해쉬함수 표준 | 제2부 : 해쉬함수 알고리즘 표준 (HAS-160)

TTA

한국정보통신기술협회

TTA Standard

정보통신단체표준
TTAS.KO-12.0011/R1

제정일 : 2000년 12월 20일

해쉬함수표준 - 제2부 : 해쉬함수알고리즘표준(HAS-160)

(HASH FUNCTION STANDARD - PART 2 : HASH FUNCTION ALGORITHM STANDARD(HAS-160))

TTA 한국정보통신기술협회
Telecommunications Technology Association

서 문

1. 표준의 목적

이 표준은 정보처리시스템 및 정보통신망 환경에서 임의의 길이의 비트 열을 고정된 길이(160비트)의 출력값인 해쉬코드로 압축시키는 해쉬알고리즘 표준(HAS-160)을 규정한다.

2. 참조 표준 및 권고

2.1 국제표준

ISO/IEC 10118-1(1994) : Information Technology-Security techniques Hash functions - Part I : General

ISO/IEC 10118-2(1994) : Information Technology-Security techniques Hash functions - Part 2 : Hash functions using an n-bit block cipher algorithm

ISO/IEC 10118-3(1996) : Information Technology-Security techniques Hash functions - Part 3 : Dedicated hash function

ISO/IEC 10118-4(1996) : Information Technology-Security techniques Hash functions - Part 4 : Hash functions using modular arithmetic

2.2. 국내표준 : KIS 133 : 해쉬함수 표준/기본구조

2.3. 기타 : 해당사항 없음

3. 국제표준과의 비교

3.1 국제표준과의 관련성 : 해당사항 없음

3.2 상기 국제표준 등에 대한 추가 사항 : 해당사항 없음

4. 지적재산권 관련사항 : 해당사항 없음

5. 적합인증 관련사항 : 해당사항 없음

6. 표준의 이력

판 수	제·개정일	개정판 내용
제 1 판	1998년 10월 27일	제정
제 2 판	2000년 12월 20일	개정

Preface

1. Purpose of Standard

This standard specifies HAS-160(Hash Algorithm Standard) that provides methods to compress bit strings with arbitrary lengths into a hash code with fixed lengths(160 bits).

2. Recommended Recommendations and/or Standards

2.1 International Standards

ISO/IEC 10118-1(1994) : Information Technology-Security techniques Hash functions - Part I : General

ISO/IEC 10118-2(1994) : Information Technology-Security techniques Hash functions - Part 2 : Hash functions using an n-bit block cipher algorithm

ISO/IEC 10118-3(1996) : Information Technology-Security techniques Hash functions - Part 3 : Dedicated hash function

ISO/IEC 10118-4(1996) : Information Technology-Security techniques Hash functions - Part 4 : Hash functions using modular arithmetic

2.2 Domestic Standards : KIS 133 : Hash function Standard / Framework

2.3 Other Standards : None

3. Relationship to International Standards

3.1 The Relationship of international standards : None

3.2 Additional requirements to the international standard : None

4. The statement of intellectual Property Rights : None

5. The Statement of conformance testing and certification : None

6. The history of standard

Edition	Issued date	Contents
The 1st edition	1998. 10. 27.	Established
The 2nd edition	2000. 12. 20.	Revised

목 차
CONTENTS

2000. 12. 20.

해쉬함수표준 - 제2부 : 해쉬함수알고리즘표준(HAS-160)
HASH FUNCTION STANDARD - PART 2 : HASH FUNCTION ALGORITHM STANDARD(HAS-160)

1. 개요

해쉬알고리즘은 임의의 길이의 비트 열을 고정된 길이의 출력값인 해쉬코드로 압축시키는 알고리즘으로써 다음의 성질을 만족한다 :

1. 주어진 출력에 대하여 입력값을 구하는 것이 계산상 불가능하다.
2. 주어진 입력에 대하여 같은 출력을 내는 또다른 입력을 찾아내는 것이 계산상 불가능하다.
   암호학적 응용에 사용되는 대부분의 해쉬알고리즘은 위의 두 성질뿐만 아니라 이보다 강한 충돌저항성을 지닐 것이 요구된다. 즉,
3. 같은 출력을 내는 임의의 서로 다른 두 입력 메시지를 찾는 것이 계산상 불가능하다(collision-resistance).
   암호학적 해쉬알고리즘의 충돌 저항성은 전자서명에서 송신자 외의 제 3자에 의한 문서위조를 방지하는 부인방지 서비스를 제공하기 위한 필수적인 요구조건이 된다.

해쉬알고리즘은 크게 DES와 같은 블록 암호알고리즘에 기초한 해쉬알고리즘과 전용 해쉬알고리즘으로 나눌 수 있다. 블록 암호알고리즘을 이용한 해쉬알고리즘은 이미 구현되어 사용되고 있는 블록 암호알고리즘을 사용할 수 있다는 이점이 있으나, 대부분의 블록 암호알고리즘의 속도가 그리 빠르지 않을뿐더러 이를 기본함수로 이용한 해쉬알고리즘의 경우 블록 암호알고리즘보다도 훨씬 더 속도가 떨어지므로 현재는 대부분의 응용에서 전용 해쉬알고리즘이 주로 이용된다.

대부분의 전용 해쉬알고리즘은 덧붙이기, 분할, 반복연산의 과정을 거쳐 계산된다. 임의의 길이의 메시지 X를 입력단위의 배수가 되도록 덧붙이기 하여 t개의 입력 블록 $(X_1, \cdots, X_t)$으로 분할한다. 해쉬코드는 각 블록 X_i에 대해 연쇄변수를 주어진 초기값(IV)으로 초기화한 후 압축함수를 반복적으로 적용하여 계산되며, 그 처리과정은 다음과 같이 기술할 수 있다.

$$H_0 = IV,$$

$$H_i = f(H_{i-1}, X_i),\ 1 \le i \le t,$$

$$h(X) = H_t$$

여기서 f 는 h의 압축함수이며, H_i는 단계 $i-1$과 단계 i의 중간 계산값이다.

2. 적용범위

이 표준에서 규정되는 해쉬알고리즘은 정보처리시스템 또는 정보통신망 환경에서 인증, 무결성 및 부인방지 등의 정보보호 서비스를 제공하는 전자서명에 활용할 수 있다. 또한 이 표준에 따르는 정보통신 기기 및 관련 응용 개발시 구현결과의 구현적합성 여부를 확인하는 데 이용할 수 있다.

3. 준용표준

KIS 133 : 해쉬함수 표준/기본구조

4. 정의

이 표준이 참조하고 있는 준용표준에서 정의된 다음 정의들이 이 표준에서 사용된다.

가. 덧붙이기 : 데이터 열에 부가적인 비트들을 덧붙이는 것
나. 디지털 서명 : 데이터의 수신자로 하여금 그 데이터의 발신원과 무결성을 증명하고 위조를 방지하도록 하기 위하여 암호학적으로 변환되거나 또는 원래의 데이터에 추가된 데이터
다. 충돌회피 해쉬알고리즘 : 같은 출력(해쉬코드)을 가지는 서로 다른 두 개의 입력 데이터 열을 찾는 것이 계산상 불가능한 것을 특징으로 갖는 해쉬알고리즘
라. 해쉬코드 : 해쉬알고리즘의 출력 비트열
마. 해쉬알고리즘 : 데이터 열을 다음의 두가지 성질을 만족하는 고정된 길이의 해쉬코드로 대응시키는 함수

1) 주어진 해쉬코드에 대하여 이와 동일한 해쉬코드를 생성하는 데이터 열을 찾아내는 것은 계산상 실행 불가능하다.
2) 주어진 데이터 열에 대하여 같은 해쉬코드를 생성하는 또 다른 데이터 열을 찾아내는 것은 계산상 실행 불가능하다.

바. 워드 : 32 비트 열

사. 블록 : 해쉬알고리즘에서 압축함수의 입력 비트 수 (512비트임)

5. 기호와 표기

5.1 일반적인 기호 및 표기

영대문자로 표시된 기호는 음이 아닌 정수값, 혹은 그에 상응하는 비트 열이다. 본 표준에서 사용될 기호는 다음과 같다.

