교과목 : 정보보호

5. Hash Functions

2023학년도 2학기
이석환 교수

Contents

1. Hash Function

2. Non-Crypto Hash

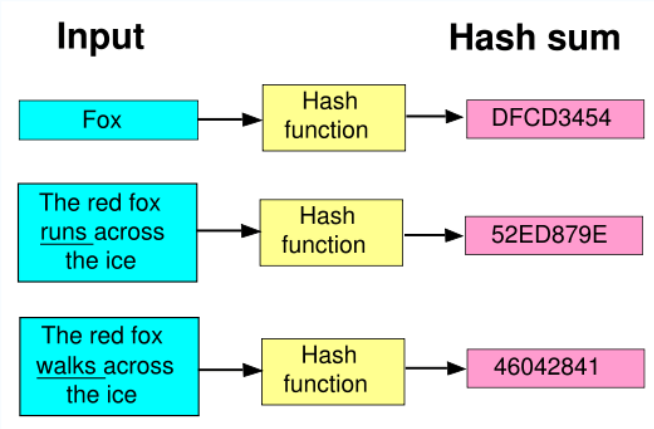
3. Crypto Hash (SHA512, SHA3)

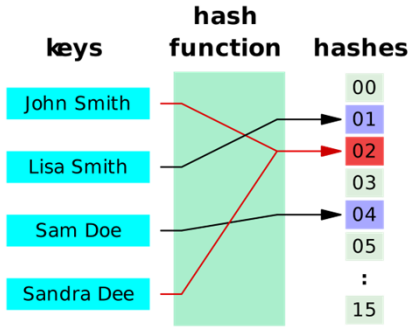
4. HMAC (Hash-based Message Authentication)

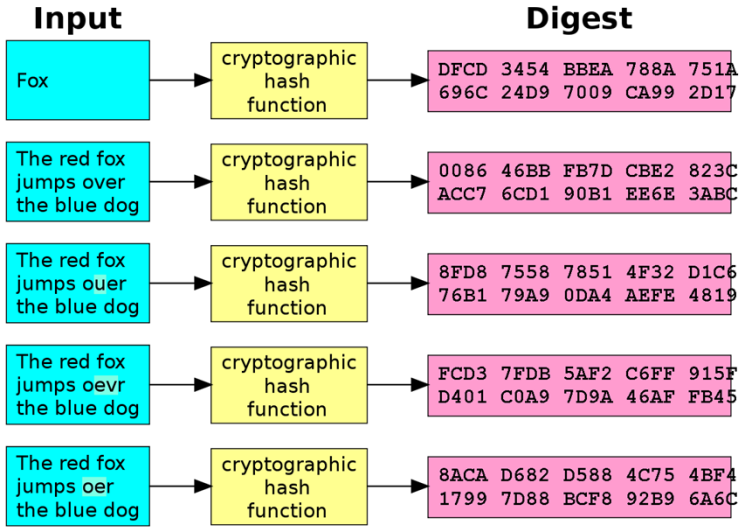
1. Hash Function?

- A hash function is a reproducible method of turning some kind of data into a (relatively) small number that may serve as a digital "fingerprint" of the data. (임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수)
- Crypto Hash function: a hash function with certain additional security properties to make it suitable for use as various info security applications
  - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
  - SHA-3 : SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256

A cryptographic hash function is a kind that is used in cryptography. Its hash value is a fixed-size, alphanumeric string, and may also be called a 'message digest', 'digital fingerprint', 'digest' or 'checksum'.







[Wikipedia]

A cryptographic hash function (specifically SHA-1) at work. A small change in the input (in the word "over") drastically changes the output (digest) (so-called avalanche effect).

Hash Function Motivation

- Public key notation
- ✓ $[M]_{Alice}$: Sign message M with Alice's **private key**
- ✓ $\{M\}_{Alice}$: Encrypt message M with Alice's **public key**

- Suppose Alice signs M
  - ✓ Alice sends M and $S = [M]_{Alice}$ to Bob
  - ✓ Bob verifies that $M = \{S\}_{Alice}$
  - ✓ Aside: Is it OK to just send S?
- If M is big, $[M]_{Alice}$ is costly to compute
- Suppose instead, Alice signs h(M), where h(M) is much smaller than M
  - ✓ Alice sends M and $S = [h(M)]_{Alice}$ to Bob
  - ✓ Bob verifies that $h(M) = \{S\}_{Alice}$

Crypto Hash Function

- Crypto hash function h(x) must provide the following properties

- Compression : output length is small
- Efficiency : h(x) easy to compute for any x
- One-way : given a value y, it is infeasible to find an x such that h(x) = y
- Weak collision resistance : Given a plaintext x and h(x), infeasible to find y with y≠x such that h(y)=h(x)
- Strong collision resistance : Infeasible to find any x and y, with x≠y such that h(x)=h(y)
- Lots of collisions exist, but hard to find one

Pre-Birthday Problem

- Suppose N people in a room
- How large must N be before the probability someone has same birthday as me is ≥ 1/2
  - Solve: $1/2 = 1 - (364/365)^N$ for N
  - Find N = 253

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that two or more have same birthday? (Strong collision) = match any birthday
  - ✓ The probability that no two out of n people share a birthday

$$\bar{p} = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right), \cdots \left(\frac{365-(n-1)}{365}\right) = \frac{365!}{(365-n)!}\frac{1}{365^n}$$

  - ✓ The probability that any two persons have the same birthday

$$p = 1 - \bar{p} = 1 - \frac{365!}{(365-n)!}\frac{1}{365^n}$$

  - ✓ Set equal to 1/2 and solve: N = 23

$19^2=361$
$20^2=400$
$23^2=529$

- Surprising? A paradox?
- Maybe not: "Should be" about sqrt(365) since we compare all pairs x and y

Of Hashes and Birthdays

- If h(x) is N bits, then 2^N different hash values are possible
- sqrt(2^N) = $2^{N/2}$
- Therefore, hash about $2^{N/2}$ random values and you expect to find a collision
- Implication: secure N bit symmetric key requires 2^{N-1} work to "break" while secure N bit hash requires $2^{N/2}$ work to "break"

