해시 함수

유튜브:https://youtu.be/60tUuWMpMno

Contents

SHA 해시함수

SHA-2 / SHA 256

SHA256 구현

SHA-3

CryptoCraft LAB

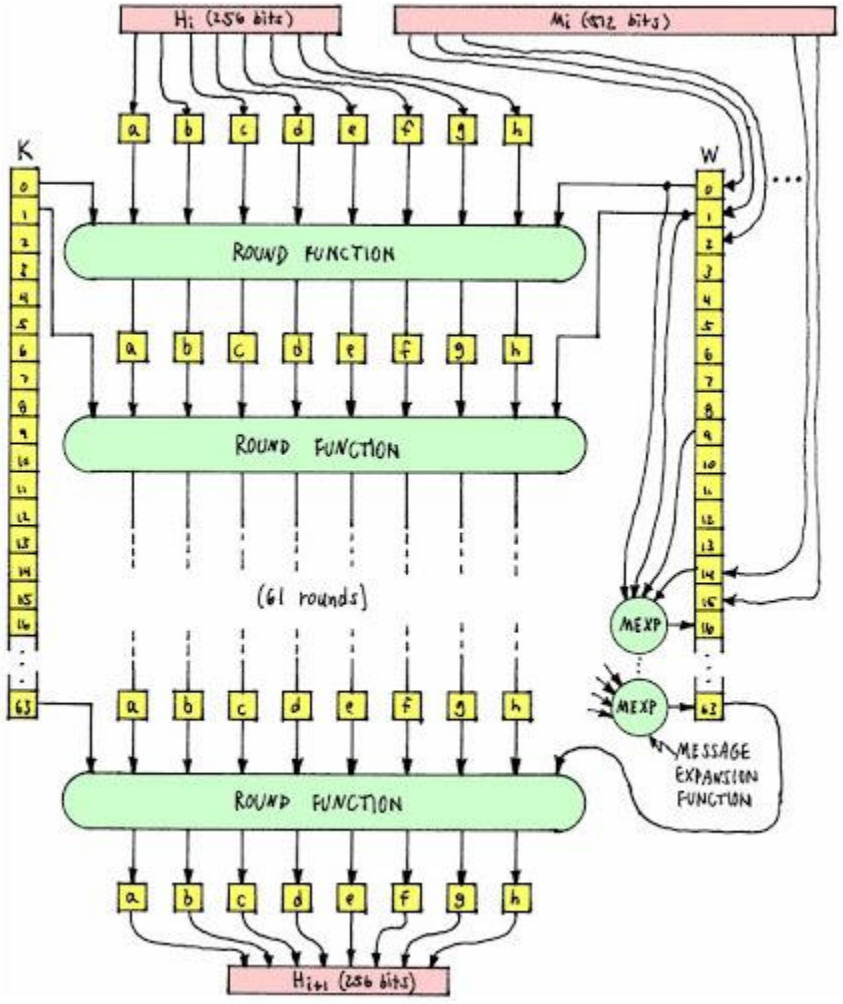
1. SHA

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

+

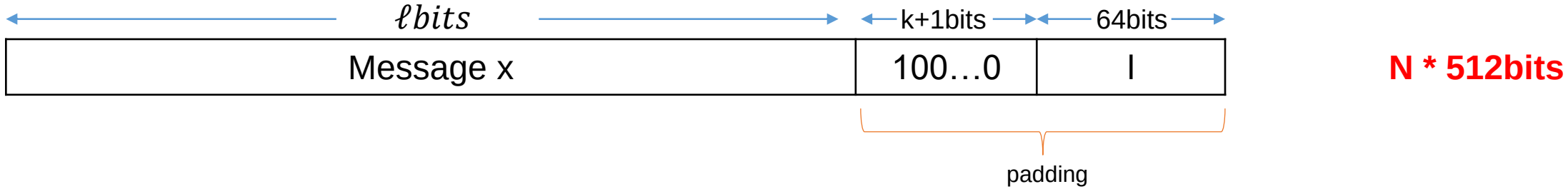
SHA-3	SHA-224
	SHA-256
	SHA-384
	SHA-512

