Sieci Komputerowe

Grzegorz Gutowski

Uniwersytet Jagielloński

2023/24

Bezpieczeństwo

- Poufność
- Integralność
- Tożsamość
- ...

Bezpieczeństwo

- Poufność
- Integralność
- Tożsamość
- ... (możliwość zaprzeczenia)

Zagrożenia

- Podglądanie
- Modyfikacja, wstawianie, powtarzanie, usuwanie komunikatów
- Uniemożliwianie komunikacji

Bezpieczeństwo

- używać niebezpiecznych komunikacji w bezpieczny sposób
- używać bezpiecznych komunikacji
- używać bezpiecznych sieci

Narzędzia kryptograficzne

- hasze kryptograficzne
- szyfrowanie symetryczne
- szyfrowanie z kluczem publicznym

Hasze kryptograficzne

SHA

MD4,MD5,SHA1,SHA2,...

Network Working Group
Request for Comments: 1320
Obsoletes: RFC 1186

R. Rivest
MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

The MD4 Message-Digest Algorithm

Status of thie 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, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents

1. Executive Summary

This document describes the MD4 message-digest algorithm [1]. 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 MD4 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.

The MD4 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD4 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD4 algorithm is being placed in the public domain for review and possible adoption as a standard.

This document replaces the October 1990 RFC 1186 [2]. The main difference is that the reference implementation of MD4 in the appendix is more portable.

For OSI-based applications, MD4's object identifier is

md4 OBJECT IDENTIFIER ::=
  {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 4}

In the X.509 type AlgorithmIdentifier [3], the parameters for MD4 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.

3. MD4 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):

```
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 three 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) = XY v XZ v YZ
H(X,Y,Z) = X xor Y xor 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" bits in the same bit position.) In each bit position G acts as a majority function: if at least two of X, Y, Z are on, then G has a "1" bit in that bit position, else G has a "0" bit. 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, and similarly each bit of g(X,Y,Z) will be independent and unbiased. The function H is the bit-wise XOR or parity" function; it has properties similar to those of F and G.

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
     CC = C
     DD = D

     /* Round 1. */
     /* Let [abcd k s] denote the operation
          a = (a + F(b,c,d) + X[k]) <<< s. */
     /* Do the following 16 operations. */
     [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
     [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
     [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
     [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]

     /* Round 2. */
     /* Let [abcd k s] denote the operation
          a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
```

```
      /* Do the following 16 operations. */
      [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
      [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
      [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
      [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]

      /* Round 3. */
      /* Let [abcd k s] denote the operation
           a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
      /* Do the following 16 operations. */
      [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
      [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
      [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
      [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]

      /* 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 */
```

Note. The value 5A..99 is a hexadecimal 32-bit constant, written with the high-order digit first. This constant represents the square root of 2. The octal value of this constant is 013240474631.

The value 6E..A1 is a hexadecimal 32-bit constant, written with the high-order digit first. This constant represents the square root of 3. The octal value of this constant is 015666365641.

See Knuth, The Art of Programming, Volume 2 (Seminumerical Algorithms), Second Edition (1981), Addison-Wesley. Table 2, page 660.

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 MD4. A reference implementation in C is given in the appendix.

4. Summary

The MD4 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 MD4 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] 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.

[2] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1186, MIT, October 1990.

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

[4] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, MIT and RSA Data Security, Inc, April 1992.