Non-Crypto Hash

2. Non-Crypto Hash?

- Data X = $(X_0, X_1, X_2, \cdots, X_{n-1})$, each X_i is a byte
- Suppose that hash(X) = $X_0+X_1+X_2+\cdots+X_{n-1}$ mod 256
  - ✓ Output is always 8 bits
- Is this secure?
  - ✓ Example: (n=2)
    - X = (10101010, 00001111) = 170+15 = 185 = 10111001
    - Hash is 10111001
    - But so is hash of Y = (00001111,10101010)
- Easy to find collisions, so not secure…

2. Non-Crypto Hash?

- Data $X = (X_0, X_1, X_2, \cdots, X_{n-1})$
- Suppose that hash is
  - $h(X) = nX_0 + (n-1)X_1 + (n-2)X_2 + \cdots + 1 \cdot X_{n-1} \bmod 256$
- Is this hash secure?
- At least
  - h(10101010, 00001111) ≠ h(00001111, 10101010)
- But hash of (00000001, 00001111) is same as hash of (00000000, 00010001)
- Not one-way, but this hash is used in the (non-crypto) application rsync

[Wikipedia] rsync : 컴퓨터 시스템 상에서 파일을 효율적으로 전송하고 동기화하기 위한 유틸리티의 하나로, 파일의 타임스탬프와 크기를 검사함으로써 이루어진다

- Redundancy check (ex: parity bit)
  - ✓ Extra data added to a message for the purposes of error detection and correction
  - ✓ Any hash function can be used as a redundancy check
- Checksum:
  - ✓ A form of redundancy check, a simple way to protect the integrity of data by detecting errors in data
  - ✓ It works by adding up the basic components of the message, and later compare it to the authentic checksum

- Example of Checksum:
  - ✓ 다음과 같이 4 바이트의 데이터가 있다고 가정: 0x25, 0x62, 0x3F, 0x52
  - ✓ 1 단계: 모든 바이트를 덧셈하면 0x118이 된다.
  - ✓ 2 단계: Carry Nibble(4bit)을 버림으로써 0x18을 만든다.
  - ✓ 3 단계: 0x18의 2의 보수를 얻음으로써 0xE8을 얻는다. 이것이 체크섬 바이트이다.
  - ✓ 체크섬 바이트를 테스트하려면 원래 그룹의 바이트에 체크섬 바이트까지 모두 더하면 0x200이 된다.
  - ✓ 다시 Carry Nibble을 버림으로써 0x000이 된다. 0x000이라는 뜻은 오류가 없다는 뜻이다. (하지만 오류가 있어도 우연히 0x000이 될 수도 있다.)

부록 참조

- Cyclic Redundancy Check (CRC)
  - ✓ A CRC "checksum" is the remainder of a binary division with no bit carry (XOR used instead of subtraction), of the message bit stream, by a predefined (short) bit stream of length *n*+1, which represents the coefficients of a polynomial with degree *n*.
  - ✓ Before the division, *n* zeros are appended to the message stream. (example)
  - ✓ CRCs and similar checksum methods are only designed to detect transmission errors
  - ✓ Not to detect intentional tampering with data
  - ✓ But CRC sometimes mistakenly used in crypto applications (WEP)

WEP(Wired Equivalent Privacy) : 무선 LAN 운용간의 보안을 위해 사용되는 알고리즘, 스트림 암호화 기법인 RC4를 사용하며, CRC-32 체크섬을 통해 무결성을 확보하였음. (2004년 사용중단(deprecated)이 선언)

- **Example of CRC: [Wikipedia 참조:https://ko.wikipedia.org/wiki/순환_중복_검사]**

n-bit 이진 CRC 계산을 위해서는 다항식(polynomial) (*n*+1)-비트 패턴의 제수(divisor)를 만든다.

다항식	제수	비트수	CRC
$x^3 + x + 1$	1011	4비트	3비트

다항식의 각 자릿수 별로 표현하면:

$1x^3 + 0x^2 + 1x + 1 => 1011$

메시지 데이터는:

```
11010011101100
```

3비트 CRC를 계산하기 첫 번째로 나누면 :

```
          1
     --------------------
1011 ) 11010011101100 000    <--- 오른쪽으로 3비트 후부터
       1011                  <--- 제수(divisor) 4비트 <= x³ +x+1
-------------------
       0110011101100 000     <--- 결과
```

나누는 과정에서 각 비트별로 XOR로 행한다. 일반적인 나누기에서의 뺄셈과는 다르다. 연산결과 첫 번째 비트가 0으로 소거 되도록 몫의 비트를 정한다. 메시지 첫 번째 비트가 1이므로 몫의 1로 하여 XOR-나누기를 한다. 이렇게 되면 첫 번째 비트가 0으로 된다.

```
1101 XOR 1011 => 0 110
```

이제 전체를 계산하며:

```
    11110001111 100   <--- 몫은 별로 의미가 없는 중간 과정이다.
--------------------
11010011101100 000    <--- 오른쪽으로 3비트 후부터
1011                  <--- 제수
0110011101100 000     <--- 결과
 1011                 <--- 제수 ...
0011110111011OO 000
  1011
0001010111011OO 000
   1011
0000000011011OO 000
    0000              <--- 0인 경우 위의 계산결과가 바뀌지 않는다. 그대로 넘겨도 된다.
0000000011011OO 000
     0000
0000000011011OO 000
      0000
0000000011011OO 000
       1011
0000000001101OO 000
        1011
0000000000111OO 000
         1011
0000000000010OO 000
          1011
0000000000001 000
           101 1
0000000000000 100   <--- 왼쪽이 모두 0이면 여기서 끝내도 된다. 뒤에 오는 것은 0은 계산에 영향이 없다.
           00 00
0000000000000 100
            0 000
---------------
0000000000000 100 <--- 나머지(remainder) 3비트, 앞에는 모두 0이 되고 뒤에 3비트가 최종 결론이다.
```

왼쪽의 나머지가 모두 0이 되도록 계속 나눈다. 최대로 입력 비트수 만큼 나누면 모두 0이 된다.

- Example of CRC: [Wikipedia 참조:https://ko.wikipedia.org/wiki/순환_중복_검사]

이제 위의 계산결과로 부터 검증을 하면 입력 메시지 다음에 CRC 결과를 붙여 나누면 나머지가 0이 된다.:

```
11010011101100 100 <--- 입력과 CRC 체크값을 붙이고
1011               <--- 나누고
0110011101100 100 <--- 결과
 1011              <--- 나누고 ...
00111011101100 100


......

00000000001110 100
          1011
00000000000101 100
           101 1
----------------
               0 <--- 나머지
```

CRC의 이론적 내용은 [부록] 참조

Crypto Hash

A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length.

