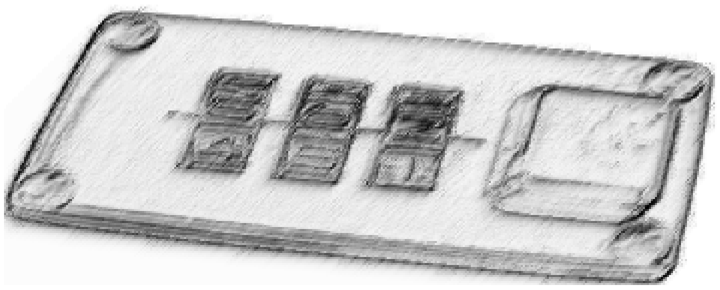
Table of Contents

1 Introduction

1.1 What is SecExMD5+



SecExMD5+ generates message digest signatures of files. Message digests are digital fingerprints commonly published by distributors of computer software and other digital documents so that end users may verify the authenticity of such documents. Any attempt by a third party to tamper with a digital document invariably alters its digital fingerprint.

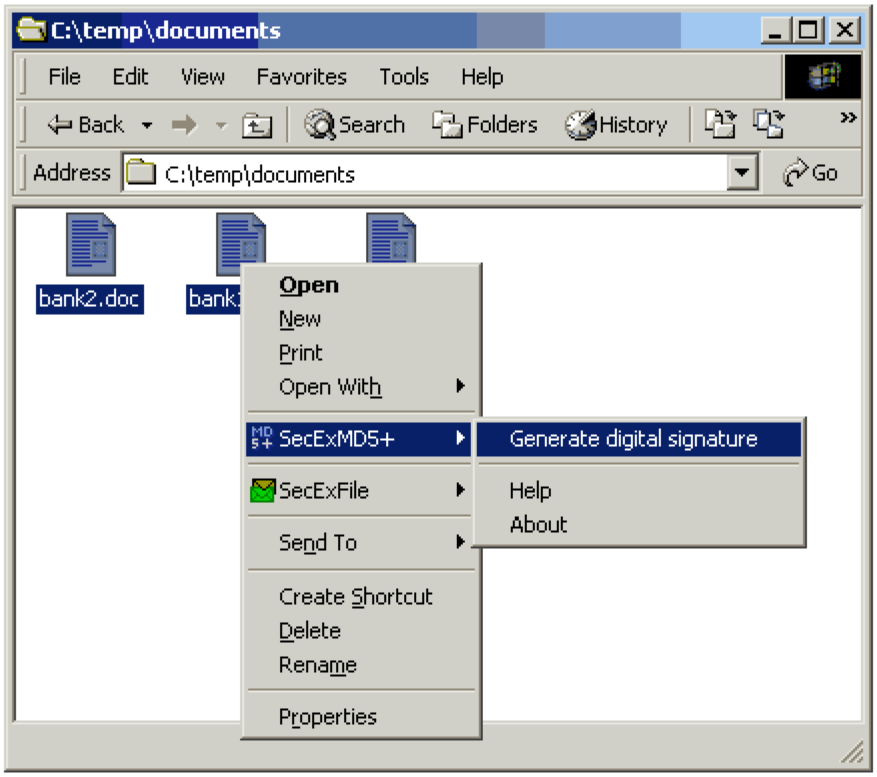
A number of competing standards for the generation of digital signatures or message digests exist. The common standards are MD5, SHA-1 and RIPEMD-160. SecExMD5+ supports all three standards.

Features of SecExMD5+ include a graphical wizard style interface, full integration with Windows explorer right-click menus, multiple file processing, and one-click email support for MAPI based email clients for easy distribution of digital signatures via email.

2 Signatures

2.1 Selecting Files

simply right-click on the file or files you wish to encrypt and select ***SecExMD5+*** & ***Generate digital signature*** from the drop down menu as shown below. This will invoke the SecExMD5 Wizard. Alternatively, you may invoke ***SecExMD5+*** - ***Generate digital signature*** from the Window ***START & Programs*** menu.

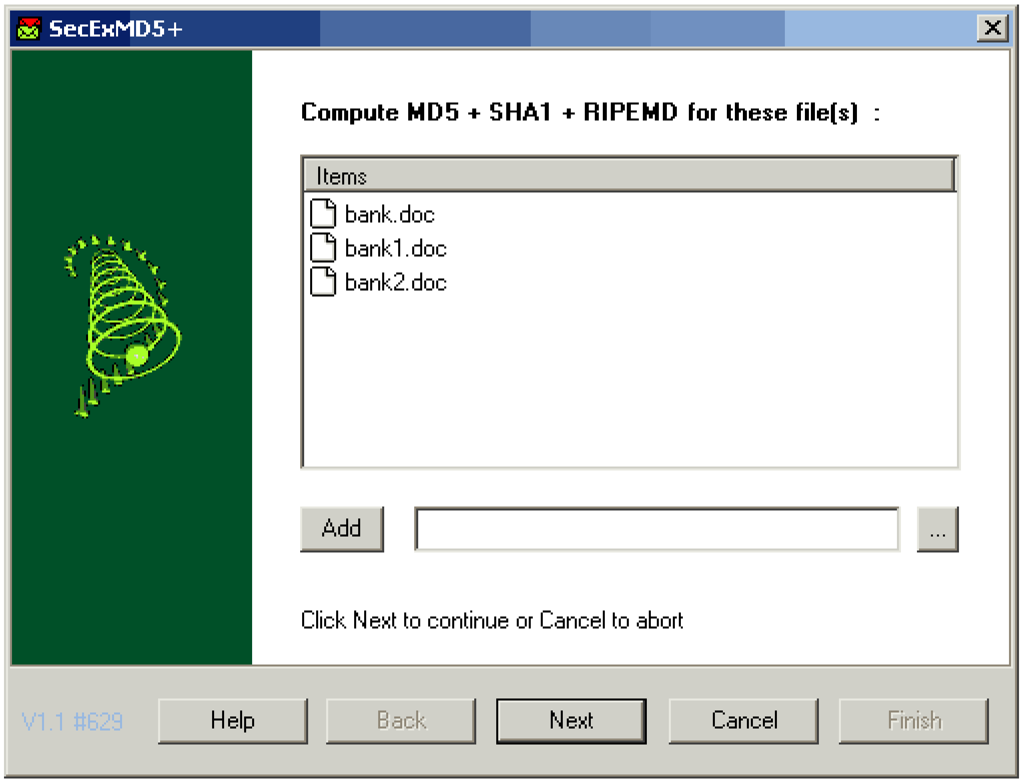


2.2 Wizard Interface

The SecExMD5+ wizard lets you review the file(s) for which you are about to create digital signatures.

