MD5

O algoritmo toma uma entrada de tamanho arbitrário e produz uma saída de 128 bits representando seu hash. É conjecturado que computacionalmente inviável gerar o mesmo hash para entradas diferentes. Destina-se a aplicações de assinatura digital, quando um grande arquivo deve ser “comprimido” de forma segura antes de ser criptografado com uma chave privada (secreta) sob um sistema de criptografia de chave pública tal como o RSA.

**Funcionamento**

1. Tem-se uma mensagem de entrada composta de b bits em que queremos calcular seu hash (b>=0). Sempre representado em little endian.

2. Um padding é realizado, primeiro bit 1 e o resto em zero, de forma que o tamanho em bits da mensagem inicial seja congruente a 448 módulo 512, 64 bits afastado de ser múltiplo de 512. O padding é sempre realizado, mesmo que a mensagem já seja congruente a 448 módulo 512.

3. O tamanho b bits é representado como 64 bits e concatenado à mensagem do passo anterior. Aqui o tamanho da mensagem é exatamente múltipla de 512 bits, sendo também múltipla de 16 palavras de 32 bits.

4. Um buffer de quatro palavras é usado para auxiliar no cálculo, devendo serem iniciados com os valores A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476.

5. Para cada bloco de 16 palavras, sendo X o bloco atual. São realizadas 64 operações de deslocamento circular a esquerda divididas em 4 passos.

#variáveis:

- sendo A,B,C e D cópias dos buffers

- k índice do bloco de 16 palavras

- s quantidade a ser deslocada circularmente à esquerda

- i índice da tabela da parte inteira de 4294967296 \* abs(sin(i+1))

- F = ((X & Y) | (~X & Z))

- G = ((X & Z) | (Y & ~Z))

- H = X ^ Y ^ Z

- I = Y ^ (X | ~Z)

/\* passo 1 \*/

Se [abcd k s i] é a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)

Faça as 16 operações

/\* [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] \*/

/\* passo 2 \*/

Se [abcd k s i] é a = b + ((a + F(b,c,d) + G[k] + T[i]) <<< s)

Faça as 16 operações

/\* [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] \*/

/\* passo 3 \*/

Se [abcd k s i] é a = b + ((a + F(b,c,d) + H[k] + T[i]) <<< s)

Faça as 16 operações

/\* [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] \*/

/\* passo 4 \*/

Se [abcd k s i] é a = b + ((a + F(b,c,d) + I[k] + T[i]) <<< s)

Faça as 16 operações

/\* [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] \*/

6. Ao fim de cada cópia, incrementar o valor original dos buffers com as cópias obtidas.

7. O valor de hash é ABCD em little endian.

**Exemplo**

Mensagem: Grupo 3, sem \0

Tamanho do arquivo: 7 bytes

Hexadecimal associado: 477275706f2033

Tamanho do arquivo com padding de bits:56 bytes

Hexadecimal associado: 477275706f203380000000000000000000000000000000000000000000000000

Tamanho com padding do tamanho original:64 bytes

477275706f203380000000000000000000000000000000000000000000000000380000000

A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476

AA=A, BB=B, CC=C, DD=D;

Como o número de iterações se dá em blocos de 16 palavras = 16\*32 = 512 bits e o tamanho da mensagem final é de 64 bytes = 64\*8 = 512 bits, é feita apenas uma iteração com 64 operações.

Na primeira operação, os valores das cópias são

AA=10325476

BB=dfd90aac

CC=efcdab89

DD=98badcfe

Na segunda..

AA=98badcfe

BB=708a5f3d

CC=dfd90aac

DD=efcdab89

Após as 64 iterações

AA=20d207f3

BB=64e1d30c

CC=c12e2c1e

DD=b3560525

A = A + AA = 0x67452301 + 0x20d207f3 = 0x88172af4

B = B + BB = 0xefcdab89 + 0x64e1d30c = 0x54af7e95

C = C + CC = 0x98badcfe + 0xc12e2c1e = 0x59e9091c

D = D + DD = 0x10325476 + 0xb3560525 = 0xc388599b

O resultado é mostrado então em little endian como ABCD, totalizando 128 bits

f42a1788957eaf541c09e9599b5988c3