Crypto Hash Design

- No collisions
  - ✓ Then we say the hash function is secure
  - ✓ Change in input should not be correlated with output
- Desired property: avalanche effect
  - ✓ Change to 1 bit of input should affect about half of output bits
  - ✓ Avalanche effect should occur after few rounds
- Efficiency
  - ✓ No efficiency, no meaning for signing appl
- Crypto hash functions consist of some number of rounds
  - ✓ Analogous to design of block ciphers
  - ✓ Similar trade-offs as the iterated block cipher
  - ✓ Want security and speed but simple rounds

Popular Crypto Hashes

- **MD(Message Digest) 5**
  - ✓ invented by Rivest
  - ✓ 128 bit output
  - ✓ MD2 → MD4 → MD5
  - ✓ MD2 and MD4 are no longer secure, due to collision found
  - ✓ Note: even MD5, collision recently found
- **SHA(Secure Hash Algorithm)-1**
  - A US government standard (similar to MD5)
  - "The world's most popular hash function"
  - 180 bit output
  - SHA-0 → SHA-1 → SHA-2 (256/512 bits output) → SHA-3
- Many others hashes, but MD5 and SHA-1 most widely used

SHA-1은 SHA 함수들 중 가장 많이 쓰이며, TLS, SSL, PGP, SSH, IPSec 등 많은 보안 프로토콜과 프로그램에서 사용되고 있다. SHA-1은 이전에 널리 사용되던 MD5를 대신해서 쓰이기도 한다. 혹자는 좀 더 중요한 기술에는 SHA-256이나 그 이상의 알고리즘을 사용할 것을 권장한다.

Table 12.1 *Characteristics of Secure Hash Algorithms (SHAs)*

Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

- SHA-512 is the version of SHA with a 512-bit message digest. This version, like the others in the SHA family of algorithms, is based on the *Merkle-Damgard scheme* (method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions.).

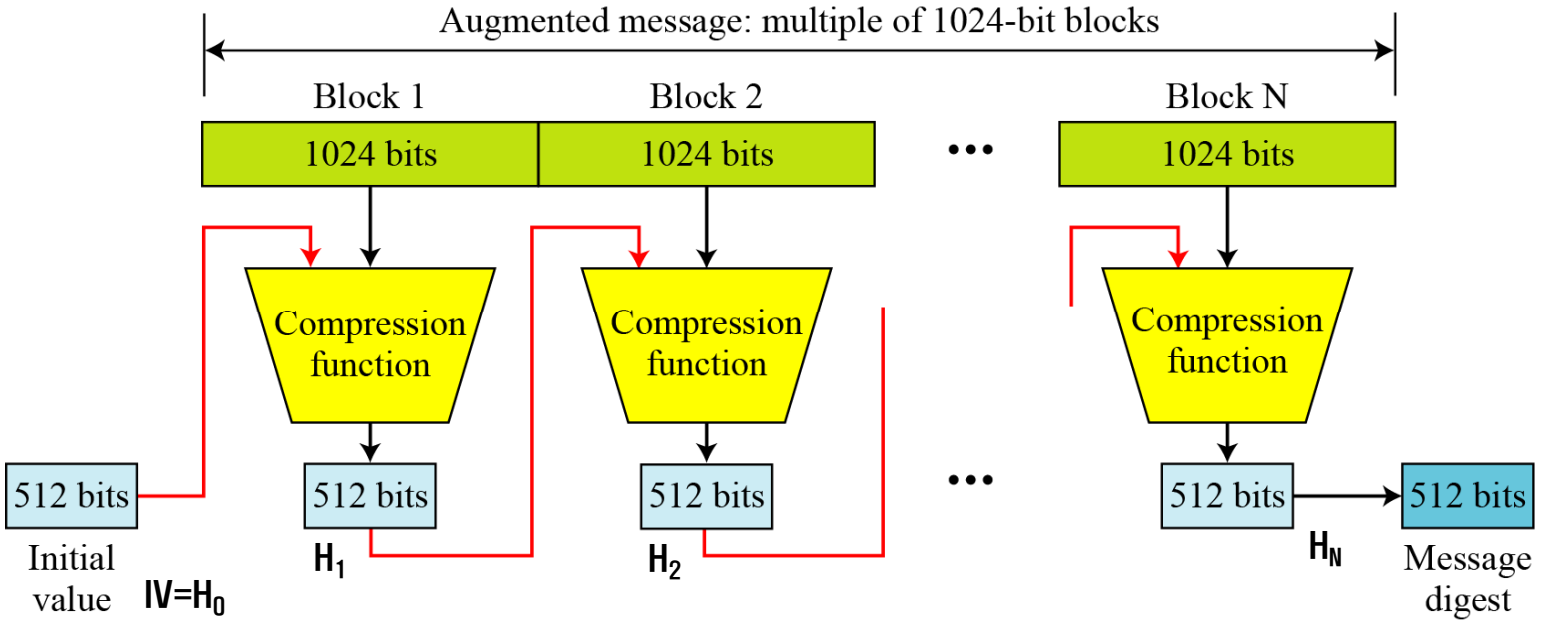


Figure 1. ***Message digest creation SHA-512***

Message Preparation

- SHA-512 insists that the length of the original message be less than 2^{128} bits.

