Network Working Group
Request for Comments: 1319
Updates: RFC 1115

B. Kaliski
RSA Laboratories

April 1992

The MD2 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

The description of MD2 is based on material prepared by John Linn and Ron Rivest. Their permission to incorporate that material is greatly appreciated.

Table of Contents

1. Executive Summary

This document describes the MD2 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 MD2 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being signed with a private (secret) key under a public-key cryptosystem such as RSA.

License to use MD2 is granted for non-commerical Internet Privacy-Enhanced Mail [1-3].

This document is an update to the August 1989 RFC 1115 [3], which also gives a reference implementation of MD2. The main differences

are that a textual description of MD2 is included, and that the reference implementation of MD2 is more portable.

For OSI-based applications, MD2's object identifier is

md2 OBJECT IDENTIFIER ::=
iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 2}

In the X.509 type AlgorithmIdentifier [4], the parameters for MD2 should have type NULL.

2. Terminology and Notation

In this document, a "byte" is an eight-bit quantity.

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 X xor Y denote the bit-wise XOR of X and Y.

3. MD2 Algorithm Description

We begin by supposing that we have a b-byte message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, and it may be arbitrarily large. We imagine the bytes 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 Bytes

The message is "padded" (extended) so that its length (in bytes) is congruent to 0, modulo 16. That is, the message is extended so that it is a multiple of 16 bytes long. Padding is always performed, even if the length of the message is already congruent to 0, modulo 16.

Padding is performed as follows: "i" bytes of value "i" are appended to the message so that the length in bytes of the padded message becomes congruent to 0, modulo 16. At least one byte and at most 16 16 bytes are appended.

At this point the resulting message (after padding with bytes) has a length that is an exact multiple of 16 bytes. Let M[0 ... N-1] denote

the bytes of the resulting message, where N is a multiple of 16.

3.2 Step 2. Append Checksum

A 16-byte checksum of the message is appended to the result of the previous step.

This step uses a 256-byte "random" permutation constructed from the digits of pi. Let S[i] denote the i-th element of this table. The table is given in the appendix.

Do the following:

```
/* Clear checksum. */
For i = 0 to 15 do:
   Set C[i] to 0.
end /* of loop on i */

Set L to 0.

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

   /* Checksum block i. */
   For j = 0 to 15 do
      Set c to M[i*16+j].
      Set C[j] to S[c xor L].
      Set L to C[j].
    end /* of loop on j */
 end /* of loop on i */
```

The 16-byte checksum C[0 ... 15] is appended to the message. Let M[0 with checksum), where N' = N + 16.

3.3 Step 3. Initialize MD Buffer

A 48-byte buffer X is used to compute the message digest. The buffer is initialized to zero.

3.4 Step 4. Process Message in 16-Byte Blocks

This step uses the same 256-byte permutation S as step 2 does.

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[16+j] to M[i*16+j].
         Set X[32+j] to (X[16+j] xor X[j]).
       end /* of loop on j */

      Set t to 0.

      /* Do 18 rounds. */
      For j = 0 to 17 do

         /* Round j. */
         For k = 0 to 47 do
            Set t and X[k] to (X[k] xor S[t]).
         end /* of loop on k */

         Set t to (t+j) modulo 256.
      end /* of loop on j */

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

3.5 Step 5. Output

The message digest produced as output is X[0 ... 15]. That is, we begin with X[0], and end with X[15].

This completes the description of MD2. A reference implementation in C is given in the appendix.

4. Summary

The MD2 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 MD2 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.

References

[1] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures", RFC 1113, DEC, IAB Privacy Task Force, August 1989.

[2] Kent, S., and J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", RFC 1114, BBNCC, DEC, IAB Privacy Task Force, August 1989.

[3] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers", RFC 1115 DEC, IAB Privacy Task Force, August 1989.

[4] 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

md2.h -- header file for MD2

md2c.c -- source code for MD2

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.

