```
From: branstad@st1.ncsl.nist.gov (Dennis Branstad)
Subject: Proposed NIST SHS
To:  Parties who may be interested in Digital Signatures

     On January 31, 1992, NIST published in the Federal Register a proposed
Secure Hash Standard (SHS) designed to work with the proposed Digital
Signature Standard (DSS).  I am taking the liberty to send you an electronic
copy of the proposed standard's announcement section, technical
specifications and appendix A.  Because of mail limitations, I left off
appendix B which is just a second example and the federal register
solication for comments.  Miles Smid is the point of contact for comments
which are due in 90 days.  The following is the core of the proposed
standard.  Note that the name implies only that the algorithm, to the best
of our knowledge, exhibits the qualities needed for a hashing algorithm
acceptable to work with the DSS.  Also note the credit to Ron Rivest.


                   SECURE HASH STANDARD (SHS)


                               Abstract

   This standard specifies a Secure Hash Algorithm (SHA) which can
be used to generate a message digest for a message.  A message
digest is a condensed version of the original message.  This
algorithm can be used with the Digital Signature Standard (DSS).

Key words: ADP security, computer security, digital signatures,
Federal Information Processing Standard, hash algorithm.

Name of Standard: Secure Hash Standard.

Category of Standard: ADP Operations, Computer Security.

Explanation: This Standard specifies a Secure Hash Algorithm (SHA),
which is necessary to ensure the security of the Digital Signature
Algorithm (DSA). When a message of any length < 2^64 bits is input,
the SHA produces a 160-bit output called a message digest.  The
message digest is then input to the DSA which computes the
```

signature for the message. Signing the message digest rather than the message often improves the efficiency of the process, because the message digest is usually much smaller than the message. The same message digest should be obtained by the verifier of the signature when the received version of the message is used as input to the SHA. The SHA is called secure because it is designed to be computationally infeasible to recover a message corresponding to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. The SHA is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 hash algorithm ("The MD4 Message Digest Algorithm," Advances in Cryptology - CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 303-311).

Approving Authority: Secretary of Commerce.

Maintenance Agency: Computer Systems Laboratory, National Institute of Standards and Technology.

Applicability: This standard is applicable to all Federal departments and agencies for the protection of unclassified information that is not subject to section 2315 of Title 10, United States Code, or section 3502(2) of Title 44, United States Code. This standard is required for use with the Digital Signature Standard and whenever a secure hash algorithm is required for federal applications. Private and commercial organizations are encouraged to adopt and use this standard.

Applications: The SHA is appropriate for use with the Digital Signature Standard, which in turn may be used in electronic mail, electronic funds transfer, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication. The SHA can also be used whenever it is necessary to generate a condensed version of a message.

Implementations: The SHA may be implemented in software, firmware, hardware, or any combination thereof. Only implementations of the SHA that are validated by NIST will be considered as complying with this standard. Information about the requirements for validating implementations of this standard can be obtained from the National Institute of Standards and Technology, Computer Systems Laboratory, Attn: SHS Validation, Gaithersburg, MD 20899.

Export Control: Implementations of this standard are subject to Federal Government export controls as specified in Title 15, Code of Federal Regulations, Parts 768 through 799. Exporters are advised to contact the Department of Commerce, Bureau of Export Administration for more information.

Patents: Implementations of the SHA in this standard may be covered by U.S. and foreign patents.

Implementation Schedule: This standard becomes effective six months after the publication date of this FIPS PUB.

Specifications: Federal Information Processing Standard (FIPS YY) Secure Hash Standard (affixed).

Cross Index:

a. FIPS PUB 46-1, Data Encryption Standard.

b. FIPS PUB 73, Guidelines for Security of Computer Applications.

c. FIPS PUB 140-1, Security Requirements for Cryptographic Modules.

d. FIPS PUB XX, Digital Signature Standard.

Qualifications: While it is the intent of this standard to specify a secure hash algorithm, 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.