This screen also lets you add and remove files and folders from the list. Click the browse files ... button to browse for additional files and then click the ***Add*** button to include them on the list.

Click ***Next*** to continue.



Compute MD5 + SHA1 + RIPEMD for these file(s) :

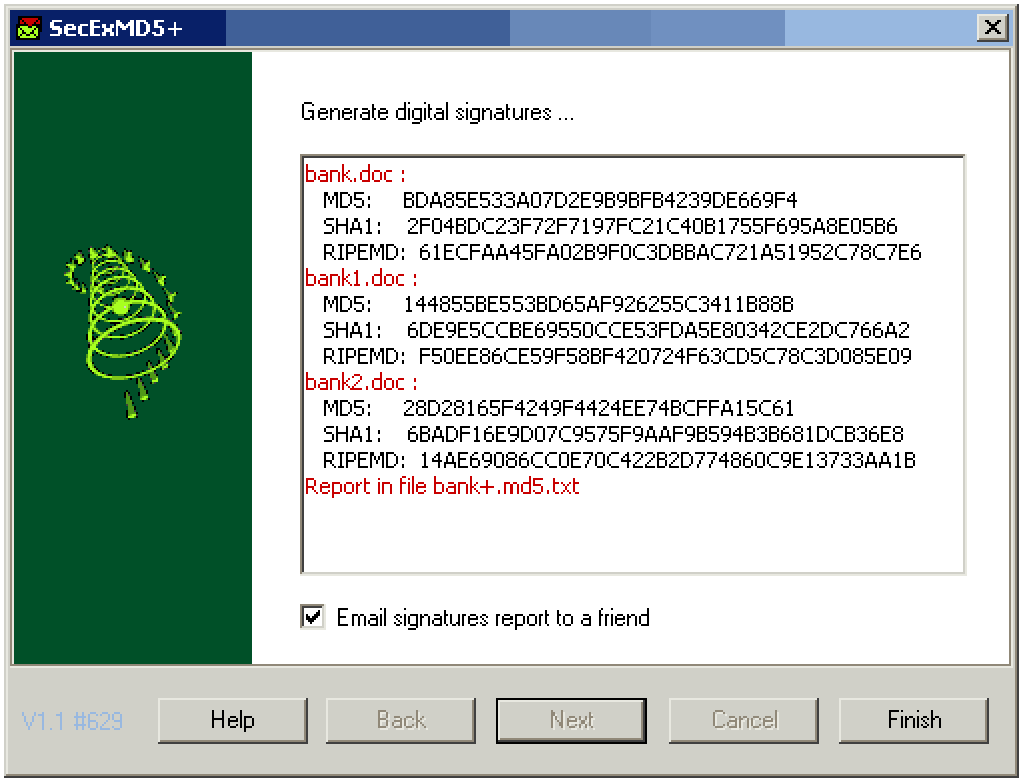
bank.doc
bank1.doc
bank2.doc

Click Next to continue or Cancel to abort

2.3 Generating Reports

SecExMD5+ will automatically generate MD5, SHA-1, and RIPEMD-160 signatures for each file you requested. Check "***Email signatures report to a friend***" if you wish to distribute the report via email.

A copy of the report is written to file as shown in the log window. Click ***Finish***.



Generate digital signatures ...

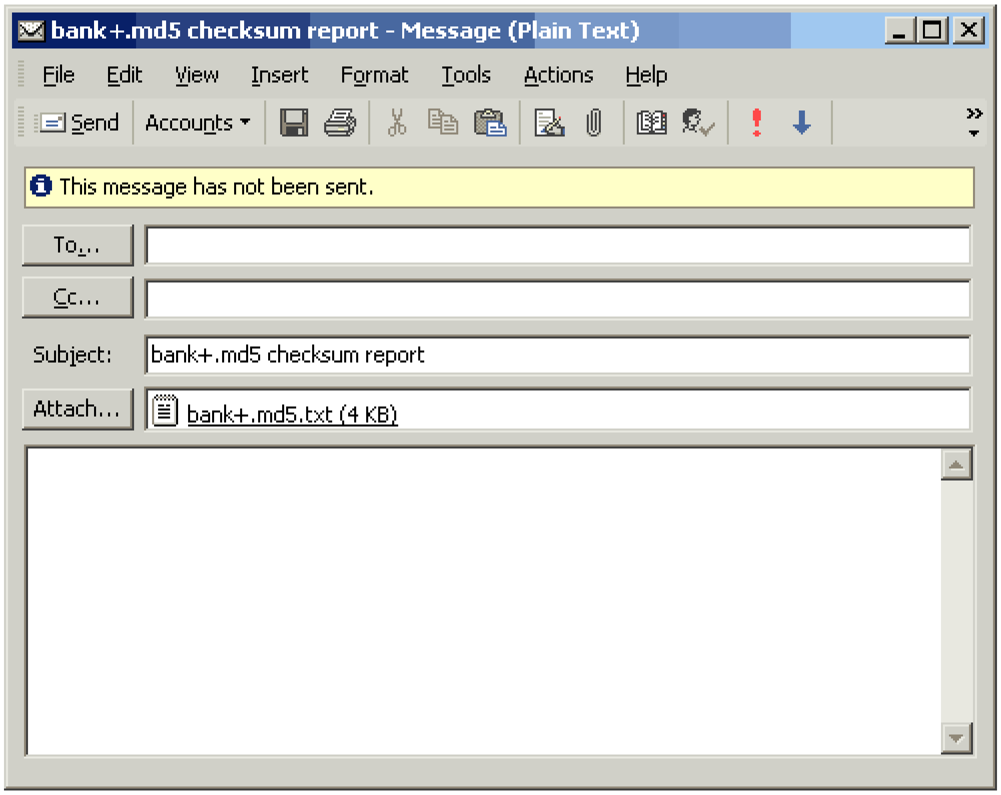
```
bank.doc :
  MD5:    BDA85E533A07D2E9B9BFB4239DE669F4
  SHA1:   2F04BDC23F72F7197FC21C40B1755F695A8E05B6
  RIPEMD: 61ECFAA45FA02B9F0C3DBBAC721A51952C78C7E6
bank1.doc :
  MD5:    144855BE553BD65AF926255C3411B88B
  SHA1:   6DE9E5CCBE69550CCE53FDA5E80342CE2DC766A2
  RIPEMD: F50EE86CE59F58BF420724F63CD5C78C3D085E09
bank2.doc :
  MD5:    28D28165F4249F4424EE74BCFFA15C61
  SHA1:   6BADF16E9D07C9575F9AAF9B594B3B681DCB36E8
  RIPEMD: 14AE69086CC0E70C422B2D774860C9E13733AA1B
Report in file bank+.md5.txt
```