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
     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 md2.h

```
/* MD2.H - header file for MD2C.C
 */

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

   License to copy and use this software is granted for
   non-commercial Internet Privacy-Enhanced Mail provided that it is
   identified as the "RSA Data Security, Inc. MD2 Message Digest
   Algorithm" in all material mentioning or referencing this software
   or this function.

   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.
 */
```

```
typedef struct {
  unsigned char state[16];                                  /* state */
  unsigned char checksum[16];                            /* checksum */
  unsigned int count;                /* number of bytes, modulo 16 */
  unsigned char buffer[16];                           /* input buffer */
} MD2_CTX;

void MD2Init PROTO_LIST ((MD2_CTX *));
void MD2Update PROTO_LIST
  ((MD2_CTX *, unsigned char *, unsigned int));
void MD2Final PROTO_LIST ((unsigned char [16], MD2_CTX *));
```

A.3 md2c.c

```
/* MD2C.C - RSA Data Security, Inc., MD2 message-digest algorithm
 */

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

   License to copy and use this software is granted for
   non-commercial Internet Privacy-Enhanced Mail provided that it is
   identified as the "RSA Data Security, Inc. MD2 Message Digest
   Algorithm" in all material mentioning or referencing this software
   or this function.

   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 "md2.h"

static void MD2Transform PROTO_LIST
  ((unsigned char [16], unsigned char [16], unsigned char [16]));
static void MD2_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD2_memset PROTO_LIST ((POINTER, int, unsigned int));

/* Permutation of 0..255 constructed from the digits of pi. It gives a
   "random" nonlinear byte substitution operation.
 */
static unsigned char PI_SUBST[256] = {
  41, 46, 67, 201, 162, 216, 124, 1, 61, 54, 84, 161, 236, 240, 6,
```

```
  19, 98, 167, 5, 243, 192, 199, 115, 140, 152, 147, 43, 217, 188,
  76, 130, 202, 30, 155, 87, 60, 253, 212, 224, 22, 103, 66, 111, 24,
  138, 23, 229, 18, 190, 78, 196, 214, 218, 158, 222, 73, 160, 251,
  245, 142, 187, 47, 238, 122, 169, 104, 121, 145, 21, 178, 7, 63,
  148, 194, 16, 137, 11, 34, 95, 33, 128, 127, 93, 154, 90, 144, 50,
  39, 53, 62, 204, 231, 191, 247, 151, 3, 255, 25, 48, 179, 72, 165,
  181, 209, 215, 94, 146, 42, 172, 86, 170, 198, 79, 184, 56, 210,
  150, 164, 125, 182, 118, 252, 107, 226, 156, 116, 4, 241, 69, 157,
  112, 89, 100, 113, 135, 32, 134, 91, 207, 101, 230, 45, 168, 2, 27,
  96, 37, 173, 174, 176, 185, 246, 28, 70, 97, 105, 52, 64, 126, 15,
  85, 71, 163, 35, 221, 81, 175, 58, 195, 92, 249, 206, 186, 197,
  234, 38, 44, 83, 13, 110, 133, 40, 132, 9, 211, 223, 205, 244, 65,
  129, 77, 82, 106, 220, 55, 200, 108, 193, 171, 250, 36, 225, 123,
  8, 12, 189, 177, 74, 120, 136, 149, 139, 227, 99, 232, 109, 233,
  203, 213, 254, 59, 0, 29, 57, 242, 239, 183, 14, 102, 88, 208, 228,
  166, 119, 114, 248, 235, 117, 75, 10, 49, 68, 80, 180, 143, 237,
  31, 26, 219, 153, 141, 51, 159, 17, 131, 20
};

static unsigned char *PADDING[] = {
  (unsigned char *)"",
  (unsigned char *)"\001",
  (unsigned char *)"\002\002",
  (unsigned char *)"\003\003\003",
  (unsigned char *)"\004\004\004\004",
  (unsigned char *)"\005\005\005\005\005",
  (unsigned char *)"\006\006\006\006\006\006",
  (unsigned char *)"\007\007\007\007\007\007\007",
  (unsigned char *)"\010\010\010\010\010\010\010\010",
  (unsigned char *)"\011\011\011\011\011\011\011\011\011",
  (unsigned char *)"\012\012\012\012\012\012\012\012\012\012",
  (unsigned char *)"\013\013\013\013\013\013\013\013\013\013\013",
  (unsigned char *)"\014\014\014\014\014\014\014\014\014\014\014\014",
  (unsigned char *)
    "\015\015\015\015\015\015\015\015\015\015\015\015\015",
  (unsigned char *)
    "\016\016\016\016\016\016\016\016\016\016\016\016\016\016",
  (unsigned char *)
    "\017\017\017\017\017\017\017\017\017\017\017\017\017\017\017",
  (unsigned char *)
    "\020\020\020\020\020\020\020\020\020\020\020\020\020\020\020\020"
};

/* MD2 initialization. Begins an MD2 operation, writing a new context.
 */
void MD2Init (context)
MD2_CTX *context;                                       /* context */
