 충분 조건을 만족하는 임의의 $Q_1, Q_3, \ldots, Q_{16}$을 선택한다. 2. 선택한 $Q_1, Q_3, \ldots, Q_{16}$을 이용해 $m_0, m_6, \ldots, m_{15}$을 구한다. 3. $Q_{17}, \ldots, Q_{21}$이 조건을 만족할 때까지 아래를 반복: (a) 충분 조건을 만족하는 Q_{17} 선택 (b) m_1을 계산

(c) Q_2 와 m_2, m_3, m_4, m_5을 계산

(d) $Q_{18}, \dots, Q_{21}$ 계산, $Q_{17}, \dots, Q_{21}$이 충분조건을 만족하면 반복 문 종료

4. $T(Q_9, m_9), T(Q_9, m_{10}), T(Q_{10}, m_{10})$을 이용해서 Q_9, Q_{10}을 충분조건을 만족하도록 바꿔가며 아래를 반복:
   (a) $m_8, m_9, m_{10}, m_{12}, m_{13}$을 계산
   (b) $Q_{22}, \dots, Q_{64}$을 계산
   (c) $Q_{22}, \dots, Q_{64}$이 충분 조건을 만족하는 지 확인, 만족한다면, 두번째 메시지 블록에서의 차분 경로를 위한 IHV 조건 확인, $T_{22}[17] = 0, T_{34}[15] = 0$을 만족하는지 확인
5. 모든 가능한 Q_9, Q_{10}에 대해 4번의 (c)가 충족되지 않는다면 1번으로 돌아가서 다시 시도

Block 2 search 알고리즘

1. 충분 조건을 만족하는 임의의 $Q_2, \dots, Q_{16}$을 선택한다.
2. 선택한 $Q_2, \dots, Q_{16}$을 이용해 $m_5, \dots, m_{15}$을 구한다.
3. $Q_{17}, \dots, Q_{21}$이 조건을 만족할 때까지 아래를 반복:
   (a) 충분 조건을 만족하는 Q_1 선택
   (b) m_1을 계산
   (c) $m_0, \dots, m_4$을 계산
   (d) $Q_{17}, \dots, Q_{21}$ 계산, $Q_{17}, \dots, Q_{21}$이 충분조건을 만족하면 반복 문 종료
4. $T(Q_9, m_9), T(Q_9, m_{10}), T(Q_{10}, m_{10})$을 이용해서 Q_9, Q_{10}을 충분조건을 만족하도록 바꿔가며 아래를 반복:
   (a) $m_8, m_9, m_{10}, m_{12}, m_{13}$을 계산
   (b) $Q_{22}, \dots, Q_{64}$을 계산
   (c) $Q_{22}, \dots, Q_{64}$이 충분 조건을 만족하는 지 확인, 만족한다면, $T_{22}[17] = 1, T_{34}[15] = 1$을 만족하는지 확인
   (e) 모든 가능한 Q_9, Q_{10}에 대해 4번의 (c)가 충족되지 않는다면 1번으로 돌아가서 다시 시도

IHV 검증하기

아래는 두번째 메시지 블록의 차분 경로 표 중 IHV 차분이다. 첫번째 메시지 블록의 연산 결과가 두번째 메시지 블록의 IHV을 구성하는데 사용되기 때문에, 두번째 메시지 블록의 Initial state가 아래에 제시된 차분을 가져야한다.

t	ΔQ_t (BSDR of δQ_t)	δF_t	δw_t	δT_t	RC_t
-3	$+2^{31}$	$\times$	$\times$	$\times$	$\times$
-2	$+2^{25}+2^{31}$	$\times$	$\times$	$\times$	$\times$
-1	$+2^{25}+2^{31}$	$\times$	$\times$	$\times$	$\times$
0	$+2^{25}+2^{31}$	$2^{25}+2^{31}$	$-$	2^{25}	7

두번째 메시지 블록의 initial state는 다음과 같이 구할 수 있다.