$+$: 워드간의 2^{32}을 법으로 하는 덧셈 연산

$X^{\lll s}$: X를 s비트 만큼 왼쪽으로 순환 이동하는 연산

$\neg$, $\vee$, $\wedge$, $\oplus$: 각각 비트별 NOT, OR, AND, XOR 연산

H_0, H_1, H_2, H_3, H_4 : 연쇄변수

K_j : j번째 단계연산에 사용되는 32비트 상수

f_j : j번째 단계연산에 사용되는 부울함수

$s_1(j)$, $s_2(j)$: j번째 단계연산에 사용되는 순환이동의 양 (비트수)

A += B : 임의의 A, B에 대하여 A + B한 값을 A에 입력

5.2 비트열-워드간의 변환

표준 해쉬알고리즘의 내부연산은 32비트의 정수간 연산이 사용된다. 즉, 임의의 길이의 비트열을 32비트 워드열로 변환하여 처리하고 그 결과를 160비트의 비트열로 변환하여 해쉬코드로 출력하게 된다. 따라서 비트열-워드열간의 변환규칙이 필요하다. 본 표준은 다음과 같은 변환규칙(little-endian convention)을 따른다.

가. 32비트 비트열 - 워드간의 변환

32비트 비트열을 4 바이트의 문자열로 보고 첫 바이트가 워드의 최하위

바이트가 된다. 예를들면 비트열

10101101 01101011 11001001 10101110 = ad6bc9ae

은 32비트 워드로 W = aec96bad가 된다. 이는 위의 4바이트 문자열을 little-endian 컴퓨터에서 unsigned long으로 type casting한 것과 같다.

나. 임의의 길이의 비트열을 32비트 워드열로 변환하고자 할 때는 이 비트열을 바이트열로 보고 첫 4바이트를 첫 워드로, 두 번째 4바이트를 두 번째 워드로 변환하는 과정을 반복한다. 예를들면 비트열

10101101 01101011 11001001 10101110 00111111 01011001 01000110 = ad6bc9ae3f5946

은 32비트 워드열로는 aec96bad 0046593f가 된다.

다. 32비트 워드열을 비트열로 변환시킬 때는 가, 나의 역과정을 따른다.

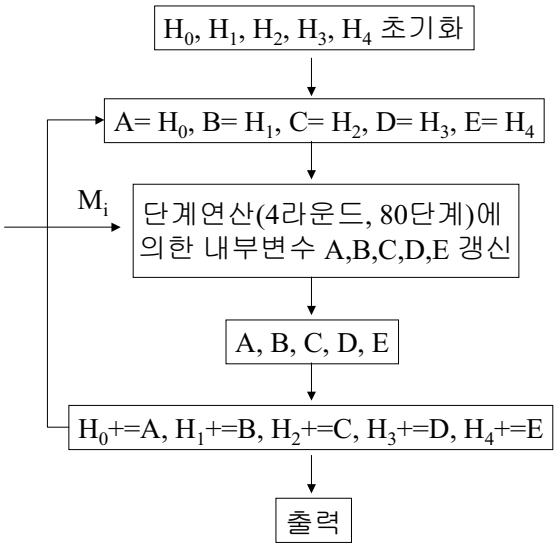
라. 64비트 정수 $Z = 2^{32}X + Y\ (0 \le X, Y < 2^{32})$는 32비트 워드열 (Y, X)로 변환된다. 즉, 하위 32비트가 첫 워드가 된다.

6. 해쉬알고리즘 기술

6.1. 전체 구조

표준 해쉬알고리즘은 임의의 길이를 가지는 입력 메시지를 512비트 블록 단위로 처리하여 160비트의 출력을 낸다. 512비트 단위 블록을 처리하는 압축함수는 모두 4 라운드, 80단계로 구성되며, 해쉬코드를 계산하는 연쇄변수는 5개이다. 또한 각 라운드에 적용될 메시지 변수의 개수는 512비트 입력블록으로부터 생성된 16워드와 이로부터 추가로 생성되는 4개의 워드를 포함하여 20개가 된다.

표준 해쉬알고리즘은 임의의 길이의 메시지 M이 입력으로 들어오면 이 M을 덧붙이기 과정을 통해 512비트의 배수로 만든 후, 512비트 블록 $M_i\ (0 \le i < t)$ 로 나눈다. 각 블록 M_i를 다음 그림과 같은 과정을 통해 압축하여 최종 블록 처리후의 연쇄변수 H_0, H_1, H_2, H_3, H_4를 비트열로 변환한 것이 해쉬코드가 된다.



6.2. 입력블록 길이 및 덧붙이기

입력메시지는 512비트 단위로 처리된다. 마지막 메시지 블록은 블록의 길이가 448비트(= 512 - 64)가 되도록 "1" 다음에 필요한 개수의 "0"을 채운 다음, 마지막 64비트는 덧붙이기 전 메시지 길이를 2^{64} 법으로 계산한 정수값이 들어간다. 예를들어 입력메세지가 다음의 비트열로 주어졌다고 하자.

10100010 00111001 11100101 01101011 11001001 10001010 10011101

이 비트열의 길이는 56이므로 1다음에 391개의 0을 덧붙여 448비트로 만들고 나머지 64비트에는 56 = 111000이 들어가게 된다. 따라서 그 결과 512비트의 32비트 워드열은 다음과 같이 주어진다.

6be539a2 809d8ac9 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000038 00000000

6.3. 초기값

해쉬알고리즘 표준의 연쇄변수의 초기화에 사용되는 값은 16진수 표기로 다음과 같다.

H_0 = 67452301
H_1 = efcdab89
H_2 = 98badcfe

H_3 = 10325476

H_4 = c3d2e1f0

6.4. 상수

각 단계연산에서 사용될 상수는 16진수 표기로 다음과 같이 주어진다. 이 상수들은 무리수 $2^{30}\sqrt{2}, 2^{30}\sqrt{3}, 2^{30}\sqrt{5}$의 정수부분을 취한 것이다.

K_j = 00000000 ($0 \le j \le 19$: 라운드 1)
K_j = 5a827999 ($20 \le j \le 39$: 라운드 2) $\lfloor 2^{30}\sqrt{2} \rfloor$
K_j = 6ed9eba1 ($40 \le j \le 59$: 라운드 3) $\lfloor 2^{30}\sqrt{3} \rfloor$
K_j = 8f1bbcdc ($60 \le j \le 79$: 라운드 4) $\lfloor 2^{30}\sqrt{5} \rfloor$

6.5. 메시지 변수 준비

각 512비트 메시지 블록 M_i는 비트열-워드열 변환규칙에 의해 16개의 워드 $X[0], X[1] \cdots, X[15]$로 변환되며, 이 16워드로부터 4개의 메시지 변수()가 $X[16], X[17], X[18], X[19]$추가로 생성된다. 각 라운드의 단계연산에 입력될 메시지 변수의 순서는 아래 표와 같이 주어진다. 4개의 추가 메시지 변수 $X[16], X[17], X[18], X[19]$는 각 라운드의 10, 15, 0, 5번째 단계연산에서 사용되며 그 값은 각 라운드의 0-4, 6-9, 11-14, 16-19 단계연산들에서 사용되는 메시지 변수들의 XOR값으로 계산된다. 예를 들어, 2번째 라운드의 $X[16], X[17], X[18], X[19]$는 다음과 같이 계산된다.

$$X[16] = X[3] \oplus X[6] \oplus X[9] \oplus X[12]$$
$$X[17] = X[15] \oplus X[2] \oplus X[5] \oplus X[8]$$
$$X[18] = X[11] \oplus X[14] \oplus X[1] \oplus X[4]$$
$$X[19] = X[7] \oplus X[10] \oplus X[13] \oplus X[0]$$

i	라운드 1	라운드 2	라운드 3	라운드 4
0	X[18]	X[18]	X[18]	X[18]
1	X[0]	X[3]	X[12]	X[7]
2	X[1]	X[6]	X[5]	X[2]
3	X[2]	X[9]	X[14]	X[13]
4	X[3]	X[12]	X[7]	X[8]
5	X[19]	X[19]	X[19]	X[19]
6	X[4]	X[15]	X[0]	X[3]
7	X[5]	X[2]	X[9]	X[14]
8	X[6]	X[5]	X[2]	X[9]
9	X[7]	X[8]	X[11]	X[4]
10	X[16]	X[16]	X[16]	X[16]
11	X[8]	X[11]	X[4]	X[15]
12	X[9]	X[14]	X[13]	X[10]
13	X[10]	X[1]	X[6]	X[5]
14	X[11]	X[4]	X[15]	X[0]
15	X[17]	X[17]	X[17]	X[17]
16	X[12]	X[7]	X[8]	X[11]
17	X[13]	X[10]	X[1]	X[6]
18	X[14]	X[13]	X[10]	X[1]
19	X[15]	X[0]	X[3]	X[12]

편의상 j번째 단계연산에서 사용될 메시지 변수를 $X[l(j)]$라 표기하자. 그러면 위의 표로부터 $l(j)$는 다음과 같이 주어짐을 알 수 있다.