SHA-256



- Message Padding
- 초기값 H
- 64개의 Word
- 64개의 고정된 상수 값 K
- 총 64번의 Round Function

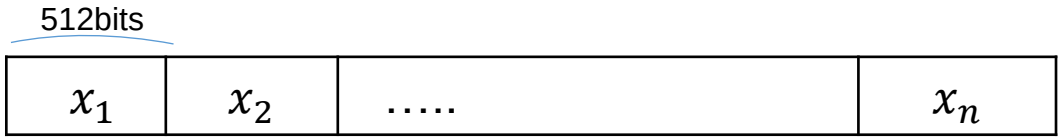
SHA-256

- 

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)
  = 하나의 chunk

초기 값 상수 H(0)

- $H_0 = [a, b, c, d, e, f, g, h]$
  =32bit(4byte) x 8 =256bit

- 가장 작은 소수 8개에 루트를 씌운 후 소수점 아래 32비트.

$$H_0^{(0)} = \texttt{6a09e667}$$
$$H_1^{(0)} = \texttt{bb67ae85}$$
$$H_2^{(0)} = \texttt{3c6ef372}$$
$$H_3^{(0)} = \texttt{a54ff53a}$$
$$H_4^{(0)} = \texttt{510e527f}$$
$$H_5^{(0)} = \texttt{9b05688c}$$
$$H_6^{(0)} = \texttt{1f83d9ab}$$
$$H_7^{(0)} = \texttt{5be0cd19}$$

상수 값 K

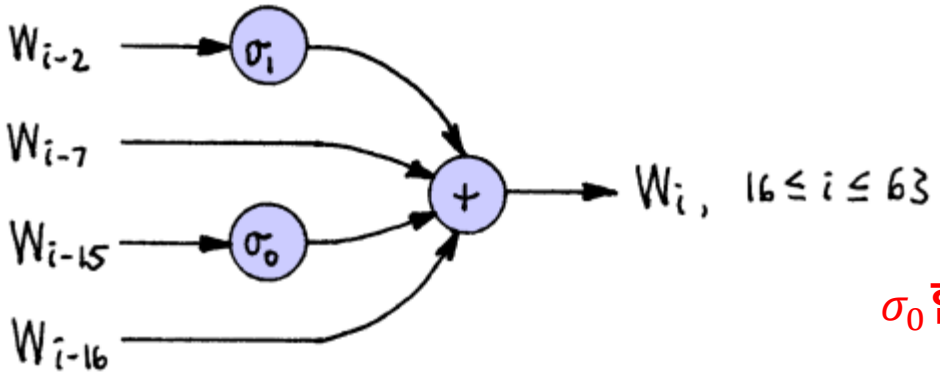
- K= 32bit x 64
- 만드는 법:

  가장 작은 소수 64개에 세제곱근을 취한 후, 소수점 아래 32bit

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Word

- $W_j = \begin{cases} x_i^{(j)} & 0 \le j \le 15 \\ MEXP(\,) & 16 \le j \le 63 \end{cases}$



MEXP(Message Expansion Function)