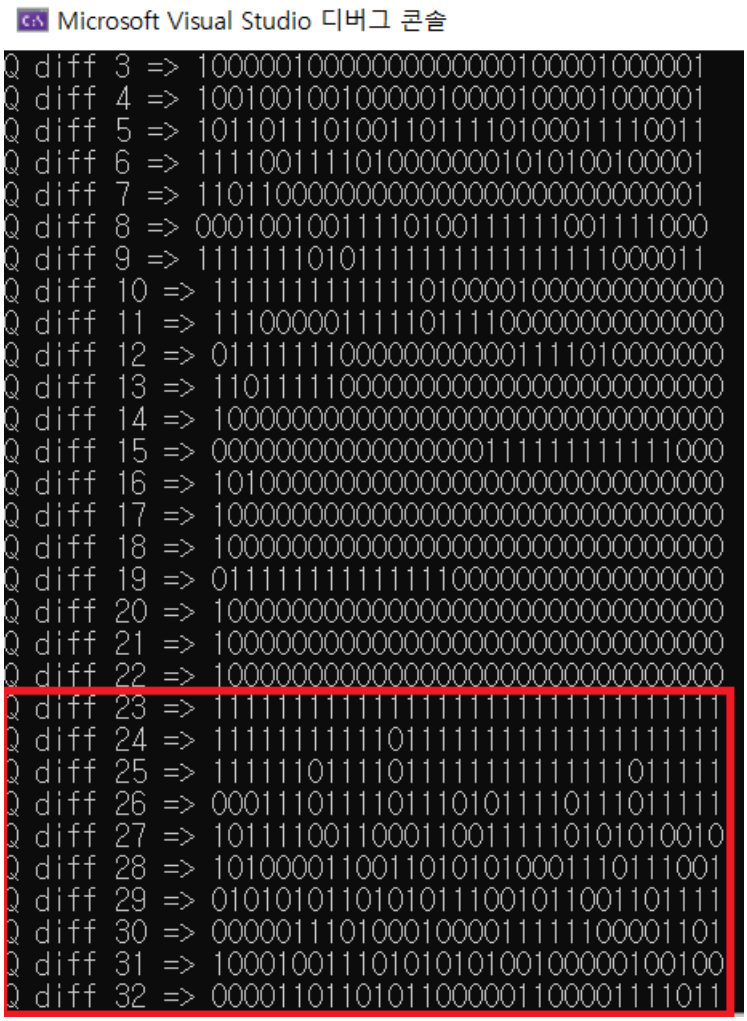
Q[-3] = 0x67452301 + 1번 블록에서의 Q[61]
Q[0] = 0xefcdab89 + Q[64]
Q[-1] = 0x98badcfe + Q[63]
Q[-2] = 0x10325476 + Q[62]

위 4개의 Q[-3], …, Q[0]을 구한 다음, 4개가 모두 충분 조건을 만족하는지 확인하면 된다.