$l(j)$ = 18, 0, 1, 2, 3,19, 4, 5, 6, 7,16, 8, 9,10,11,17,12,13,14,15 ($0 \le j \le 19$)
$l(j)$ = 18, 3, 6, 9,12,19,15, 2, 5, 8,16,11,14, 1, 4,17, 7,10,13, 0 ($20 \le j \le 39$)
$l(j)$ = 18,12, 5,14, 7,19, 0, 9, 2,11,16, 4,13, 6,15,17, 8, 1,10, 3 ($40 \le j \le 59$)
$l(j)$ = 18, 7, 2,13, 8,19, 3,14, 9, 4,16,15,10, 5, 0,17,11, 6, 1,12 ($60 \le j \le 79$)

6.6. 부울함수

3개의 부울함수가 사용되며 각 j번째 단계연산에서 사용되는 부울함수는 다음과 같다.

$f_j(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$ ($0 \le j \le 19$: 라운드 1)

$f_j(x, y, z) = x \oplus y \oplus z$ ($20 \le j \le 39$, $60 \le j \le 79$: 라운드 2, 4)

$f_j(x, y, z) = y \oplus (x \vee \neg z)$ ($40 \le j \le 59$: 라운드 3)

즉, 각 라운드에서는 동일한 부울함수가 사용되며, 라운드 2와 4에서는 같은 부울함수가 사용된다.

6.7. 순환이동의 양

각 j번째 단계연산에 필요한 순환이동의 양 s_1, s_2는 아래와 같이 정의된다.

$$s_1(j) = 5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13$$
$$(0 \le j \le 19,\ 20 \le j \le 39,\ 40 \le j \le 59,\ 60 \le j \le 79)$$

$$s_2(j) = 10 \quad (0 \le j \le 19)$$
$$s_2(j) = 17 \quad (20 \le j \le 39)$$
$$s_2(j) = 25 \quad (40 \le j \le 59)$$
$$s_2(j) = 30 \quad (60 \le j \le 79)$$

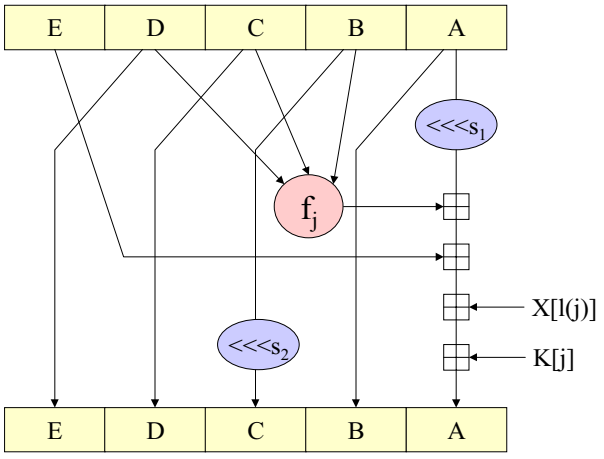
즉, 순환이동의 양 s_1은 각 라운드마다 동일한 순서로 적용되며, 순환이동의 양 s_2는 라운드 내에서는 동일한 값이 적용된다.

6.8. 단계연산

각 j번째의 단계연산은 다음과 같이 정의된다 (아래 그림 참조).

$$T \leftarrow A^{\lll s_1(j)} + f_j(B, C, D) + E + X[l(j)] + K_j;$$

$$E \leftarrow D;\ D \leftarrow C;\ C \leftarrow B^{\lll s_2(j)};\ B \leftarrow A;\ A \leftarrow T;$$



6.9. 연쇄변수 갱신과정

하나의 512비트 메시지 블록에 대해 4라운드, 80단계의 압축함수 연산이 끝나면 그 결과 A,B,C,D,E를 연쇄변수 $H_0 \sim H_4$에 더하여 다음과 같이 연쇄변수 $H_0 \sim H_4$를 갱신시킨다.

$$H_0 \mathrel{+}= A,\ H_1 \mathrel{+}= B,\ H_2 \mathrel{+}= C,\ H_3 \mathrel{+}= D,\ H_4 \mathrel{+}= E$$

6.10. 해쉬코드 출력

모든 512비트 메시지 블록을 처리한 후의 연쇄변수 H_0, H_1, H_2, H_3, H_4는 5.2절의 워드열-비트열 변환규칙에 의해 문자열로 변환된 후 해쉬코드로 출력된다. 즉 각 변수 H_i가 $H_i = h_{i3}h_{i2}h_{i1}h_{i0}$($h_{ij}$는8비트 비트열)와 같이 주어질 때 해쉬코드는 다음과 같은 비트열이 된다.

$$h_{00}h_{01}h_{02}h_{03}h_{10}h_{11}h_{12}h_{13}h_{20}h_{21}h_{22}h_{23}h_{30}h_{31}h_{32}h_{33}h_{40}h_{41}h_{42}h_{43}$$

7. 해쉬알고리즘 설계기준

해쉬알고리즘 표준은 기존의 MD4계열 해쉬알고리즘의 설계원칙이나 분석을 바탕으로 설계되었다. 압축함수의 전체적인 구조나 덧붙이기, 초기값, 상수 등은 대부분의 다른 해쉬알고리즘과 유사하다. 아래에는 해쉬알고리즘의 특징적인 면들을 바탕으로 설계기준들을 기술한다.

7.1. 메시지 변수 처리

기존 공격형태의 대부분이 메시지 변수에 대한 적용의 단순성에 기인하므로 하나의 메시지 워드가 되도록 많은 단계에 영향을 주도록 설계하였다. SHA-1에서 이미 입력된 4개의 메시지 워드의 합으로 각 단계마다 입력될 메시지 워드를 생성하는 것이 그 좋은 예이나, 지나치게 복잡한 메시지 변수 처리방법은 해쉬알고리즘의 효율성에 영향을 미칠 수 있으므로 메시지 변수가 충분한 빈도로 적용되면서도 빠르게 구현되도록 설계하였다. 주어지는 16개의 메시지 변수에서 각 라운드마다 서로 다른 4개의 메시지 변수를 XOR하여 4개

의 메시지 변수를 추가로 생성하였을 뿐만 아니라, XOR한 메시지 변수들의 조합을 모두 다르게 하여 메시지 변수 적용의 단순성을 개선하고자 하였다. 표준 해쉬알고리즘의 메시지 변수에 대한 적용순서는 RIPEMD-160의 설계기준을 참조하였으며, 각 메시지 변수가 일정한 간격을 두고 동일한 빈도로 적용되도록 하였다. 특히 4개의 메시지 변수의 XOR값으로 구해진 추가 메시지 변수들은 XOR되는 4개의 메시지 변수와 충분한 간격(5단계 이상)을 두고 적용되도록 $\{X[16], X[17], X[18], X[19]\}$의 적용순서를 규칙적으로 배열되도록 하였다. 뿐만 아니라, 동일한 메시지 변수가 각 라운드마다 인접하지 않게 적용되도록 하였다. 순환이동의 양은 거의 모든 메시지 변수에 대해서 매 라운드마다 다른 값이 적용되도록 하였다.

