Cryptographic Hash Documentation

MD5 SHA-1 SHA-256 SHA-512
SHA-224 SHA-384

Molly Taylor
42 Silicon Valley
August 2019

Contents

Chapter 1

Glossary

message: The input; the data you are hashing.

- **padded message:** The message with padding added to the end so that it is of a length that divides evenly into the block size.
- **block:** One n-bits portion of the padded message.
- **words:** The block, usually after undergoing a transformation. Typically the first 16 words are just the block, and the rest are derived from those first 16. This is the form of the block used by the compression function.

digest: The bitfield that is transformed throughout the hashing algorithm. Because it's ever-changing, this a somewhat vague term. Also sometimes called the **hash**.

- **internal state:** The part that is initialized at the beginning, and then added to after each compression of a data block.
- **checksum:** The output, the final internal state converted to hexadecimal.

compression function: The real guts of a cryptographic hashing algorithm. It runs once per block of message. It takes in the current internal state, and a set of words (ie the block). It spits out a set of numbers which are then added back into the internal state.

rounds: The sets of transformations that happen each time the compression function is run.

1.1 Variable naming conventions

- The internal state is usually an array called `h[]` (h for *hash*)
- The working variables inside the compression function are typically lettered: `a`, `b`, `c`, `d`, etc
- The array of constants used within the compression function are called `k[]`
- The words are usually called `w[]` (w for *words*)

Chapter 2

Merkle-Damgård construction

2.1 Pseudocode

Algorithm 1 Merkle-Damgård construction

```
1: function MERKLE-DAMGÅRD CONSTRUCTION(inputfd)
2:     hash = initialize hash
3:     while (block = get padded block(inputfd)) do
4:         words = word function(block)
5:         hash = compression function(hash, words)
6:     return hash
```

Algorithm 2 Compression function

```
1: function COMPRESSION FUNCTION(hash, words)
2:     workingvariables = hash
3:     for rounds, do
4:         [...]                                  ▷ algorithm specific stuff
5:     hash = hash + workingvariables
6:     return hash
```

Chapter 3

Message padding

Message padding goes as follows:

1. All the bytes of message
2. The byte `10000000`
3. Zeros to make up the length so it divides evenly into the block size
4. The number of bits in the message portion

The way the number of bits is formatted depends on the algorithm.

1. If it's a little-endian algorithm, the number is given in little algorithm. If it's a big-endian algorithm (like the SHA family), the number is given in big algorithm.
2. The number is unusually given as a 64-bit number, but it depends on the function.

Chapter 4

Helper functions

These are functions used in the hashing algorithms.

4.1 Boolean algebra

4.1.1 Majority function

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

For each bit index, that result bit according to what bit is the the majority amongst x, y, and z at this index.

4.1.2 Choose function

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

For each bit index, that result bit is according to the bit from y or z, depending on the bit from x.

$$Ch(x, y, z) = \begin{cases} x = 1 & y \\ x = 0 & z \end{cases}$$

4.1.3 Parity function

$$Par(x, y, z) = x \oplus y \oplus z$$

Parity is whether it contains an odd or even number of 1-bits.

$$\begin{cases} odd & 1 \\ even & 0 \end{cases}$$

For each bit index, that result bit is according to the parity of x, y, and z at this index.

4.2 Bit shifts

4.2.1 Standard shift

$$ShiftR^n(x) = \text{shift bits } n \text{ positions to the right}$$

Algorithm 3 Shift right

```
1: function SHIFT RIGHT(x, n)
2:     return (x >> n)
```

4.2.2 Bitwise rotation

Also called a circular shift.

$$RotR^n(x) = \text{rotate (circular shift) bits } n \text{ positions to the right}$$
$$RotL^n(x) = \text{rotate (circular shift) bits } n \text{ positions to the left}$$

Algorithm 4 rotate left

```
1: function ROTATE LEFT(x, n)
2:     bits = the number of bits in the bitfield x
3:     return (x << n) | (x >> (bits - n))
```

Algorithm 5 Rotate right

```
1: function ROTATE RIGHT(x, n)
2:     bits = the number of bits in the bitfield x
3:     return (x >> n) | (x << (bits - n))
```

Chapter 5

MD5

Endian	Little endian
Digest	128 bits (4 × 32-bits) 16 bytes (4 × 4-bytes)
Block	512 bits 64 bytes
Words	512 bits (16 × 32-bits) 64 bytes (16 × 4-bytes)
Rounds	64

5.1 Words

The 16 words are 32-bit sections of the message block.

$$0 \leq i \leq 15 \quad \Big\{ W[i] = M[i]$$

5.2 Compression function

$$\begin{aligned} tmp &= b + RotL^{s[i \mod 4]}(a + F(b,c,d) + K[i] + W[g]) \\ a &= d \\ d &= c \\ c &= b \\ b &= tmp \end{aligned}$$