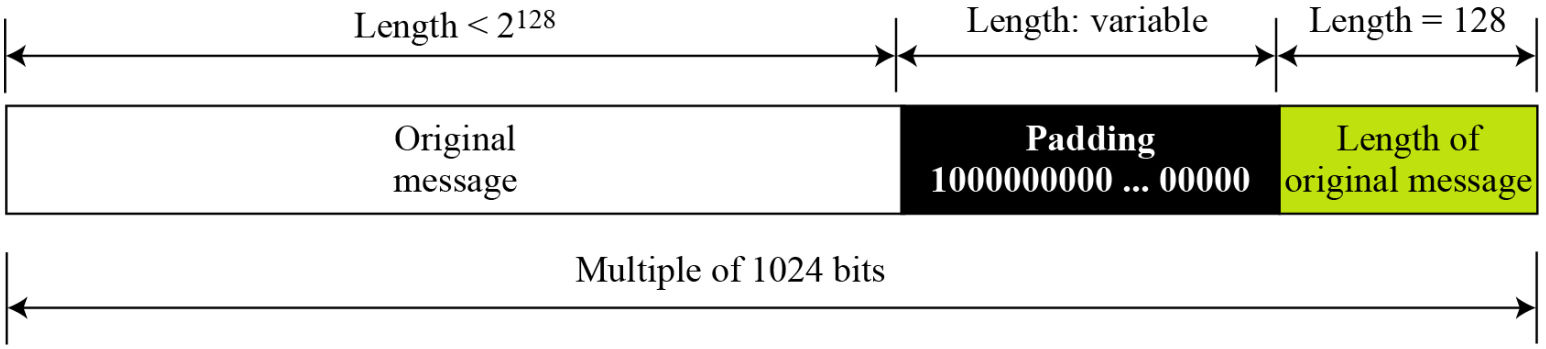
SHA-512 creates a 512-bit message digest out of a message less than 2^{128}.

Step1. Append padding bits

- Message is padded so that its length is congruent to 896 modulo 1024 [length ≡ 896(mod 1024)].
- Padding bits (length from 1 to 1024) : a single 1 bit followed by the necessary number of 0 bits

Step2. Append length

- A block of 128bits : Contains the length of original message in bits (before padding)



- The sequence of 1024-bit blocks M_1, M_2, ... M_N, so
- Total length of the expanded message : Nx1024bits

Figure 2. ***Padding and length field in SHA-512***

Step3. Initialize hash buffer

- Use 512-bit buffer to hold intermediate and final results
- Represent the buffer as 8 words (64-bit registers a, b, c, d, e, f, g, h)

The registers are initialized to 64-bit integers

Table 12.2 *Values of constants in message digest initialization of SHA-512*

Buffer	*Value (in hexadecimal)*	*Buffer*	*Value (in hexadecimal)*
A_0	6A09E667F3BCC908	E_0	510E527FADE682D1
B_0	BB67AE8584CAA73B	F_0	9B05688C2B3E6C1F
C_0	3C6EF372EF94F828	G_0	1F83D9ABFB41BD6B
D_0	A54FE53A5F1D36F1	H_0	5BE0CD19137E2179

IV=H_0

Word

1 Block = 1024bits = 16 words

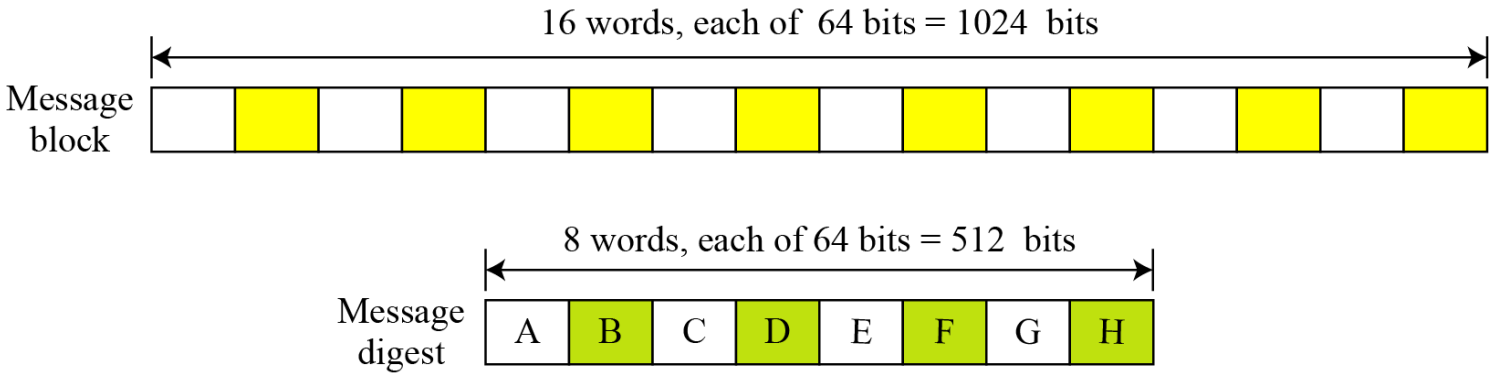
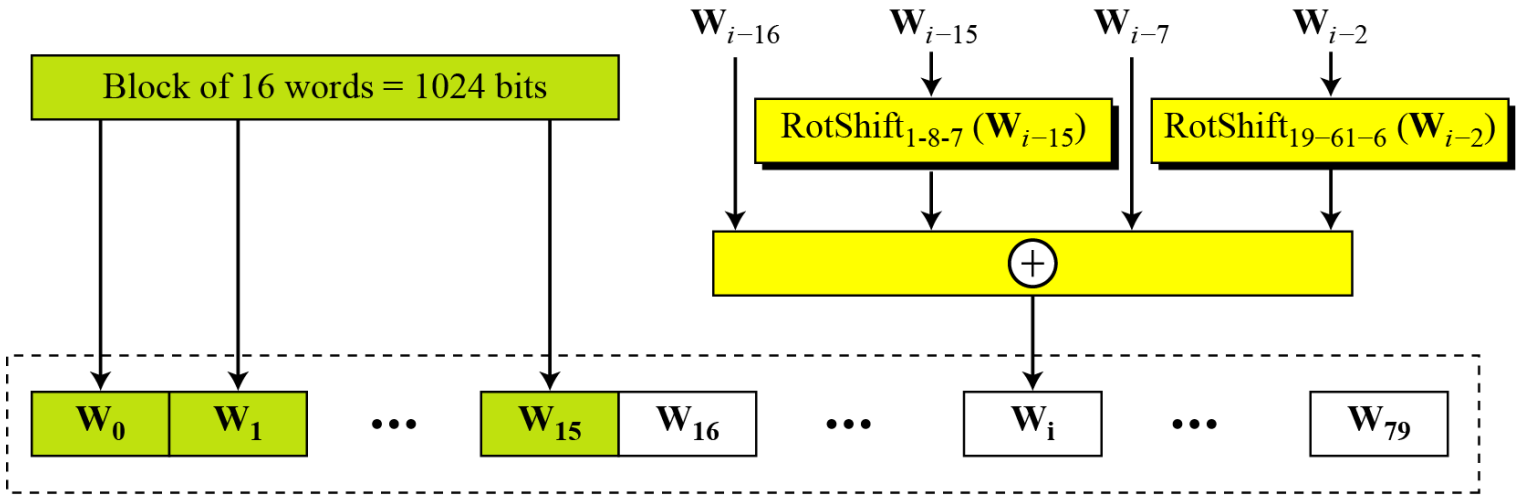


Figure 3. ***A message block and the digest as words***

Word Expansion

- Derive the 64-bit word values W_t from the 1024-bit message



RotShift$_{l\text{-}m\text{-}n}$ (x): RotR$_l$ (x) $\oplus$ RotR$_m$ (x) $\oplus$ ShL$_n$ (x)

RotR$_i$ (x): Right-rotation of the argument x by i bits
ShL$_i$ (x): Shift-left of the argument x by i bits and padding the left by 0's.