Waiver Procedure: Under certain exceptional circumstances, the heads of Federal departments and agencies may approve waivers to Federal Information Processing Standards (FIPS). The head of such agency may redelegate such authority only to a senior official designated pursuant to section 3506(b) of Title 44, United States Code. Waiver shall be granted only when:

a. Compliance with a standard would adversely affect the accomplishment of the mission of an operator of a Federal computer system; or

b. Compliance with a standard would cause a major adverse financial impact on the operator which is not offset by Government-wide savings.

Agency heads may act upon a written waiver request containing the information detailed above. Agency heads may also act without a written waiver request when they determine that conditions for meeting the standard cannot be met. Agency heads may approve waivers only by a written decision which explains the basis on which the agency head made the required finding(s). A copy of each decision, with procurement sensitive or classified portions clearly identified, shall be sent to: National Institute of Standards and Technology; ATTN: FIPS Waiver Decisions, Technology Building, Room B-154, Gaithersburg, MD 20899.

In addition, notice of each waiver granted and each delegation of authority to approve waivers shall be sent promptly to the Committee on Government Operations of the House of Representatives and the Committee on Government Affairs of the Senate and shall be published promptly in the Federal Register.

When the determination on a waiver applies to the procurement of equipment and/or services, a notice of the waiver determination must be published in the Commerce Business Daily as a part of the notice of solicitation for offers of an acquisition or, if the waiver determination is made after that notice is published, by amendment to such notice.

A copy of the waiver, any supporting documents, the document approving the waiver and any accompanying documents, with such deletions as the agency is authorized and decides to make under 5 United States Code Section 552(b), shall be part of the procurement documentation and retained by the agency.

Specifications for a

SECURE HASH STANDARD (SHS)

1. INTRODUCTION

The Secure Hash Algorithm (SHA) specified in this standard is necessary to ensure the security of the Digital Signature Standard. When a message of length < 2^64 bits is input, the SHA produces a 160-bit representation of the message called the message digest. The message digest is used during generation of a signature for the message. The same message digest should be computed for the received version of the message, during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The SHA is designed to have the following properties: it is computationally infeasible to recover a message corresponding to a given message digest, or to find two different messages which produce the same message digest.

2. BIT STRINGS AND INTEGERS

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

a. hex digit = 0, 1, ... , 9, a, ... , f. A hex digit is the representation of a 4-bit string. Examples: 7 = 0111, a = 1010.

b. word = 32-bit string b(31) b(30) ... b(0). A word may be represented

as a sequence of 8 hex digits. To convert the latter to a 32-bit string, each hex digit is converted to a 4-bit string as in (a). Example:

a103fe23 = 1010 0001 0000 0011 1111 1110 0010 0011.

A word is also the representation of an unsigned integer between 0 and 2^32 - 1, inclusive, with the most significant bit first. Example: the hex string a103fe23 represents

```
      the decimal integer 2701393443.

   c. block = 512-bit string.  A block may be represented as a
      sequence of 16 words.

   d. integer: each integer x in the standard will satisfy 0 <=
      x < 2^64.  For the purpose of this standard, "integer" and
      "unsigned integer" are equivalent.  If an integer x
      satisfies 0 <= x < 2^32, x may be represented as a word as in
      (b).  If 2^32 <= x < 2^64, then x = 2^32 y + z where 0 <= y < 2^32
      and 0 <= z < 2^32.  Hence y and z can be represented as words A and B,

      respectively, and x can be represented as the pair of
      words (A,B).

   Suppose 0 <= x < 2^32.  To convert x to a 32-bit string, the
following algorithm may be employed:

      y(0) = x;
      for i = 0 to 31 do
        {
        b(i) = 1 if y(i) is odd, b(i) = 0 if y(i) is even;
        y(i+1) = (y(i) - b(i))/2;
        }

   Then x has the 32-bit representation b(31) b(30) ... b(0).  Example:

      25 = 00000000 00000000 00000000 00011001
         = hex 00000019.

   If 2^32 <= x < 2^64, the 2-word representation of x is obtained
similarly.
Example:

      2^35 + 25 = 8 * 2^32 + 25
                = 00000000 00000000 00000000 00001000
                  00000000 00000000 00000000 00011001
                = hex 00000008 00000019.

   Conversely, the string b(31) b(30) ... b(0) represents the integer

      b(31) * 2^31 + b(30) * 2^30 + ... + b(1) * 2 + b(0).

                       3. OPERATIONS ON WORDS

   The following logical operators will be applied to words:

      AND    =  bitwise logical and.

      OR     =  bitwise logical inclusive-or.

      XOR   =  bitwise logical exclusive-or.

      ~x    =  bitwise logical complement of x.

   Example:
```

```
             01101100101110011101001001111011
       XOR   01100101110000010110100110110111
             --------------------------------
         =   00001001011110001011101111001100.
```