[✓] Email signatures report to a friend

2.4 E-Mailing Reports

SecExMD5+ supports one-click emailing of digital signatures reports. If you check "***Email signatures report to a friend***" on the reports page, SecExMD5+ will open the compose message window in your favorite email client and enclose the newly generated report as a file attachment, ready for sending.

In order to use this feature, you need MAPI compliant email software such as MS Outlook or Eudora Mail.


bank+.md5 checksum report - Message (Plain Text)
File Edit View Insert Format Tools Actions Help
Send Accounts
To...
Cc...
Attach...

This message has not been sent.

Subject: bank+.md5 checksum report

bank+.md5.txt (4 KB)

3 Technical

3.1 MD5 Message-Digest Algorithm

The MD5 message digest algorithm specifies a standard for producing a unique 128 bit fingerprint or digital signature for documents of arbitrary length. MD5 was designed by Professor Ronald L. Rivest of MIT and if specified in RFC 1321.

3.2 US Secure Hash Algorithm 1 (SHA1)

The Secure Hash Algorithm 1 (SHA1) has been officially adopted by the United States government as the Secure Hash Standard of the Federal Information Processing Standard (FIPS) and is specified in RFC 3174. See also FIPS 180-1. SHA1 uses a 160 bit digest which is designed to replace earlier 128 bit message digests such as MD5. Successful attacks on the latter have been demonstrated in cryptographic literature.

Potential users of SHA1 are adivsed that in contrast to RIPEMD-160 the design criteria as well as an earlier attack on the SHA-1 algorithm are generally held to be secret. The use of RIPEMD-160 is therefore recommended over the use of SHA-1.

3.3 RIPEMD Cryptographic Hash Function

RIPEMD-160 is a cryptographic hash function designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel in the framework of the European Union's RACE Integrity Primitives Evaluation (RIPE) project. RIPEMD-160 is a 160 bit replacement for earlier 128 bit message digests such as MD5.

While MD5 is still widely used to verify the authenticity of files and data, successful attacks on the integrity of the MD5 message digests have been demonstrated. See 2nd ACM Conference on Computer and Communications Security, ACM Press, 1994, pp. 210-218. While the costs of carrying out such an attack using high end computing equipment were estimated at US $10 million, these costs are deemed to half every 18 months. All users of message digests are therefore urged to upgrade to either RIPEMD-160 or the United States Secure Hash Algorithm SHA-1. However, users are adivsed that in contrast to RIPEMD-160 the design criteria as well as an earlier attack on the SHA-1 algorithm are generally held to be secret. It is for this reason that the designers of the SecExMail email cipher, for example, have chosen the RIPEMD-160 algorithm.

3.4 RFC 3174

Network Working Group	D. Eastlake, 3rd
Request for Comments: 3174	Motorola
Category: Informational	P. Jones
	Cisco Systems
	September 2001

US Secure Hash Algorithm 1 (SHA1)

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

The purpose of this document is to make the SHA-1 (Secure Hash Algorithm 1) hash algorithm conveniently available to the Internet community. The United States of America has adopted the SHA-1 hash algorithm described herein as a Federal Information Processing Standard. Most of the text herein was taken by the authors from FIPS 180-1. Only the C code implementation is "original".

Acknowledgements

Most of the text herein was taken from [FIPS 180-1]. Only the C code implementation is "original" but its style is similar to the previously published MD4 and MD5 RFCs [RFCs 1320, 1321].

The SHA-1 is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm [MD4] and is modeled after that algorithm [RFC 1320].

Useful comments from the following, which have been incorporated herein, are gratefully acknowledged:

Tony Hansen
Garrett Wollman

Eastlake & Jones Informational [Page 1]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

Table of Contents

1. Overview of Contents

NOTE: The text below is mostly taken from [FIPS 180-1] and assertions therein of the security of SHA-1 are made by the US Government, the author of [FIPS 180-1], and not by the authors of this document.

This document specifies a Secure Hash Algorithm, SHA-1, for computing a condensed representation of a message or a data file. When a message of any length < 2^64 bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then, for example, be input to a signature algorithm which generates or verifies 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 in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital

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-1 is called secure because it is computationally infeasible to find a message which corresponds 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.

Eastlake & Jones Informational [Page 2]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

Section 2 below defines the terminology and functions used as building blocks to form SHA-1.