Figure 4. ***Word expansion in SHA-512***

3. Crypto Hash – SHA512

Step4. Process message in 1024-bit (128-byte) blocks – Compression function

- 80rounds
- Each round : Input of 512-bit buffer, abcdefgh,
- The buffer has the intermediate hash value, H_{i-1}. at input to the first round.
- Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently.
- Each round also makes use of an additive constant K_t, where 0 ⋯ t ⋯ 79 indicates one of the 80 rounds.
- These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers.
- The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data. (Table 11.4).

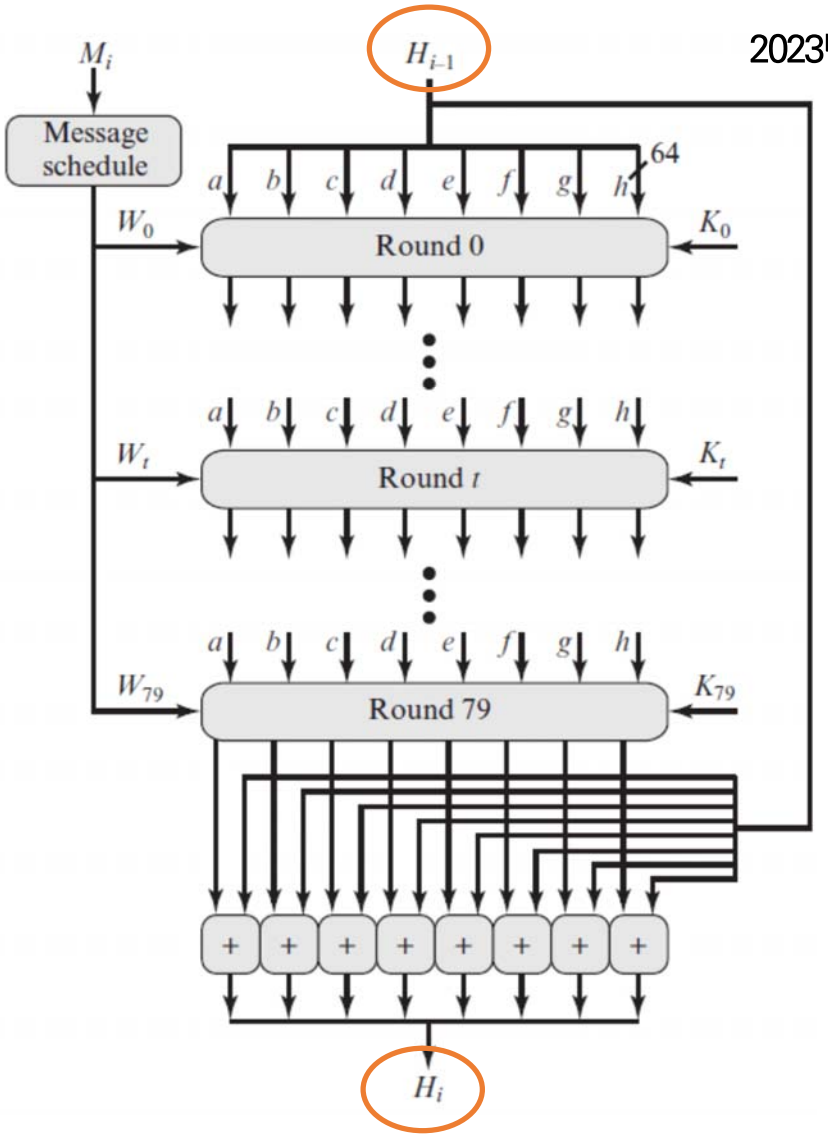


Figure 5. SHA-512 Processing of a Single 1024-Bit Block

Step5. Output

- After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digests

Summarize the behavior of SHA-512

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

IV = initial value of the abcdefgh buffer, defined in step 3

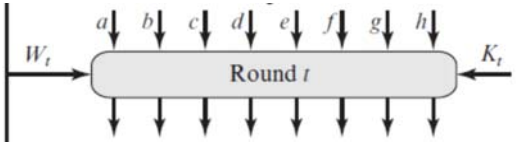
abcdefgh_i = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value

SHA512 Round Function



Majority(A,B,C)

$$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$$

- True only of the majority (two or three) of the arguments are true

Conditional(E,F,G)

$$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$$

- If e then f, else g

Rotate function

$$\text{Rotate (A): } \text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$$

$$\text{Rotate (E): } \text{RotR}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$$

- $\text{RotR}_i(x)$: Circular right shift (rotation) of the 64-bit argument x by i bits