Another operation on words is A + B. This is defined as follows: words A and B represent integers x and y, where 0 <= x < 2^32 and 0 <= y < 2^32. Compute

```
      z = (x + y) mod 2^32.
```

Then 0 <= z < 2^32. Convert z to a word, C, and define A + B = C.

Another function on words is S(n,X), where X is a word and n is an integer with 0 <= n < 32. This is defined by

```
      S(n,X) = (X << n) OR (X >> 32-n).
```

In the above, X << n is obtained as follows: discard the leftmost n bits of X and then pad the result with n zeroes on the right (the result will still be 32 bits). X >> m is obtained by discarding the rightmost m bits of X and then padding the result with m zeroes on the left. Thus S(n,X) is equivalent to a circular shift of X by n positions to the left.

4. MESSAGE PADDING

The SHA takes bit strings as input. Thus, for the purpose of this standard, a message will be considered to be a bit string. The length of the message is the number of bits (the empty message has length 0). If the number of bits in a message is a multiple of 8, for compactness we can represent the message in hex.

Suppose a message has length L < 2^64. Before it is input to the SHA, the message is padded on the right as follows:

a. "1" is appended. Example: if the original message is "01010011", this is padded to "010100111".

b. If necessary, "0"s are then appended until the number of bits in the padded message is congruent to 448 modulo 512. Example: suppose the original message is the bit string

```
   01100001 01100010 01100011 01100100 01100101.
```

   After step (a) this gives

```
   01100001 01100010 01100011 01100100 01100101 1.
```

   The number of bits in the above is 41; we pad with 407 "0"s to make the length of the padded message congruent to 448 modulo 512. This gives (in hex)

```
   61626364 65800000 00000000 00000000
   00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00000000
```

```
00000000 00000000.
```

Note that the padding is arranged so that at this point the padded message contains 16s + 14 words, for some s >= 0.

c. Obtain the 2-word representation of L = the number of bits in the original message. If L < 2^32 then the first word is all zeroes. Append these two words to the padded message. Example: suppose the original message is as in (b). Then L = 40 (note that L is computed before any padding). The two-word representation of 40 is hex 00000000 00000028. Hence the final padded message is hex

```
61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028.
```

The final padded message will contain 16N words for some N > 0. Example: in (c) above, the final padded message has N = 1. The final padded message may be regarded as a sequence of N blocks M(1), M(2), ... , M(N), where each M(i) contains 16 words and M(1) is leftmost.

5. FUNCTIONS USED

A sequence of logical functions f(0,x,y,z), ... , f(79,x,y,z) is used in the SHA. Each f operates on three 32-bit words {x,y,z} and produces a 32-bit word as output. f(t,x,y,z) is defined as follows: for words x,y,z,

```
f(t,x,y,z) = (x AND y) OR (~x AND z)             (0  <= t <= 19)

f(t,x,y,z) = x XOR y XOR z                       (20 <= t <= 39)

f(t,x,y,z) = (x AND y) OR (x AND z) OR (y AND z)    (40 <= t <= 59)

f(t,x,y,z) = x XOR y XOR z                       (60 <= t <= 79).
```

6. CONSTANTS USED

A sequence of constant words K(0), K(1), ... , K(79) is used in the SHA. In hex these are given by

```
K(t) = 5a827999         (0  <= t <= 19)

K(t) = 6ed9eba1         (20 <= t <= 39)

K(t) = 8f1bbcdc         (40 <= t <= 59)

K(t) = ca62c1d6         (60 <= t <= 79).
```

7. COMPUTING THE MESSAGE DIGEST

The message digest is computed using the final padded message. The computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labeled A,B,C,D,E. The words of the second 5-word buffer are labeled h0, h1, h2, h3, h4. The words of the 80-word sequence are labeled W(0), W(1), ... , W(79). A single word buffer TEMP is also employed.