7.2. 단계연산

단계연산은 기본적으로 SHA-1을 사용하였다. 이는 SHA-1의 단계연산 방법이 기존의 알려진 공격에 대해 더 강한 저항성을 보여 주며, 또한 parallel 혹은 pipeline processing시에도 훨씬 이점이 있기 때문이다.

7.3. 부울함수

단계연산에 사용한 부울함수는 기존의 MD4 계열 해쉬알고리즘에서 널리 사용되는 3변수 부울함수 중에서 계산에 필요한 단위연산의 수가 3을 넘지 않는 함수를 사용하였다 (첫 부울함수는 다음과 같이 계산할 수 있다: $(x \wedge y) \vee (\neg x \wedge z) = (x \wedge (y \vee z)) \vee z$). 참고로 단위연산의 수는 $\neg, \vee, \wedge, \oplus$를 동일한 하나의 단위연산으로 보았을 때 부울함수의 계산에 필요한 연산의 수를 나타낸다. 모든 부울함수는 0-1 균형성을 만족한다. 라운드 1에 사용되는 부울함수는 SAC(Strict Avalanche Criterion)을 만족하며, 3변수 부울함수의 최대 비선형도인 2를 가진다. 라운드 3에 사용되는 부울함수는 SAC은 만족하지 않으나, 비선형도는 최대값 2를 갖는다. 라운드 2와 4에 사용되는 부울함수는 가장 계산량이 적은 부울함수로 높은 비선형도를 갖는 부울함수들 사이에 중복 사용하도록 하였다.

7.4. 문자열과 정수간의 변환

해쉬알고리즘 표준은 32비트 프로세서를 기반구조로 하며, little-endian 구조의 컴퓨터를 기본으로 설계하였다. 따라서 대부분의 워크스테이션과 같은 big-endian 구조에서는 추가적으로 메시지 워드들의 바이트 순서를 역순으로 바꾸어 주어야 한다. 이는 해쉬알고리즘의 개발 추세에 의한 것으로 대개의 MD4 계열 해쉬알고리즘들은 32비트 단위와 little-endian구조로 설계되어 있기 때문이다.

부기 1. 해쉬알고리즘 의사코드

1.1 정의

해쉬할 메시지가 t개의 512비트 블록 $\{M_i\}$ $(0 \le i < t)$로 이루어 졌다고 가정하자 (여기서 마지막 메시지 블럭은 메시지 덧붙이기 과정을 통해 448비트의 메시지와 64비트의 메시지 길이를 덧붙여서 생성된 512비트 블록이다). 각 i번째 메시지 블록은 비트열-정수간의 변환규칙에 의해 32비트 정수로 변환된 16개의 워드 $M_i = \{X_i[j]\}$ $(0 \le j < 16)$로 구성된다. 각 j번째 단계연산에서 사용되는 함수 및 상수 f_j, K_j, $s_1(j)$, $s_2(j)$, $l(j)$에 대한 정의는 다음과 같다.

□ 부울 함수 및 라운드 상수

$$
\begin{aligned}
f_j(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) && (0 \le j \le 19)\\
f_j(x,y,z) &= x \oplus y \oplus z && (20 \le j \le 39,\ 60 \le j \le 79)\\
f_j(x,y,z) &= y \oplus (x \vee \neg z) && (40 \le j \le 59)
\end{aligned}
$$

$$
\begin{aligned}
K_j &= 00000000 \quad (0 \le j \le 19)\\
K_j &= 5a827999 \quad (20 \le j \le 39) \quad \lfloor 2^{30}\sqrt{2} \rfloor\\
K_j &= 6ed9eba1 \quad (40 \le j \le 59) \quad \lfloor 2^{30}\sqrt{3} \rfloor\\
K_j &= 8f1bbcdc \quad (60 \le j \le 79) \quad \lfloor 2^{30}\sqrt{5} \rfloor
\end{aligned}
$$

□ 순환이동 양

$$
\begin{aligned}
s_1(j) &= 5,11,7,15,6,13,8,14,7,12,9,11,8,15,6,12,9,14,5,13\\
&\qquad (0 \le j \le 19,\ 20 \le j \le 39,\ 40 \le j \le 59,\ 60 \le j \le 79)
\end{aligned}
$$

$$
\begin{aligned}
s_2(j) &= 10 \quad (0 \le j \le 19)\\
s_2(j) &= 17 \quad (20 \le j \le 39)\\
s_2(j) &= 25 \quad (40 \le j \le 59)\\
s_2(j) &= 30 \quad (60 \le j \le 79)
\end{aligned}
$$

□ 메시지 변수 적용순서

$$
\begin{aligned}
l(j) &= 18,0,1,2,3,19,4,5,6,7,16,8,9,10,11,17,12,13,14,15 \quad (0 \le j \le 19)\\
l(j) &= 18,3,6,9,12,19,15,2,5,8,16,11,14,1,4,17,7,10,13,0 \quad (20 \le j \le 39)\\
l(j) &= 18,12,5,14,7,19,0,9,2,11,16,4,13,6,15,17,8,1,10,3 \quad (40 \le j \le 59)\\
l(j) &= 18,7,2,13,8,19,3,14,9,4,16,15,10,5,0,17,11,6,1,12 \quad (60 \le j \le 79)
\end{aligned}
$$

1.2 의사코드

입력 : 입력 메시지 M이 덧붙이기 과정을 통해 $16t$개의 32비트 워드열 $\{X_i[j]\}$ $(0 \le i < t,\ 0 \le j < 16)$로 변환되었다고 가정.

$H_0 = 67452301;\ H_1 = efcdab89;\ H_2 = 98badcfe;\ H_3 = 10325476;\ H_4 = c3d2e1f0;$
for i = 0 to t-1 {
$A = H_0;\ B = H_1;\ C = H_2;\ D = H_3;\ E = H_4;$
for r = 0 to 3 {
j = 20 · r ;

$X_i[16] = X_i[l(j+1)] \oplus X_i[l(j+2)] \oplus X_i[l(j+3)] \oplus X_i[l(j+4)];$
$X_i[17] = X_i[l(j+6)] \oplus X_i[l(j+7)] \oplus X_i[l(j+8)] \oplus X_i[l(j+9)];$
$X_i[18] = X_i[l(j+11)] \oplus X_i[l(j+12)] \oplus X_i[l(j+13)] \oplus X_i[l(j+14)];$
$X_i[19] = X_i[l(j+16)] \oplus X_i[l(j+17)] \oplus X_i[l(j+18)] \oplus X_i[l(j+19)];$

for k = 0 to 19 {
j = 20 · r + k ;
$T = A^{\lll s_1(j)} + f_j(B,C,D) + E + X_i[l(j)] + K_j;$
$E = D;\ D = C;\ C = B^{\lll s_2(j)};\ B = A;\ A = T;$
}
}
$H_0 \mathrel{+}= A;\ H_1 \mathrel{+}= B;\ H_2 \mathrel{+}= C;\ H_3 \mathrel{+}= D;\ H_4 \mathrel{+}= E;$
}

출력 : 32비트 워드열 H_0,H_1,H_2,H_3,H_4를 순서대로 비트열로 변환후 출력

부기 2. 참조구현의 계산값 예

부기 2.1 참조구현 값

hash("")
= 30 79 64 ef 34 15 1d 37 c8 04 7a de c7 ab 50 f4 ff 89 76 2d
hash("a")
= 48 72 bc bc 4c d0 f0 a9 dc 7c 2f 70 45 e5 b4 3b 6c 83 0d b8
hash("abc")
= 97 5e 81 04 88 cf 2a 3d 49 83 84 78 12 4a fc e4 b1 c7 88 04
hash("message digest")
= 23 38 db c8 63 8d 31 22 5f 73 08 62 46 ba 52 9f 96 71 0b c6
hash("abcdefghijklmnopqrstuvwxyz")
= 59 61 85 c9 ab 67 03 d0 d0 db b9 87 02 bc 0f 57 29 cd 1d 3c
hash("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234
56789")
= cb 5d 7e fb ca 2f 02 e0 fb 71 67 ca bb 12 3a f5 79 57 64 e5
hash("12345678901234567890123456789012345678901234567890123456789
01234567890")
= 07 f0 5c 8c 07 73 c5 5c a3 a5 a6 95 ce 6a ca 4c 43 89 11 b5
hash(a million of "a")
= d6 ad 6f 06 08 b8 78 da 9b 87 99 9c 25 25 cc 84 f4 c9 f1 8d