5.3 Rounds

$$\begin{aligned} 0 \leq i \leq 15 \quad & \begin{cases} F(b,c,d) = (b \wedge c) \vee (\neg b \wedge d) \\ g = i \\ s = \{7, 12, 17, 22\} \end{cases} \\ 16 \leq i \leq 31 \quad & \begin{cases} F(b,c,d) = (b \wedge d) \vee (c \wedge \neg d) \\ g = (5 \times i + 1) \mod 16 \\ s = \{5, 9, 14, 20\} \end{cases} \\ 32 \leq i \leq 47 \quad & \begin{cases} F(b,c,d) = b \oplus c \oplus d \\ g = (3 \times i + 5) \mod 16 \\ s = \{4, 11, 16, 23\} \end{cases} \\ 48 \leq i \leq 63 \quad & \begin{cases} F(b,c,d) = c \oplus (b \vee \neg d) \\ g = (7 \times i) \mod 16 \\ s = \{6, 10, 15, 21\} \end{cases} \end{aligned}$$

5.4 Constants

Table 5.1: Initialization values

```
h[4] = {
    0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476
}
```

Table 5.2: K values

```
k[64] = {
    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
    0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
    0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
    0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
    0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
    0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
    0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
    0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
    0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
    0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
    0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
    0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
    0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391
}
```

Table 5.3: Shifts

```
s[4][4] = {
    {7, 12, 17, 22},
    {5,  9, 14, 20},
    {4, 11, 16, 23},
    {6, 10, 15, 21}
}
```

Chapter 6

SHA-1

Endian	Big endian
Digest	256 bits (8 × 32-bits) 32 bytes (8 × 4-bytes)
Block	512 bits 64 bytes
Words	2048 bits (64 × 32-bits) 256 bytes (64 × 4-bytes)
Rounds	64

6.1 Words

The first 16 words are 32-bit sections of the message block. The rest of the words are derived from those original 16.

$$W[i] = \begin{cases} 0 \leq i \leq 15 & M[i] \\ 16 \leq i \leq 79 & RotL^1(W[i-3] \oplus W[i-8] \oplus W[i-14] \oplus W[i-16]) \end{cases}$$

6.2 Compression function

$$\begin{aligned} tmp &= RotL^5(a) + F(b,c,d) + W[i] + K + e \\ e &= d \\ d &= c \\ c &= RotL^{30}(b) \\ b &= a \\ a &= tmp \end{aligned}$$

6.3 Rounds

$$\begin{aligned} 0 \leq i \leq 19 &\quad \begin{cases} F(b,c,d) = Ch(b,c,d) \\ K = \text{0x5A827999} \end{cases} \\ 20 \leq i \leq 39 &\quad \begin{cases} F(b,c,d) = Par(b,c,d) \\ K = \text{0x6ED9EBA1} \end{cases} \\ 40 \leq i \leq 59 &\quad \begin{cases} F(b,c,d) = Maj(b,c,d) \\ K = \text{0x8F1BBCDC} \end{cases} \\ 60 \leq i \leq 79 &\quad \begin{cases} F(b,c,d) = Par(b,c,d) \\ K = \text{0xCA62C1D6} \end{cases} \end{aligned}$$

6.4 Constants

Table 6.1: Initialization values

```
h[5] = {
    0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0
}
```

Table 6.2: K values

```
k[4] = {
    0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xca62c1d6
}
```

Chapter 7

SHA-256

Endian	Big endian
Digest	256 bits (8 × 32-bits) 32 bytes (8 × 4-bytes)
Block	512 bits 64 bytes
Words	2560 bits (80 × 32-bits) 320 bytes (80 × 4-bytes)
Rounds	80

$$\Sigma_0(x) = RotR^2(x) \ \oplus RotR^{13}(x) \oplus RotR^{22}(x)$$
$$\Sigma_1(x) = RotR^6(x) \ \oplus RotR^{11}(x) \oplus RotR^{25}(x)$$
$$\sigma_0(x) = RotR^7(x) \ \oplus RotR^{18}(x) \oplus ShiftR^3(x)$$
$$\sigma_1(x) = RotR^{17}(x) \oplus RotR^{19}(x) \oplus ShiftR^{10}(x)$$

7.1 Words

The first 16 words are 64-bit sections of the message block. The rest of the words are derived from those original 16.

$$W[i] = \begin{cases} 0 \leq i \leq 15 & M[i] \\ 16 \leq i \leq 63 & \sigma_1(W[i-2]) + W[i-7] + \sigma_0(W[i-15]) + W[i-16] \end{cases}$$

