FIPS PUB 180-4

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

Secure Hash Standard (SHS)

CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPTOGRAPHY

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

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.FIPS.180-4

August 2015

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie E. May, Under Secretary for Standards and Technology and Director

Federal Information
Processing Standards Publication 180-4

August 2015

Announcing the

SECURE HASH STANDARD

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106), and the Computer Security Act of 1987 (Public Law 100-235).

1. Name of Standard: Secure Hash Standard (SHS) (FIPS PUB 180-4).

2. Category of Standard: Computer Security Standard, Cryptography.

3. Explanation: This Standard specifies secure hash algorithms - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 - for computing a condensed representation of electronic data (message). When a message of any length less than 2^{64} bits (for SHA-1, SHA-224 and SHA-256) or less than 2^{128} bits (for SHA-384, SHA-512, SHA-512/224 and SHA-512/256) is input to a hash algorithm, the result is an output called a message digest. The message digests range in length from 160 to 512 bits, depending on the algorithm. Secure hash algorithms are typically used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, or in the generation of random numbers (bits).

The hash algorithms specified in this Standard are called secure because, for a given algorithm, it is computationally infeasible 1) to find a message that corresponds to a given message digest, or 2) to find two different messages that produce the same message digest. Any change to a message will, with a very high probability, result in a different message digest. This will result in a verification failure when the secure hash algorithm is used with a digital signature algorithm or a keyed-hash message authentication algorithm.

This Standard supersedes FIPS 180-3 [FIPS 180-3].

4. Approving Authority: Secretary of Commerce.

5. Maintenance Agency: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL).

$M^{(i)}$ 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.

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

>> Right-shift operation, where $x >> n$ 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 $ROTL^n(x)=(x << n) \vee (x >> w - n)$.

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 $ROTR^n(x)=(x >> n) \vee (x << w - n)$.

3. NOTATION AND CONVENTIONS

3.1 Bit Strings and Integers

The following terminology related to bit strings and integers will be used.

1. A *hex digit* is an element of the set {0, 1,…, 9, a,…, f}. A hex digit is the representation of a 4-bit string. For example, the hex digit "7" represents the 4-bit string "0111", and the hex digit "a" represents the 4-bit string "1010".

2. A *word* is a *w*-bit string that may be represented as a sequence of hex digits. To convert a word to hex digits, each 4-bit string is converted to its hex digit equivalent, as described in (1) above. For example, the 32-bit string

   ```
   1010 0001 0000 0011 1111 1110 0010 0011
   ```

   can be expressed as "a103fe23", and the 64-bit string

   ```
   1010 0001 0000 0011 1111 1110 0010 0011
   0011 0010 1110 1111 0011 0000 0001 1010
   ```

   can be expressed as "a103fe2332ef301a".

   Throughout this specification, the "big-endian" convention is used when expressing both 32- and 64-bit words, so that within each word, the most significant bit is stored in the left-most bit position.

3. An *integer* may be represented as a word or pair of words. A word representation of the message length, ℓ, in bits, is required for the padding techniques of Sec. 5.1.

   An integer between 0 and 2^{32}-1 *inclusive* may be represented as a 32-bit word. The least significant four bits of the integer are represented by the right-most hex digit of the word representation. For example, the integer $291=2^8 + 2^5 + 2^1 + 2^0=256+32+2+1$ is represented by the hex word "00000123".

   The same holds true for an integer between 0 and 2^{64}-1 *inclusive*, which may be represented as a 64-bit word.

   If Z is an integer, $0 \le Z < 2^{64}$, then $Z=2^{32}X + Y$, where $0 \le X < 2^{32}$ and $0 \le Y < 2^{32}$. Since X and Y can be represented as 32-bit words x and y, respectively, the integer Z can be represented as the pair of words (x, y). This property is used for SHA-1, SHA-224 and SHA-256.

4. FUNCTIONS AND CONSTANTS

4.1 Functions

This section defines the functions that are used by each of the algorithms. Although the SHA-224, SHA-256, SHA-384,SHA-512, SHA-512/224 and SHA-512/256 algorithms all use similar functions, their descriptions are separated into sections for SHA-224 and SHA-256 (Sec. 4.1.2) and for SHA-384, SHA-512, SHA-512/224 and SHA-512/256 (Sec. 4.1.3), since the input and output for these functions are words of different sizes. Each of the algorithms include $Ch(x, y, z)$ and $Maj(x, y, z)$ functions; the exclusive-OR operation ($\oplus$) in these functions may be replaced by a bitwise OR operation ($\vee$) and produce identical results.

4.1.1 SHA-1 Functions

SHA-1 uses a sequence of logical functions, $f_0, f_1, \ldots, f_{79}$. Each function f_t, where $0 \le t \le 79$, operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output. The function $f_t(x, y, z)$ is defined as follows:

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z)=(x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z)=x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z)=x \oplus y \oplus z & 60 \le t \le 79. \end{cases} \quad (4.1)$$

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.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \quad (4.2)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (4.3)$$

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

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

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

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

If $t \geq 16$ then
{

$$W_s = ROTL^1(W_{(s+13)\wedge MASK} \oplus W_{(s+8)\wedge MASK} \oplus W_{(s+2)\wedge MASK} \oplus W_s)$$

}

$$T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_s$$
$$e = d$$
$$d = c$$
$$c = ROTL^{30}(b)$$
$$b = a$$
$$a = T$$

}

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

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$

}

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 160-bit message digest of the message, M, is

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)}$$

6.2 SHA-256

SHA-256 may be used to hash a message, M, having a length of ℓ bits, where $0 \leq \ell < 2^{64}$. 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,…, W_{63}. The eight working variables are labeled ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g***, and ***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, $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$, 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, ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g***, and ***h***, with the (i-1)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

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

6.3 SHA-224

SHA-224 may be used to hash a message, M, having a length of ℓ bits, where $0 \le \ell < 2^{64}$. The function is defined in the exact same manner as SHA-256 (Section 6.2), with the following two exceptions:

1. The initial hash value, $H^{(0)}$, shall be set as specified in Sec. 5.3.2; and