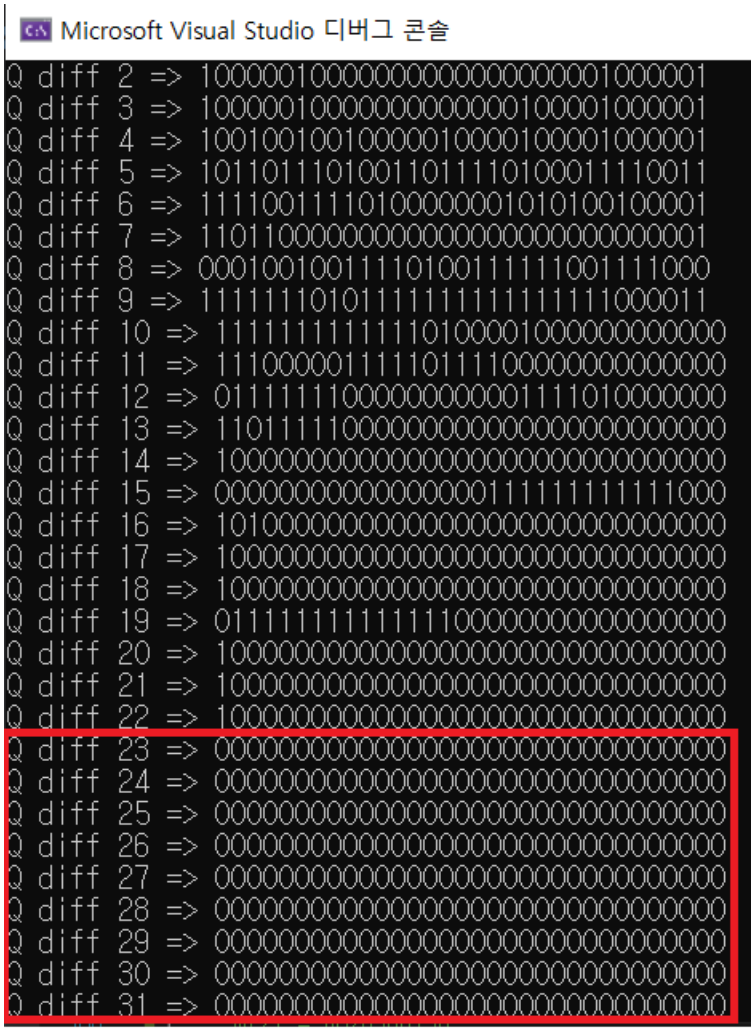
T[22]의 17번째 비트, T[34]의 15번째 비트 검증

[3]에서는 [1]과 다른 3개의 두번째 메시지 블록의 차분 경로를 제시하면서, T22, T34 조건에 대한 구체적인 언급을 하지 않았는데, 이번 구현에서 사용하는 nr.3 두번째 메시지 블록 차분 경로는 step 22에서 [1]에서의 경로와 다른 차분을 가지기 때문에 T22와 T34에 대한 조건이 달라야 한다. [1]의 첫번째, 두번째 메시지 블록에서의 T22, T34에 대한 조건인 $T_{22}[17] = 0, T_{34}[15] = 0$을 [3]에서 제시한 nr.3 두번째 블록에 적용하게 되면, step 22에서 원하는 차분을 얻어낼 수 없다.

아래 이미지를 통해, $T_{22}[17] = 0, T_{34}[15] = 0$의 조건을 두번째 블록에 적용하게 되면, Q[22] 이후로 잘못된 차분이 등장함을 확인할 수 있다(원하는 Q[23]의 차분은 0인데, 나온 차분은 2^32-1).



대신에, $T_{22}[17] = 1, T_{34}[15] = 1$ 조건을 적용하게 되면, Q[22] 이후로 정상적인 차분이 나옴을 확인할 수 있었다.



구현 결과

아래는 위 과정들을 통해 찾은 같은 해시 값을 갖지만, 서로 다른 내용의 128byte 메시지 2개이다.

1.

39ae591d0d0ddacd77f68010d427411d2a25b3821ba8737af12eb2152e676fe641ed2305fbf3b3e18bada47d2b9b42120cd9415ceff2bd20d4a37732a3fe3df8d1a7efc761c3a101798d7479b7fddf5626ff286b096636edc91a06c9ecc601f23ec646d7f0d11f5d888341b2be18daf23616d53f3d68df5a16cfc44816555e48

2.

39ae591d0d0ddacd77f68010d427411d2a25b3021ba8737af12eb2152e676fe641ed2305fbf3b3e18bada47d2b1b43120cd9415ceff2bd20d4a377b2a3fe3df8d1a7efc761c3a101798d7479b7fddf5626ff28eb096636edc91a06c9ecc601f23ec646d7f0d11f5d888341b2be98d9f23616d53f3d68df5a16cfc4c816555