2. Definitions of Bit Strings and Integers

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

a. 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. Examples: 7 = 0111, A = 1010.

b. A word equals a 32-bit string which may be represented as a sequence of 8 hex digits. To convert a word to 8 hex digits each 4-bit string is converted to its hex equivalent as described in (a) above. Example:

   1010 0001 0000 0011 1111 1110 0010 0011 = A103FE23.

c. An integer between 0 and 2^32 - 1 inclusive may be represented as a word. The least significant four bits of the integer are represented by the right-most hex digit of the word representation. Example: the integer 291 = 2^8+2^5+2^1+2^0 = 256+32+2+1 is represented by the hex word, 00000123.

   If z is an integer, 0 <= z < 2^64, then z = (2^32)x + y where 0 <= x < 2^32 and 0 <= y < 2^32. Since x and y can be represented as words X and Y, respectively, z can be represented as the pair of words (X,Y).

d. block = 512-bit string. A block (e.g., B) may be represented as a sequence of 16 words.

3. Operations on Words

The following logical operators will be applied to words:

a. Bitwise logical word operations

   X AND Y = bitwise logical "and" of X and Y.

   X OR Y = bitwise logical "inclusive-or" of X and Y.

   X XOR Y = bitwise logical "exclusive-or" of X and Y.

   NOT X = bitwise logical "complement" of X.

Example:

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

b. The operation X + Y is defined as follows: words X and Y represent integers x and y, where 0 <= x < 2^32 and 0 <= y < 2^32. For positive integers n and m, let n mod m be the remainder upon dividing n by m. Compute

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

Then 0 <= z < 2^32. Convert z to a word, Z, and define Z = X + Y.

c. The circular left shift operation S^n(X), where X is a word and n is an integer with 0 <= n < 32, is defined by

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

In the above, X << n is obtained as follows: discard the left-most n bits of X and then pad the result with n zeroes on the right (the result will still be 32 bits). X >> n is obtained by discarding the right-most n bits of X and then padding the result with n 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

SHA-1 is used to compute a message digest for a message or data file that is provided as input. The message or data file should be considered to be a bit string. The length of the message is the number of bits in the message (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. The purpose of message padding is to make the total length of a padded message a multiple of 512. SHA-1 sequentially processes blocks of 512 bits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 512 * n. The 64-bit integer is the length of the original message. The padded message is then processed by the SHA-1 as n 512-bit blocks.

Eastlake & Jones   Informational   [Page 4]

RFC 3174   US Secure Hash Algorithm 1 (SHA1)   September 2001

Suppose a message has length l < 2^64. Before it is input to the SHA-1, the message is padded on the right as follows:

a. "1" is appended. Example: if the original message is "01010000", this is padded to "010100001".

b. "0"s are appended. The number of "0"s will depend on the original length of the message. The last 64 bits of the last 512-bit block are reserved

for the length l of the original message.

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.
```

Since l = 40, the number of bits in the above is 41 and 407 "0"s are appended, making the total now 448. This gives (in hex)

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

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 padded message will contain 16 * n words for some n > 0. The padded message is regarded as a sequence of n blocks M(1) , M(2), first characters (or bits) of the message.

Eastlake & Jones Informational [Page 5]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

5. Functions and Constants Used

A sequence of logical functions f(0), f(1),..., f(79) is used in SHA-1. Each f(t), 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. f(t;B,C,D) is defined as follows: for words B, C, D,

```
f(t;B,C,D) = (B AND C) OR ((NOT B) AND D)         ( 0 <= t <= 19)

f(t;B,C,D) = B XOR C XOR D                        (20 <= t <= 39)

f(t;B,C,D) = (B AND C) OR (B AND D) OR (C AND D)  (40 <= t <= 59)

f(t;B,C,D) = B XOR C XOR D                        (60 <= t <= 79).
```

A sequence of constant words K(0), K(1), ... , K(79) is used in the SHA-1. 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).
```

6. Computing the Message Digest

The methods given in 6.1 and 6.2 below yield the same message digest. Although using method 2 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 (c). There are other computation methods which give identical results.

6.1 Method 1

The message digest is computed using the message padded as described in section 4. The computation is described using 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.

Eastlake & Jones Informational [Page 6]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

Before processing any blocks, the H's 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 left-most word.

b. For t = 16 to 79 let

   W(t) = S^1(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.

6.2 Method 2

The method 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:

```
a. Divide M(i) into 16 words W[0], ... , W[15], where W[0] is the
   left-most word.

b. Let A = H0, B = H1, C = H2, D = H3, E = H4.

c. For t = 0 to 79 do

   s = t AND MASK;

   if (t >= 16) W[s] = S^1(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;

d. Let H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4
   + E.
```

7. C Code

Below is a demonstration implementation of SHA-1 in C. Section 7.1 contains the header file, 7.2 the C code, and 7.3 a test driver.

7.1 .h file

```
/*
 *  sha1.h
 *
 *  Description:
 *      This is the header file for code which implements the Secure
 *      Hashing Algorithm 1 as defined in FIPS PUB 180-1 published
 *      April 17, 1995.
 *
 *      Many of the variable names in this code, especially the
 *      single character names, were used because those were the names
 *      used in the publication.
 *
 *      Please read the file sha1.c for more information.
 *
 */
```

Eastlake & Jones   Informational   [Page 8]