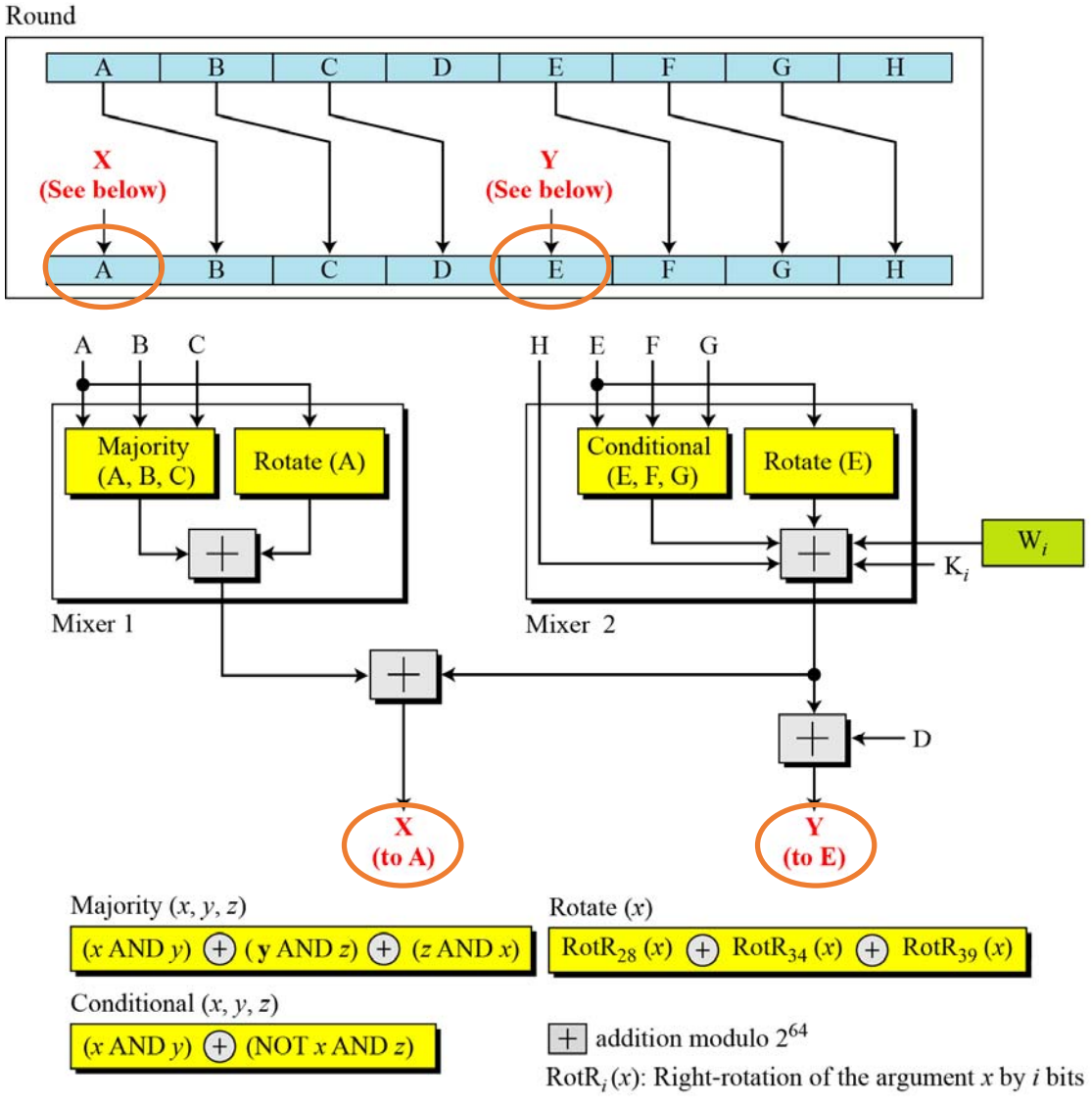


Figure 6. ***Structure of each round in SHA-512***

Table 12.3 *Eighty constants used for eighty rounds in SHA-512*

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

There are 80 constants, K_0 to K_{79}, each of 64 bits. Similar These values are calculated from the first 80 prime numbers (2, 3,…, 409). For example, the 80th prime is 409, with the cubic root $(409)^{1/3} = 7.42291412044$. Converting this number to binary with only 64 bits in the fraction part, we get

$$(111.0110\ 1100\ 0100\ 0100\ \ldots\ 0111)_2 \rightarrow (7.6C44198C4A475817)_{16}$$

The fraction part: $(6C44198C4A475817)_{16}$

With a message digest of 512 bits, SHA-512 expected to be resistant to all attacks, including collision attacks.

SHA512 Algorithm

The padded message consists blocks $M_1, M_2, \ldots, M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1}, \ldots, M_{i,15}$. All addition is performed modulo 2^{64}.

$H_{0,0} = 6A09E667F3BCC908$ $H_{0,4} = 510E527FADE682D1$
$H_{0,1} = BB67AE8584CAA73B$ $H_{0,5} = 9B05688C2B3E6C1F$
$H_{0,2} = 3C6EF372FE94F82B$ $H_{0,6} = 1F83D9ABFB41BD6B$
$H_{0,3} = A54FF53A5F1D36F1$ $H_{0,7} = 5BE0CD19137E2179$

for $i = 1$ **to** N

1. Prepare the message schedule W
for $t = 0$ **to** 15
$W_t = M_{i,t}$
for $t = 16$ **to** 79
$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$

2. Initialize the working variables

$a = H_{i-1,0}$ $e = H_{i-1,4}$
$b = H_{i-1,1}$ $f = H_{i-1,5}$
$c = H_{i-1,2}$ $g = H_{i-1,6}$
$d = H_{i-1,3}$ $h = H_{i-1,7}$

3. Perform the main hash computation
for $t = 0$ **to** 79

$$T_1 = h + \text{Ch}(e, f, g) + \left(\Sigma_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\Sigma_0^{512} a\right) + \text{Maj}(a, b, c)$$

