FOREWORD

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Federal Information Security Management Act (FISMA) of 2002.

Comments concerning FIPS publications are welcomed and should be addressed to the Director, Information Technology Laboratory, National Institute of Standards and Technology, 100 Bureau Drive, Stop 8900, Gaithersburg, MD 20899-8900.

Charles H. Romine, Director
Information Technology Laboratory
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.

Key words: computer security, cryptography, message digest, hash function, hash algorithm, Federal Information Processing Standards, Secure Hash Standard.
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).


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.

6. **Applicability:** This Standard is applicable to all Federal departments and agencies for the protection of sensitive unclassified information that is not subject to Title 10 United States Code Section 2315 (10 USC 2315) and that is not within a national security system as defined in Title 40 United States Code Section 11103(a)(1) (40 USC 11103(a)(1)). Either this Standard or Federal Information Processing Standard (FIPS) 202 must be implemented wherever a secure hash algorithm is required for Federal applications, including as a component within other cryptographic algorithms and protocols. This Standard may be adopted and used by non-Federal Government organizations.


8. **Implementations:** The secure hash algorithms specified herein may be implemented in software, firmware, hardware or any combination thereof. Only algorithm implementations that are validated by NIST will be considered as complying with this standard. Information about the validation program can be obtained at [http://csrc.nist.gov/groups/STM/index.html](http://csrc.nist.gov/groups/STM/index.html).

9. **Implementation Schedule:** Guidance regarding the testing and validation to FIPS 180-4 and its relationship to FIPS 140-2 can be found in IG 1.10 of the Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program at [http://csrc.nist.gov/groups/STM/cmvp/index.html](http://csrc.nist.gov/groups/STM/cmvp/index.html).

10. **Patents:** Implementations of the secure hash algorithms in this standard may be covered by U.S. or foreign patents.

11. **Export Control:** Certain cryptographic devices and technical data regarding them are subject to Federal export controls. Exports of cryptographic modules implementing this standard and technical data regarding them must comply with these Federal regulations and be licensed by the Bureau of Export Administration of the U.S. Department of Commerce. Information about export regulations is available at: [http://www.bis.doc.gov/index.htm](http://www.bis.doc.gov/index.htm).

12. **Qualifications:** While it is the intent of this Standard to specify general security requirements for generating a message digest, conformance to this Standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. This Standard will be reviewed every five years in order to assess its adequacy.

13. **Waiver Procedure:** The Federal Information Security Management Act (FISMA) does not allow for waivers to a FIPS that is made mandatory by the Secretary of Commerce.

14. **Where to Obtain Copies of the Standard:** This publication is available electronically by accessing [http://csrc.nist.gov/publications/](http://csrc.nist.gov/publications/). Other computer security publications are available at the same web site.
Specifications for the

SECURE HASH STANDARD

Table of Contents

1. INTRODUCTION ..................................................................................................................3

2. DEFINITIONS.......................................................................................................................4
   2.1 GLOSSARY OF TERMS AND ACRONYMS ........................................................................4
   2.2 ALGORITHM PARAMETERS, SYMBOLS, AND TERMS .......................................................4
      2.2.1 Parameters ..............................................................................................................4
      2.2.2 Symbols and Operations .......................................................................................5

3. NOTATION AND CONVENTIONS .....................................................................................7
   3.1 BIT STRINGS AND INTEGERS ......................................................................................7
   3.2 OPERATIONS ON WORDS ..........................................................................................8

4. FUNCTIONS AND CONSTANTS .......................................................................................10
   4.1 FUNCTIONS ................................................................................................................10
      4.1.1 SHA-1 Functions ..................................................................................................10
      4.1.2 SHA-224 and SHA-256 Functions ........................................................................10
      4.1.3 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 Functions ......................11
   4.2 CONSTANTS ...............................................................................................................11
      4.2.1 SHA-1 Constants ................................................................................................11
      4.2.2 SHA-224 and SHA-256 Constants .......................................................................11
      4.2.3 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 Constants ....................12

5. PREPROCESSING ............................................................................................................13
   5.1.Padding the Message .....................................................................................................13
      5.1.1 SHA-1, SHA-224 and SHA-256 ........................................................................13
      5.1.2 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 ......................................13
   5.2.Parsing the Message ...................................................................................................14
      5.2.1 SHA-1, SHA-224 and SHA-256 ........................................................................14
      5.2.2 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 ......................................14
   5.3.Setting the Initial Hash Value ($H^0$) ..........................................................................14
      5.3.1 SHA-1 ................................................................................................................14
      5.3.2 SHA-224 ............................................................................................................14
      5.3.3 SHA-256 ............................................................................................................15
      5.3.4 SHA-384 ............................................................................................................15
      5.3.5 SHA-512 ............................................................................................................15
      5.3.6 SHA-512/1 ..........................................................................................................15

6. SECURE HASH ALGORITHMS ........................................................................................18
   6.1 SHA-1 ........................................................................................................................18
      6.1.1 SHA-1 Preprocessing .........................................................................................18
      6.1.2 SHA-1 Hash Computation ..................................................................................18
1. INTRODUCTION

This Standard specifies secure hash algorithms, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256. All of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a message digest. These algorithms enable the determination of a message's integrity: any change to the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

Each algorithm can be described in two stages: preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into $m$-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

The algorithms differ most significantly in the security strengths that are provided for the data being hashed. The security strengths of these hash functions and the system as a whole when each of them is used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, can be found in [SP 800-57] and [SP 800-107].