부기 2.2 상세 구현 예 1

M = "abc" = 61 62 63

X[0] = 80636261
X[1] = 00000000
X[2] = 00000000
X[3] = 00000000
X[4] = 00000000
X[5] = 00000000
X[6] = 00000000
X[7] = 00000000
X[8] = 00000000
X[9] = 00000000
X[10] = 00000000
X[11] = 00000000
X[12] = 00000000
X[13] = 00000000
X[14] = 00000018
X[15] = 00000000

Round 1 :: X[16] = 80636261
Round 1 :: X[17] = 00000000
Round 1 :: X[18] = 00000000
Round 1 :: X[19] = 00000018

Round 2 :: X[16] = 00000000
Round 2 :: X[17] = 00000000
Round 2 :: X[18] = 00000018
Round 2 :: X[19] = 80636261

Round 3 :: X[16] = 00000018
Round 3 :: X[17] = 80636261
Round 3 :: X[18] = 00000000
Round 3 :: X[19] = 00000000

Round 4 :: X[16] = 00000000
Round 4 :: X[17] = 00000018
Round 4 :: X[18] = 80636261
Round 4 :: X[19] = 00000000

H0 = 67452301

H1 = efcdab89
H2 = 98badcfe
H3 = 10325476
H4 = c3d2e1f0

	A	B	C	D	E
j = 0 ::	45321f1a	67452301	36ae27bf	98badcfe	10325476
j = 1 ::	e04d88ff	45321f1a	148c059d	36ae27bf	98badcfe
j = 2 ::	f60b82ab	e04d88ff	c87c6914	148c059d	36ae27bf
j = 3 ::	ccd02fd8	f60b82ab	3623ff81	c87c6914	148c059d
j = 4 ::	870fe765	ccd02fd8	2e0aafd8	3623ff81	c87c6914
j = 5 ::	038d19e6	870fe765	40bf6333	2e0aafd8	3623ff81
j = 6 ::	eb4d513d	038d19e6	3f9d961c	40bf6333	2e0aafd8
j = 7 ::	c6199cc0	eb4d513d	3467980e	3f9d961c	40bf6333
j = 8 ::	826359a2	c6199cc0	3544f7ad	3467980e	3f9d961c
j = 9 ::	a99e52d0	826359a2	66730318	3544f7ad	3467980e
j = 10 ::	28d842cf	a99e52d0	8d668a09	66730318	3544f7ad
j = 11 ::	c6c273fb	28d842cf	794b42a6	8d668a09	66730318
j = 12 ::	d655c964	c6c273fb	610b3ca3	794b42a6	8d668a09
j = 13 ::	eb2425da	d655c964	09cfef1b	610b3ca3	794b42a6
j = 14 ::	63a4b6e3	eb2425da	57259359	09cfef1b	610b3ca3
j = 15 ::	f0693e36	63a4b6e3	90976bac	57259359	09cfef1b
j = 16 ::	f0d180b3	f0693e36	92db8d8e	90976bac	57259359
j = 17 ::	4831dd1b	f0d180b3	a4f8dbc1	92db8d8e	90976bac
j = 18 ::	39ad9cba	4831dd1b	4602cfc3	a4f8dbc1	92db8d8e
j = 19 ::	2b3ba486	39ad9cba	c7746d20	4602cfc3	a4f8dbc1
j = 20 ::	1fcb2490	2b3ba486	3974735b	c7746d20	4602cfc3
j = 21 ::	cee58557	1fcb2490	490c5677	3974735b	c7746d20
j = 22 ::	046c945c	cee58557	49203f96	490c5677	3974735b
j = 23 ::	aceedbe0	046c945c	0aaf9dcb	49203f96	490c5677
j = 24 ::	2728fe3c	aceedbe0	28b808d9	0aaf9dcb	49203f96
j = 25 ::	d2c6ef67	2728fe3c	b7c159dd	28b808d9	0aaf9dcb
j = 26 ::	e4732e6e	d2c6ef67	fc784e51	b7c159dd	28b808d9
j = 27 ::	e8563479	e4732e6e	decfa58d	fc784e51	b7c159dd
j = 28 ::	0422d61c	e8563479	5cddc8e6	decfa58d	fc784e51
j = 29 ::	eea0e13e	0422d61c	68f3d0ac	5cddc8e6	decfa58d
j = 30 ::	ab216b59	eea0e13e	ac380845	68f3d0ac	5cddc8e6
j = 31 ::	ed2649af	ab216b59	c27ddd41	ac380845	68f3d0ac
j = 32 ::	af24b8a7	ed2649af	d6b35642	c27ddd41	ac380845
j = 33 ::	5cf71c1c	af24b8a7	935fda4c	d6b35642	c27ddd41
j = 34 ::	458f929a	5cf71c1c	714f5e49	935fda4c	d6b35642
j = 35 ::	e9470c4c	458f929a	3838b9ee	714f5e49	935fda4c
j = 36 ::	88f362f4	e9470c4c	25348b1f	3838b9ee	714f5e49

j = 37 :: 98da38db 88f362f4 1899d28e 25348b1f 3838b9ee
j = 38 :: 63608a5f 98da38db c5e911e6 1899d28e 25348b1f
j = 39 :: 57114f38 63608a5f 71b731b4 c5e911e6 1899d28e
j = 40 :: 745f8524 57114f38 bec6c114 71b731b4 c5e911e6
j = 41 :: 928b2f98 745f8524 70ae229e bec6c114 71b731b4
j = 42 :: 2bfa870f 928b2f98 48e8bf0a 70ae229e bec6c114
j = 43 :: 485b83bd 2bfa870f 3125165f 48e8bf0a 70ae229e
j = 44 :: 8543cf31 485b83bd 1e57f50e 3125165f 48e8bf0a
j = 45 :: 0234fa06 8543cf31 7a90b707 1e57f50e 3125165f
j = 46 :: f4d7e359 0234fa06 630a879e 7a90b707 1e57f50e
j = 47 :: 6a7ddb44 f4d7e359 0c0469f4 630a879e 7a90b707
j = 48 :: 194bd76a 6a7ddb44 b3e9afc6 0c0469f4 630a879e
j = 49 :: d771855c 194bd76a 88d4fbb6 b3e9afc6 0c0469f4
j = 50 :: 33743c28 d771855c d43297ae 88d4fbb6 b3e9afc6
j = 51 :: e7edeff5 33743c28 b9aee30a d43297ae 88d4fbb6
j = 52 :: 67f27cb1 e7edeff5 5066e878 b9aee30a d43297ae
j = 53 :: 39004ed5 67f27cb1 ebcfdbdf 5066e878 b9aee30a
j = 54 :: 6cd12861 39004ed5 62cfe4f9 ebcfdbdf 5066e878
j = 55 :: b229d753 6cd12861 aa72009d 62cfe4f9 ebcfdbdf
j = 56 :: 05dbaade b229d753 c2d9a250 aa72009d 62cfe4f9
j = 57 :: f1d5af33 05dbaade a76453ae c2d9a250 aa72009d
j = 58 :: ee9d7f0d f1d5af33 bc0bb755 a76453ae c2d9a250
j = 59 :: 276963ea ee9d7f0d 67e3ab5e bc0bb755 a76453ae
j = 60 :: d9855335 276963ea 7ba75fc3 67e3ab5e bc0bb755
j = 61 :: b0eeba74 d9855335 89da58fa 7ba75fc3 67e3ab5e
j = 62 :: 9a54f69e b0eeba74 766154cd 89da58fa 7ba75fc3
j = 63 :: d568200c 9a54f69e 2c3bae9d 766154cd 89da58fa
j = 64 :: 330c25d9 d568200c a6953da7 2c3bae9d 766154cd
j = 65 :: e9feeb40 330c25d9 355a0803 a6953da7 2c3bae9d
j = 66 :: 5b05bcdf e9feeb40 4cc30976 355a0803 a6953da7
j = 67 :: 3550bb91 5b05bcdf 3a7fbad0 4cc30976 355a0803
j = 68 :: 9a8c9cf2 3550bb91 d6c16f37 3a7fbad0 4cc30976
j = 69 :: 7f9c5e70 9a8c9cf2 4d542ee4 d6c16f37 3a7fbad0
j = 70 :: 037235cc 7f9c5e70 a6a3273c 4d542ee4 d6c16f37
j = 71 :: 8bf6e3d6 037235cc 1fe7179c a6a3273c 4d542ee4
j = 72 :: 8d89c7b7 8bf6e3d6 00dc8d73 1fe7179c a6a3273c
j = 73 :: ae682415 8d89c7b7 a2fdb8f5 00dc8d73 1fe7179c
j = 74 :: f9182e75 ae682415 e36271ed a2fdb8f5 00dc8d73
j = 75 :: 02d79705 f9182e75 6b9a0905 e36271ed a2fdb8f5
j = 76 :: 5327d673 02d79705 7e460b9d 6b9a0905 e36271ed
j = 77 :: 7f26992f 5327d673 40b5e5c1 7e460b9d 6b9a0905
j = 78 :: 4d5d23ff 7f26992f d4c9f59c 40b5e5c1 7e460b9d
j = 79 :: 9d3c3b96 4d5d23ff dfc9a64b d4c9f59c 40b5e5c1

