해시 함수

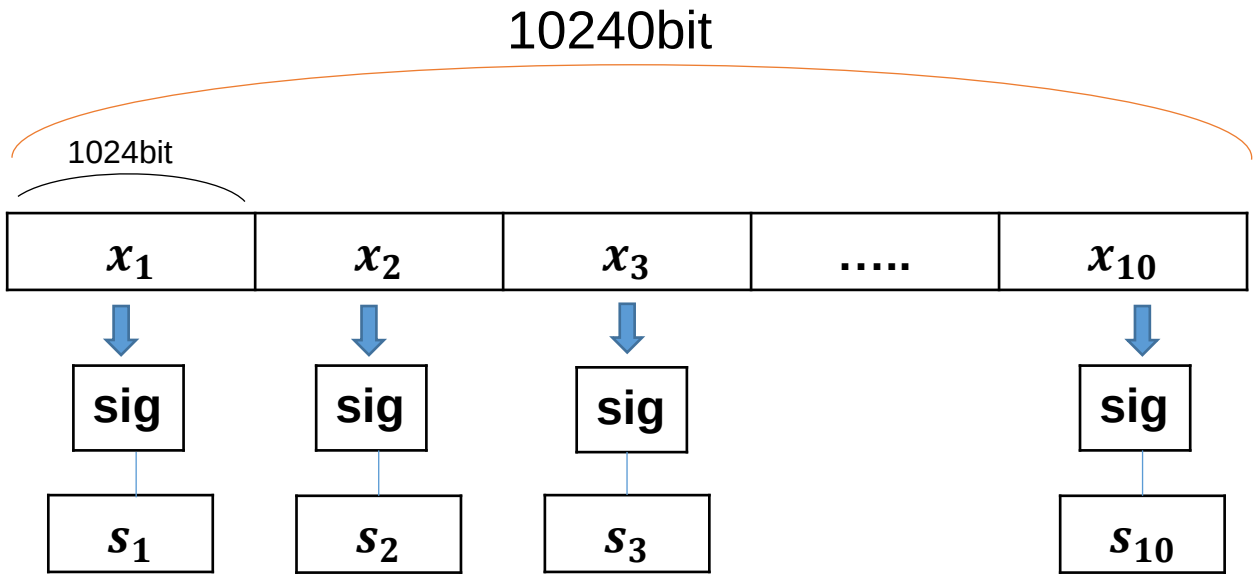
https://youtu.be/pbcHjQJhRoA

Contents

CryptoCraft LAB

해시함수 필요성

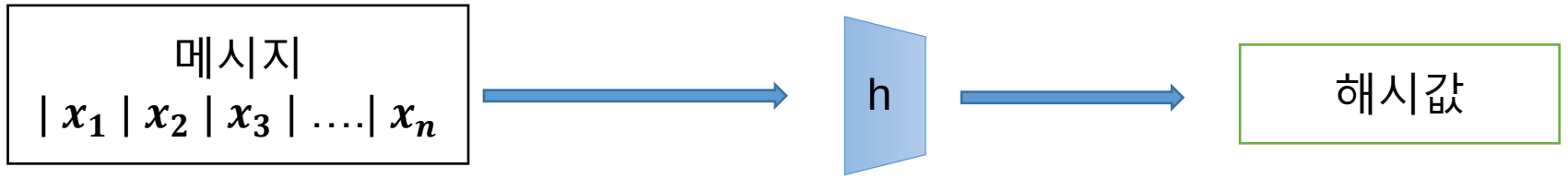
- 긴 메시지 서명



1.높은 계산부담
2.메시지 오버헤드(2배)
(한 메시지 당 1개의 서명 필요)

⇒ 해시함수 필요

해시함수 원리



Variable 가변길이

Fixed (고정길이)

Alice

Bob

$z = h(x)$

$s = sig_{k_{pr,B}}(z)$

(x,s)

$z' = h(x)$

$ver_{k_{pub,B}}(s, z') =$ 참/거짓

해시함수 안전성

1. 역상저항성(preimage resistance) or 일방향성(one-way ness)