Additionally, the algorithms differ in terms of the size of the blocks and words of data that are used during hashing or message digest sizes. Figure 1 presents the basic properties of these hash algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message Size (bits)</th>
<th>Block Size (bits)</th>
<th>Word Size (bits)</th>
<th>Message Digest Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>$&lt; 2^{64}$</td>
<td>512</td>
<td>32</td>
<td>160</td>
</tr>
<tr>
<td>SHA-224</td>
<td>$&lt; 2^{64}$</td>
<td>512</td>
<td>32</td>
<td>224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>$&lt; 2^{64}$</td>
<td>512</td>
<td>32</td>
<td>256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>SHA-512/224</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>224</td>
</tr>
<tr>
<td>SHA-512/256</td>
<td>$&lt; 2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>256</td>
</tr>
</tbody>
</table>

Figure 1: Secure Hash Algorithm Properties
2. DEFINITIONS

2.1 Glossary of Terms and Acronyms

- **Bit**: A binary digit having a value of 0 or 1.
- **Byte**: A group of eight bits.
- **FIPS**: Federal Information Processing Standard.
- **NIST**: National Institute of Standards and Technology.
- **SHA**: Secure Hash Algorithm.
- **SP**: Special Publication
- **Word**: A group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the secure hash algorithm.

2.2 Algorithm Parameters, Symbols, and Terms

2.2.1 Parameters

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

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

- $H_i^{(i)}$: The $i^{th}$ hash value. $H_0^{(i)}$ is the initial hash value; $H_f^{(i)}$ is the final hash value and is used to determine the message digest.

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

- $K_i$: Constant value to be used for the iteration $i$ of the hash computation.

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

- $\ell$: 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.

\begin{align*}
\land & \quad \text{Bitwise AND operation.} \\
\lor & \quad \text{Bitwise OR ("inclusive-OR") operation.} \\
\oplus & \quad \text{Bitwise XOR ("exclusive-OR") operation.} \\
\neg & \quad \text{Bitwise complement operation.} \\
+ & \quad \text{Addition modulo } 2^w. \\
<< & \quad \text{Left-shift operation, where } x << n \text{ is obtained by discarding the left-most } n \text{ bits of the word } x \text{ and then padding the result with } n \text{ zeroes on the right.} \\
>> & \quad \text{Right-shift operation, where } x >> n \text{ is obtained by discarding the right-most } n \text{ bits of the word } x \text{ and then padding the result with } n \text{ zeroes on the left.}
\end{align*}

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 $0 \leq n < w$, is defined by $ROTL^n(x) = (x << n) \lor (x >> w - n)$.

**$ROTR^n(x)$** The rotate right (circular right shift) operation, where $x$ is a $w$-bit word and $n$ is an integer with $0 \leq n < w$, is defined by $ROTR^n(x) = (x >> n) \lor (x << w - n)$.
$SHR^w(x)$ The *right shift* operation, where $x$ is a $w$-bit word and $n$ is an integer with $0 \leq n < w$, is defined by $SHR^w(x) = x >> n$. 
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, \( t \), 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, \( 0 \leq Z < 2^{64} \), then \( Z = 2^{32}X + Y \), where \( 0 \leq X < 2^{32} \) and \( 0 \leq Y < 2^{32} \). 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.
If $Z$ is an integer, $0 \leq Z < 2^{128}$, then $Z=2^{64}X + Y$, where $0 \leq X < 2^{64}$ and $0 \leq Y < 2^{64}$. Since $X$ and $Y$ can be represented as 64-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-384, SHA-512, SHA-512/224 and SHA-512/256.

4. For the secure hash algorithms, the size of the message block - $m$ bits - depends on the algorithm.

a) For SHA-1, SHA-224 and SHA-256, each message block has 512 bits, which are represented as a sequence of sixteen 32-bit words.

b) For SHA-384, SHA-512, SHA-512/224 and SHA-512/256 each message block has 1024 bits, which are represented as a sequence of sixteen 64-bit words.

3.2 Operations on Words

The following operations are applied to $w$-bit words in all five secure hash algorithms. SHA-1, SHA-224 and SHA-256 operate on 32-bit words ($w=32$), and SHA-384, SHA-512, SHA-512/224 and SHA-512/256 operate on 64-bit words ($w=64$).

1. Bitwise logical word operations: $\land$, $\lor$, $\oplus$, and $\neg$ (see Sec. 2.2.2).

2. Addition modulo $2^w$.

The operation $x + y$ is defined as follows. The words $x$ and $y$ represent integers $X$ and $Y$, where $0 \leq X < 2^w$ and $0 \leq Y < 2^w$. For positive integers $U$ and $V$, let $U \mod V$ be the remainder upon dividing $U$ by $V$. Compute

$$Z=(X + Y) \mod 2^w.$$  

Then $0 \leq Z < 2^w$. Convert the integer $Z$ to a word, $z$, and define $z=x + y$.