H0 = 67452301 + 9d3c3b96 = 04815e97
H1 = efcdab89 + 4d5d23ff = 3d2acf88
H2 = 98badcfe + dfc9a64b = 78848349
H3 = 10325476 + d4c9f59c = e4fc4a12
H4 = c3d2e1f0 + 40b5e5c1 = 0488c7b1

해쉬코드 = 97 5e 81 04 88 cf 2a 3d 49 83 84 78 12 4a fc e4 b1 c7 88 04

부기 2.3 상세 구현 예 2

```
M ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" =
        41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50
        51 52 53 54 55 56 57 58 59 5a 61 62 63 64 65 66
        67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
        77 78 79 7a 30 31 32 33 34 35 36 37 38 39

First 512-bit block =
        41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50
        51 52 53 54 55 56 57 58 59 5a 61 62 63 64 65 66
        67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
        77 78 79 7a 30 31 32 33 34 35 36 37 38 39 80 00

X[ 0] = 44434241
X[ 1] = 48474645
X[ 2] = 4c4b4a49
X[ 3] = 504f4e4d
X[ 4] = 54535251
X[ 5] = 58575655
X[ 6] = 62615a59
X[ 7] = 66656463
X[ 8] = 6a696867
X[ 9] = 6e6d6c6b
X[10] = 7271706f
X[11] = 76757473
X[12] = 7a797877
X[13] = 33323130
X[14] = 37363534
X[15] = 00803938


Round 1 :: X[16] = 10000000
Round 1 :: X[17] = 08003a3e
Round 1 :: X[18] = 00000010
Round 1 :: X[19] = 7efd454b

Round 2 :: X[16] = 263a0008
Round 2 :: X[17] = 7ef54d43
Round 2 :: X[18] = 5d575553
Round 2 :: X[19] = 6365677d

Round 3 :: X[16] = 737d7f75
```

Round 3 :: X[17] = 10101010
Round 3 :: X[18] = 05800000
Round 3 :: X[19] = 00101000

Round 4 :: X[16] = 7375777d
Round 4 :: X[17] = 5d474543
Round 4 :: X[18] = 6ee55d43
Round 4 :: X[19] = 262a1018

H0 = 67452301
H1 = efcdab89
H2 = 98badcfe
H3 = 10325476
H4 = c3d2e1f0

	A	B	C	D	E
j = 0 ::	45321f2a	67452301	36ae27bf	98badcfe	10325476
j = 1 ::	a42de8df	45321f2a	148c059d	36ae27bf	98badcfe
j = 2 ::	2e82b8b2	a42de8df	c87ca914	148c059d	36ae27bf
j = 3 ::	6fff365d	2e82b8b2	b7a37e90	c87ca914	148c059d
j = 4 ::	4ba724d9	6fff365d	0ae2c8ba	b7a37e90	c87ca914
j = 5 ::	c6f7606b	4ba724d9	fcd975bf	0ae2c8ba	b7a37e90
j = 6 ::	4c192962	c6f7606b	9c93652e	fcd975bf	0ae2c8ba
j = 7 ::	6a2e27d3	4c192962	dd81af1b	9c93652e	fcd975bf
j = 8 ::	52d226db	6a2e27d3	64a58930	dd81af1b	9c93652e
j = 9 ::	1b0c07d6	52d226db	b89f4da8	64a58930	dd81af1b
j = 10 ::	3a48e8f9	1b0c07d6	489b6d4b	b89f4da8	64a58930
j = 11 ::	bef208d3	3a48e8f9	301f586c	489b6d4b	b89f4da8
j = 12 ::	89b0db3b	bef208d3	23a3e4e9	301f586c	489b6d4b
j = 13 ::	4b59f37f	89b0db3b	c8234efb	23a3e4e9	301f586c
j = 14 ::	27351bac	4b59f37f	c36cee26	c8234efb	23a3e4e9
j = 15 ::	40c9d040	27351bac	67cdfd2d	c36cee26	c8234efb
j = 16 ::	bd8b4521	40c9d040	d46eb09c	67cdfd2d	c36cee26
j = 17 ::	2f344be5	bd8b4521	27410103	d46eb09c	67cdfd2d
j = 18 ::	eaf360a3	2f344be5	2d1486f6	27410103	d46eb09c
j = 19 ::	6e586a18	eaf360a3	d12f94bc	2d1486f6	27410103
j = 20 ::	c0f085e5	6e586a18	c147d5e6	d12f94bc	2d1486f6
j = 21 ::	da45a825	c0f085e5	d430dcb0	c147d5e6	d12f94bc
j = 22 ::	866f084e	da45a825	0bcb81e1	d430dcb0	c147d5e6
j = 23 ::	141df495	866f084e	504bb48b	0bcb81e1	d430dcb0
j = 24 ::	8e993129	141df495	109d0cde	504bb48b	0bcb81e1
j = 25 ::	44a3e18a	8e993129	e92a283b	109d0cde	504bb48b
j = 26 ::	c65e076c	44a3e18a	62531d32	e92a283b	109d0cde