$x \rightarrow h(x) \rightarrow z \quad / \; x = h'(z)$ → infeasible (거의 불가능)

2. 약한 충돌 저항성(weak collision resistance)

$x_1 \rightarrow h(x) \rightarrow z_1$

$x_2 \rightarrow h(x) \rightarrow z_2$

$x_1 \neq x_2 \,,\; h(x_1) = h(x_2)$

3.강한 충돌 저항성(strong collision resistance)

$x_1 \rightarrow h(x) \rightarrow z_1$

$x_2 \rightarrow h(x) \rightarrow z_2$

$x_1 \neq x_2 \,,\; h(x_1) = h(x_2)$

강한 충돌저항성

- 생일 공격 문제와 밀접.

P(두명의 생일이 같지 않음) = $1 - \frac{1}{365}$

P(세명의 생일이 같지 않음) = $\left(1 - \frac{1}{365}\right) * \left(1 - \frac{2}{365}\right)$

P(충돌 x) = $\left(1 - \frac{1}{365}\right) * \left(1 - \frac{2}{365}\right) * \ldots \left(1 - \frac{t-1}{365}\right)$

P(최소 한번의 충돌) = $1 - \left(1 - \frac{1}{365}\right) * \left(1 - \frac{2}{365}\right) * \ldots \left(1 - \frac{t-1}{365}\right) \geq 0.5$ t=23

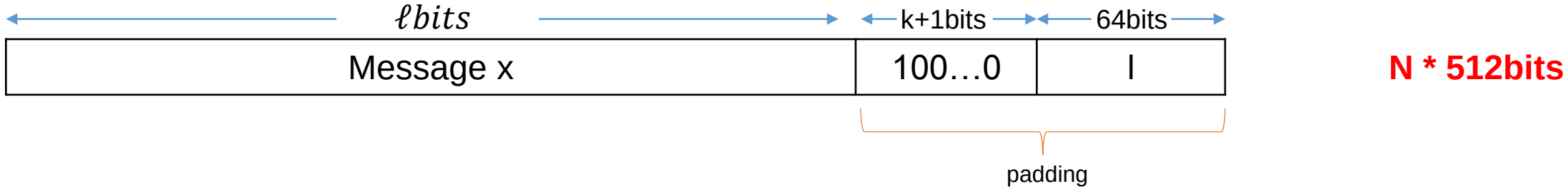
- 해시 함수 충돌 탐색

- P(충돌x)=$\left(1 - \frac{1}{2^n}\right) * \left(1 - \frac{2}{2^n}\right) * \ldots \left(1 - \frac{t-1}{2^n}\right) = \Pi_{i=1}^{t-1}\left(1 - \frac{i}{2^n}\right) \approx \Pi_{i=1}^{t-1} e^{-\frac{i}{2^n}} = e^{-\frac{t(t-1)}{2^{n+1}}}$
- 적어도 한번 이상 충돌
- $\lambda = 1 - P(\text{충돌x}) \approx 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$
- $t \approx \sqrt{2^{n+1}\ln\left(\frac{1}{1-\lambda}\right)}$

SHA

알고리즘		결과[비트]	입력[비트]	라운드 수	충돌 탐색 여부
MD5		128	512	64	O
SHA-1		160	512	80	O
SHA-2	SHA-224	224	512	64	X
	SHA-256	256	512	64	X
	SHA-384	384	1024	80	X
	SHA-512	512	1024	80	X