To generate the message digest, the 16-word blocks M(1), M(2), ... , M(N) defined in Section 4 are processed in order. The processing of each M(i) involves 80 steps.

Before processing any blocks, the {hj} are initialized as follows: in hex,

```
h0 = 67452301

h1 = efcdab89

h2 = 98badcfe

h3 = 10325476

h4 = c3d2e1f0.
```

Now M(1), M(2), ... , M(N) are processed. To process M(i), we proceed as follows:

a. Divide M(i) into 16 words W(0), W(1), ... , W(15), where W(0) is the

   leftmost word.

b. For t = 16 to 79 let W(t) = W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16).

c. Let A = h0, B = h1, C = h2, D = h3, E = h4.

d. For t = 0 to 79 do

```
TEMP = S(5,A) + f(t,B,C,D) + E + W(t) + K(t);

E = D;  D = C;  C = S(30,B);  B = A; A = TEMP;
```

e. Let h0 = h0 + A, h1 = h1 + B, h2 = h2 + C, h3 = h3 + D, h4 = h4 + E.

After processing M(N), the message digest is the 160-bit string represented by the 5 words

```
h0 h1 h2 h3 h4.
```

The above assumes that the sequence W(0), ... , W(79) is implemented as an array of eighty 32-bit words. This is efficient from the standpoint of minimization of execution time, since the addresses of W(t-3), ... , W(t-16) in step (b) are easily computed. If space is at a premium, an alternative is to regard { W(t) } as a circular queue, which may be implemented using an array of sixteen 32-bit words W[0], ... W[15]. In this case, in hex let MASK = 0000000f. Then processing of M(i) is as follows:

aa. Divide M(i) into 16 words W[0], ... , W[15], where W[0] is the leftmost word.

bb. Let A = h0, B = h1, C = h2, D = h3, E = h4.

cc. For t = 0 to 79 do

```
s = t AND MASK;

if (t >= 16) W[s] = W[(s + 13) AND MASK] XOR W[(s + 8) AND

   MASK] XOR W[(s + 2) AND MASK] XOR W[s];

TEMP = S(5,A) + f(t,B,C,D) + E + W[s] + K(t);

E = D; D = C; C = S(30,B); B = A; A = TEMP;
```

dd. Let h0 = h0 + A, h1 = h1 + B, h2 = h2 + C, h3 = h3 + D, h4 = h4 + E.

Both (a) - (d) and (aa) - (dd) yield the same message digest. Although using (aa) - (dd) saves sixty-four 32-bit words of storage, it is likely to lengthen execution time due to the increased complexity of the address computations for the { W[t] } in step (cc). Other computation methods which give identical results may be implemented in conformance with the standard.

Examples are given in the appendices.

APPENDIX A. A SAMPLE MESSAGE AND ITS MESSAGE DIGEST

This appendix is for informational purposes only and is not required to meet the standard.

Let the message be the ASCII binary-coded form of "abc", i.e.

01100001 01100010 01100011.

