Secure Hash Standard (SHS)

CATEGORY: COMPUTER SECURITY  SUBCATEGORY: CRYPTOGRAPHY

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

October 2008
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.

Cita Furlani, Director
Information Technology Laboratory
Abstract
This standard specifies five 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 180-3

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 five secure hash algorithms - SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 - 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 and SHA-512) 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 five 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 180-2 [FIPS 180-2].

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 44 United States Code Section 3502(2) (44 USC 3502(2)). This standard shall be implemented whenever a secure hash algorithm is required for Federal applications, including use by other cryptographic algorithms and protocols. The adoption and use of this Standard is available to private and commercial 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.

9. Implementation Schedule: Guidance regarding the testing and validation to FIPS 180-3 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.

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.

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 Federal Information Processing Standards (FIPS) that are 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/. 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 and SHA-512 Functions ......................................................................................10
   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 and SHA-512 Constants ......................................................................................11

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 and SHA-512 .......................................................................................................13
   5.2 Parsing the Padded Message .........................................................................................................14
      5.2.1 SHA-1, SHA-224 and SHA-256 ..........................................................................................14
      5.2.2 SHA-384 and SHA-512 .......................................................................................................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

6. **SECURE HASH ALGORITHMS** .........................................................................................................17
   6.1 SHA-1 ...........................................................................................................................................17
      6.1.1 SHA-1 Preprocessing ...........................................................................................................17
      6.1.2 SHA-1 Hash Computation ..................................................................................................17
      6.1.3 Alternate Method for Computing a SHA-1 Message Digest ...............................................19
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>SHA-256 Preprocessing</td>
</tr>
<tr>
<td>6.2.1</td>
<td>SHA-256 Hash Computation</td>
</tr>
<tr>
<td>6.2.2</td>
<td>SHA-256 Hash Computation</td>
</tr>
<tr>
<td>6.3</td>
<td>SHA-224 Preprocessing</td>
</tr>
<tr>
<td>6.4</td>
<td>SHA-512 Preprocessing</td>
</tr>
<tr>
<td>6.4.1</td>
<td>SHA-512 Hash Computation</td>
</tr>
<tr>
<td>6.4.2</td>
<td>SHA-512 Hash Computation</td>
</tr>
<tr>
<td>6.5</td>
<td>SHA-384 Preprocessing</td>
</tr>
</tbody>
</table>

7. TRUNCATION OF A MESSAGE DIGEST ................................................................. 25

APPENDIX A: ADDITIONAL INFORMATION ............................................................ 26

A.1 SECURITY OF THE SECURE HASH ALGORITHMS ......................................... 26
A.2 IMPLEMENTATION NOTES ............................................................................. 26
A.3 OBJECT IDENTIFIERS .................................................................................. 26

APPENDIX B: REFERENCES ................................................................................ 27
1. INTRODUCTION

This Standard specifies five secure hash algorithms, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. All five 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 five algorithms differ most significantly in the security strengths that are provided for the data being hashed. The security strengths of these five 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 five algorithms differ in terms of the size of the blocks and words of data that are used during hashing. 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>
</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.
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.

\( a, b, c, \ldots, h \) 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.

\( H^{(i)}_{j} \) The \( j \)th word of the \( i \)th hash value, where \( H^{(i)}_{0} \) 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.

\( \ell \) 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.

$M^{(i)}_j$ The $j^{th}$ word of the $i^{th}$ message block, where $M^{(i)}_0$ 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.
- $\lor$ 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 $0 \leq n < w$, is defined by $ROTL^n(x) = (x \ll n) \lor (x \gg w - n)$.

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

$\text{SHR}^n(x)$ 

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

6
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, \(\ell\), 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 and SHA-512.

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 and SHA-512, 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 and SHA-512 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 \( SHR^n(x) \), where \( x \) is a \( w \)-bit word and \( n \) is an integer with \( 0 \leq n < w \), is defined by

\[
SHR^n(x) = x >> n.
\]

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

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

\[
ROTR^n(x) = (x >> n) \lor (x << w - n).
\]
Thus, $\text{ROTR}^n(x)$ 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, and SHA-512 algorithms.