3. The right shift operation $\text{SHR}^n(x)$, where $x$ is a $w$-bit word and $n$ is an integer with $0 \leq n < w$, is defined by

$$\text{SHR}^n(x)=x \gg n.$$  

This operation is used in the SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms.

4. The rotate right (circular right shift) operation $\text{ROTR}^n(x)$, where $x$ is a $w$-bit word and $n$ is an integer with $0 \leq n < w$, is defined by

$$\text{ROTR}^n(x)=(x \gg n) \lor (x << w - n).$$
Thus, $ROTR^n(x)$ is equivalent to a circular shift (rotation) of $x$ by $n$ positions to the right.

This operation is used by the SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms.

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

$$ROTL^w(x) = (x \ll n) \lor (x \gg w - n).$$

Thus, $ROTL^w(x)$ is equivalent to a circular shift (rotation) of $x$ by $n$ positions to the left.

This operation is used only in the SHA-1 algorithm.

6. Note the following equivalence relationships, where $w$ is fixed in each relationship:

$$ROTL^w(x) \approx ROTR^{n-w}(x)$$

$$ROTR^n(x) \approx ROTL^{n-w}(x)$$
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 (\(\lor\)) 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 \(0 \leq t \leq 79\), 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:

\[
f_t(x, y, z) = \begin{cases} 
Ch(x, y, z) = (x \land y) \oplus (\neg x \land z) & 0 \leq t \leq 19 \\
Parity(x, y, z) = x \lor y \lor z & 20 \leq t \leq 39 \\
Maj(x, y, z) = (x \land y) \lor (x \land z) \lor (y \land z) & 40 \leq t \leq 59 \\
Parity(x, y, z) = x \lor y \lor z & 60 \leq t \leq 79.
\end{cases}
\]  

(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.

\[
\begin{align}
Ch(x, y, z) &= (x \land y) \oplus (\neg x \land z) & (4.2) \\
Maj(x, y, z) &= (x \land y) \lor (x \land z) \lor (y \land z) & (4.3) \\
\sum^{1256}_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) & (4.4) \\
\sum^{1256}_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) & (4.5) \\
\sigma^0_0^{1256}(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) & (4.6) \\
\sigma^1_1^{1256}(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) & (4.7)
\end{align}
\]
4.1.3 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 Functions

SHA-384, SHA-512, SHA-512/224 and SHA-512/256 use six logical functions, where each function operates on 64-bit words, which are represented as \(x, y,\) and \(z\). The result of each function is a new 64-bit word.

\[
Ch(x, y, z) = (x \land y) \oplus (\neg x \land z) \quad (4.8)
\]
\[
Maj(x, y, z) = (x \land y) \oplus (x \land z) \oplus (y \land z) \quad (4.9)
\]
\[
\sum_0^{[512]}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \quad (4.10)
\]
\[
\sum_1^{[512]}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \quad (4.11)
\]
\[
\sigma_0^{[512]}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x) \quad (4.12)
\]
\[
\sigma_1^{[512]}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x) \quad (4.13)
\]

4.2 Constants

4.2.1 SHA-1 Constants

SHA-1 uses a sequence of eighty constant 32-bit words, \(K_0, K_1, \ldots, K_{79}\), which are given by

\[
K_t = \begin{cases} 
5a827999 & 0 \leq t \leq 19 \\
6ed9eba1 & 20 \leq t \leq 39 \\
8f1bbcdc & 40 \leq t \leq 59 \\
ca62c1d6 & 60 \leq t \leq 79
\end{cases} \quad (4.14)
\]

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 72be56d4 80deb1fe 9bdcc067 c19bf174
e4b8a05c eefbe494 b0536a5d 2f6a8823 5c06fcdb d68ced37 9576f178 c3c8c57d
dff00b03 7c04a4c2 e0f01172 7069f594 c8b63ce9 c7c6705a 5a1a5f58 55c0321f
18b86c2f 7f0f0ad4 d12f431c c47c3d1d f20b5e8c 19a4c8de 2da91e1b 2bde3786
58c76f88 a30c1945 4e0a8272 f63efb61 7211c671 e0ee6df0 b0e036b8 276dfc3d
8b2730c1 9af6b25b 54f20f4c 27c1c3a5 8da0ac07 795aace6 99f47ec0 68437d62
2f592700 1b80d7a6 f8725888 5c60abb0 84c628e5 d2a5671a e0b6534f 169e25d0
a8b707ca 56e3f519 b10a7748 ffe22730 7defb688 c7a71b30 7d63e1c2 aec26902
d1e10159 a8a1b8a3 5f0f6129 c6b8b769 8c295302 643a396c0d04803f
d7192578 e9144c96 gg58a11c d7c6c4b5 0d928982 13a940a4 589b01a5 1b3f3076
70d4c5b7 405a84e5 808adae3 95ed5f99 c8f5e8c0 ee665f174d73eb14 268c2077
23b77b88 163e431a b0f35b63 19924489 659df555 5e7daa0d b872d881 75408a6f
2967e41f 8fa6b234 b6229568 56df0f6b 5972f528 15ca1983 17e0f3ac 927573d7
d7e ply0b958 15848c18 c665f6aa b82a0b75 fed2f80c 67178f2
4.2.3 SHA-384, SHA-512, SHA-512/224 and SHA-512/256 Constants