RFC 3174   US Secure Hash Algorithm 1 (SHA1)   September 2001

```
#ifndef _SHA1_H_
#define _SHA1_H_

#include <stdint.h>
/*
 * If you do not have the ISO standard stdint.h header file, then you
 * must typdef the following:
 *    name              meaning
 *  uint32_t         unsigned 32 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 *  int_least16_t    integer of >= 16 bits
 *
 */

#ifndef _SHA_enum_
#define _SHA_enum_
enum
{
    shaSuccess = 0,
    shaNull,            /* Null pointer parameter */
    shaInputTooLong,    /* input data too long */
    shaStateError       /* called Input after Result */
};
#endif
#define SHA1HashSize 20

/*
 *  This structure will hold context information for the SHA-1
 *  hashing operation
 */
typedef struct SHA1Context
{
    uint32_t Intermediate_Hash[SHA1HashSize/4]; /* Message Digest  */

    uint32_t Length_Low;            /* Message length in bits      */
    uint32_t Length_High;           /* Message length in bits      */

                               /* Index into message block array   */
    int_least16_t Message_Block_Index;
    uint8_t Message_Block[64];      /* 512-bit message blocks      */

    int Computed;               /* Is the digest computed?         */
    int Corrupted;             /* Is the message digest corrupted? */
} SHA1Context;

/*
 *  Function Prototypes
 */
```

```
int SHA1Reset(  SHA1Context *);
int SHA1Input(  SHA1Context *,
                const uint8_t *,
                unsigned int);
int SHA1Result( SHA1Context *,
                uint8_t Message_Digest[SHA1HashSize]);

#endif
```

7.2 .c file

```
/*
 *  sha1.c
 *
 *  Description:
 *      This file implements the Secure Hashing Algorithm 1 as
 *      defined in FIPS PUB 180-1 published April 17, 1995.
 *
 *      The SHA-1, produces a 160-bit message digest for a given
 *      data stream.  It should take about 2**n steps to find a
 *      message with the same digest as a given message and
 *      2**(n/2) to find any two messages with the same digest,
 *      when n is the digest size in bits.  Therefore, this
 *      algorithm can serve as a means of providing a
 *      "fingerprint" for a message.
 *
 *  Portability Issues:
 *      SHA-1 is defined in terms of 32-bit "words".  This code
 *      uses <stdint.h> (included via "sha1.h" to define 32 and 8
 *      bit unsigned integer types.  If your C compiler does not
 *      support 32 bit unsigned integers, this code is not
 *      appropriate.
 *
 *  Caveats:
 *      SHA-1 is designed to work with messages less than 2^64 bits
 *      long.  Although SHA-1 allows a message digest to be generated
 *      for messages of any number of bits less than 2^64, this
 *      implementation only works with messages with a length that is
 *      a multiple of the size of an 8-bit character.
 *
 */
```

```
#include "sha1.h"

/*
 *  Define the SHA1 circular left shift macro
 */
#define SHA1CircularShift(bits,word) \
                (((word) << (bits)) | ((word) >> (32-(bits))))

/* Local Function Prototyptes */
void SHA1PadMessage(SHA1Context *);
void SHA1ProcessMessageBlock(SHA1Context *);

/*
 *  SHA1Reset
 *
 *  Description:
 *      This function will initialize the SHA1Context in preparation
 *      for computing a new SHA1 message digest.
 *
 *  Parameters:
 *      context: [in/out]
 *          The context to reset.
 *
 *  Returns:
 *      sha Error Code.
 *
 */
int SHA1Reset(SHA1Context *context)
{
    if (!context)
    {
        return shaNull;
    }

    context->Length_Low             = 0;
    context->Length_High            = 0;
    context->Message_Block_Index    = 0;

    context->Intermediate_Hash[0]   = 0x67452301;
    context->Intermediate_Hash[1]   = 0xEFCDAB89;
    context->Intermediate_Hash[2]   = 0x98BADCFE;
    context->Intermediate_Hash[3]   = 0x10325476;
    context->Intermediate_Hash[4]   = 0xC3D2E1F0;

    context->Computed   = 0;
    context->Corrupted  = 0;
```