7.2 Compression function

$$\begin{aligned} tmp_1 &= h + \Sigma_1(e) + Ch(e,f,g) + K[i] + W[i] \\ tmp_2 &= \Sigma_0(a) + Maj(a,b,c) \\ h &= g \\ g &= f \\ f &= e \\ e &= d + tmp_1 \\ d &= c \\ c &= b \\ b &= a \\ a &= tmp_1 + tmp_2 \end{aligned}$$

7.3 Constants

Table 7.1: Initialization values

```
h[8] = {
    0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
    0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19
}
```

Table 7.2: K values

```
k[64] = {
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
    0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
    0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
    0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
    0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
    0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
}
```

Chapter 8

SHA-224

SHA-224 is a variant of SHA-256. It's just the same, except:

- The internal state has different initialization values.
- It uses the same size digest all the way through, but at the very end, it only takes a portion of that for the checksum output. SHA-224 uses the first 224 of 256 bits.

Table 8.1: Initialization values

```
h[8] = {
    0xc1059ed8, 0x367cd507, 0x3070dd17, 0xf70e5939,
    0xffc00b31, 0x68581511, 0x64f98fa7, 0xbefa4fa4
}
```

Chapter 9

SHA-512

Endian	Big endian
Digest	512 bits (8 × 64-bits) 64 bytes (8 × 8-bytes)
Block	1024 bits 128 bytes
Words	5120 bits (80 × 64-bits) 640 bytes (80 × 8-bytes)
Rounds	80

In SHA-512 message padding, the number of bits in the message is given as a 128-bit number. (Or—functionally—as 64-bits of 0-padding and 64-bits of length.)

$$\begin{aligned}
\Sigma_0(x) &= RotR^{28}(x) \oplus RotR^{34}(x) \oplus RotR^{39}(x) \\
\Sigma_1(x) &= RotR^{14}(x) \oplus RotR^{18}(x) \oplus RotR^{41}(x) \\
\sigma_0(x) &= RotR^{1}(x) \ \oplus RotR^{8}(x) \ \oplus ShiftR^{7}(x) \\
\sigma_1(x) &= RotR^{19}(x) \oplus RotR^{61}(x) \oplus ShiftR^{6}(x)
\end{aligned}$$

9.1 Words

The first 16 words are 64-bit sections of the message block. The rest of the words are derived from those original 16.

$$W[i] = \begin{cases} 0 \leq i \leq 15 & M[i] \\ 16 \leq i \leq 63 & \sigma_1(W[i-2]) + W[i-7] + \sigma_0(W[i-15]) + W[i-16] \end{cases}$$

9.2 Compression function

$$\begin{aligned}
tmp_1 &= h + \Sigma_1(e) + Ch(e,f,g) + K[i] + W[i] \\
tmp_2 &= \Sigma_0(a) + Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + tmp_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= tmp_1 + tmp_2
\end{aligned}$$

9.3 Constants

Table 9.1: Initialization values

```
h[8] = {
    0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b, 0xa54ff53a5f1d36f1,
    0x510e527fade682d1, 0x9b05688c2b3e6c1f, 0x1f83d9abfb41bd6b, 0x5be0cd19137e2179
}
```

Table 9.2: K values

```
k[80] = {
    0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc,
    0x3956c25bf348b538, 0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118,
    0xd807aa98a3030242, 0x12835b0145706fbe, 0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,
    0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235, 0xc19bf174cf692694,
    0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
    0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5,
    0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4,
    0xc6e00bf33da88fc2, 0xd5a79147930aa725, 0x06ca6351e003826f, 0x142929670a0e6e70,
    0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed, 0x53380d139d95b3df,
    0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
    0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30,
    0xd192e819d6ef5218, 0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,
    0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8,
    0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373, 0x682e6ff3d6b2b8a3,
    0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
    0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b,
    0xca273eceea26619c, 0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178,
    0x06f067aa72176fba, 0x0a637dc5a2c898a6, 0x113f9804bef90dae, 0x1b710b35131c471b,
    0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc, 0x431d67c49c100d4c,
    0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817
}
```

Chapter 10

SHA-384

SHA-384 is a variant of SHA-512. It's just the same, except:

- The internal state has different initialization values.
- It uses the same size digest all the way through, but at the very end, it only takes a portion of that for the checksum output. SHA-384 uses the first 384 of 512-bits.

Table 10.1: Initialization values

```
h[8] = {
    0xcbbb9d5dc1059ed8, 0x629a292a367cd507, 0x9159015a3070dd17, 0x152fecd8f70e5939,
    0x67332667ffc00b31, 0x8eb44a8768581511, 0xdb0c2e0d64f98fa7, 0x47b5481dbefa4fa4
}
```