$=\sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{i-15}) + W_{i-16}$

σ_0 함수

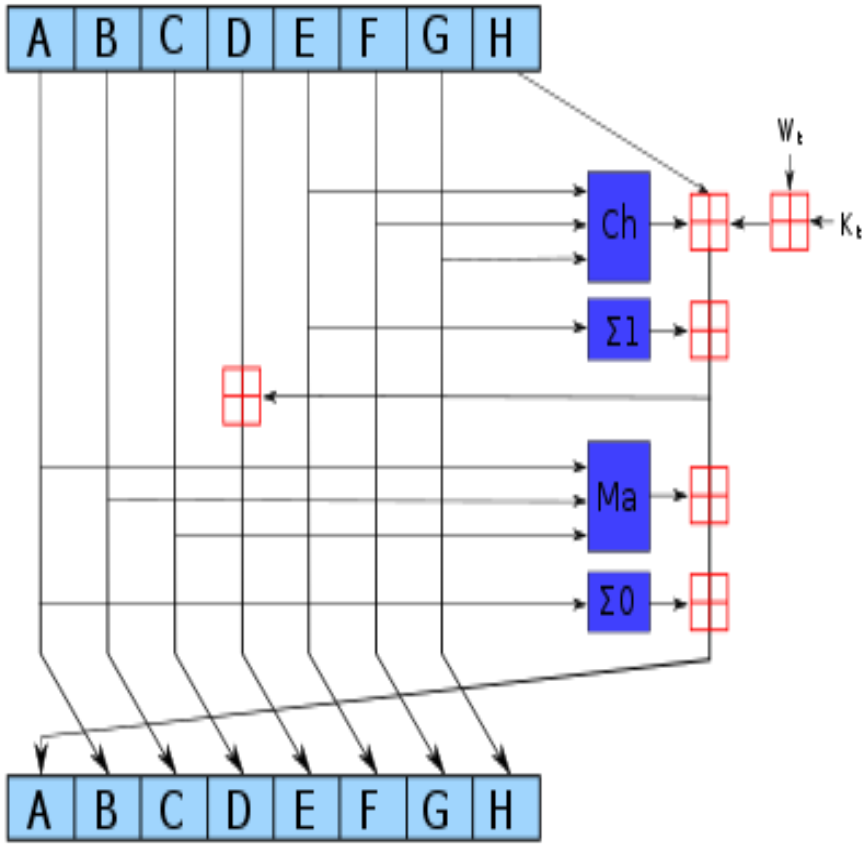
= (X right-rotate 7) xor (X right-rotate 18) xor (X right-shift 3)

σ_1 함수

= (X right-rotate 17) xor (X right-rotate 19) xor (X right-shift 10)

Round Function

- SHA-2 의 Round Function



- Σ0(X) = (X right-rotate 2) xor (X right-rotate 13) xor (X right-rotate 22)
- Σ1(X) = (X right-rotate 6) xor (X right-rotate 11) xor (X right-rotate 25)
- Ch(X,Y,Z) = (X and Y) xor ((not X) and Z)
- Maj(X,Y,Z) = (X and Y) xor (X and Z) xor (Y and Z)

- ch =Choose(E,F,G)
- s= K +Sigma1(E)+ ch +H +w
- maj =Majority(A,B,C)
- A' = Sigma0(A) +maj +s
- E' = D + s

SHA 256 구현

```
168  // Add padding to message.
169  void PreProcess(vector<uint8_t>& message)
170  {
171      auto L = static_cast<uint64_t>(message.size());
172
173      // Append single '1' bit and seven '0' bits.
174      message.push_back(0b10000000);
175
176      // Append (K * 8) '0' bits, where L + 1 + K + 8 is a multiple of 64.
177      auto K = 64 - (((L % 64) + 9) % 64);
178      if (K == 64) K = 0;
179
180      for (int i = 0; i < K; ++i)
181      {
182          message.push_back(0);
183      }
184
185      // Append the bit length of original message.
186      assert(L <= UINT64_MAX / 8);
187      uint64_t bitLengthInBigEndian = ChangeEndian(L * 8);
188      auto ptr = reinterpret_cast<uint8_t*>(&bitLengthInBigEndian);
189
190      message.insert(end(message), ptr, ptr + 8);
191      assert(message.size() % 64 == 0);
```

```
194  array<uint32_t, 8> Process(vector<uint8_t> const& message)
195  {
196      assert(message.size() % 64 == 0);
197
198      const auto K = Make_K();
199      const auto blockCount = message.size() / 64;
200
201      auto digest = Make_H0();
202
203      for (int i = 0; i < blockCount; ++i)
204      {
205          auto W = Make_W(reinterpret_cast<const uint8_t(&)[64]>(message[i * 64]));
206          auto H = digest;
207
208          for (int r = 0; r < 64; ++r)
209          {
210              H = Round(H, K[r], W[r]);
211          }
212
213          for (int i = 0; i < 8; ++i)
214          {
215              digest[i] += H[i];
216          }
217      }
218
219      return digest;
220  }
```