SHA-384, SHA-512, SHA-512/224 and SHA-512/256 use the same sequence of eighty constant 64-bit words, $K_0^{(512)}, K_1^{(512)}, \ldots, K_{79}^{(512)}$. These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers. In hex, these constant words are (from left to right)

\[
\begin{align*}
428a2f98d728ae22 & \quad 7137449123ef65cd \quad b5c0fbcfec4d3b2f \quad e9b5daba58189dbbc \\
3956c25bf348b538 & \quad 59f111f8b605d019 \quad 923f82a4af194f9b \quad ab1c5ed5da6d118 \\
st \quad d807aa98a0302422 & \quad 12835b0145706fbee \quad 243185b4e4e4b28c \quad 550c7dc3d5fffb4e2e \\
72be5d7f279b96f & \quad 80deb1fe3b1696b1 \quad 9bdc06725c71235c \quad c19bf174cf692694 \\
e49b69c19ef14ad2 & \quad efbe4786384f25e3 \quad 0fc15dc68b8cd5b5 \quad 240ca1cc77ac9c65 \\
2de9c6f592b0275 & \quad 4a7484aa6c6ea6483 \quad 55b0a9dcd0d31fd4 \quad 76f988e931153b5 \\
983e5152ee66d3ab & \quad a831c66d2db43210 \quad b50327ec898fb213f \quad bf597fca7bee0e4 \\
c600bf33da88f0c2 & \quad d5a79147b93a90d5 \quad 06ca6351e003e026f \quad 142929610a0e6e70 \\
27b70a8546d22f6c & \quad 2eb1b21385c26c926 \quad 4d2e6dfc5ac42aed \quad 53880d134d45b3d1f \\
65a73548ba6f32ac & \quad 766a0ab3c77b2a8 \quad 0c2c92e47d8ee6 \quad 92722c8514022353b \\
a2bfe8a1afcf10364 & \quad a81a664bbc423001 \quad c24eb670dfe89791 \quad c76c51a30654be30 \\
d19e919d6ef15218 & \quad d699062b56b5a910 \quad f40e35855771202a \quad 106aa07032bbd1b8 \\
19a4c16b9bd32d0c8 & \quad 1e376c9511a4b5 \quad 2748774cdf8eeb99 \quad 34b0bcb61c548a6 \\
391c0bbc3c59a63 & \quad 4ed8aae43418acb \quad 5b9cca4f77e3e73 \quad 682e6ff3d6b2b8a3 \\
74f8f2ee5defb2f6c & \quad 766a6554f4172f60 \quad 84c87814af0ab72 \quad 8cc70299a6439ec \\
90befffa23631e28 & \quad a4506cecede82bde9 \quad bef9a3f7b2c67915 \quad c671f82e375522b \\
ca273ecc26619c & \quad d18b987c721c0c207 \quad eeda7dd6cde0eble \quad f57d4f79ee6ed178 \\
05f067a72176f22b & \quad 0a637dc5a3c896ae \quad 113f9804bef90daeb \quad 1b701b35131c477b \\
28db77f523047d84 & \quad 32caab7b40c72493 \quad 3c9ebe0af15c9bebc \quad 431d67c57c0204a4 \\
4cc5d4beeh3e42b6 & \quad 59f299c66c652e2a \quad 5fcb6fabc3ad6faec \quad 6c44198c4475817 
\end{align*}
\]
5. PREPROCESSING

Preprocessing consists of three steps: padding the message, \( M \) (Sec. 5.1), parsing the message into message blocks (Sec. 5.2), and setting the initial hash value, \( H^{(i)} \) (Sec. 5.3).

5.1 Padding the Message

The purpose of this padding is to ensure that the padded message is a multiple of 512 or 1024 bits, depending on the algorithm. Padding can be inserted before hash computation begins on a message, or at any other time during the hash computation prior to processing the block(s) that will contain the padding.

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

Suppose that the length of the message, \( M \), is \( \ell \) 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 \( \ell + 1 + k = 448 \text{mod} \ 512 \). Then append the 64-bit block that is equal to the number \( \ell \) 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

\[
\begin{array}{cccccccc}
0110001 & 01100010 & 01100011 & 1 & 00...00 & 00...011000 \\
"a" & "b" & "c" & & & \ell = 24 \\
\end{array}
\]

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

5.1.2 SHA-384, SHA-512, SHA-512/224 and SHA-512/256

Suppose the length of the message \( M \), in bits, is \( \ell \) 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 \( \ell + 1 + k = 896 \text{mod} \ 1024 \). Then append the 128-bit block that is equal to the number \( \ell \) 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 \( 896 - (24 + 1) = 871 \) zero bits, and then the message length, to become the 1024-bit padded message

\[
\begin{array}{cccccccc}
01100001 & 01100010 & 01100011 & 1 & 00...00 & 00...011000 \\
"a" & "b" & "c" & & & \ell = 24 \\
\end{array}
\]

The length of the padded message should now be a multiple of 1024 bits.
5.2 Parsing the Message
The message and its padding must be parsed into $N m$-bit blocks.