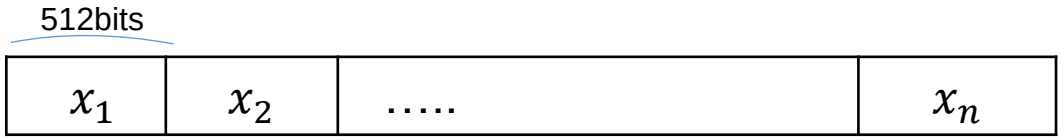
SHA-1

- 

N * 512bits

$$k \equiv 512 - 64 - 1 - \ell = 448 - (\ell + 1)\ mod\ 512$$

- Padding 이후 x



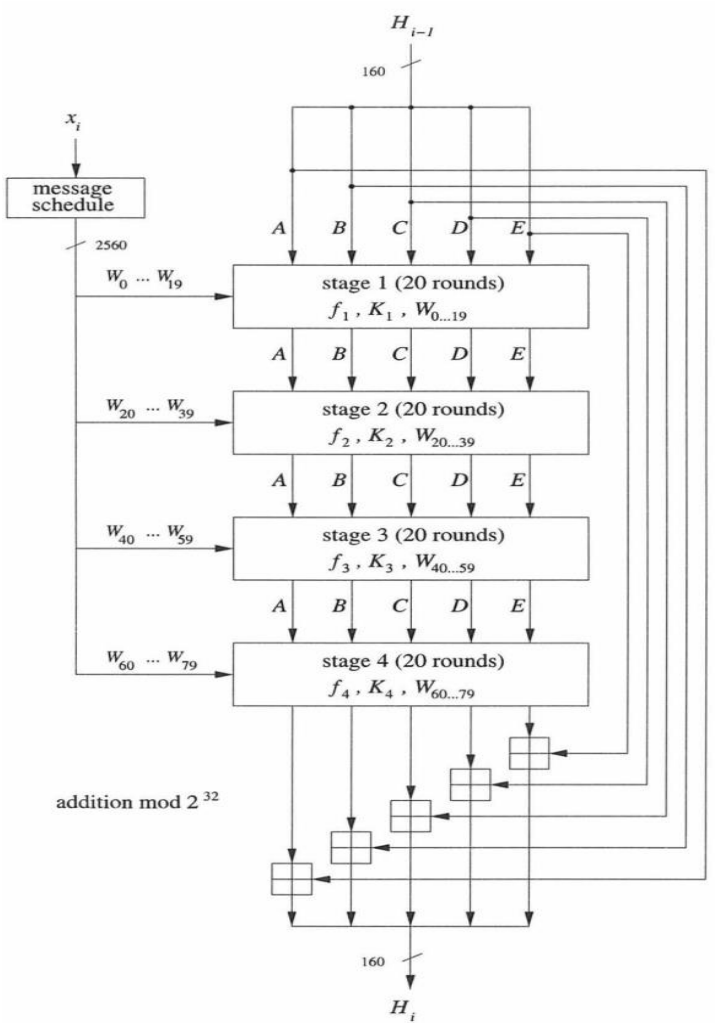
word

- $x_i = (x_i^{(0)}, x_i^{(1)}, \ldots\ldots, x_i^{(15)})$ $x_i^{(k)}$ 는 32 비트 워드이다. 1개의 블록 = 32(bits) * 16(word)

- 초기 값 H_0

$$A = H_0^{(0)} = 67452301$$
$$B = H_0^{(1)} = \mathrm{EFCDAB89}$$
$$C = H_0^{(2)} = 98BADCFE$$
$$D = H_0^{(3)} = 10325476$$
$$\mathrm{E} = H_0^{(4)} = C3D2E1F0$$

SHA-1



$$W_j = \begin{cases} x_i^{(j)} & 0 \le j \le 15 \\ \left(W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3}\right)_{\lll 1} & 16 \le j \le 79 \end{cases}$$

