# C | SHA
## - Implementing Secure Hash Algorithm in C -

Ji Yong-Hyeon

**Department of Information Security, Cryptology, and Mathematics**
College of Science and Technology
Kookmin University

December 25, 2023

# Acknowledgements

# Contents

# Chapter 1

# Hash Function

## 1.1 Cryptographic Hash Function

## 1.2 Hash Function Structure

### 1.2.1 Padding

To encrypt a message of any length in '$\lambda$' bits, the message must first be divided into segments, each exactly $\lambda$ bits in length.

$$m \longrightarrow \underbrace{m \parallel \text{pad}}_{t \cdot \lambda - \text{bit}}.$$

Let $k = \text{Bitlen}(m)$. Then

| Type of Padding | Definition | Application |
|---|---|---|
| Zeros | $m \parallel 0^{\lambda-(k \mod \lambda)}$ | |
| One-Zeros | $m \,\|\|\, 1 \,\|\|\, 0^{l-1-(\text{Bitlen}(m) \mod l)}$ | LSH |
| One-Zeros-Bitlen | $m \,\|\|\, 1 \,\|\|\, 0^{l-1-(\text{Bitlen}(m) \mod l)}$ | MD5, SHA-1, SHA-2 |
| Zeros-Bitlen | $m \,\|\|\, 1 \,\|\|\, 0^{l-1-(\text{Bitlen}(m) \mod l)}$ | |
| One-Zeros-One | $m \,\|\|\, 1 \,\|\|\, 0^{l-1-(\text{Bitlen}(m) \mod l)}$ | SHA-3 |

### 1.2.2 Merkle-Damgård Transform

Consider a function

$$f : \{0,1\}^{n+\lambda} = \{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^n.$$

**Algorithm 1:** Hash Function based on Merkle-Damgård Transformation

**Input:** Input message $M \in \{0,1\}^*$
**Result:** Hash value of the input message $H \in \{0,1\}^n$

```
M_1, M_2, ..., M_t ← Pad(M);                // M_i ∈ {0,1}^λ
H ← IV;                                     // Initialize Chaining Variable
for i ← 1 to n do
    H ← f(H, M_i);                          // Compression Function
end
return H;
```

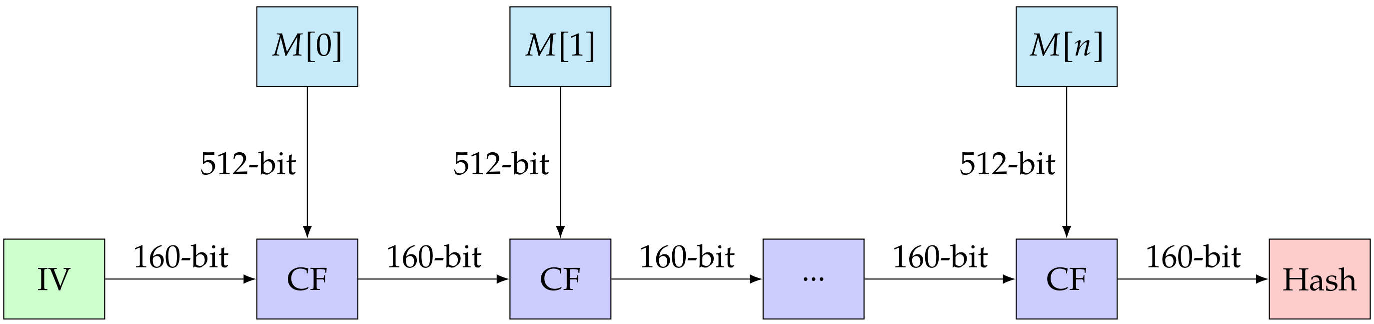
# Chapter 2

# SHA Family

| Algorithm | Year | Developer | Design | Status |
|---|---|---|---|---|
| SHA-0 | 1993 | NSA | MD+ARX | Broken |
| SHA-1 | 1995 | NSA | MD+ARX | Broken |
| SHA-2 | 2001 | NSA | MD+ARX | |
| SHA-3 | 2015 | Industry | Sponge | |

Table 2.1: SHA Algorithm Versions

## 2.1 SHA-1




Code 2.1: Key Expansion in C (General ver.)

```c
// Define the SHA1 message digest structure
typedef struct {
    uint32_t state[5];
    uint32_t count[2];
    uint8_t buffer[64];
} sha1_t;

// Initialize the SHA1 message digest with a given seed
void sha1_init(sha1_t *sha1, uint32_t seed) {
    memset(sha1->state, 0, sizeof(sha1->state));
    sha1->count[0] = seed;
    sha1->count[1] = (seed >> 8) | ((seed & 0xff) << 24);
}

```

```
// Update the SHA1 message digest with a given block of data
void sha1_update(sha1_t *sha1, const void *data, size_t len) {
    while (len >= 64) {
        uint32_t words[16];
        for (int i = 0; i < 16; i++) {
            words[i] = ((uint32_t *)data)[i];
        }
        sha1->state[0] += words[0];
        sha1->state[1] += words[1];
        sha1->state[2] += words[2];
        sha1->state[3] += words[3];
        sha1->state[4] += words[4];
        for (int i = 0; i < 64; i++) {
            sha1->buffer[i] ^= ((uint8_t *)&words[i])[i & 3];
        }
        len -= 64;
        data += 64;
    }
}

// Finalize the SHA1 message digest and return the resulting hash
void sha1_final(sha1_t *sha1, uint8_t *hash) {
    sha1->state[0] = (sha1->state[0] & 0xff000000) | ((sha1->state[0] >> 24) & 0x00ffffff);
    sha1->state[1] = (sha1->state[1] & 0x00ffffff) | ((sha1->state[1] << 8) & 0xff000000);
    sha1->state[2] = (sha1->state[2] & 0x00ffffff) | ((sha1->state[2] << 16) & 0xff000000);
    sha1->state[3] = (sha1->state[3] & 0x00ffffff) | ((sha1->state[3] << 24) & 0xff000000);
    sha1->count[0] += 64;
    for (int i = 0; i < 5; i++) {
        hash[i] = (uint8_t)(sha1->state[i] >> 24);
        hash[i + 4] = (uint8_t)(sha1->state[i] >> 16);
        hash[i + 8] = (uint8_t)(sha1->state[i] >> 8);
        hash[i + 12] = (uint8_t)sha1->state[i];
    }
}
```

# Appendix A

# Additional Data A

## A.1 Substitution-BOX

```
static const u8 s_box[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
    0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
    0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
    0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
    0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
    0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
    0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
    0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
    0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
    0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
    0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
    0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
    0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
    0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
    0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
    0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
    0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

```c
static const u8 inv_s_box[256] = {
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
    0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
    0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
    0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
    0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
    0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
    0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a,
    0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
    0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
    0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
    0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
    0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
    0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
    0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
    0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
    0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
    0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};
```