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 $M_{15}^{(i)}$.

5.2.2 SHA-384, SHA-512, SHA-512/224 and SHA-512/256
For SHA-384, SHA-512, SHA-512/224 and SHA-512/256, the message and its padding are parsed into $N$ 1024-bit blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block $i$ are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

5.3 Setting the Initial Hash Value ($H^{(0)}$)
Before hash computation begins for each of the secure hash algorithms, the initial hash value, $H^{(0)}$, must be set. The size and number of words in $H^{(0)}$ depends on the message digest size.

5.3.1 SHA-1
For SHA-1, the initial hash value, $H^{(0)}$, shall consist of the following five 32-bit words, in hex:

$$
H_0^{(0)} = 67452301 \\
H_1^{(0)} = efcdab89 \\
H_2^{(0)} = 98badcfe \\
H_3^{(0)} = 10325476 \\
H_4^{(0)} = c3d2e1f0
$$

5.3.2 SHA-224
For SHA-224, the initial hash value, $H^{(0)}$, shall consist of the following eight 32-bit words, in hex:

$$
H_0^{(0)} = c1059ed8 \\
H_1^{(0)} = 367cd507 \\
H_2^{(0)} = 3070dd17 \\
H_3^{(0)} = f70e5939 \\
H_4^{(0)} = ffc00b31 \\
H_5^{(0)} = 68581511 \\
H_6^{(0)} = 64f98fa7
$$
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:

\[
\begin{align*}
H_0^{(0)} &= 6a09e667 \\
H_1^{(0)} &= bb67ae85 \\
H_2^{(0)} &= 3c6ef372 \\
H_3^{(0)} &= a54ff53a \\
H_4^{(0)} &= 510e527f \\
H_5^{(0)} &= 9b05688c \\
H_6^{(0)} &= 1f83d9ab \\
H_7^{(0)} &= 5be0cd19
\end{align*}
\]

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.

5.3.4 SHA-384
For SHA-384, the initial hash value, \( H^{(0)} \), shall consist of the following eight 64-bit words, in hex:

\[
\begin{align*}
H_0^{(0)} &= cbb9d5dc1059ed8 \\
H_1^{(0)} &= 629a292a367cd507 \\
H_2^{(0)} &= 9159015a3070dd17 \\
H_3^{(0)} &= 152fecd8f70e5939 \\
H_4^{(0)} &= 67332667ffcc00b31 \\
H_5^{(0)} &= 8eb44a8768581511 \\
H_6^{(0)} &= db0c2e0d64f98fa7 \\
H_7^{(0)} &= 47b5481dbefa4fa4
\end{align*}
\]

These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the ninth through sixteenth prime numbers.

5.3.5 SHA-512
For SHA-512, the initial hash value, \( H^{(0)} \), shall consist of the following eight 64-bit words, in hex:

\[
\begin{align*}
H_0^{(0)} &= 6a09e667f3bcc908 \\
H_1^{(0)} &= bb67ae8584caa73b
\end{align*}
\]
\[ H_2^{(0)} = 3c6ef372fe94f82b \]
\[ H_3^{(0)} = a54ff53a5f1d36f1 \]
\[ H_4^{(0)} = 510e527fade682d1 \]
\[ H_5^{(0)} = 9b05688c2b3e6c1f \]
\[ H_6^{(0)} = 1f83d9abfb41bd6b \]
\[ H_7^{(0)} = 5be0cd19137e2179 \]

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

5.3.6 SHA-512/t

“SHA-512/t” is the general name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. Each hash function requires a distinct initial hash value. This section provides a procedure for determining the initial value for SHA-512/t for a given value of t.

For SHA-512/t, t is any positive integer without a leading zero such that t < 512, and t is not 384. For example: t is 256, but not 0256, and “SHA-512/256” is “SHA-512/256” (an 11 character long ASCII string), which is equivalent to 53 48 41 2D 35 31 32 2F 32 35 36 in hexadecimal.

The initial hash value for SHA-512/t, for a given value of t, shall be generated by the SHA-512/t IV Generation Function below.

SHA-512/t IV Generation Function

(begins)

Denote \( H_0^{(0)} \) to be the initial hash value of SHA-512 as specified in Section 5.3.5 above.

Denote \( H_0^{(0)'} \) to be the initial hash value computed below.

\( H_0^{(0)} \) is the IV for SHA-512/t.

For \( i = 0 \) to 7

\[
H_i^{(0)'} = H_i^{(0)} \oplus \text{a5a5a5a5a5a5a5 (in hex).}
\]

\( H_0^{(0)} \) = SHA-512 (“SHA-512/t”) using \( H_0^{(0)'} \) as the IV, where \( t \) is the specific truncation value.

(ends)
SHA-512/224 (t = 224) and SHA-512/256 (t = 256) are approved hash algorithms. Other SHA-512/t hash algorithms with different t values may be specified in [SP 800-107] in the future as the need arises. Below are the IVs for SHA-512/224 and SHA-512/256.