e48

1, 2의 MD5 해시 값=>7edbf2a913ded4286184100be8ca4806

```
89ae591d0d0ddacd77f68010d427411d2a25b3821ba8737af12eb2152e676fe641ed2305fbf3b3e18bada47d2b9b42120cd9415ceff2bd20d4a37732
a3fe3df8d1a7efc761c3a101798d7479b7fddf5626ff286b096636edc91a06c9ecc601f23ec646d7f0d11f5d888341b2be18daf23616d53f3d68df5a
16cfc44816555e48
128
89 ae 59 1d 0d 0d da cd 77 f6 80 10 d4 27 41 1d 2a 25 b3 82 1b a8 73 7a f1 2e b2 15 2e 67 6f e6 41 ed 23 05 fb f3 b3 e1
8b ad a4 7d 2b 9b 42 12 0c d9 41 5c ef f2 bd 20 d4 a3 77 32 a3 fe 3d f8
d1 a7 ef c7 61 c3 a1 01 79 8d 74 79 b7 fd df 56 26 ff 28 6b 09 66 36 ed c9 1a 06 c9 ec c6 01 f2 3e c6 46 d7 f0 d1 1f 5d
68 83 41 b2 be 18 da f2 36 16 d5 3f 3d 68 df 5a 16 cf c4 48 16 55 5e 48
2
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00
=> 7edbf2a913ded4286184100be8ca4806
89ae591d0d0ddacd77f68010d427411d2a25b3021ba8737af12eb2152e676fe641ed2305fbf3b3e18bada47d2b1b43120cd9415ceff2bd20d4a377b2
a3fe3df8d1a7efc761c3a101798d7479b7fddf5626ff28eb096636edc91a06c9ecc601f23ec646d7f0d11f5d888341b2be98d9f23616d53f3d68df5a
16cfc4c816555e48
128
89 ae 59 1d 0d 0d da cd 77 f6 80 10 d4 27 41 1d 2a 25 b3 02 1b a8 73 7a f1 2e b2 15 2e 67 6f e6 41 ed 23 05 fb f3 b3 e1
8b ad a4 7d 2b 1b 43 12 0c d9 41 5c ef f2 bd 20 d4 a3 77 b2 a3 fe 3d f8
d1 a7 ef c7 61 c3 a1 01 79 8d 74 79 b7 fd df 56 26 ff 28 eb 09 66 36 ed c9 1a 06 c9 ec c6 01 f2 3e c6 46 d7 f0 d1 1f 5d
68 83 41 b2 be 98 d9 f2 36 16 d5 3f 3d 68 df 5a 16 cf c4 c8 16 55 5e 48
2
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00
=> 7edbf2a913ded4286184100be8ca4806
```

공격 시나리오 구현

위에서 구한 충돌이 발생하는 메시지들을 이용해, 서로 다르게 작동하지만, 같은 해시 값을 갖는 프로그램 두 개를 구현할 수 있다.

benign_exe.c 코드

```c
#include <stdio.h>
unsigned char a[32 + 128] = {
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x39, 0xae, 0x59, 0x1d, 0x0d, 0x0d, 0xda, 0xcd, 0x77, 0xf6, 0x80, 0x10, 0xd4, 0x27, 0x41, 0x1d,
0x2a, 0x25, 0xb3, 0x82, 0x1b, 0xa8, 0x73, 0x7a, 0xf1, 0x2e, 0xb2, 0x15, 0x2e, 0x67, 0x6f, 0xe6,
0x41, 0xed, 0x23, 0x05, 0xfb, 0xf3, 0xb3, 0xe1, 0x8b, 0xad, 0xa4, 0x7d, 0x2b, 0x9b, 0x42, 0x12,
0x0c, 0xd9, 0x41, 0x5c, 0xef, 0xf2, 0xbd, 0x20, 0xd4, 0xa3, 0x77, 0x32, 0xa3, 0xfe, 0x3d, 0xf8,
0xd1, 0xa7, 0xef, 0xc7, 0x61, 0xc3, 0xa1, 0x01, 0x79, 0x8d, 0x74, 0x79, 0xb7, 0xfd, 0xdf, 0x56,
0x26, 0xff, 0x28, 0x6b, 0x09, 0x66, 0x36, 0xed, 0xc9, 0x1a, 0x06, 0xc9, 0xec, 0xc6, 0x01, 0xf2,
0x3e, 0xc6, 0x46, 0xd7, 0xf0, 0xd1, 0x1f, 0x5d, 0x88, 0x83, 0x41, 0xb2, 0xbe, 0x18, 0xda, 0xf2,
0x36, 0x16, 0xd5, 0x3f, 0x3d, 0x68, 0xdf, 0x5a, 0x16, 0xcf, 0xc4, 0x48, 0x16, 0x55, 0x5e, 0x48
};
unsigned char b[32 + 128] = {
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x39, 0xae, 0x59, 0x1d, 0x0d, 0x0d, 0xda, 0xcd, 0x77, 0xf6, 0x80, 0x10, 0xd4, 0x27, 0x41, 0x1d,
0x2a, 0x25, 0xb3, 0x82, 0x1b, 0xa8, 0x73, 0x7a, 0xf1, 0x2e, 0xb2, 0x15, 0x2e, 0x67, 0x6f, 0xe6,
0x41, 0xed, 0x23, 0x05, 0xfb, 0xf3, 0xb3, 0xe1, 0x8b, 0xad, 0xa4, 0x7d, 0x2b, 0x9b, 0x42, 0x12,
0x0c, 0xd9, 0x41, 0x5c, 0xef, 0xf2, 0xbd, 0x20, 0xd4, 0xa3, 0x77, 0x32, 0xa3, 0xfe, 0x3d, 0xf8,
0xd1, 0xa7, 0xef, 0xc7, 0x61, 0xc3, 0xa1, 0x01, 0x79, 0x8d, 0x74, 0x79, 0xb7, 0xfd, 0xdf, 0x56,
0x26, 0xff, 0x28, 0x6b, 0x09, 0x66, 0x36, 0xed, 0xc9, 0x1a, 0x06, 0xc9, 0xec, 0xc6, 0x01, 0xf2,
0x3e, 0xc6, 0x46, 0xd7, 0xf0, 0xd1, 0x1f, 0x5d, 0x88, 0x83, 0x41, 0xb2, 0xbe, 0x18, 0xda, 0xf2,
0x36, 0x16, 0xd5, 0x3f, 0x3d, 0x68, 0xdf, 0x5a, 0x16, 0xcf, 0xc4, 0x48, 0x16, 0x55, 0x5e, 0x48
};
int main()
{
    int size = 100;
    int flag = 1;
    for (int i = 0; i < size; i++)
        if (a[i] != b[i])
            flag = 0;
    if (flag)
        printf("이 프로그램은 안전합니다.₩n");
    else
        printf("모든 시스템 파일 삭제!₩n");
}
```