j = 27 :: 0920d6da c65e076c c3148947 62531d32 e92a283b
j = 28 :: 9388f846 0920d6da 0ed98cbc c3148947 62531d32
j = 29 :: 7bb13b8b 9388f846 adb41241 0ed98cbc c3148947
j = 30 :: d72d809a 7bb13b8b f08d2711 adb41241 0ed98cbc
j = 31 :: 725e605c d72d809a 7716f762 f08d2711 adb41241
j = 32 :: ee836e69 725e605c 0135ae5b 7716f762 f08d2711
j = 33 :: 4f091795 ee836e69 c0b8e4bc 0135ae5b 7716f762
j = 34 :: 1740cd2d 4f091795 dcd3dd06 c0b8e4bc 0135ae5b
j = 35 :: 3ae274da 1740cd2d 2f2a9e12 dcd3dd06 c0b8e4bc
j = 36 :: 2b440566 3ae274da 9a5a2e81 2f2a9e12 dcd3dd06
j = 37 :: 3ab41628 2b440566 e9b475c4 9a5a2e81 2f2a9e12
j = 38 :: 6c0c6c05 3ab41628 0acc5688 e9b475c4 9a5a2e81
j = 39 :: a06ccd40 6c0c6c05 2c507568 0acc5688 e9b475c4
j = 40 :: 3d17a198 a06ccd40 0ad818d8 2c507568 0acc5688
j = 41 :: aa645397 3d17a198 8140d99a 0ad818d8 2c507568
j = 42 :: a222c158 aa645397 307a2f43 8140d99a 0ad818d8
j = 43 :: e019e372 a222c158 2f54c8a7 307a2f43 8140d99a
j = 44 :: 1dec1fb1 e019e372 b1444582 2f54c8a7 307a2f43
j = 45 :: 655a0199 1dec1fb1 e5c033c6 b1444582 2f54c8a7
j = 46 :: f6b31c29 655a0199 623bd83f e5c033c6 b1444582
j = 47 :: 72da30c0 f6b31c29 32cab403 623bd83f e5c033c6
j = 48 :: db3b55d3 72da30c0 53ed6638 32cab403 623bd83f
j = 49 :: a8fa93ca db3b55d3 80e5b461 53ed6638 32cab403
j = 50 :: 8a281e20 a8fa93ca a7b676ab 80e5b461 53ed6638
j = 51 :: b05855f0 8a281e20 9551f527 a7b676ab 80e5b461
j = 52 :: ca802c35 b05855f0 4114503c 9551f527 a7b676ab
j = 53 :: 4af6b1a9 ca802c35 e160b0ab 4114503c 9551f527
j = 54 :: e1e3a3ae 4af6b1a9 6b950058 e160b0ab 4114503c
j = 55 :: 2fa439b0 e1e3a3ae 5295ed63 6b950058 e160b0ab
j = 56 :: aa9577de 2fa439b0 5dc3c747 5295ed63 6b950058
j = 57 :: 72dbd9de aa9577de 605f4873 5dc3c747 5295ed63
j = 58 :: 5a3ebcce 72dbd9de bd552aef 605f4873 5dc3c747
j = 59 :: 3735a1ad 5a3ebcce bce5b7b3 bd552aef 605f4873
j = 60 :: a0a2b9ca 3735a1ad 968faf33 bce5b7b3 bd552aef
j = 61 :: e6045a60 a0a2b9ca 4dcd686b 968faf33 bce5b7b3
j = 62 :: 165a6ddd e6045a60 a828ae72 4dcd686b 968faf33
j = 63 :: 93adc4e5 165a6ddd 39811698 a828ae72 4dcd686b
j = 64 :: bab79c49 93adc4e5 45969b77 39811698 a828ae72
j = 65 :: 40b1fbc6 bab79c49 64eb7139 45969b77 39811698
j = 66 :: 66b25e08 40b1fbc6 6eade712 64eb7139 45969b77
j = 67 :: ee621520 66b25e08 902c7ef1 6eade712 64eb7139
j = 68 :: 2bb2f2e2 ee621520 19ac9782 902c7ef1 6eade712
j = 69 :: e92e154d 2bb2f2e2 3b988548 19ac9782 902c7ef1

```
j = 70 :: f86f2f44 e92e154d 8aecbcb8 3b988548 19ac9782
j = 71 :: 7b1ce216 f86f2f44 7a4b8553 8aecbcb8 3b988548
j = 72 :: 62cfdfbd 7b1ce216 3e1bcbd1 7a4b8553 8aecbcb8
j = 73 :: a18b2de4 62cfdfbd 9ec73885 3e1bcbd1 7a4b8553
j = 74 :: 72892a81 a18b2de4 58b3f7ef 9ec73885 3e1bcbd1
j = 75 :: 2526c7a6 72892a81 2862cb79 58b3f7ef 9ec73885
j = 76 :: f43fcc35 2526c7a6 5ca24aa0 2862cb79 58b3f7ef
j = 77 :: 8f24d2b2 f43fcc35 8949b1e9 5ca24aa0 2862cb79
j = 78 :: 06345c67 8f24d2b2 7d0ff30d 8949b1e9 5ca24aa0
j = 79 :: 6d26f10f 06345c67 a3c934ac 7d0ff30d 8949b1e9

H0 = 67452301 + 6d26f10f = d46c1410
H1 = efcdab89 + 06345c67 = f60207f0
H2 = 98badcfe + a3c934ac = 3c8411aa
H3 = 10325476 + 7d0ff30d = 8d424783
H4 = c3d2e1f0 + 8949b1e9 = 4d1c93d9


Second 512-bit Block =
        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 00 f0 01 00 00 00 00 00 00


X[ 0] = 00000000
X[ 1] = 00000000
X[ 2] = 00000000
X[ 3] = 00000000
X[ 4] = 00000000
X[ 5] = 00000000
X[ 6] = 00000000
X[ 7] = 00000000
X[ 8] = 00000000
X[ 9] = 00000000
X[10] = 00000000
X[11] = 00000000
X[12] = 00000000
X[13] = 00000000
X[14] = 000001f0
X[15] = 00000000
```

```
Round 1 :: X[16] = 00000000
Round 1 :: X[17] = 00000000
Round 1 :: X[18] = 00000000
Round 1 :: X[19] = 000001f0

Round 2 :: X[16] = 00000000
Round 2 :: X[17] = 00000000
Round 2 :: X[18] = 000001f0
Round 2 :: X[19] = 00000000

Round 3 :: X[16] = 000001f0
Round 3 :: X[17] = 00000000
Round 3 :: X[18] = 00000000
Round 3 :: X[19] = 00000000

Round 4 :: X[16] = 00000000
Round 4 :: X[17] = 000001f0
Round 4 :: X[18] = 00000000
Round 4 :: X[19] = 00000000

H0 = d46c1410
H1 = f60207f0
H2 = 3c8411aa
H3 = 8d424783
H4 = 4d1c93d9
              A        B        C        D        E
j =  0 :: 17df5796 d46c1410 081fc3d8 3c8411aa 8d424783
j =  1 :: b08af9fb 17df5796 b0504351 081fc3d8 3c8411aa
j =  2 :: 9a51d2da b08af9fb 7d5e585f b0504351 081fc3d8
j =  3 :: 21e76b5b 9a51d2da 2be7eec2 7d5e585f b0504351
j =  4 :: 997ae4e0 21e76b5b 474b6a69 2be7eec2 7d5e585f
j =  5 :: e53e5c47 997ae4e0 9dad6c87 474b6a69 2be7eec2
j =  6 :: 496da530 e53e5c47 eb938265 9dad6c87 474b6a69
j =  7 :: aa2a9d89 496da530 f9711f94 eb938265 9dad6c87
j =  8 :: 9eef38b1 aa2a9d89 b694c125 f9711f94 eb938265
j =  9 :: d2701f68 9eef38b1 aa7626a8 b694c125 f9711f94
j = 10 :: 8426d2dc d2701f68 bce2c67b aa7626a8 b694c125
j = 11 :: a591cc2e 8426d2dc c07da349 bce2c67b aa7626a8
j = 12 :: f526dbb8 a591cc2e 9b4b7210 c07da349 bce2c67b
j = 13 :: ec2ca44f f526dbb8 4730ba96 9b4b7210 c07da349
j = 14 :: 1b1071d4 ec2ca44f 9b6ee3d4 4730ba96 9b4b7210
j = 15 :: 2da56e95 1b1071d4 b2913fb0 9b6ee3d4 4730ba96
j = 16 :: 248c9881 2da56e95 41c7506c b2913fb0 9b6ee3d4
```