5.3.6.1 SHA-512/224

For SHA-512/224, the initial hash value, $H^{(0)}$, shall consist of the following eight 64-bit words, in hex:

\[
\begin{align*}
H_0^{(0)} &= 8C3D37C819544DA2 \\
H_1^{(0)} &= 73E1996689DCD4D6 \\
H_2^{(0)} &= 1DFAB7AE32FF9C82 \\
H_3^{(0)} &= 679DD514582F9FCF \\
H_4^{(0)} &= 0F6D2B697BD44DA8 \\
H_5^{(0)} &= 77E36F7304C48942 \\
H_6^{(0)} &= 3F9D85A86A1D36C8 \\
H_7^{(0)} &= 1112E6AD91D692A1 
\end{align*}
\]

These words were obtained by executing the SHA-512/t IV Generation Function with $t = 224$.

5.3.6.2 SHA-512/256

For SHA-512/256, the initial hash value, $H^{(0)}$, shall consist of the following eight 64-bit words, in hex:

\[
\begin{align*}
H_0^{(0)} &= 22312194FC2BF72C \\
H_1^{(0)} &= 9F555FA3C84C64C2 \\
H_2^{(0)} &= 2393B86B6F53B151 \\
H_3^{(0)} &= 963877195940EABD \\
H_4^{(0)} &= 96283E2A88EFE3 \\
H_5^{(0)} &= BE5E1E2553B63992 \\
H_6^{(0)} &= 2B0199FC2CB5B8AA \\
H_7^{(0)} &= 0EB72DDC81C52CA2 
\end{align*}
\]

These words were obtained by executing the SHA-512/t IV Generation Function with $t = 256$. 
6. SECURE HASH ALGORITHMS

In the following sections, the hash algorithms are not described in ascending order of size. SHA-256 is described before SHA-224 because the specification for SHA-224 is identical to SHA-256, except that different initial hash values are used, and the final hash value is truncated to 224 bits for SHA-224. The same is true for SHA-512, SHA-384, SHA-512/224 and SHA-512/256, except that the final hash value is truncated to 224 bits for SHA-512/224, 256 bits for SHA-512/256 or 384 bits for SHA-384.

For each of the secure hash algorithms, there may exist alternate computation methods that yield identical results; one example is the alternative SHA-1 computation described in Sec. 6.1.3. Such alternate methods may be implemented in conformance to this standard.

6.1 SHA-1

SHA-1 may be used to hash a message, \( M \), having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{64} \). The algorithm uses 1) a message schedule of eighty 32-bit words, 2) five working variables of 32 bits each, and 3) a hash value of five 32-bit words. The final result of SHA-1 is a 160-bit message digest.

The words of the message schedule are labeled \( W_0, W_1, \ldots, W_{79} \). The five working variables are labeled \( a, b, c, d, \) and \( e \). The words of the hash value are labeled \( H_0^{(i)}, H_1^{(i)}, \ldots, H_4^{(i)} \), 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-1 also uses a single temporary word, \( T \).

6.1.1 SHA-1 Preprocessing

1. Set the initial hash value, \( H^{(0)} \), as specified in Sec. 5.3.1.

2. The message is padded and parsed as specified in Section 5.

6.1.2 SHA-1 Hash Computation

The SHA-1 hash computation uses functions and constants previously defined in Sec. 4.1.1 and Sec. 4.2.1, 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, \( \{ W_i \} \):

\[
W_i = \begin{cases} 
    M_i & 0 \leq i \leq 15 \\
    \text{ROTL}^1(W_{i-1} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & 16 \leq i \leq 79 
\end{cases}
\]

2. Initialize the five working variables, \( a, b, c, d, \) and \( e \), with the \((i-1)\)th hash value:

\[
\begin{align*}
    a &= H_0^{(i-1)} \\
    b &= H_1^{(i-1)} \\
    c &= H_2^{(i-1)} \\
    d &= H_3^{(i-1)} \\
    e &= H_4^{(i-1)}
\end{align*}
\]

3. For \( i = 0 \) to \( 79 \):

\[
\begin{align*}
    T &= \text{ROTL}^i(a) + f_i(b,c,d) + e + K_i + W_i \\
    e &= d \\
    d &= c \\
    c &= \text{ROTL}^{30}(b) \\
    b &= a \\
    a &= T
\end{align*}
\]

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

\[
\begin{align*}
    H_0^{(i)} &= a + H_0^{(i-1)} \\
    H_1^{(i)} &= b + H_1^{(i-1)} \\
    H_2^{(i)} &= c + H_2^{(i-1)} \\
    H_3^{(i)} &= d + H_3^{(i-1)} \\
    H_4^{(i)} &= e + H_4^{(i-1)}
\end{align*}
\]
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

\[
H_3^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)}\]

### 6.1.3 Alternate Method for Computing a SHA-1 Message Digest

The SHA-1 hash computation method described in Sec. 6.1.2 assumes that the message schedule \(W_0, W_1, \ldots, W_{79}\) is implemented as an array of eighty 32-bit words. This is efficient from the standpoint of the minimization of execution time, since the addresses of \(W_{8,13}, \ldots, W_{13,15}\) in step (2) of Sec. 6.1.2 are easily computed.