APPENDIX A - Reference Implementation

This appendix contains the following files:

```
global.h -- global header file

md4.h -- header file for MD4

md4c.c -- source code for MD4

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 MD4Transform can be replaced with

Szyfrowanie symetryczne

XOR

$c = m \oplus K$

Szyfrowanie symetryczne

XOR

$c = m \oplus K$

AES

$c = \text{shuffle}_K^{14}(m)$

Szyfrowanie symetryczne

XOR

$c = m \oplus K$

AES

$c = \text{shuffle}_K^{14}(m)$

ECB

- $c_i \leftarrow \text{enc}(m_i)$

Szyfrowanie symetryczne

XOR

$c = m \oplus K$

AES

$c = \text{shuffle}_K^{14}(m)$

CBC

- v
- $c_1 \leftarrow \text{enc}(m_1 \oplus v)$
- $c_2 \leftarrow \text{enc}(m_2 \oplus c_1)$
- $\ldots$

CTR

- v
- $c_i \leftarrow m_i \oplus \text{enc}(v + i)$

THEORETICAL COMPUTER SCIENCE Jagiellonian University

Szyfrowanie symetryczne

Diffie-Hellman

- Ustalone g, p
- $A \mapsto B : g^a \pmod{p}$
- $B \mapsto A : g^b \pmod{p}$
- Wspólny sekret to g^{ab}

Szyfrowanie symetryczne

Needham-Schroeder

- A (B) dzieli K_{AS} (K_{BS}) z S
- $A \mapsto S : A, B, N_A$
- $S \mapsto A : \text{enc}_{K_{AS}}(N_A, K_{AB}, B, \text{enc}_{K_{BS}}(K_{AB}, A))$
- $A \mapsto B : \text{enc}_{K_{BS}}(K_{AB}, A)$
- $B \mapsto A : \text{enc}_{K_{AB}}(N_B)$
- $A \mapsto B : \text{enc}_{K_{AB}}(N_B - 1)$

Szyfrowanie symetryczne

Needham-Schroeder (Kerberos style)

- A (B) dzieli K_{AS} (K_{BS}) z S
- $A \mapsto S : A, B, N_A$
- $S \mapsto A : \mathrm{enc}_{K_{AS}}(N_A, K_{AB}, B, \mathrm{enc}_{K_{BS}}(K_{AB}, A, T))$
- $A \mapsto B : \mathrm{enc}_{K_{BS}}(K_{AB}, A, T)$
- $B \mapsto A : \mathrm{enc}_{K_{AB}}(N_B)$
- $A \mapsto B : \mathrm{enc}_{K_{AB}}(N_B - 1)$

Szyfrowanie symetryczne

Needham-Schroeder

- A (B) dzieli K_{AS} (K_{BS}) z S
- $A \mapsto B : A,$
- $B \mapsto A : \mathrm{enc}_{K_{BS}}(A, N_B)$
- $A \mapsto S : A, B, N_A, \mathrm{enc}_{K_{BS}}(A, N_B)$
- $S \mapsto A : \mathrm{enc}_{K_{AS}}(N_A, K_{AB}, B, \mathrm{enc}_{K_{BS}}(K_{AB}, A, N_B))$
- $A \mapsto B : \mathrm{enc}_{K_{BS}}(K_{AB}, A, N_B)$
- $B \mapsto A : \mathrm{enc}_{K_{AB}}(N_B)$
- $A \mapsto B : \mathrm{enc}_{K_{AB}}(N_B - 1)$

THEORETICAL COMPUTER SCIENCE Jagiellonian University

Szyfrowanie z kluczem publicznym

RSA

- $(m^e)^d \equiv m \pmod{n}$
- $n = p \cdot q$
- $\phi(n) = (p-1)(q-1)$
- $e = 65537$
- $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$
- Klucz publiczny: n, e
- Klucz prywatny: d

RSA+AES

Podpis cyfrowy

- $m' = m + \mathrm{dec}_A(m)$

Podpis cyfrowy

- $m' = m + \mathrm{dec}_A(m)$
- $m' = m + \mathrm{dec}_A(h(m))$

Autoryzacja

- hasłem

Autoryzacja

hasłem

kluczem

Autoryzacja

hasłem

kluczem

podpisanym kluczem

PGP

Wiadomość

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQIcBAABAgAGBQJaYHbdAAoJEAcDy+l6QCADLXAP/i/Shjbn5NAXI3b4NbKJUOaq
VuZxp0YAvHbg2FWNn3YXk/lnQ8Bktg4x7UYUObmy5sDIwYlqcKJD3YmsoEiScoZb
A7FDOQTm4bWayMw0c4cHKbKLMxMERF8bR9GhwrHa49gcT4MT+8ATtql12dmW614W
37wRWapt5E11BK3XdZo8m5H267mJPY4m9ixcvCJM9fkHIXEXNeUpbaruLxBEEPP1
5W25+10GuRuaf0LVloJL/aOwJtf2pZRsegKrR5eyPC6ymXBXiuG8yEeQn+lzE2N1
q+/WPliDnRAv21//CUIeeVZlciACl0LmnH14Prda7xhDBZeulTjE+XGAVtuqfcAS
r7cjlSGqtmRDIPV0V2auvbR/Y1TdycPKI2wAslEpztgJ/nMoo4I/wUbwlhV1P3aD
NHicvCy/57bN81x74e3xYXZMTtJsvvLFcovSm80i7pL3n1J2oGArqpMcE7pif2Ha
1L6KSewOCEqXLBsCIyofTBQ4kBqzcDDdL9K10xr0hxLKGCbPKSWPrd4Oxe61Kp89
nkiG4rAvDcDOBMDnhbdBanuK/Ucv1Y1CRyEqMeqnCaPa2OaTvGhnIrwMIoZIKDEK
FkD1TGo1wLSU3Sb0pO/xMJeHcmpVIBd1ACn7Yw+3+dLjt17zo9yDqOuHay4cvd3k
RqsL1j84o0TGWM7XAOAG
=/Fws
-----END PGP SIGNATURE-----
```

SSL

- negocjacja połączenia
- wybór algorytmów szyfrowania
- B przedstawia klucz publiczny i certyfikat
- generowanie sekretu M, $A \mapsto B : \mathrm{enc}_B(M)$
- klucze generowane na podstawie M : E_A, E_B, K_A, K_B
- fragmenty $A \mapsto B$ są szyfrowane symetrycznie E_B i podpisywane K_B.

Python

```python
#!/usr/bin/env python3
import socket
import ssl

hostname = 'satori.tcs.uj.edu.pl'
context = ssl.create_default_context()

with socket.create_connection((hostname, 443)) as sock:
    with context.wrap_socket(sock, server_hostname=hostname) as ssock:
        print(ssock.cipher())
        print(ssock.getpeercert())
        ssock.sendall(b'\r\n'.join([ bytes(line, 'utf-8') for line in [
            f'GET / HTTP/1.1',
            f'Host: {hostname}',
            f'Connection: close',
            f'',
            f'',
        ]]))
        response=b''
        while True:
            buf = ssock.recv()
            print(buf)
            response += buf
            if buf == b'':
                break
        print(response)
```

Python

```python
#!/usr/bin/env python3
import socket
import ssl
import traceback

context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('server.crt', 'server.key')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('127.0.0.1', 8443))
    sock.listen(5)
    with context.wrap_socket(sock, server_side=True) as ssock:
        while True:
            try:
                sconn, addr = ssock.accept()
                print(sconn.cipher())
            except ssl.SSLError:
                traceback.print_exc()
```

THEORETICAL COMPUTER SCIENCE
Jagiellonian University

Gdzie to wszystko jest zaimplementowane?