This message has length L = 24. In step (a) of Section 4, we append "1", giving a new length of 25. In step (b) we append 423 "0"s. In step (c) we append hex 00000000 00000018, the 2-word representation of 24. Thus the final padded message consists of one block, so that N = 1 in the notation of Section 4. The single block has hex words

```
W[ 0] = 61626380
W[ 1] = 00000000
W[ 2] = 00000000
W[ 3] = 00000000
W[ 4] = 00000000
W[ 5] = 00000000
W[ 6] = 00000000
W[ 7] = 00000000
W[ 8] = 00000000
W[ 9] = 00000000
W[10] = 00000000
W[11] = 00000000
W[12] = 00000000
```

```
W[13] = 00000000
W[14] = 00000000
W[15] = 00000018

   Initial hex values of h:

         h0        h1        h2        h3        h4

      67452301  efcdab89  98badcfe  10325476  c3d2e1f0

   Hex values of A,B,C,D,E after pass t of the "for t = 0 to 79"
loop (step (d) or (cc)) in Section 7:

            A        B        C        D        E

t =  0:  0116fc33 67452301 7bf36ae2 98badcfe 10325476
t =  1:  8990536d 0116fc33 59d148c0 7bf36ae2 98badcfe
t =  2:  a1390f08 8990536d c045bf0c 59d148c0 7bf36ae2
t =  3:  cdd8e11b a1390f08 626414db c045bf0c 59d148c0
t =  4:  cfd499de cdd8e11b 284e43c2 626414db c045bf0c
t =  5:  3fc7ca40 cfd499de f3763846 284e43c2 626414db
t =  6:  993e30c1 3fc7ca40 b3f52677 f3763846 284e43c2
t =  7:  9e8c07d4 993e30c1 0ff1f290 b3f52677 f3763846
t =  8:  4b6ae328 9e8c07d4 664f8c30 0ff1f290 b3f52677
t =  9:  8351f929 4b6ae328 27a301f5 664f8c30 0ff1f290
t = 10:  fbda9e89 8351f929 12dab8ca 27a301f5 664f8c30
t = 11:  63188fe4 fbda9e89 60d47e4a 12dab8ca 27a301f5
t = 12:  4607b664 63188fe4 7ef6a7a2 60d47e4a 12dab8ca
t = 13:  9128f695 4607b664 18c623f9 7ef6a7a2 60d47e4a
t = 14:  196bee77 9128f695 1181ed99 18c623f9 7ef6a7a2
t = 15:  20bdd62f 196bee77 644a3da5 1181ed99 18c623f9
t = 16:  ed2ff4a3 20bdd62f c65afb9d 644a3da5 1181ed99
t = 17:  565df73c ed2ff4a3 c82f758b c65afb9d 644a3da5
t = 18:  550b1e7f 565df73c fb4bfd28 c82f758b c65afb9d
t = 19:  fe0f9e4b 550b1e7f 15977dcf fb4bfd28 c82f758b
t = 20:  b4d4c943 fe0f9e4b d542c79f 15977dcf fb4bfd28
t = 21:  43993572 b4d4c943 ff83e792 d542c79f 15977dcf
t = 22:  f7106486 43993572 ed353250 ff83e792 d542c79f
t = 23:  775924e6 f7106486 90e64d5c ed353250 ff83e792
t = 24:  45a7ef23 775924e6 bdc41921 90e64d5c ed353250
t = 25:  ccead674 45a7ef23 9dd64939 bdc41921 90e64d5c
t = 26:  02d0c6d1 ccead674 d169fbc8 9dd64939 bdc41921
t = 27:  070c437f 02d0c6d1 333ab59d d169fbc8 9dd64939
t = 28:  301e90be 070c437f 40b431b4 333ab59d d169fbc8
t = 29:  b898c685 301e90be c1c310df 40b431b4 333ab59d
t = 30:  669723e2 b898c685 8c07a42f c1c310df 40b431b4
t = 31:  d9316f96 669723e2 6e2631a1 8c07a42f c1c310df
t = 32:  db81a5c7 d9316f96 99a5c8f8 6e2631a1 8c07a42f
t = 33:  99c8dfb2 db81a5c7 b64c5be5 99a5c8f8 6e2631a1