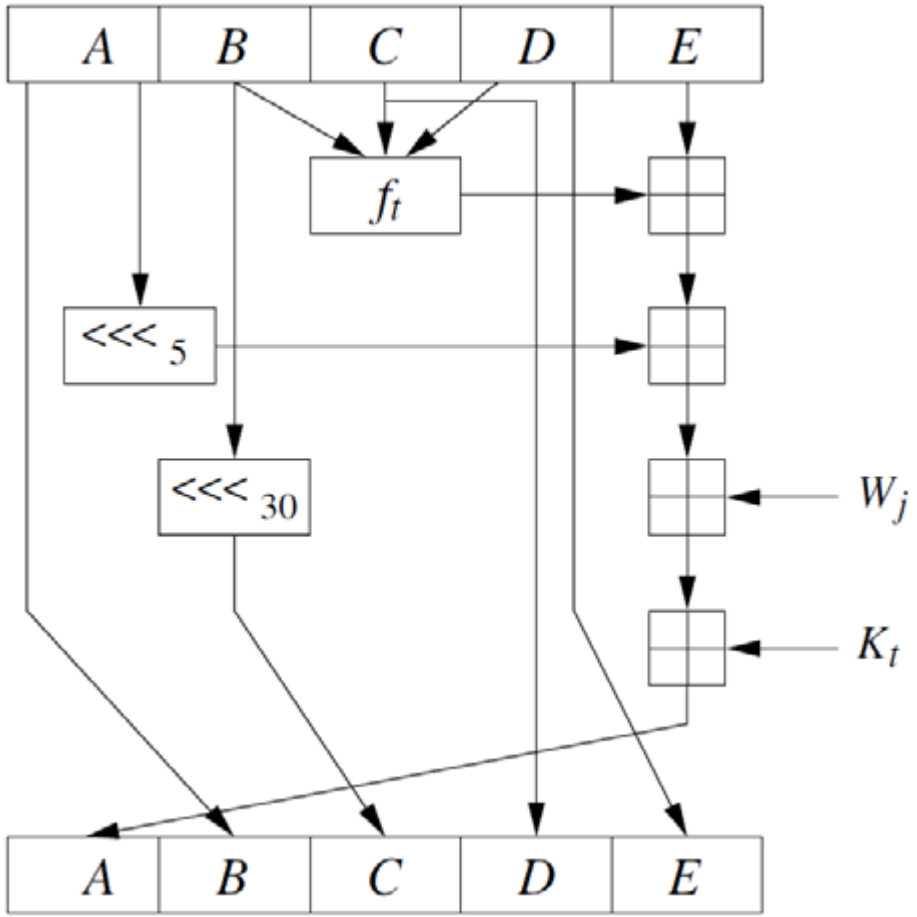
$W_0 \sim W_{79}$: 80라운드 각 라운드에 하나씩 사용

각 스테이지에 20개 words 인 이유:

⇒20라운드를 돌기위해

SHA-1

- SHA-1의 단계 t의 라운드 j



- 초기 값 H_0

$$A = H_0^{(0)} = 67452301$$
$$B = H_0^{(1)} = \text{EFCDAB89}$$
$$C = H_0^{(2)} = 98BADCFE$$
$$D = H_0^{(3)} = 10325476$$
$$\text{E} = H_0^{(4)} = C3D2E1F0$$

$$A, B, C, D, E$$
$$= (E + f_t(B, C, D) + (A)_{\lll 5} + W_j + K_t, A, (B)_{\lll 30}, C, D$$

Stage t	Round j	Constant K_t	Function f_t
1	0…19	$K_1 = 5A827999$	$f_1(B,C,D) = (B \wedge C) \vee (B \wedge D)$
2	20…39	$K_2 = 6ED9EBA1$	$f_2(B,C,D) = B \oplus C \oplus D$
3	40…59	$K_3 = 8F1BBCDC$	$f_3(B,C,D)$ $= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
4	60…79	$K_4 = CA62C1D6$	$f_4(B,C,D) = (B \wedge C) \vee (B \wedge D)$