SHA 256 구현

```
31  uint32_t RotateRight(uint32_t x, uint32_t n)
32  {
33      return (x >> n) | (x << (32 - n));
34  }
35
36  // σ0 (Small Sigma 0)
37  uint32_t SSigma_0(uint32_t x)
38  {
39      return RotateRight(x, 7) ^ RotateRight(x, 18) ^ (x >> 3);
40  }
41
42  // σ1 (Small Sigma 1)
43  uint32_t SSigma_1(uint32_t x)
44  {
45      return RotateRight(x, 17) ^ RotateRight(x, 19) ^ (x >> 10);
46  }
```

```
48  // Σ0 (Big Sigma 0)
49  uint32_t BSigma_0(uint32_t x)
50  {
51      return RotateRight(x, 2) ^ RotateRight(x, 13) ^ RotateRight(x, 22);
52  }
53
54  // Σ1 (Big Sigma 1)
55  uint32_t BSigma_1(uint32_t x)
56  {
57      return RotateRight(x, 6) ^ RotateRight(x, 11) ^ RotateRight(x, 25);
58  }
59
60  // Let X, Y, Z to a bit of x, y, z each.
61  // If X == 1, take Y. Otherwise take Z.
62  uint32_t Choose(uint32_t x, uint32_t y, uint32_t z)
63  {
64      return (x & y) ^ (~x & z);
65  }
66
67  // Let X, Y, Z to a bit of x, y, z each.
68  // Take a bit of majority among X, Y, and Z.
69  uint32_t Majority(uint32_t x, uint32_t y, uint32_t z)
70  {
71      return (x & y) ^ (x & z) ^ (y & z);
72  }
```

SHA 256 구현

```
78  array<uint32_t, 8> Make_H0()
79  {
80      const double kPrimeList[] = { 2, 3, 5, 7, 11, 13, 17, 19 };
81      static_assert(sizeof(kPrimeList) / sizeof(*kPrimeList) == 8, "");
82
83      array<uint32_t, 8> H;
84
85      for (int i = 0; i < 8; ++i)
86      {
87          auto v = sqrt(kPrimeList[i]);
88
89          v -= static_cast<uint32_t>(v);
90          v *= pow(16, 8);
91
92          H[i] = static_cast<uint32_t>(v);
93      }
94
95      return H;
96  }
```

```
98  array<uint32_t, 64> Make_K()
99  {
100     double kPrimeList[] = {
101         2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
102         31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
103         73, 79, 83, 89, 97, 101, 103, 107, 109, 113,
104         127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
105         179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
106         233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
107         283, 293, 307, 311
108     };
109     static_assert(sizeof(kPrimeList) / sizeof(*kPrimeList) == 64, "");
110
111     array<uint32_t, 64> K;
112
113     for (int i = 0; i < 64; ++i)
114     {
115         auto v = cbrt(kPrimeList[i]);
116
117         v -= static_cast<uint32_t>(v);
118         v *= pow(16, 8);
119
120         K[i] = static_cast<uint32_t>(v);
121     }
122
123     return K;
124 }
125
```

SHA 256 구현

```
126 array<uint32_t, 64> Make_W(const uint8_t (&M)[64])
127 {
128     array<uint32_t, 64> W;
129
130     for (int i = 0; i < 16; ++i)
131     {
132         W[i] = ChangeEndian(reinterpret_cast<uint32_t const&>(M[i * 4]));
133     }
134
135     for (int i = 16; i < 64; ++i)
136     {
137         // MEXP (Message Expansion Function)
138         W[i] = SSigma_1(W[i - 2]) + W[i - 7] + SSigma_0(W[i - 15]) + W[i - 16];
139     }
140
141     return W;
142 }
143
```