t = 34:  6be6ae07 99c8dfb2 f6e06971 b64c5be5 99a5c8f8
t = 35:  c01cc62c 6be6ae07 a67237ec f6e06971 b64c5be5
t = 36:  6433fdd0 c01cc62c daf9ab81 a67237ec f6e06971
t = 37:  0a33ccf7 6433fdd0 3007318b daf9ab81 a67237ec
t = 38:  4bf58dc8 0a33ccf7 190cff74 3007318b daf9ab81
t = 39:  ebbd5233 4bf58dc8 c28cf33d 190cff74 3007318b
t = 40:  825a3460 ebbd5233 12fd6372 c28cf33d 190cff74
t = 41:  b62cbb93 825a3460 faef548c 12fd6372 c28cf33d
```

```
t = 42:  aa3f9707 b62cbb93 20968d18 faef548c 12fd6372
t = 43:  fe1d0273 aa3f9707 ed8b2ee4 20968d18 faef548c
t = 44:  57ad526b fe1d0273 ea8fe5c1 ed8b2ee4 20968d18
t = 45:  93ebbe3f 57ad526b ff87409c ea8fe5c1 ed8b2ee4
t = 46:  f9adf47b 93ebbe3f d5eb549a ff87409c ea8fe5c1
t = 47:  875586d2 f9adf47b e4faef8f d5eb549a ff87409c
t = 48:  d0a22ffb 875586d2 fe6b7d1e e4faef8f d5eb549a
t = 49:  c12b6426 d0a22ffb a1d561b4 fe6b7d1e e4faef8f
t = 50:  ebc90281 c12b6426 f4288bfe a1d561b4 fe6b7d1e
t = 51:  e7d0ec05 ebc90281 b04ad909 f4288bfe a1d561b4
t = 52:  7cb98e55 e7d0ec05 7af240a0 b04ad909 f4288bfe
t = 53:  0d48dba2 7cb98e55 79f43b01 7af240a0 b04ad909
t = 54:  c2d477bf 0d48dba2 5f2e6395 79f43b01 7af240a0
t = 55:  236bd48d c2d477bf 835236e8 5f2e6395 79f43b01
t = 56:  9b4364d6 236bd48d f0b51def 835236e8 5f2e6395
t = 57:  5b8c33c9 9b4364d6 48daf523 f0b51def 835236e8
t = 58:  be2a4656 5b8c33c9 a6d0d935 48daf523 f0b51def
t = 59:  8ff296db be2a4656 56e30cf2 a6d0d935 48daf523
t = 60:  c10c8993 8ff296db af8a9195 56e30cf2 a6d0d935
t = 61:  6ac23cbf c10c8993 e3fca5b6 af8a9195 56e30cf2
t = 62:  0708247d 6ac23cbf f0432264 e3fca5b6 af8a9195
t = 63:  35d201f8 0708247d dab08f2f f0432264 e3fca5b6
t = 64:  969b2fc8 35d201f8 41c2091f dab08f2f f0432264
t = 65:  3cac6514 969b2fc8 0d74807e 41c2091f dab08f2f
t = 66:  14cd9a35 3cac6514 25a6cbf2 0d74807e 41c2091f
t = 67:  ba564047 14cd9a35 0f2b1945 25a6cbf2 0d74807e
t = 68:  c241f74d ba564047 4533668d 0f2b1945 25a6cbf2
t = 69:  2896b70f c241f74d ee959011 4533668d 0f2b1945
t = 70:  564bbed1 2896b70f 70907dd3 ee959011 4533668d
t = 71:  8fa15d5a 564bbed1 ca25adc3 70907dd3 ee959011
t = 72:  9a226c11 8fa15d5a 5592efb4 ca25adc3 70907dd3
t = 73:  f0b94489 9a226c11 a3e85756 5592efb4 ca25adc3
t = 74:  1809d5e2 f0b94489 66889b04 a3e85756 5592efb4
t = 75:  b86c5a40 1809d5e2 7c2e5122 66889b04 a3e85756
t = 76:  dfe7e487 b86c5a40 86027578 7c2e5122 66889b04
t = 77:  70286c07 dfe7e487 2e1b1690 86027578 7c2e5122
t = 78:  24ff7ed5 70286c07 f7f9f921 2e1b1690 86027578
t = 79:  9a1f95a8 24ff7ed5 dc0a1b01 f7f9f921 2e1b1690

   After processing, values of h:

          h0        h1        h2        h3        h4

       0164b8a9  14cd2a5e  74c4f7ff   082c4d97  f1edf880

   Message digest =  0164b8a9 14cd2a5e 74c4f7ff 082c4d97 f1edf880
```