Eastlake & Jones Informational [Page 11]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

```
    return shaSuccess;
}

/*
 *  SHA1Result
 *
 *  Description:
 *      This function will return the 160-bit message digest into the
 *      Message_Digest array  provided by the caller.
 *      NOTE: The first octet of hash is stored in the 0th element,
 *            the last octet of hash in the 19th element.
 *
 *  Parameters:
 *      context: [in/out]
 *          The context to use to calculate the SHA-1 hash.
 *      Message_Digest: [out]
 *          Where the digest is returned.
 *
 *  Returns:
 *      sha Error Code.
 *
 */
int SHA1Result( SHA1Context *context,
                uint8_t Message_Digest[SHA1HashSize])
{
    int i;

    if (!context || !Message_Digest)
    {
        return shaNull;
    }

    if (context->Corrupted)
    {
        return context->Corrupted;
    }

    if (!context->Computed)
    {
        SHA1PadMessage(context);
        for(i=0; i<64; ++i)
        {
            /* message may be sensitive, clear it out */
            context->Message_Block[i] = 0;
        }
        context->Length_Low = 0;    /* and clear length */
        context->Length_High = 0;
        context->Computed = 1;
    }

    for(i = 0; i < SHA1HashSize; ++i)
    {
        Message_Digest[i] = context->Intermediate_Hash[i>>2]
                            >> 8 * ( 3 - ( i & 0x03 ) );
    }

    return shaSuccess;
}

/*
 *  SHA1Input
 *
 *  Description:
 *      This function accepts an array of octets as the next portion
 *      of the message.
 *
 *  Parameters:
 *      context: [in/out]
 *          The SHA context to update
 *      message_array: [in]
 *          An array of characters representing the next portion of
 *          the message.
 *      length: [in]
 *          The length of the message in message_array
 *
 *  Returns:
 *      sha Error Code.
 *
 */
int SHA1Input(    SHA1Context    *context,
                  const uint8_t  *message_array,
                  unsigned       length)
{
    if (!length)
    {
        return shaSuccess;
    }

    if (!context || !message_array)
    {
        return shaNull;
    }

    if (context->Computed)
    {
        context->Corrupted = shaStateError;
```

```
Eastlake & Jones           Informational                    [Page 13]

RFC 3174          US Secure Hash Algorithm 1 (SHA1)      September 2001


        return shaStateError;
    }

    if (context->Corrupted)
    {
         return context->Corrupted;
    }
    while(length-- && !context->Corrupted)
    {
    context->Message_Block[context->Message_Block_Index++] =
                    (*message_array & 0xFF);

    context->Length_Low += 8;
    if (context->Length_Low == 0)
    {
        context->Length_High++;
        if (context->Length_High == 0)
        {
            /* Message is too long */
            context->Corrupted = 1;
        }
    }

    if (context->Message_Block_Index == 64)
    {
        SHA1ProcessMessageBlock(context);
    }

    message_array++;
    }

    return shaSuccess;
}

/*
 *  SHA1ProcessMessageBlock
 *
 *  Description:
 *      This function will process the next 512 bits of the message
 *      stored in the Message_Block array.
 *
 *  Parameters:
 *      None.
 *
 *  Returns:
 *      Nothing.
 *
 *  Comments:
```

```
 *      Many of the variable names in this code, especially the
 *      single character names, were used because those were the
 *      names used in the publication.
 *
 *
 */
void SHA1ProcessMessageBlock(SHA1Context *context)
{
    const uint32_t K[] =    {       /* Constants defined in SHA-1   */

                            0x5A827999,
                            0x6ED9EBA1,
                            0x8F1BBCDC,
                            0xCA62C1D6
                            };
    int           t;                 /* Loop counter                */
    uint32_t      temp;              /* Temporary word value        */
    uint32_t      W[80];             /* Word sequence               */
    uint32_t      A, B, C, D, E;     /* Word buffers                */

    /*
     *  Initialize the first 16 words in the array W
     */
    for(t = 0; t < 16; t++)
    {
        W[t] = context->Message_Block[t * 4] << 24;
        W[t] |= context->Message_Block[t * 4 + 1] << 16;
        W[t] |= context->Message_Block[t * 4 + 2] << 8;
        W[t] |= context->Message_Block[t * 4 + 3];
    }

    for(t = 16; t < 80; t++)
    {
       W[t] = SHA1CircularShift(1,W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16]);
    }

    A = context->Intermediate_Hash[0];
    B = context->Intermediate_Hash[1];
    C = context->Intermediate_Hash[2];
    D = context->Intermediate_Hash[3];
    E = context->Intermediate_Hash[4];

    for(t = 0; t < 20; t++)
    {
        temp =  SHA1CircularShift(5,A) +
                ((B & C) | ((~B) & D)) + E + W[t] + K[0];
        E = D;
        D = C;
        C = SHA1CircularShift(30,B);
Eastlake & Jones            Informational                   [Page 15]

RFC 3174          US Secure Hash Algorithm 1 (SHA1)      September 2001


        B = A;
        A = temp;
    }

    for(t = 20; t < 40; t++)
    {
        temp = SHA1CircularShift(5,A) + (B ^ C ^ D) + E + W[t] + K[1];
        E = D;
        D = C;
        C = SHA1CircularShift(30,B);
        B = A;
        A = temp;
    }

    for(t = 40; t < 60; t++)
    {
        temp = SHA1CircularShift(5,A) +
               ((B & C) | (B & D) | (C & D)) + E + W[t] + K[2];
        E = D;
        D = C;
        C = SHA1CircularShift(30,B);
        B = A;
        A = temp;
    }

    for(t = 60; t < 80; t++)
    {
        temp = SHA1CircularShift(5,A) + (B ^ C ^ D) + E + W[t] + K[3];
        E = D;
        D = C;
        C = SHA1CircularShift(30,B);
        B = A;
        A = temp;
    }

    context->Intermediate_Hash[0] += A;
    context->Intermediate_Hash[1] += B;
    context->Intermediate_Hash[2] += C;
    context->Intermediate_Hash[3] += D;
    context->Intermediate_Hash[4] += E;

    context->Message_Block_Index = 0;
}


/*
 *  SHA1PadMessage
 *
```

```
Eastlake & Jones           Informational                  [Page 16]

RFC 3174          US Secure Hash Algorithm 1 (SHA1)      September 2001


 *  Description:
 *      According to the standard, the message must be padded to an even
 *      512 bits.  The first padding bit must be a '1'.  The last 64
 *      bits represent the length of the original message.  All bits in
 *      between should be 0.  This function will pad the message
 *      according to those rules by filling the Message_Block array
 *      accordingly.  It will also call the ProcessMessageBlock function
 *      provided appropriately.  When it returns, it can be assumed that
 *      the message digest has been computed.
 *
 *  Parameters:
 *      context: [in/out]
 *          The context to pad
 *      ProcessMessageBlock: [in]
 *          The appropriate SHA*ProcessMessageBlock function
 *  Returns:
 *      Nothing.
 *
 */

void SHA1PadMessage(SHA1Context *context)
{
    /*
     *  Check to see if the current message block is too small to hold
     *  the initial padding bits and length.  If so, we will pad the
     *  block, process it, and then continue padding into a second
     *  block.
     */
    if (context->Message_Block_Index > 55)
    {
        context->Message_Block[context->Message_Block_Index++] = 0x80;
        while(context->Message_Block_Index < 64)
        {
            context->Message_Block[context->Message_Block_Index++] = 0;
        }

        SHA1ProcessMessageBlock(context);

        while(context->Message_Block_Index < 56)
        {
            context->Message_Block[context->Message_Block_Index++] = 0;
        }
    }
    else
    {
        context->Message_Block[context->Message_Block_Index++] = 0x80;
        while(context->Message_Block_Index < 56)
        {
            context->Message_Block[context->Message_Block_Index++] = 0;
        }
    }

    /*
     *  Store the message length as the last 8 octets
     */
    context->Message_Block[56] = context->Length_High >> 24;
    context->Message_Block[57] = context->Length_High >> 16;
    context->Message_Block[58] = context->Length_High >> 8;
    context->Message_Block[59] = context->Length_High;
    context->Message_Block[60] = context->Length_Low >> 24;
    context->Message_Block[61] = context->Length_Low >> 16;
    context->Message_Block[62] = context->Length_Low >> 8;
    context->Message_Block[63] = context->Length_Low;

    SHA1ProcessMessageBlock(context);
}
```