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

$$\text{ROTL}^n(x) = (x << n) \lor (x >> w - n).$$

Thus, $\text{ROTL}^n(x)$ 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:

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

$$\text{ROTR}^n(x) \approx \text{ROTL}^{w-n}(x)$$
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, and SHA-512 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 and SHA-512 (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 $0 \leq t < 79$, operates on three 32-bit words, $x, y, z$, and produces a 32-bit word as output. The function $f_t(x, y, z)$ is defined as follows:

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

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 $x, y, z$. The result of each function is a new 32-bit word.

$$
\begin{align*}
  Ch(x, y, z) &= (x \land y) \oplus (\neg x \land z) \\
  Maj(x, y, z) &= (x \land y) \oplus (x \land z) \oplus (y \land z)
\end{align*}
$$

$$
\begin{align*}
  \sum_{0}^{(256)}(x) &= ROTr^2(x) \oplus ROTr^{13}(x) \oplus ROTr^{22}(x) \\
  \sum_{1}^{(256)}(x) &= ROTr^6(x) \oplus ROTr^{11}(x) \oplus ROTr^{25}(x) \\
  \sigma_{0}^{(256)}(x) &= ROTr^7(x) \oplus ROTr^{18}(x) \oplus SHR^3(x) \\
  \sigma_{1}^{(256)}(x) &= ROTr^{17}(x) \oplus ROTr^{19}(x) \oplus SHR^{10}(x)
\end{align*}
$$

4.1.3 SHA-384 and SHA-512 Functions

SHA-384 and SHA-512 both use six logical functions, where each function operates on 64-bit words, which are represented as $x, y, z$. The result of each function is a new 64-bit word.
\[ \text{Ch}(x, y, z) = (x \land y) \oplus (\neg x \land z) \quad (4.8) \]
\[ \text{Maj}(x, y, z) = (x \land y) \oplus (x \land z) \oplus (y \land z) \quad (4.9) \]
\[ \sum_0^{(512)}(x) = \text{ROTR}^{28}(x) \oplus \text{ROTR}^{34}(x) \oplus \text{ROTR}^{39}(x) \quad (4.10) \]
\[ \sum_1^{(512)}(x) = \text{ROTR}^{14}(x) \oplus \text{ROTR}^{18}(x) \oplus \text{ROTR}^{41}(x) \quad (4.11) \]
\[ \sigma_0^{(512)}(x) = \text{ROTR}^{1}(x) \oplus \text{ROTR}^{8}(x) \oplus \text{SHR}^{7}(x) \quad (4.12) \]
\[ \sigma_1^{(512)}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^{6}(x) \quad (4.13) \]

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)
\[
\begin{align*}
428a2f98 & \quad 71374491 \quad b5c0fbcf \quad e9b5dba5 \quad 3956c25b \quad 59f111f1 \quad 923f82a4 \quad ab1c5ed5 \\
d807aa98 & \quad 12835b01 \quad 243185be \quad 550c7dc3 \quad 72be5d74 \quad 80deb1fe \quad 9bdc06a7 \quad c19bf174 \\
e4b82fbf & \quad efebe476 \quad 0fc19dc6 \quad 240ca1cc \quad 2de92c6f \quad 4a7484aa \quad 5cb0a9dc \quad 76f98da \\
983e5152 & \quad a831c66d \quad b00327c8 \quad bf597fc7 \quad c6e00bf3 \quad d5a79147 \quad 06ca6351 \quad 14292967 \\
27b70a85 & \quad 2eb1b2138 \quad 4d2c6d6c \quad 5338d13 \quad 650a7354 \quad 766a0abb \quad 81c2c92e \quad 92722c85 \\
a2be8a1 & \quad a81a664b \quad c2b8b70 \quad c76c51a3 \quad d192e819 \quad d6990624 \quad f40e3585 \quad 106aa070 \\
19a4c116 & \quad 1e376c08 \quad 2748774c \quad 34b0cb5 \quad 391c0cb3 \quad 4ed8aa4a \quad 5b9cca4f \quad 682e6ff3 \\
748f82ee & \quad 78a5636f \quad 84c87814 \quad 8cc70208 \quad 90befffa \quad a4506ceb \quad bef9a3f7 \quad c67178f2 
\end{align*}
\]