However, if memory is limited, an alternative is to regard \(\{W_i\}\) as a circular queue that may be implemented using an array of sixteen 32-bit words, \(W_0, W_1, \ldots, W_{15}\). The alternate method that is described in this section yields the same message digest as the SHA-1 computation method described in Sec. 6.1.2. Although this alternate method saves sixty-four 32-bit words of storage, it is likely to lengthen the execution time due to the increased complexity of the address computations for the \(\{W_i\}\) in step (3).

For this alternate SHA-1 method, let \(MASK = \text{0000000F}\) (in hex). As in Sec. 6.1.1, addition is performed modulo \(2^{32}\). Assuming that the preprocessing as described in Sec. 6.1.1 has been performed, the processing of \(M^{(N)}\) is as follows:

For \(i = 1\) to \(N\):

1. For \(i = 0\) to \(15\):
   
   \[
   W_i = M_i^{(i)}
   \]

2. Initialize the five working variables, \(a, b, c, d,\) and \(e\), with the \((i-1)^{\text{st}}\) 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)}
   \]

3. For \(i = 0\) to \(79\):
   
   \[
   s = t \land MASK
   \]
If \( i \geq 16 \) then
\[
\{ \\
W_z = ROTL(3)(W_{(i+3)\&\text{M sponge}} \oplus W_{(i+8)\&\text{M sponge}} \oplus W_{(i+12)\&\text{M sponge}} \oplus W_{(i+15)\&\text{M sponge}}} \\
\}
\]

\[ T = ROTL(3)(a) + f(a, b, c, d) + e + K + W_z \]
\[ e = d \]
\[ d = c \]
\[ c = ROTL(3)(b) \]
\[ b = a \]
\[ a = T \]
\}

4. Compute the \( i^{th} \) intermediate hash value \( H_i^{(i)} \):
\[ H_\text{in}^{(i)} = a + H_0^{(i-1)} \]
\[ H_1^{(i)} = b + H_1^{(i-1)} \]
\[ H_2^{(i)} = c + H_2^{(i-1)} \]
\[ H_3^{(i)} = d + H_3^{(i-1)} \]
\[ H_4^{(i)} = e + H_4^{(i-1)} \]
\}

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 \( \ell \) 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 \( H_0^{(1)}, H_1^{(1)}, ..., H_7^{(1)} \), 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, \(\{W_i\}\):

\[
W_i = \begin{cases} 
M_i^{(i)} & 0 \leq i \leq 15 \\
\sigma_1^{(254)}(W_{i-2}) + W_{i-3} + \sigma_0^{(256)}(W_{i-5}) + W_{i-6} & 16 \leq i \leq 63 
\end{cases}
\]

2. Initialize the eight working variables, \(a, b, c, d, e, f, g,\) and \(h\), with the \((i-1)\)th hash value:

\[
\begin{align*}
a &= H_0^{(i-1)} \\
b &= H_1^{(i-1)} \\
c &= H_2^{(i-1)} \\
d &= H_3^{(i-1)} \\
e &= H_4^{(i-1)} \\
f &= H_5^{(i-1)} \\
g &= H_6^{(i-1)} \\
h &= H_7^{(i-1)}
\end{align*}
\]
3. For $i=0$ to $N-1$
   \begin{align*}
   T_1 &= h + \sum^{256}_{i=0} e_i + Ch(e, f, g) + K_i^{256} + W_i \\
   T_2 &= \sum^{256}_{i=0} a_i + 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 \\
   \end{align*}

4. Compute the $i^{th}$ intermediate hash value $H^{(i)}$
   \begin{align*}
   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)} \\
   \end{align*}

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

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

### 6.3 SHA-224

SHA-224 may be used to hash a message, $M$, having a length of $\ell$ bits, where $0 \leq \ell < 2^{64}$. 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
2. The 224-bit message digest is obtained by truncating the final hash value, \( H(N) \), to its left-most 224 bits:
\[
H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)}
\]

6.4 SHA-512

SHA-512 may be used to hash a message, \( M \), having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{128} \). The algorithm uses 1) a message schedule of eighty 64-bit words, 2) eight working variables of 64 bits each, and 3) a hash value of eight 64-bit words. The final result of SHA-512 is a 512-bit message digest.

The words of the message schedule are labeled \( W_0, W_1, \ldots, W_{79} \). 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-512 also uses two temporary words, \( T_1 \) and \( T_2 \).

6.4.1 SHA-512 Preprocessing

1. Set the initial hash value, \( H^{(0)} \), as specified in Sec. 5.3.5.

2. The message is padded and parsed as specified in Section 5.

6.4.2 SHA-512 Hash Computation

The SHA-512 hash computation uses functions and constants previously defined in Sec. 4.1.3 and Sec. 4.2.3, respectively. Addition (+) is performed modulo \( 2^{64} \).