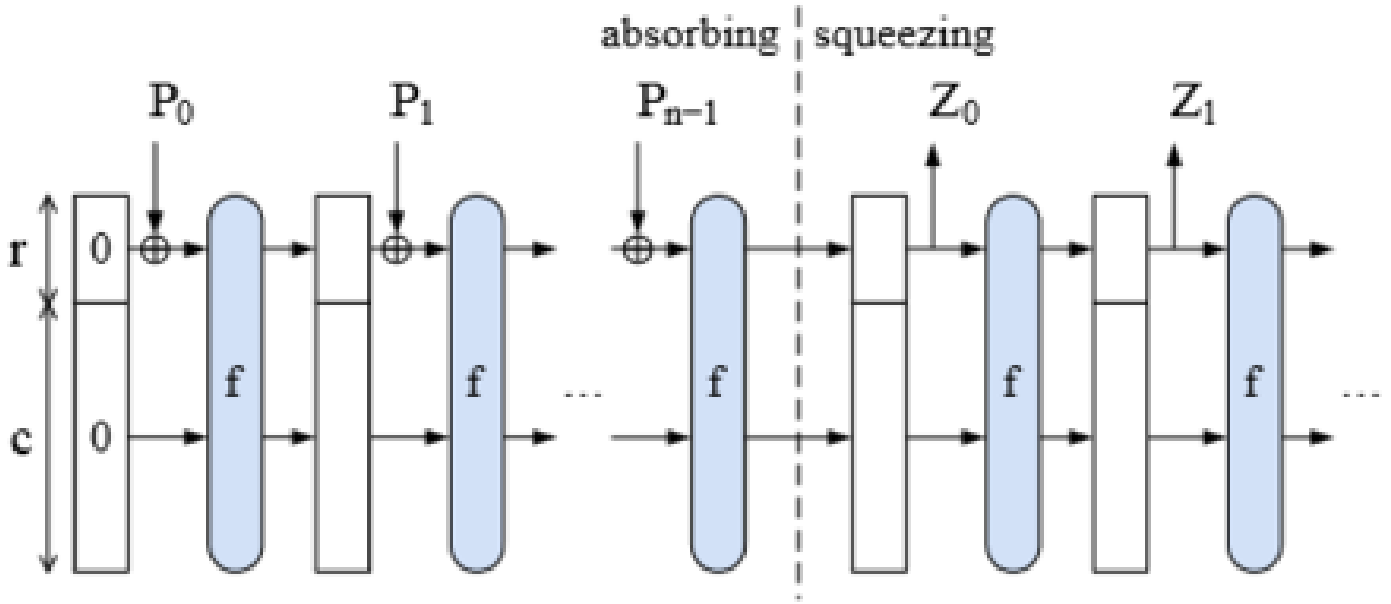
```
144 array<uint32_t, 8> Round(array<uint32_t, 8> const& H, uint32_t K, uint32_t W)
145 {
146     array<uint32_t, 8> nH; // next H
147
148     auto a= H[0];
149     auto b= H[1];
150     auto c= H[2];
151     auto d= H[3];
152     auto e= H[4];
153     auto f= H[5];
154     auto g= H[6];
155     auto h= H[7];
156
157     auto maj = Majority(a, b, c);
158     auto ch = Choose(e, f, g);
159     auto s = K + BSigma_1(e) + ch + h + W;
160
161     nH[0] = BSigma_0(a) + maj + s;
162     nH[1] = a;
163     nH[2] = b;
164     nH[3] = c;
165     nH[4] = d + s;
166     nH[5] = e;
167     nH[6] = f;
168     nH[7] = g;
169
170     return nH;
171 }
```

- 출처
https://github.com/taeguk/blockchain_study/blob/master/hash/sha256.cpp

SHA-3

- SHA1, SHA2 와 매우 다른 알고리즘
- 공개적인 방식을 통해 모집한 알고리즘으로 선정
- Keccak 알고리즘
- 스펀지 구조

absorbing phase / squeezing phase



SHA-3로의 전환

- **2015/8/5 NIST 에서 SHA-3 암호화 해시 함수 표준 발표**
- **Sha-1 에서 sha-2로의 마이그레이션: 2016~ 2017**

=> SHA-1에서 SHA-3로 바로 마이그레이션 되지 않은 이유?

1) SHA-3 를 지원하는 소프트웨어 및 하드웨어 전무
2) SHA-2 에 비해 느린 속도

Q & A