EXCERPT · FIPS PUB 180-4 · AUGUST 2015

Secure Hash Standard
SHA-256

National Institute of Standards and Technology
Information Technology Laboratory · Gaithersburg, MD 20899-8900



Abstract This standard specifies hash algorithms that can be used to generate digests of messages. The digests are used to detect whether messages have been changed since the digests were generated.

2.2.1 Parameters

The following parameters are used in the secure hash algorithm specifications in this Standard.

$a, b, c, \ldots, h$ Working variables that are the w-bit words used in the computation of the hash values, $H^{(i)}$.

$H^{(i)}$ The i^{th} hash value. $H^{(0)}$ is the *initial* hash value; $H^{(N)}$ is the *final* hash value and is used to determine the message digest.

$H_j^{(i)}$ The j^{th} word of the i^{th} hash value, where $H_0^{(i)}$ is the left-most word of hash value i.

K_t Constant value to be used for the iteration t of the hash computation.

k Number of zeroes appended to a message during the padding step.

ℓ Length of the message, M, in bits.

m Number of bits in a message block, $M^{(i)}$.

M Message to be hashed.

$M^{(i)}$ Message block i, with a size of m bits.

$M_j^{(i)}$ The j^{th} word of the i^{th} message block, where $M_0^{(i)}$ is the left-most word of message block i.

n Number of bits to be rotated or shifted when a word is operated upon.

N Number of blocks in the padded message.

T Temporary w-bit word used in the hash computation.

w Number of bits in a word.

W_t The t^{th} w-bit word of the message schedule.

2.2.2 Symbols and Operations

The following symbols are used in the secure hash algorithm specifications; each operates on w-bit words.

$\wedge$ Bitwise AND operation.

$\vee$ Bitwise OR ("inclusive-OR") operation.

$\oplus$ Bitwise XOR ("exclusive-OR") operation.

$\neg$ Bitwise complement operation.

$+$ Addition modulo 2^w.

$\ll$ Left-shift operation, where $x \ll n$ is obtained by discarding the left-most n bits of the word x and then padding the result with n zeroes on the right.

$\gg$ Right-shift operation, where $x \gg n$ is obtained by discarding the right-most n bits of the word x and then padding the result with n zeroes on the left.

The following operations are used in the secure hash algorithm specifications:

$ROTL^n(x)$ The *rotate left* (circular left shift) operation, where x is a w-bit word and n is an integer with $0 \le n < w$, is defined by $ROTL^n(x) = (x \ll n) \vee (x \gg w - n)$.

$ROTR^n(x)$ The *rotate right* (circular right shift) operation, where x is a w-bit word and n is an integer with $0 \le n < w$, is defined by $ROTR^n(x) = (x \gg n) \vee (x \ll w - n)$.

$SHR^n(x)$ The right shift operation, where x is a w-bit word and n is an integer with $0 \le n < w$, is defined by $SHR^n(x) = x \gg n$.

4.1.2 SHA-224 and SHA-256 Functions

SHA-224 and SHA-256 both use six logical functions, where *each function operates on 32-bit words*, which are represented as x, y, and z. The result of each function is a new 32-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{4.2}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{4.3}$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{4.4}$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{4.5}$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \tag{4.6}$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \tag{4.7}$$

4.2.2 SHA-224 and SHA-256 Constants

SHA-224 and SHA-256 use the same sequence of sixty-four constant 32-bit words, $K_0^{\{256\}}, K_1^{\{256\}}, \ldots, K_{63}^{\{256\}}$. These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. In hex, these constant words are (from left to right)

```
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
```

5.1.1 SHA-1, SHA-224 and SHA-256

Suppose that the length of the message, M, is ℓ bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $\ell + 1 + k \equiv 448 \bmod 512$. Then append the 64-bit block that is equal to the number ℓ expressed using a binary representation. For example, the (8-bit ASCII) message "**abc**" has length $8 \times 3 = 24$, so the message is padded with a one bit, then $448 - (24 + 1) = 423$ zero bits, and then the message length, to become the 512-bit padded message

$$\underbrace{\texttt{01100001}}_{\text{"a"}}\ \underbrace{\texttt{01100010}}_{\text{"b"}}\ \underbrace{\texttt{01100011}}_{\text{"c"}}\ 1\ \overbrace{\texttt{00}\ldots\texttt{00}}^{423}\ \overbrace{\texttt{00}\ldots\texttt{0}\,\underbrace{\texttt{11000}}_{\ell = 24}}^{64}$$

The length of the padded message should now be a multiple of 512 bits.

5.2.1 SHA-1, SHA-224 and SHA-256

For SHA-1, SHA-224 and SHA-256, the message and its padding are parsed into N 512-bit blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

5.3.3 SHA-256

For SHA-256, the initial hash value, $H^{(0)}$, shall consist of the following eight 32-bit words, in hex:

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

These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.

6.2 SHA-256

SHA-256 may be used to hash a message, M, having a length of ℓ bits, where $0 \le \ell < 2^{64}$. The algorithm uses 1) a message schedule of sixty-four 32-bit words, 2) eight working variables of 32 bits each, and 3) a hash value of eight 32-bit words. The final result of SHA-256 is a 256-bit message digest.

The words of the message schedule are labeled $W_0, W_1, \ldots, W_{63}$. The eight working variables are labeled $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$, $\boldsymbol{e}$, $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{h}$. The words of the hash value are labeled $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$, which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. SHA-256 also uses two temporary words, T_1 and T_2.

6.2.1 SHA-256 Preprocessing

1. Set the initial hash value, $H^{(0)}$, as specified in Sec. 5.3.3.
2. The message is padded and parsed as specified in Section 5.

6.2.2 SHA-256 Hash Computation

The SHA-256 hash computation uses functions and constants previously defined in Sec. 4.1.2 and Sec. 4.2.2, respectively. Addition (+) is performed modulo 2^{32}.

Each message block, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, is processed in order, using the following steps:

For i=1 to N:

{

1. Prepare the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

2. Initialize the eight working variables, $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$, $\boldsymbol{e}$, $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{h}$, with the $(i\text{-}1)^{\text{st}}$ hash value:

$a = H_0^{(i-1)}$
$b = H_1^{(i-1)}$
$c = H_2^{(i-1)}$
$d = H_3^{(i-1)}$
$e = H_4^{(i-1)}$
$f = H_5^{(i-1)}$
$g = H_6^{(i-1)}$
$h = H_7^{(i-1)}$

3. For t=0 to 63:

{

$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$
$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$
$h = g$
$g = f$
$f = e$
$e = d + T_1$
$d = c$
$c = b$
$b = a$
$a = T_1 + T_2$

}

4. Compute the i^{th} intermediate hash value $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$
$H_1^{(i)} = b + H_1^{(i-1)}$
$H_2^{(i)} = c + H_2^{(i-1)}$
$H_3^{(i)} = d + H_3^{(i-1)}$
$H_4^{(i)} = e + H_4^{(i-1)}$
$H_5^{(i)} = f + H_5^{(i-1)}$
$H_6^{(i)} = g + H_6^{(i-1)}$
$H_7^{(i)} = h + H_7^{(i-1)}$

}

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, M, is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)}$$

doi.org/10.6028/NIST.FIPS.180-4