malicious_exe.c 코드

```c
#include <stdio.h>
unsigned char a[32+128] = {
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x39, 0xae, 0x59, 0x1d, 0x0d, 0x0d, 0xda, 0xcd, 0x77, 0xf6, 0x80, 0x10, 0xd4, 0x27, 0x41, 0x1d,
0x2a, 0x25, 0xb3, 0x82, 0x1b, 0xa8, 0x73, 0x7a, 0xf1, 0x2e, 0xb2, 0x15, 0x2e, 0x67, 0x6f, 0xe6,
0x41, 0xed, 0x23, 0x05, 0xfb, 0xf3, 0xb3, 0xe1, 0x8b, 0xad, 0xa4, 0x7d, 0x2b, 0x9b, 0x42, 0x12,
0x0c, 0xd9, 0x41, 0x5c, 0xef, 0xf2, 0xbd, 0x20, 0xd4, 0xa3, 0x77, 0x32, 0xa3, 0xfe, 0x3d, 0xf8,
0xd1, 0xa7, 0xef, 0xc7, 0x61, 0xc3, 0xa1, 0x01, 0x79, 0x8d, 0x74, 0x79, 0xb7, 0xfd, 0xdf, 0x56,
0x26, 0xff, 0x28, 0x6b, 0x09, 0x66, 0x36, 0xed, 0xc9, 0x1a, 0x06, 0xc9, 0xec, 0xc6, 0x01, 0xf2,
0x3e, 0xc6, 0x46, 0xd7, 0xf0, 0xd1, 0x1f, 0x5d, 0x88, 0x83, 0x41, 0xb2, 0xbe, 0x18, 0xda, 0xf2,
0x36, 0x16, 0xd5, 0x3f, 0x3d, 0x68, 0xdf, 0x5a, 0x16, 0xcf, 0xc4, 0x48, 0x16, 0x55, 0x5e, 0x48
};
unsigned char b[32+128] = {
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
0x39, 0xae, 0x59, 0x1d, 0x0d, 0x0d, 0xda, 0xcd, 0x77, 0xf6, 0x80, 0x10, 0xd4, 0x27, 0x41, 0x1d,
0x2a, 0x25, 0xb3, 0x02, 0x1b, 0xa8, 0x73, 0x7a, 0xf1, 0x2e, 0xb2, 0x15, 0x2e, 0x67, 0x6f, 0xe6,
0x41, 0xed, 0x23, 0x05, 0xfb, 0xf3, 0xb3, 0xe1, 0x8b, 0xad, 0xa4, 0x7d, 0x2b, 0x1b, 0x43, 0x12,
0x0c, 0xd9, 0x41, 0x5c, 0xef, 0xf2, 0xbd, 0x20, 0xd4, 0xa3, 0x77, 0xb2, 0xa3, 0xfe, 0x3d, 0xf8,
0xd1, 0xa7, 0xef, 0xc7, 0x61, 0xc3, 0xa1, 0x01, 0x79, 0x8d, 0x74, 0x79, 0xb7, 0xfd, 0xdf, 0x56,
0x26, 0xff, 0x28, 0xeb, 0x09, 0x66, 0x36, 0xed, 0xc9, 0x1a, 0x06, 0xc9, 0xec, 0xc6, 0x01, 0xf2,
0x3e, 0xc6, 0x46, 0xd7, 0xf0, 0xd1, 0x1f, 0x5d, 0x88, 0x83, 0x41, 0xb2, 0xbe, 0x98, 0xd9, 0xf2,
0x36, 0x16, 0xd5, 0x3f, 0x3d, 0x68, 0xdf, 0x5a, 0x16, 0xcf, 0xc4, 0xc8, 0x16, 0x55, 0x5e, 0x48
};
int main()
{
    int size = 100;
    int flag = 1;
    for (int i = 0; i < size; i++)
        if (a[i] != b[i])
            flag = 0;
    if (flag)
        printf("이 프로그램은 안전합니다.₩n");
    else
        printf("모든 시스템 파일 삭제!₩n");
}
```