SHA-1 구현

```cpp
int main(int argc, char* argv[]) {
    if (argc < 2) {
        fprintf(stderr, "usage: %s {filename}\n", argv[0]);
        return 0;
    }

    m = fileContents(argv[1]);

    padding();

    var64 chunkNums = m.size() / CHUNK_BYTES;    //512bits 크기의 블록 개수
    for (unsigned int i = 0; i < chunkNums; i++) {
        processChunck(i);
    }

    // Hex std::string
    ostringstream hh;
    for (unsigned int i = 0; i < DIGEST_NUMS; i++) {
        hh << hex << setfill('0') << setw(8);
        hh << h[i];
    }

    cout << hh.str() << endl;
}
```

```cpp
using var64 = uint64_t;

// CHUNK_BYTES = WORDS_BYTES * WORDS_NUMS
constexpr unsigned int CHUNK_BYTES = 64; // 512 bit
constexpr unsigned int WORDS_BYTES = 4; // 32 bit
constexpr unsigned int WORDS_NUMS = 16;
constexpr unsigned int DIGEST_NUMS = 5;

string m;    //암호화 할 데이터
var64 ml;         //m의 원래 크기 bit size로 나타냄

void padding() {
// Step1
    // m에 padding을 추가하기 전 원래 메시지 length (bit size)
    ml = m.size()*8;

    // message에 '1' 비트 1개 추가 (실제론 0b10000000 추가)
    // 이 다음 과정에서 m + ml이 64bytes(512bit)로 나누어 떨어지도록
    // '0'비트를 계속 추가하기 때문에 미리 byte 단위로 추가 함.
    m += (char)0x80;

    while (m.size() % CHUNK_BYTES != CHUNK_BYTES - sizeof(ml)) {
        m += (char)0x00;
    }

    for (int i=7; i>=0; i--) {
        char byte = (ml >> 8*i) & 0xff;
        m += byte;
    }

    // 이제 m은 64bytes(512bit) 즉 CHUNK_BYTES로 나누어 떨어진다.
}
```

SHA-1 구현

```
59  vector<var> chunkToWords(const string& chunk) {
60      vector<var> words(WORDS_NUMS);
61
62  // Step2
63      for (unsigned int i = 0; i < WORDS_NUMS; i++) {
64          words[i] = (chunk[4*i+3] & 0xff)
65                   | (chunk[4*i+2] & 0xff)<<8
66                   | (chunk[4*i+1] & 0xff)<<16
67                   | (chunk[4*i+0] & 0xff)<<24;
68      }
69
70      return words;
71  }
72
73  //초기값 H=IV(A,B,C,D,E)
74  var h[DIGEST_NUMS] = {
75      0x67452301,
76      0xEFCDAB89,
77      0x98BADCFE,
78      0x10325476,
79      0xC3D2E1F0,
80  };
81
82  void processChunck(unsigned int index) { //index는 블록number (총 블록개수만큼 이 함수가 호출됨.)
83      string chunk = m.substr(index * CHUNK_BYTES, CHUNK_BYTES);
84      vector<var> w = chunkToWords(chunk);
```

```
86  // Step3
87      // Extend the sixteen 32-bit words into eighty 32-bit words:
88      for (unsigned int i = 16; i <= 79; i++) {
89          var word = w[i-3] xor w[i-8] xor w[i-14] xor w[i-16];
90          word = BYTE_ROTATE_LEFT32(word, 1);
91          w.push_back(word);
92      }
93
94      // Initialize hash value for this chunk:
95      var a = h[0];
96      var b = h[1];
97      var c = h[2];
98      var d = h[3];
99      var e = h[4];
100     var f, k;
101
```

SHA-1 구현

```
104     for (unsigned int i = 0; i <= 79; i++) {
105         if (0 <= i && i <= 19) {
106             f = (b bitand c) bitor (~b bitand d);
107             k = 0x5A827999;
108         }
109         else if (20 <= i && i <= 39) {
110             f = b xor c xor d;
111             k = 0x6ED9EBA1;
112         }
113         else if (40 <= i && i <= 59) {
114             f = (b bitand c) bitor (b bitand d) bitor (c bitand d);
115             k = 0x8F1BBCDC;
116         }
117         else if (60 <= i && i <= 79) {
118             f = b xor c xor d;
119             k = 0xCA62C1D6;
120         }
121
122 //A,B,C,D,E -> (E+f+(A)<<<6+Wj+ Kt),A,(B)<<<30,C,D
123         var temp = BYTE_ROTATE_LEFT32(a, 5) + f + e + k + w[i];
124         e = d;
125         d = c;
126         c = BYTE_ROTATE_LEFT32(b, 30);
127         b = a;
128         a = temp;
129     }
130
131     // Add this chunk's hash to result so far:
132     h[0] = h[0] + a;
133     h[1] = h[1] + b;
134     h[2] = h[2] + c;
135     h[3] = h[3] + d;
136     h[4] = h[4] + e;
137 }
```

- 출처 https://github.com/JangTaeHwan/AlgorithmTraining/blob/master/Mathematical/sha/sha1.cpp

Q & A