4.2.3 SHA-384 and SHA-512 Constants
SHA-384 and SHA-512 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)
\[
\begin{align*}
428a2f987d78ae22 & \quad 7137449123ef65cd \quad b5c0fbcfecd3b2f \quad e9b5dba58189dbbc \\
d807aa98 & \quad 12835b01 \quad 243185be \quad 550c7dc3 \quad 72be5d74 \quad 80deb1fe \quad 9bdc06a7 \quad c19bf174 \\
e4b82fbf & \quad efebe476 \quad 0fc19dc6 \quad 240ca1cc \quad 2de92c6f \quad 4a7484aa \quad 5cb0a9dc \quad 76f98da \\
983e5152 & \quad a831c66d \quad b00327c8 \quad bf597fc7 \quad c6e00bf3 \quad d5a79147 \quad 06ca6351 \quad 14292967 \\
27b70a85 & \quad 2eb1b2138 \quad 4d2c6d6c \quad 5338d13 \quad 650a7354 \quad 766a0abb \quad 81c2c92e \quad 92722c85 \\
a2be8a1 & \quad a81a664b \quad c2b8b70 \quad c76c51a3 \quad d192e819 \quad d6990624 \quad f40e3585 \quad 106aa070 \\
19a4c116 & \quad 1e376c08 \quad 2748774c \quad 34b0cb5 \quad 391c0cb3 \quad 4ed8aa4a \quad 5b9cca4f \quad 682e6ff3 \\
748f82ee & \quad 78a5636f \quad 84c87814 \quad 8cc70208 \quad 90befffa \quad a4506ceb \quad bef9a3f7 \quad c67178f2 
\end{align*}
\]
5. **PREPROCESSING**

Preprocessing shall take place before hash computation begins. This preprocessing consists of three steps: padding the message, $M$ (Sec. 5.1), parsing the padded message into message blocks (Sec. 5.2), and setting the initial hash value, $H^0$ (Sec. 5.3).

5.1 **Padding the Message**

The message, $M$, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512 or 1024 bits, depending on the algorithm.

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 $\ell + 1 + k \equiv 448 \text{ mod } 512$. 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

\[
\begin{array}{cccccccc}
01100001 & 0110010 & 0110011 & 1 & 00...00 & 00...011000 \\
\text{“a”} & \text{“b”} & \text{“c”} & & & \ell = 24
\end{array}
\]

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

5.1.2 **SHA-384 and SHA-512**

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 $\ell + 1 + k \equiv 896 \text{ mod } 1024$. 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

\[
\begin{array}{cccccccc}
01100001 & 0110010 & 0110011 & 1 & 00...00 & 00...011000 \\
\text{“a”} & \text{“b”} & \text{“c”} & & & \ell = 24
\end{array}
\]

The length of the padded message should now be a multiple of 1024 bits.
5.2 Parsing the Padded Message

After a message has been padded, it must be parsed into $N m$-bit blocks before the hash computation can begin.

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

For SHA-1, SHA-224 and SHA-256, the padded message is parsed into $N / 512$-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. 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 and SHA-512

For SHA-384 and SHA-512, the padded message is parsed into $N / 1024$-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. 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:

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

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:

$$
H_0^{(0)} = \text{c1059ed8} \\
H_1^{(0)} = \text{367cd507} \\
H_2^{(0)} = \text{3070dd17} \\
H_3^{(0)} = \text{f70e5939} \\
H_4^{(0)} = \text{ff00b31} \\
H_5^{(0)} = \text{68581511}
$$
\[ H_6^{(0)} = 64f98fa7 \]
\[ H_7^{(0)} = befa4fa4 \]

### 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:

\[
\begin{align*}
H_0^{(0)} &= cbbb9d5dc1059ed8 \\
H_1^{(0)} &= 629a292a367cd507 \\
H_2^{(0)} &= 9159015a3070dd17 \\
H_3^{(0)} &= 152fec8f70e5939 \\
H_4^{(0)} &= 67332667ffc00b31 \\
H_5^{(0)} &= 8eb44a8768581511 \\
H_6^{(0)} &= db0c2e0d64f98fa7 \\
H_7^{(0)} &= 47b5481dbefa4fa4
\end{align*}
\]

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:

\[ H_0^{(0)} = 6a09e667f3bcc908 \]
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.
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 and SHA-384, except that the final hash value is truncated to 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 $W_0$, $W_1$, ..., $W_{79}$. The five working variables are labeled $a$, $b$, $c$, $d$, and $e$. The words of the hash value are labeled $H_0^{(i)}$, $H_1^{(i)}$, ..., $H_4^{(i)}$, 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. Pad the message, $M$, according to Sec. 5.1.1;
2. Parse the padded message into $N$ 512-bit message blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$, according to Sec. 5.2.1; and
3. Set the initial hash value, $H^{(0)}$, as specified in Sec. 5.3.1.

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 $2^{32}$.

After preprocessing is completed, 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, $\{W_i\}$:
\[
\begin{align*}
W_t &= \begin{cases} 
M_t^{(i)} & 0 \leq t \leq 15 \\
\text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \leq t \leq 79
\end{cases}
\end{align*}
\]

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 \(t=0\) to \(79:\)

\[
\begin{align*}
T &= \text{ROTL}^5(a) + f_t(b, c, d) + e + K_t + W_t \\
e &= d \\
d &= c \\
c &= \text{ROTL}^{30}(b) \\
b &= a \\
a &= T
\end{align*}
\]

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

\[
H^{(N)}_0 \parallel H^{(N)}_1 \parallel H^{(N)}_2 \parallel H^{(N)}_3 \parallel H^{(N)}_4
\]
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 \( W_{t-3}, \ldots, W_{t-16} \) in step (2) of Sec. 6.1.2 are easily computed.

