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

Acknowledgements

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 pre-specified 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 dscription 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 non-negative 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 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-byte 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 C[j] xor 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..N′−1] be the message with padding and checksum appended, 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

Do the following:

```
/* Process each 16-byte 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.

Bibliography

[1] J. Linn, “Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures”. RFC 1113, DEC, IAB Privacy Task Force, Aug. 01, 1989.

[2] S. Kent and J. Linn, “Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management”. RFC 1114, BBNCC, DEC, IAB Privacy Task Force, Aug. 01, 1989.

[3] J. Linn, “Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers”. RFC 1115, DEC, IAB Privacy Task Force, Aug. 01, 1989.

[4] “The Directory - Authentication Framework”. CCIT Recommendation X.509 (1988).