j = 17 :: 55247e1b 248c9881 95ba54b6 41c7506c b2913fb0
j = 18 :: 9cec55f6 55247e1b 32620492 95ba54b6 41c7506c
j = 19 :: 5d4028bf 9cec55f6 91f86d54 32620492 95ba54b6
j = 20 :: d7b8245a 5d4028bf abed39d8 91f86d54 32620492
j = 21 :: b55cd11b d7b8245a 517eba80 abed39d8 91f86d54
j = 22 :: c80f1bc9 b55cd11b 48b5af70 517eba80 abed39d8
j = 23 :: 40ec5c63 c80f1bc9 a2376ab9 48b5af70 517eba80
j = 24 :: 09a62ae9 40ec5c63 3793901e a2376ab9 48b5af70
j = 25 :: 3dddf101 09a62ae9 b8c681d8 3793901e a2376ab9
j = 26 :: 619e20be 3dddf101 55d2134c b8c681d8 3793901e
j = 27 :: eb0f05b3 619e20be e2027bbb 55d2134c b8c681d8
j = 28 :: 711a1daf eb0f05b3 417cc33c e2027bbb 55d2134c
j = 29 :: 9aa1412a 711a1daf 0b67d61e 417cc33c e2027bbb
j = 30 :: ba085316 9aa1412a 3b5ee234 0b67d61e 417cc33c
j = 31 :: 893067a5 ba085316 82553542 3b5ee234 0b67d61e
j = 32 :: 99557b90 893067a5 a62d7410 82553542 3b5ee234
j = 33 :: 00f1cf6e 99557b90 cf4b1260 a62d7410 82553542
j = 34 :: 097ea83b 00f1cf6e f72132aa cf4b1260 a62d7410
j = 35 :: 23cf8de4 097ea83b 9edc01e3 f72132aa cf4b1260
j = 36 :: 296cefb2 23cf8de4 507612fd 9edc01e3 f72132aa
j = 37 :: 7af5d598 296cefb2 1bc8479f 507612fd 9edc01e3
j = 38 :: baebe95b 7af5d598 df6452d9 1bc8479f 507612fd
j = 39 :: e67dc4d1 baebe95b ab30f5eb df6452d9 1bc8479f
j = 40 :: 6c25e610 e67dc4d1 b775d7d2 ab30f5eb df6452d9
j = 41 :: bef8dae2 6c25e610 a3ccfb89 b775d7d2 ab30f5eb
j = 42 :: 65db689f bef8dae2 20d84bcc a3ccfb89 b775d7d2
j = 43 :: b8c30d8a 65db689f c57df1b5 20d84bcc a3ccfb89
j = 44 :: 7dec56e2 b8c30d8a 3ecbb6d1 c57df1b5 20d84bcc
j = 45 :: 9e974045 7dec56e2 1571861b 3ecbb6d1 c57df1b5
j = 46 :: b425fce9 9e974045 c4fbd8ad 1571861b 3ecbb6d1
j = 47 :: 6744b0c3 b425fce9 8b3d2e80 c4fbd8ad 1571861b
j = 48 :: 5abca4ea 6744b0c3 d3684bf9 8b3d2e80 c4fbd8ad
j = 49 :: a2d323ff 5abca4ea 86ce8961 d3684bf9 8b3d2e80
j = 50 :: 98d058e5 a2d323ff d4b57949 86ce8961 d3684bf9
j = 51 :: f44f7316 98d058e5 ff45a647 d4b57949 86ce8961
j = 52 :: 89bb04a6 f44f7316 cb31a0b1 ff45a647 d4b57949
j = 53 :: 05b184d6 89bb04a6 2de89ee6 cb31a0b1 ff45a647
j = 54 :: 6a988871 05b184d6 4d137609 2de89ee6 cb31a0b1
j = 55 :: 5d3736d1 6a988871 ac0b6309 4d137609 2de89ee6
j = 56 :: 6228183f 5d3736d1 e2d53110 ac0b6309 4d137609
j = 57 :: 7f1fca1b 6228183f a2ba6e6d e2d53110 ac0b6309
j = 58 :: dc6f42ab 7f1fca1b 7ec45030 a2ba6e6d e2d53110
j = 59 :: 3ba023e9 dc6f42ab 36fe3f94 7ec45030 a2ba6e6d

j = 60 :: 3a2fd57f 3ba023e9 f71bd0aa 36fe3f94 7ec45030
j = 61 :: 86d1d3b4 3a2fd57f 4ee808fa f71bd0aa 36fe3f94
j = 62 :: b2dfe3e2 86d1d3b4 ce8bf55f 4ee808fa f71bd0aa
j = 63 :: 7edb1506 b2dfe3e2 21b474ed ce8bf55f 4ee808fa
j = 64 :: f2a969c5 7edb1506 acb7f8f8 21b474ed ce8bf55f
j = 65 :: 7eb909a3 f2a969c5 9fb6c541 acb7f8f8 21b474ed
j = 66 :: 2b8229c3 7eb909a3 7caa5a71 9fb6c541 acb7f8f8
j = 67 :: 63ea1937 2b8229c3 dfae4268 7caa5a71 9fb6c541
j = 68 :: ac654fa8 63ea1937 cae08a70 dfae4268 7caa5a71
j = 69 :: d7657342 ac654fa8 d8fa864d cae08a70 dfae4268
j = 70 :: f82fc887 d7657342 2b1953ea d8fa864d cae08a70
j = 71 :: fcc72df2 f82fc887 b5d95cd0 2b1953ea d8fa864d
j = 72 :: 9633fde2 fcc72df2 fe0bf221 b5d95cd0 2b1953ea
j = 73 :: 703bdee2 9633fde2 bf31cb7c fe0bf221 b5d95cd0
j = 74 :: 2af69707 703bdee2 a58cff78 bf31cb7c fe0bf221
j = 75 :: 611f0e82 2af69707 9c0ef7b8 a58cff78 bf31cb7c
j = 76 :: 9fdf2ce1 611f0e82 cabda5c1 9c0ef7b8 a58cff78
j = 77 :: 378d8146 9fdf2ce1 9847c3a0 cabda5c1 9c0ef7b8
j = 78 :: ea0027da 378d8146 67f7cb38 9847c3a0 cabda5c1
j = 79 :: 271249bb ea0027da 8de36051 67f7cb38 9847c3a0

H0 = 67452301 + 271249bb = fb7e5dcb
H1 = efcdab89 + ea0027da = e0022fca
H2 = 98badcfe + 8de36051 = ca6771fb
H3 = 10325476 + 67f7cb38 = f53a12bb
H4 = c3d2e1f0 + 9847c3a0 = e5645779

해쉬코드 = cb 5d 7e fb ca 2f 02 e0 fb 71 67 ca bb 12 3a f5 79 57 64 e5

표준작성 공헌자

표준 번호 : TTAS.KO-12.0011/R1

이 표준의 제·개정 및 발간을 위해 아래와 같이 여러분들이 공헌하였습니다.

구 분	성 명	위원회 및 직위	연락처	소속사
과제 제안		암호기술연구반(SG10.02)		한국정보보호센터
표준 초안 제출	김 승 주	암호기술연구반 의장	(02)3488-4066	한국정보보호센터
표준 초안 검토 및 작성	김 승 주	암호기술연구반 의장	(02)3488-4066	한국정보보호센터
	박 성 준	암호기술연구반 부의장	(02)3453-1114	(주)비씨큐어
	이 성 재	암호기술연구반 간사	(02)3488-4171	한국정보보호센터
	권 현 조	암호기술연구반 위원	(02)3488-4271	한국정보보호센터
	이 인 수	암호기술연구반 위원	(02)3453-1114	(주)비씨큐어
	신 상 욱	암호기술연구반 참관자	(042)860-5820	한국전자통신연구원
	임 채 훈	암호기술연구반 참관자	(02)578-8925	(주)퓨처시스템
표준안 편집 및 감수	김 승 주	암호기술연구반 의장	(02)3488-4066	한국정보보호센터
	박 성 준	암호기술연구반 부의장	(02)3453-1114	(주)비씨큐어
	이 성 재	암호기술연구반 간사	(02)3488-4171	한국정보보호센터
	권 현 조	암호기술연구반 위원	(02)3488-4271	한국정보보호센터
	신 상 욱	암호기술연구반 참관자	(042)860-5820	한국전자통신연구원
	임 채 훈	암호기술연구반 참관자	(02)578-8925	(주)퓨처시스템
	노 재 훈	암호기술연구반 참관자	(031)450-7885	LG정보통신(주)
표준안 심의	이 홍 섭	정보보호기술위원회 의장	(02)3488-4150	한국정보보호센터
	김 석 우	정보보호기술위원회 부의장	(031)450-5025	한세대학교
	김 학 범	정보보호기술위원회 위원	(02)3488-4213	한국정보보호센터
	김 승 주	정보보호기술위원회 위원	(02)3488-4066	한국정보보호센터
	이 인 숙	정보보호기술위원회 위원	(042)870-8047	한국통신
	신 제 호	정보보호기술위원회 특별위원	(02)2260-3336	동국대학교
	최 용 락	정보보호기술위원회 특별위원	(042)280-2541	대전대학교
	고 승 철	정보보호기술위원회 위원	(02)3488-4100	한국정보보호센터
	이 경 구	정보보호기술위원회 위원	(02)3488-4030	한국정보보호센터
사무국 담당	권 미 희	정보기술표준부 부장	(02)723-7073	한국정보통신기술협회
	이 문 철	정보기술표준부	(02)723-7073	한국정보통신기술협회