However, if memory is limited, an alternative is to regard \( \{ W_t \} \) 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 \( \{ W_t \} \) 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 \( M(i) \) is as follows:

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

1. For \( t = 0 \) to 15:
   
   \[
   W_t = M_{(i)}^{(t)}
   \]

2. Initialize the five working variables, \( a, b, c, d, \) and \( e \), with the \((i-1)\)th hash value:
   
   \[
   a = H_0^{(i-1)} \quad b = H_1^{(i-1)} \quad c = H_2^{(i-1)} \quad d = H_3^{(i-1)} \quad e = H_4^{(i-1)}
   \]

3. For \( t = 0 \) to 79:
   
   \[
   s = t \land MASK
   \]
   
   If \( t \geq 16 \) then
   
   \[
   W_s = \text{ROTL}^1(W_{(s+13)\land MASK} \oplus W_{(s+8)\land MASK} \oplus W_{(s+2)\land MASK} \oplus W_s)
   \]

19
\[ T = ROTL^7 (a) + f_i (b, c, d) + e + K_i + W_s \]
\[ e = d \]
\[ d = c \]
\[ c = ROTL^{30} (b) \]
\[ b = a \]
\[ a = T \]

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

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

### 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 \( W_0, W_1, \ldots, W_{63} \). 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, \( H_0^{(0)} \), replaced by each successive intermediate hash value (after each message block is processed), \( H_0^{(i)} \), and ending with the final hash value, \( H_0^{(N)} \). SHA-256 also uses two temporary words, \( T_1 \) and \( T_2 \).

#### 6.2.1 SHA-256 Preprocessing

1. Pad the message, \( M \), according to Sec. 5.1.1;
2. Parse the padded message into $N$ 512-bit message blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$, according to Sec. 5.2.1; and

3. Set the initial hash value, $H^{(0)}$, as specified in Sec. 5.3.3.

### 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 $2^{32}$.

After preprocessing is completed, 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, $\{W_i\}$:

\[
W_i = \begin{cases} 
M_i^{(i)} & 0 \leq t \leq 15 \\
\sigma_6^{(256)}(W_{t-2}) + W_{t-7} + \sigma_6^{(256)}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63
\end{cases}
\]

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

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

3. For $t=0$ to $63$:

\{
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

$$H^{(N)}_0 \parallel H^{(N)}_1 \parallel H^{(N)}_2 \parallel H^{(N)}_3 \parallel H^{(N)}_4 \parallel H^{(N)}_5 \parallel H^{(N)}_6 \parallel H^{(N)}_7$$

### 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:
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 \( H^{(i)}_0, H^{(i)}_1, \ldots, H^{(i)}_7 \), 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. Pad the message, \( M \), according to Sec. 5.1.2;
2. Parse the padded message into \( N \) 1024-bit message blocks, \( M^{(1)}, M^{(2)}, \ldots, M^{(N)} \), according to Sec. 5.2.2; and
3. Set the initial hash value, \( H^{(0)} \), as specified in Sec. 5.3.5.

**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 \( 2^{64} \).

After preprocessing is completed, 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, \( \{W_i\} \):

\[
W_i = \begin{cases} 
M^{(i)}_t & 0 \leq t \leq 15 \\
\sigma_1^{(512)}(W_{t-2}) + W_{t-7} + \sigma_0^{(512)}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 79 
\end{cases}
\]

2. Initialize the eight working variables, \( a, b, c, d, e, f, g, \) and \( h \), with the \((i-1)^{st}\) 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 79:
\[
\begin{align*}
T_1 &= h + \sum_{i=0}^{(512)} (e) + Ch(e, f, g) + K_i^{(512)} + W_t \\
T_2 &= \sum_{i=0}^{(512)} (a) + Maj(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{align*}
\]

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 $M^{(N)}$), 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:

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

### 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.
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, and SHA-512 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 and SHA-512 algorithms are posted at http://csrc.nist.gov/groups/ST/crypto_apps_infra/csor/algorithms.html.
APPENDIX B: REFERENCES


[SP 800-57] NIST Special Publication (SP) 800-57, Part 1, Recommendation for Key Management: General, August 2005.