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, \( \{W_i\} \):

\[
W_i = \begin{cases} 
M_t^{(i)} & 0 \leq t \leq 15 \\
\sigma_1^{(51)}(W_{i-2}) + W_{i-7} + \sigma_6^{(51)}(W_{i-15}) + W_{i-16} & 16 \leq t \leq 79
\end{cases}
\]

2. Initialize the eight working variables, \( a, b, c, d, e, f, g, \) and \( h \), with the \((i-1)^{\text{th}}\) hash value:
\[ a = H^{(r-1)}_0 \]
\[ b = H^{(r-1)}_1 \]
\[ c = H^{(r-1)}_2 \]
\[ d = H^{(r-1)}_3 \]
\[ e = H^{(r-1)}_4 \]
\[ f = H^{(r-1)}_5 \]
\[ g = H^{(r-1)}_6 \]
\[ h = H^{(r-1)}_7 \]

3. For \(i=0\) to 79:

\[
T_1 = h + \sum^{(512)}_{i=1} (e) + Ch(e, f, g) + K^{(512)}_i + W_i
\]
\[
T_2 = \sum^{(512)}_{i=0} (a) + Maj(a, b, c)
\]
\[ h = g \]
\[ g = f \]
\[ f = e \]
\[ e = d + T_i \]
\[ d = c \]
\[ c = b \]
\[ b = a \]
\[ a = T_i + T_2 \]

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

\[
H^{(i)}_0 = a + H^{(r-1)}_0
\]
\[
H^{(i)}_1 = b + H^{(r-1)}_1
\]
\[
H^{(i)}_2 = c + H^{(r-1)}_2
\]
\[
H^{(i)}_3 = d + H^{(r-1)}_3
\]
\[
H^{(i)}_4 = e + H^{(r-1)}_4
\]
\[
H^{(i)}_5 = f + H^{(r-1)}_5
\]
\[
H^{(i)}_6 = g + H^{(r-1)}_6
\]
\[
H^{(i)}_7 = h + H^{(r-1)}_7
\]
After repeating steps one through four a total of \( N \) times (i.e., after processing \( M^{(N)} \)), the resulting 512-bit message digest of the message, \( M \), is

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

### 6.5 SHA-384

SHA-384 may be used to hash a message, \( M \), having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{128} \). The algorithm is defined in the exact same manner as SHA-512 (Sec. 6.4), with the following two exceptions:

1. The initial hash value, \( H^{(0)} \), shall be set as specified in Sec. 5.3.4; and
2. The 384-bit message digest is obtained by truncating the final hash value, \( H^{(N)} \), to its left-most 384 bits:

\[
H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)}
\]

### 6.6 SHA-512/224

SHA-512/224 may be used to hash a message, \( M \), having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{118} \). The algorithm is defined in the exact same manner as SHA-512 (Sec. 6.4), with the following two exceptions:

1. The initial hash value, \( H^{(0)} \), shall be set as specified in Sec. 5.3.6.1; and
2. The 224-bit message digest is obtained by truncating the final hash value, \( H^{(N)} \), to its left-most 224 bits.

### 6.7 SHA-512/256

SHA-512/256 may be used to hash a message, \( M \), having a length of \( \ell \) bits, where \( 0 \leq \ell < 2^{118} \). The algorithm is defined in the exact same manner as SHA-512 (Sec. 6.4), with the following two exceptions:

1. The initial hash value, \( H^{(0)} \), shall be set as specified in Sec. 5.3.6.2; and
2. The 256-bit message digest is obtained by truncating the final hash value, \( H^{(N)} \), to its left-most 256 bits.
7. **TRUNCATION OF A MESSAGE DIGEST**

Some application may require a hash function with a message digest length different than those provided by the hash functions in this Standard. In such cases, a truncated message digest may be used, whereby a hash function with a larger message digest length is applied to the data to be hashed, and the resulting message digest is truncated by selecting an appropriate number of the leftmost bits. For guidelines on choosing the length of the truncated message digest and information about its security implications for the cryptographic application that uses it, see SP 800-107 [SP 800-107].
APPENDIX A: Additional Information

A.1 Security of the Secure Hash Algorithms

The security of the five hash algorithms, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 is discussed in [SP 800-107].

A.2 Implementation Notes


A.3 Object Identifiers

Object identifiers (OIDs) for the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 algorithms are posted at http://csrc.nist.gov/groups/ST/crypto_apps_infra/csor/algorithm.html.
APPENDIX B: REFERENCES


APPENDIX C: Technical Changes from FIPS 180-3

1. In FIPS 180-3, padding was inserted before hash computation begins. FIPS 140-4 removed this restriction. Padding can be inserted before hash computation begins or at any other time during the hash computation prior to processing the message block(s) containing the padding.

2. FIPS 180-4 adds two additional algorithms: SHA-512/224 and SHA-512/256 to the Standard and the method for determining the initial value for SHA-512/t for a given value of t.
ERRATUM

The following change has been incorporated into FIPS 180-4, as of the date indicated in the table.

<table>
<thead>
<tr>
<th>DATE</th>
<th>TYPE</th>
<th>CHANGE</th>
<th>PAGE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/9/2014</td>
<td>Editorial</td>
<td>Change &quot;t &lt; 79&quot; to &quot;t \leq 79&quot;</td>
<td>Page 10, Section 4.1.1, Line 1</td>
</tr>
</tbody>
</table>