$h = g$
$g = f$
$f = e$
$e = d + T_1$
$d = c$
$c = b$
$b = a$
$a = T_1 + T_2$

4. Compute the intermediate hash value

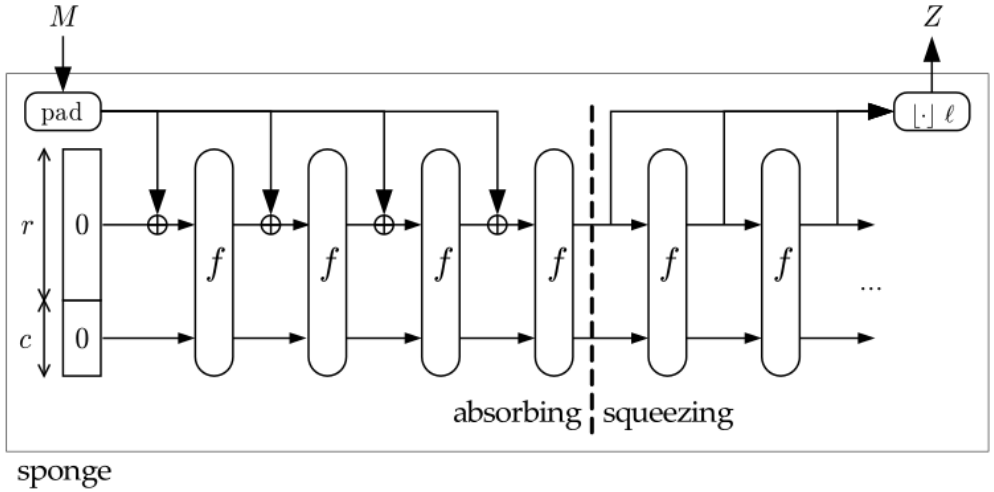
$H_{i,0} = a + H_{i-1,0}$ $H_{i,4} = e + H_{i-1,4}$
$H_{i,1} = b + H_{i-1,1}$ $H_{i,5} = f + H_{i-1,5}$
$H_{i,2} = c + H_{i-1,2}$ $H_{i,6} = g + H_{i-1,6}$
$H_{i,3} = d + H_{i-1,3}$ $H_{i,7} = h + H_{i-1,7}$

return $\{H_{N,0} \| H_{N,1} \| H_{N,2} \| H_{N,3} \| H_{N,4} \| H_{N,5} \| H_{N,6} \| H_{N,7}\}$

Latest Crypto Hash

- **SHA-3 (called Keccak)**
  - ✓ 2015년 8월 5일 NIST는 SHA-3을 신규 해싱표준 (Hashing Standard)로 공개 발표
  - ✓ SHA3-224, SHA3-256, SHA3-384 와 SHA3-512 등 4개 암호화 해시 알고리즘 그리고 SHAKE128과 SHAKE256, 두 개의 확장가능한 출력함수 (Extendable-output functions) (XOFs)로 구성되어 있다. Sponge 구조로 이루어졌기에 Sponge 함수라고도 한다.
  - ✓ Absorbing (absorption). 입력 메시지 M은 정해진 bitrate r에 따라 b-bit permutation f 출력과 패딩 p와 XOR 연산 과정 통해 흡수
  - ✓ Squeezing(pressing). permutation 결과로 해시값 z을 출력

r (bit rate), c(capacity)는 b = r+c을 만족해야 함
b∈25,50,100,200,400,800,1600



해시함수의 스펀지 구조
(https://en.bitcoinwiki.org/wiki/SHA-3)

SHA-3 예제 (https://ko.wikipedia.org/wiki/SHA-3)

미국 국립표준기술연구소가 게시한 예제 해시 값[3]

```
SHA3-224("")
6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7
SHA3-256("")
a7ffc6f8bf1ed76651c14756a061d662f580ff4de43b49fa82d80a4b80f8434a
SHA3-384("")
0c63a75b845e4f7d01107d852e4c2485c51a50aaaa94fc61995e71bbee983a2ac3713831264adb47fb6bd1e058d5f004
SHA3-512("")
a69f73cca23a9ac5c8b567dc185a756e97c982164fe25859e0d1dcc1475c80a615b2123af1f5f94c11e3e9402c3ac558f500199d95b6d3e301758586281dcd26
SHAKE128("", 256)
7f9c2ba4e88f827d616045507605853ed73b8093f6efbc88eb1a6eacfa66ef26
SHAKE256("", 512)
46b9dd2b0ba88d13233b3feb743eeb243fcd52ea62b81b82b50c27646ed5762fd75dc4ddd8c0f200cb05019d67b592f6fc821c49479ab48640292eacb3b7c4be
```

1비트를 변경하면 출력값의 각 비트가 50% 확률로 변경되어, 쇄도 효과 때문에 상당한 변화가 일어난다.:

```
SHAKE128("The quick brown fox jumps over the lazy dog", 256)
f4202e3c5852f9182a0430fd8144f0a74b95e7417ecae17db0f8cfeed0e3e66e
SHAKE128("The quick brown fox jumps over the lazy dof", 256)
853f4538be0db9621a6cea659a06c1107b1f83f02b13d18297bd39d7411cf10c

달라진 비트는 총 126개이며, 이는 전체 비트 수 256개의 약 49.22%이다.
```

KISA에서는 해시함수 SHA-3를 쉽게 활용할 수 있도록, 소스 코드(C/C++, Java, ASP, JSP, PHP)를 배포
https://seed.kisa.or.kr/kisa/Board/79/detailView.do

HMAC

HMAC (Hash-based Message Authentication)

- Recall MAC is for message integrity
  - The final encrypted block, CBC residue
- We can not send M and h(M) together
  - Trudy can change M to M′ and h(M) to h(M′)

- How can we solve the above problem?
- One solution: make hash depend on the Key, so called hashed MAC, HMAC
  - The key is known only to sender and receiver

HMAC (Hash-based Message Authentication)

- How to compute HMAC?
- Two obvious choices
  - ✓ Case1: h(K,M)
  - ✓ Case2: h(M,K)
- But the two cases both have potential problems

HMAC: Case 1: HMAC as h(K,M)

- Crypto Hashes hash messages in blocks
  - ✓ Ex: MD5, SHA-1, Tiger: 512 bits/block
  - ✓ If $M=(B_1,B_2)$,
    then $h(M)=F(F(A, B_1), B_2)=F(h(B_1), B_2)$ — ①
    for some func F, where A is fixed initial constant
- If $M' = (M,X)$
  - ✓ Trudy can find $h(K,M')$ from $h(K,M)$ using ① without knowing K, since
  - ✓ $h(K,M') = h(K,M,X) = F(h(K,M), X)$
    where func F is known
- So, this case has potential problem

HMAC: Case 2: HMAC as h(M,K)

- It can solve the Case1′s problem
- But, if it has collision i.e. ∃some M′ with h(M′)=h(M) then by ①
  - ✓ h(M,K)=F(h(M), K)=F(h(M′), K)=h(M′,K)
- It not so serious case since if collision occurs, then the hash func is insecure and will not be available.
- If we can eliminate these attacks, then we should do so

HMAC: The Right Way

- Prevent the potential problems by slightly modifying it as described in *RFC 2104*
- The right way for HMAC in *RFC 2104*
  - ✓ Let B be the block length of hash, in bytes
    - B = 64 byes(=512 bits) for MD5 and SHA-1 and Tiger
  - ✓ Let ipad = 0x36 repeated B times and
    opad = 0x5C repeated B times
  - ✓ Then
    HMAC(M,K) = H(K⊕opad, H(K⊕ipad, M))