7.3 Test Driver

The following code is a main program test driver to exercise the code in sha1.c.

```
/*
 *  sha1test.c
 *
 *  Description:
 *      This file will exercise the SHA-1 code performing the three
 *      tests documented in FIPS PUB 180-1 plus one which calls
 *      SHA1Input with an exact multiple of 512 bits, plus a few
 *      error test checks.
 *
 *  Portability Issues:
 *      None.
 *
 */

#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include "sha1.h"

/*
 *  Define patterns for testing
 */
#define TEST1   "abc"
#define TEST2a  "abcdbcdecdefdefgefghfghighijhi"
```

Eastlake & Jones   Informational   [Page 18]

RFC 3174   US Secure Hash Algorithm 1 (SHA1)   September 2001

```
#define TEST2b "jkijkljklmklmnlmnomnopnopq"
#define TEST2   TEST2a TEST2b
#define TEST3   "a"
#define TEST4a  "01234567012345670123456701234567"
#define TEST4b  "01234567012345670123456701234567"
    /* an exact multiple of 512 bits */
#define TEST4   TEST4a TEST4b
char *testarray[4] =
{
    TEST1,
    TEST2,
    TEST3,
    TEST4
};
long int repeatcount[4] = { 1, 1, 1000000, 10 };
char *resultarray[4] =
{
    "A9 99 3E 36 47 06 81 6A BA 3E 25 71 78 50 C2 6C 9C D0 D8 9D",
    "84 98 3E 44 1C 3B D2 6E BA AE 4A A1 F9 51 29 E5 E5 46 70 F1",
    "34 AA 97 3C D4 C4 DA A4 F6 1E EB 2B DB AD 27 31 65 34 01 6F",
    "DE A3 56 A2 CD DD 90 C7 A7 EC ED C5 EB B5 63 93 4F 46 04 52"
};

int main()
{
    SHA1Context sha;
    int i, j, err;
    uint8_t Message_Digest[20];

    /*
     *  Perform SHA-1 tests
     */
    for(j = 0; j < 4; ++j)
    {
        printf( "\nTest %d: %d, '%s'\n",
                j+1,
                repeatcount[j],
                testarray[j]);

        err = SHA1Reset(&sha);
        if (err)
        {
            fprintf(stderr, "SHA1Reset Error %d.\n", err );
            break;    /* out of for j loop */
        }

        for(i = 0; i < repeatcount[j]; ++i)
        {
```

Eastlake & Jones Informational [Page 19]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

```
            err = SHA1Input(&sha,
                  (const unsigned char *) testarray[j],
                  strlen(testarray[j]));
            if (err)
            {
                fprintf(stderr, "SHA1Input Error %d.\n", err );
                break;    /* out of for i loop */
            }
        }

        err = SHA1Result(&sha, Message_Digest);
        if (err)
        {
            fprintf(stderr,
            "SHA1Result Error %d, could not compute message digest.\n",
            err );
        }
        else
        {
            printf("\t");
            for(i = 0; i < 20 ; ++i)
            {
                printf("%02X ", Message_Digest[i]);
            }
            printf("\n");

        }
        printf("Should match:\n");
        printf("\t%s\n", resultarray[j]);
    }

    /* Test some error returns */
    err = SHA1Input(&sha,(const unsigned char *) testarray[1], 1);
    printf ("\nError %d. Should be %d.\n", err, shaStateError );
    err = SHA1Reset(0);
    printf ("\nError %d. Should be %d.\n", err, shaNull );
    return 0;
}
```

8. Security Considerations

This document is intended to provide convenient open source access by the Internet community to the United States of America Federal Information Processing Standard Secure Hash Function SHA-1 [FIPS 180-1]. No independent assertion of the security of this hash function by the authors for any particular use is intended.

Eastlake & Jones Informational [Page 20]

RFC 3174 US Secure Hash Algorithm 1 (SHA1) September 2001

References

[FIPS 180-1] "Secure Hash Standard", United States of American, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, April 1993.

[MD4] "The MD4 Message Digest Algorithm," Advances in Cryptology - CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 303-311.

[RFC 1320] Rivest, R., "The MD4 Message-Digest Algorithm", RFC 1320, April 1992.

[RFC 1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC 1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Requirements for Security", RFC 1750, December 1994.

Authors' Addresses

Donald E. Eastlake, 3rd
Motorola
155 Beaver Street
Milford, MA 01757 USA

Phone: +1 508-634-2066 (h)
+1 508-261-5434 (w)
Fax: +1 508-261-4777
EMail: Donald.Eastlake@motorola.com

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Road
Research Triangle Park, NC 27709 USA

Phone: +1 919 392 6948
EMail: paulej@packetizer.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Eastlake & Jones Informational [Page 22]

3.5 RFC 1321

Network Working Group R. Rivest
Request for Comments: 1321 MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

The MD5 Message-Digest Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Acknowlegements

We would like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, David Chaum, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents

1. Executive Summary

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

Rivest [Page 1]

RFC 1321 MD5 Message-Digest Algorithm April 1992

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

For OSI-based applications, MD5's object identifier is

```
md5 OBJECT IDENTIFIER ::=
  iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

In the X.509 type AlgorithmIdentifier [3], the parameters for MD5 should have type NULL.

2. Terminology and Notation

In this document a "word" is a 32-bit quantity and a "byte" is an eight-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo-2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

Rivest [Page 2]

RFC 1321 MD5 Message-Digest Algorithm April 1992

3. MD5 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

3.1 Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

3.2 Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3.3 Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

Rivest [Page 3]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B
```

Rivest [Page 4]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  CC = C
  DD = D

  /* Round 1. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
  [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
  [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
  [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

  /* Round 2. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
  [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
  [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
  [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

  /* Round 3. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
  [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
  [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
  [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