{
```

```
  context->count = 0;
  MD2_memset ((POINTER)context->state, 0, sizeof (context->state));
  MD2_memset
    ((POINTER)context->checksum, 0, sizeof (context->checksum));
}

/* MD2 block update operation. Continues an MD2 message-digest
     operation, processing another message block, and updating the
     context.
 */
void MD2Update (context, input, inputLen)
MD2_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Update number of bytes mod 16 */
  index = context->count;
  context->count = (index + inputLen) & 0xf;

  partLen = 16 - index;

  /* Transform as many times as possible.
    */
  if (inputLen >= partLen) {
    MD2_memcpy
      ((POINTER)&context->buffer[index], (POINTER)input, partLen);
    MD2Transform (context->state, context->checksum, context->buffer);

    for (i = partLen; i + 15 < inputLen; i += 16)
      MD2Transform (context->state, context->checksum, &input[i]);

    index = 0;
  }
  else
    i = 0;

  /* Buffer remaining input */
  MD2_memcpy
    ((POINTER)&context->buffer[index], (POINTER)&input[i],
     inputLen-i);
}

/* MD2 finalization. Ends an MD2 message-digest operation, writing the
     message digest and zeroizing the context.
 */
void MD2Final (digest, context)
```

```
unsigned char digest[16];                          /* message digest */
MD2_CTX *context;                                         /* context */
{
  unsigned int index, padLen;

  /* Pad out to multiple of 16.
   */
  index = context->count;
  padLen = 16 - index;
  MD2Update (context, PADDING[padLen], padLen);

  /* Extend with checksum */
  MD2Update (context, context->checksum, 16);

  /* Store state in digest */
  MD2_memcpy ((POINTER)digest, (POINTER)context->state, 16);

  /* Zeroize sensitive information.
   */
  MD2_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD2 basic transformation. Transforms state and updates checksum
     based on block.
 */
static void MD2Transform (state, checksum, block)
unsigned char state[16];
unsigned char checksum[16];
unsigned char block[16];
{
  unsigned int i, j, t;
  unsigned char x[48];

  /* Form encryption block from state, block, state ^ block.
   */
  MD2_memcpy ((POINTER)x, (POINTER)state, 16);
  MD2_memcpy ((POINTER)x+16, (POINTER)block, 16);
  for (i = 0; i < 16; i++)
    x[i+32] = state[i] ^ block[i];

  /* Encrypt block (18 rounds).
   */
  t = 0;
  for (i = 0; i < 18; i++) {
    for (j = 0; j < 48; j++)
      t = x[j] ^= PI_SUBST[t];
    t = (t + i) & 0xff;
  }
```

```
  /* Save new state */
  MD2_memcpy ((POINTER)state, (POINTER)x, 16);

  /* Update checksum.
   */
  t = checksum[15];
  for (i = 0; i < 16; i++)
    t = checksum[i] ^= PI_SUBST[block[i] ^ t];

  /* Zeroize sensitive information.
   */
  MD2_memset ((POINTER)x, 0, sizeof (x));
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */
static void MD2_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 MD2_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
#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;
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;

  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
    ("12345678901234567890123456789012345678901234567890\
12345678901234567890123456789012345678901234567890");
}

/* Digests a file and prints the result.
 */
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];
{
  unsigned int i;

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

A.5 Test suite

The MD2 test suite (driver option "-x") should print the following results:

```
MD2 test suite:
MD2 ("") = 8350e5a3e24c153df2275c9f80692773
MD2 ("a") = 32ec01ec4a6dac72c0ab96fb34c0b5d1
MD2 ("abc") = da853b0d3f88d99b30283a69e6ded6bb
MD2 ("message digest") = ab4f496bfb2a530b219ff33031fe06b0
MD2 ("abcdefghijklmnopqrstuvwxyz") = 4e8ddff3650292ab5a4108c3aa47940b
MD2 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
da33def2a42df13975352846c30338cd
MD2 ("12345678901234567890123456789012345678901234567890123456789012345
78901234567890") = d5976f79d83d3a0dc9806c3c66f3efd8
```

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 MD2 and a public-key cryptosystem.

Author's Address

Burton S. Kaliski Jr.
RSA Laboratories (a division of RSA Data Security, Inc.)
10 Twin Dolphin Drive
Redwood City, CA 94065

Phone: (415) 595-8782
FAX: (415) 595-4126
EMail: burt@rsa.com