HMAC: The Right Way

$$MAC(text)_t = HMAC(K, text)_t = H((K_0 \oplus opad) \,||\, H((K_0 \oplus ipad) \,||\, text))_t$$

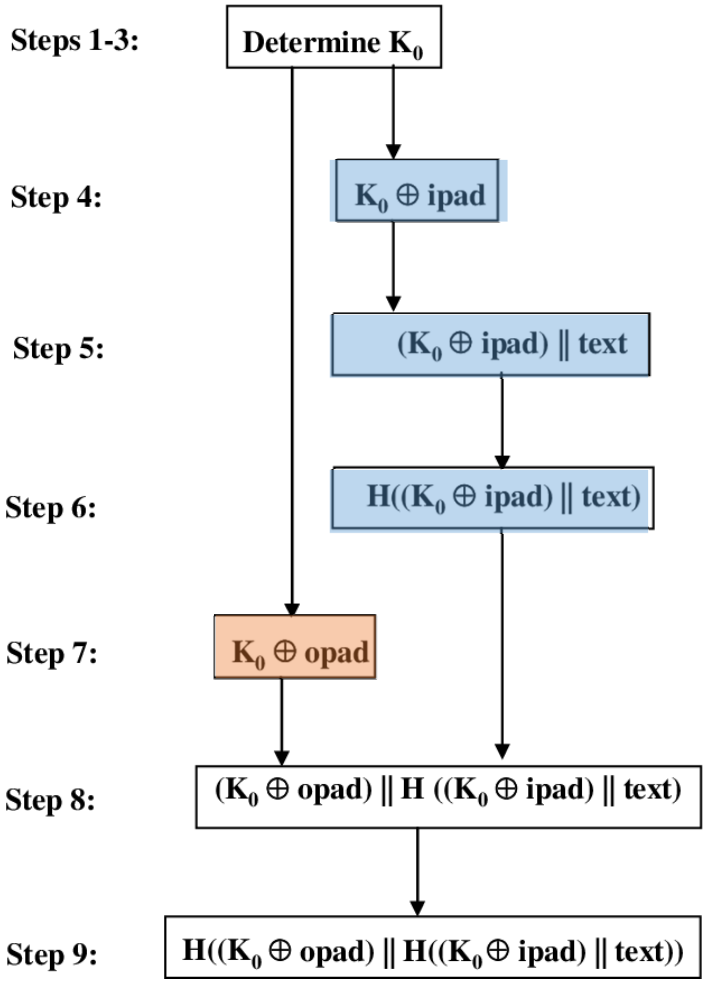
HMAC uses the following parameters:

B: Block size (in bytes) of the input to the Approved hash function.
H: An Approved hash function.
ipad: Inner pad; the byte 0x36' repeated B times.
K: Secret key shared between the originator and the intended receiver(s).
K_0: The key K after any necessary pre-processing to form a B byte key.
L: Block size (in bytes) of the output of the Approved hash function.
opad: Outer pad; the byte 0x5c' repeated B times.
t : The number of bytes of MAC.
text: The data on which the HMAC is calculated; text does not include the padded key.
The length of text is n bits, where $0 <= n < 2^B - 8B$.
x' N: Hexadecimal notation, where each symbol in the string 'N' represents 4 binary bits.
|| : Concatenation
⊕ : Exclusive-Or operation.

HMAC: The Right Way

$MAC(text)_t = HMAC(K, text)_t$
$= H((K_0 \oplus opad) \ || \ H((K_0 \oplus ipad) \ || \ text))_t$

STEPS	STEP-BY-STEP DESCRIPTION
Step 1	If the length of K = B: set K_0 = K. Else adjust K ; Step 2 –Step 3 omit
Step4	Exclusive-Or K_0 with ipad to produce a B-byte string: $\mathbf{K_0 \oplus ipad}$.
Step 5	Append the stream of data 'text' to the string resulting from step 4: $\mathbf{(K_0 \oplus ipad) \ || \ text}$.
Step 6	Apply H to the stream generated in step 5: $\mathbf{H((K_0 \oplus ipad) \ || \ text)}$.
Step 7	Exclusive-Or K_0 with opad: $\mathbf{K_0 \oplus opad}$.
Step 8	Append the result from step 6 to step 7: $\mathbf{(K_0 \oplus opad) \ || \ H((K_0 \oplus ipad) \ || \ text)}$
Step 9	Apply H to the result from step 8: $\mathbf{H((K_0 \oplus opad) \ || \ H((K_0 \oplus ipad) \ || \ text))}$.
Step 10	Select the leftmost t bytes of the result of step 9 as the MAC.



Hash Uses

4. Hash Uses

- Authentication (HMAC)
  - ✓ Password storage
- Message integrity (HMAC)
  - ✓ Blockchain
- Message fingerprint
- Data corruption detection
  - ✓ File integrity verification
- Digital signature efficiency
- Anything you can do with symmetric crypto ???

Online Auction

- Bidder (입찰자) : Alice, Bob and Charlie
- Alice plans to bid A, Bob B and Charlie C
- They don't trust that bids will stay secret
- Solution?
  - ✓ Alice, Bob, Charlie submit hashes h(A), h(B), h(C)
  - ✓ All hashes received and posted online
  - ✓ Then bids A, B and C revealed
- Hashes don't reveal bids(입찰가) (one way)
- Can't change bid after hash sent (collision)

Video Integrity Verification (VIC) Using Blockchain

S. Ghimire, J.Y. Choi, B. Lee, "Using Blockchain for Improved Video Integrity Verification," IEEE Transactions on Multimedia, 2020

- Video IVM based on a concept of a blockchain with ECC, storing the hash of video segments in a chronological order while recording videos captured using CCTV or ADR, etc. a data hash section,
- Data hash section : Generates a hash and message authentication code (MAC) with the HMAC algorithm for each video segment. The video segments are hashed by a secure hash algorithm (SHA-256), followed by the HMAC algorithm with a data HMAC key (denoted as *dK*), ensuring the confidentiality and authenticity of the video.
- Key encryption section, block key generation section, and blockchain storage section.