benign_exe.c에서 배열 a와 b는 같은 내용을 가지고 있지만, malicious_exe.c에서 배열 a와 b는 다른 내용을 가지고 있다. 따라서, 각각을 실행했을 때 실행 결과는 다음과 같다.

```
[redacted]₩MD5_attack₩program>benign.exe
이 프로그램은 안전합니다.

[redacted]₩MD5_attack₩program>malicious.exe
모든 시스템 파일 삭제!
```

benign.exe와 malicious.exe의 MD5 해시 값이 같게 나오는 것을 확인할 수 있었다.

```
Microsoft Visual Studio 디버그 콘솔
Benign program MD5 hash:
=> 651c2364fe4cc7b4a413a9bab39ba1ba
Malicios program MD5 hash:
=> 651c2364fe4cc7b4a413a9bab39ba1ba
```

소스 코드

소스 코드 저장소: https://github.com/sepaper/MD5_Collision_Attack

결론

위 과정들을 통해 같은 해시 값을 갖지만, 서로 다른 128byte 메시지를 찾는데 성공하였다. Intel® Core™ i5-8265U 1.60GHz CPU를 탑재한 노트북에서 첫번째 블록을 찾는 데 걸린 시간은 대략 3분, 두번째 블록을 찾는 데 걸린 시간은 대략 5분 정도였고, 프로그램이 작동하는 동안 사용한 CPU 점유율은 25%~30%였던 점을 감안하면, 코드를 더 최적화하고, 더 좋은 CPU를 사용한다면 [2]와 [3]에서 제시했던 것과 마찬가지로, 몇 초 만에 충돌을 찾아낼 수 있을 것이라고 생각한다.

참고문헌

[1] Xiaoyun Wang and Hongbo Yu, How to break MD5 and other hash functions, EUROCRYPT 2005 (Ronald Cramer, ed.), LNCS, vol. 3494, Springer, 2005, pp. 19–35.

[2] Vlastimil Klima, Tunnels in hash functions: MD5 collisions within a minute, Cryptology ePrint Archive, Report 2006/105, 2006, http://eprint.iacr.org/2006/105

[3] Marc Stevens. On collisions for md5. Master's thesis, Eindhoven University of Technology, 6 2007

[4] Seed Lab - MD5 Collision Attack Lab, Wenliang Du, Syracuse University, 2018