  /* Round 4. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
  [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
  [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
  [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* of loop on i */
```

Rivest [Page 5]

RFC 1321 MD5 Message-Digest Algorithm April 1992

3.5 Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD5. A reference implementation in C is given in the appendix.

4. Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

5. Differences Between MD4 and MD5

The following are the differences between MD4 and MD5:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function g in round 2 was changed from (XY v XZ v YZ) to (XZ v Y not(Z)) to make g less symmetric.

4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

Rivest [Page 6]

RFC 1321 MD5 Message-Digest Algorithm April 1992

References

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

APPENDIX A - Reference Implementation

This appendix contains the following files taken from RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h -- global header file

md5.h -- header file for MD5

md5c.c -- source code for MD5

For more information on RSAREF, send email to <rsaref@rsa.com>.

The appendix also includes the following file:

mddriver.c -- test driver for MD2, MD4 and MD5

The driver compiles for MD5 by default but can compile for MD2 or MD4 if the symbol MD is defined on the C compiler command line as 2 or 4.

The implementation is portable and should work on many different plaforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader. For example, on "little-endian" platforms where the lowest-addressed byte in a 32-bit word is the least significant and there are no alignment restrictions, the call to Decode in MD5Transform can be replaced with a typecast.

A.1 global.h

```
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already
```

Rivest [Page 7]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.
```

```
Rivest                                          [Page 8]

RFC 1321          MD5 Message-Digest Algorithm          April 1992


These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));

A.3 md5c.c

/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */
```

```
Rivest                                    [Page 9]

RFC 1321         MD5 Message-Digest Algorithm          April 1992


#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
Rivest                                            [Page 10]

RFC 1321            MD5 Message-Digest Algorithm           April 1992


 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                   /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                   /* context */
unsigned char *input;                          /* input block */
unsigned int inputLen;                  /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))
```

Rivest [Page 11]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
   < ((UINT4)inputLen << 3))
 context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.
*/
  if (inputLen >= partLen) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Buffer remaining input */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                /* message digest */
MD5_CTX *context;                            /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64.
*/
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);
```

Rivest [Page 12]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
```

Rivest [Page 13]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Round 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Round 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information.
```

Rivest [Page 14]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
*/
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - test driver for MD2, MD4 and MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
rights reserved.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* The following makes MD default to MD5 if it has not already been
  defined with C compiler flags.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
```

Rivest [Page 16]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Length of test block, number of test blocks.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Main driver.

Arguments (may be any combination):
  -sstring - digests string
  -t       - runs time trial
  -x       - runs test script
  filename - digests file
  (none)   - digests standard input
 */
int main (argc, argv)
int argc;
```

Rivest [Page 17]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Digests a string and prints the result.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Measures the time to digest TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte
  blocks.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;
```

Rivest [Page 18]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  printf
 ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
  TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Initialize block */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

  /* Start timer */
  time (&startTime);

  /* Digest blocks */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
 MDUpdate (&context, block, TEST_BLOCK_LEN);
  MDFinal (digest, &context);

  /* Stop timer */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
 ("Speed = %ld bytes/second\n",
  (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Digests a reference suite of strings and prints the results.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString
 ("1234567890123456789012345678901234567890\
1234567890123456789012345678901234567890");
}

/* Digests a file and prints the result.
```

Rivest [Page 19]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

  else {
 MDInit (&context);
 while (len = fread (buffer, 1, 1024, file))
   MDUpdate (&context, buffer, len);
 MDFinal (digest, &context);

 fclose (file);

 printf ("MD%d (%s) = ", MD, filename);
 MDPrint (digest);
 printf ("\n");
  }
}

/* Digests the standard input and prints the result.
 */
static void MDFilter ()
{
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
 MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
  printf ("\n");
}

/* Prints a message digest in hexadecimal.
 */
static void MDPrint (digest)
unsigned char digest[16];
{
```

Rivest [Page 20]

RFC 1321 MD5 Message-Digest Algorithm April 1992

```
  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Test suite

The MD5 test suite (driver option "-x") should print the following results:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("123456789012345678901234567890123456789012345678901234567890123456
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Security Considerations

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Author's Address

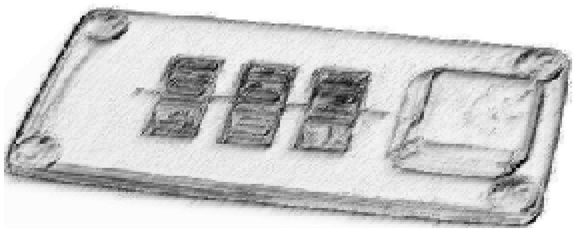
Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu

Rivest [Page 21]

4 About

4.1 About SecExMD5+



SecExMD5+
Version 1.1
Copyright © 2002, Bytefusion Ltd.
All Rights Reserved

4.2 About Bytefusion Ltd.

Bytefusion Ltd.
22 Duke Street
Douglas, IOM
IM1 2AY
British Isles

Inquiries: sales@bytefusion.com

4.3 Acknowledgements

- **RIPEMD-160**

The RIPE message digest was written by Antoon Bosselaers for Katholieke Universiteit Leuven, Department of Electrical Engineering ESAT/COSIC, Belgium. License conditions ask us to quote the following :

```
"RIPEMD-160 software written by Antoon Bosselaers,
available at http://www.esat.kuleuven.ac.be/~cosicart/ps/AB-9601/ "
```

- **OpenSSL Project**

  SecExFile contains cryptographic software from the OpenSSL project at www.openssl.org which is licensed under a "*BSD-style*" open source licenses. These licenses asks us to state the following :

  ```
  "This product includes software developed by the OpenSSL Project for use
  in the OpenSSL Toolkit. (http://www.openssl.org/)"

  "This product includes cryptographic software written by  Eric Young
  (eay@cryptsoft.com). This product includes software written by Tim Hudson
  (tjh@cryptsoft.com)."
  ```

  SecExFile is an independent, derived work and no endorsement of SecExFile by the OpenSSL project is implied. The full text of the OpenSSL license and the original SSLeay License is reproduced below.