MD5摘要算法

> MD5不能称作加密算法，因为不能还原出原本的密码。

MD5算法在逆向中经常遇到，所以了解其背后的原理和加密特征是有必要的，因此专门整理一下。

以下以c语言的其中一种实现为例。

原理

MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。

在MD5算法中:

- 第一步：增加填充。首先需要对信息进行填充，使其位长对512求`mod`的结果等于448。因此，信息的位长（Bits Length）将被扩展至`N*512+448`，N为一个非负整数，N可以是零。填充的方法如下，在信息的后面填充一个1和无数个0，直到满足上面的条件时才停止用0对信息的填充。
- 第二步：补足长度。然后，将数据长度转换为`64bit`的数值，如果长度超过`64bit`所能表示的数据长度的范围，值保留最后`64bit`，增加到前面填充的数据后面，使得最后的数据为`512bit`的整数倍。也就是`32bit`的16倍的整数倍。在`RFC1321`中，`32bit`称为一个`word`。

经过这两步的处理，现在的信息的位长=`N*512+448+64=(N+1)*512`，即长度恰好是512的整数倍。这样做的原因是为满足后面处理中对信息长度的要求。

- 第三步：初始化变量。用到4个变量，分别为`A、B、C、D`，均为`32bit`长。初始化为:

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

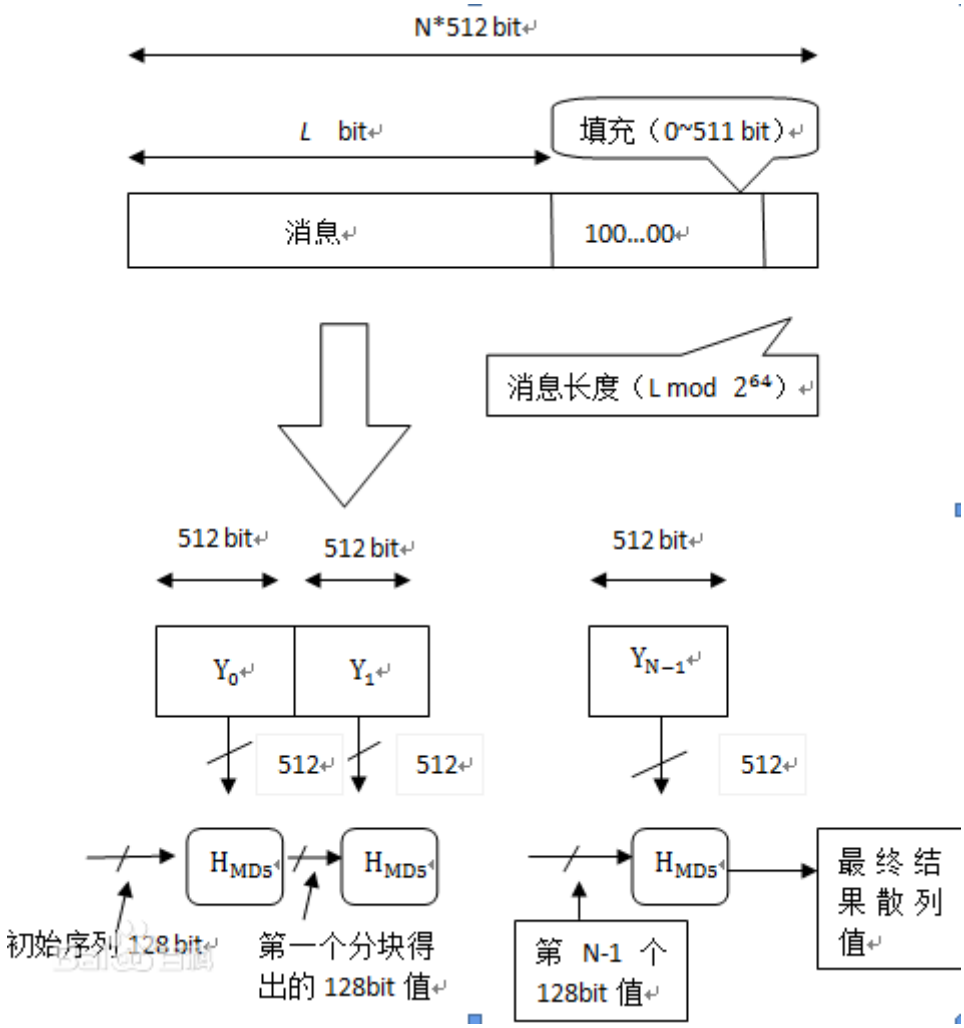
- 第四步：数据处理。首先定义4个辅助函数:

```
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))
```

其中：`XY`表示按位与，`X v Y`表示按位或，`not(X)`表示按位取反。`xor`表示按位异或。函数中的`X、Y、Z`均为`32bit`。

定义一个需要用到的数组:

> `T(i)`，`i` 取值 `1-64`，`T(i)` 等于 `abs(sin(i))` 的 `4294967296` 倍的整数部分，`i` 为弧度。 假设前三步处理后的数据长度为 `32*16*Nbit`
>
> - 第五步：输出。最后得到的ABCD为输出结果，共128bit。A为低位，D为高位。
>
> 总体流程如下图所示，表示第i个分组，每次的运算都由前一轮的128位结果值和第i块512bit值进行运算。初始的128位值为初始链接变量，这些参数用于第一轮的运算，以大端字节序来表示，他们分别为：`A=0x01234567`，`B=0x89ABCDEF`，`C=0xFEDCBA98`，`D=0x76543210`。



code

md5.h

```
#ifndef MD5_H
#define MD5_H

typedef struct
{
    unsigned int count[2];
    unsigned int state[4];
    unsigned char buffer[64];
}MD5_CTX;


#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))
#define ROTATE_LEFT(x,n) ((x << n) | (x >> (32-n)))
#define FF(a,b,c,d,x,s,ac) \
          { \
          a += F(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define GG(a,b,c,d,x,s,ac) \
          { \
          a += G(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define HH(a,b,c,d,x,s,ac) \
          { \
          a += H(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define II(a,b,c,d,x,s,ac) \
          { \
          a += I(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
void MD5Init(MD5_CTX *context);
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int
inputlen);
void MD5Final(MD5_CTX *context, unsigned char digest[16]);
void MD5Transform(unsigned int state[4], unsigned char block[64]);
void MD5Encode(unsigned char *output, unsigned int *input, unsigned int
len);
void MD5Decode(unsigned int *output, unsigned char *input, unsigned int
len);

#endif
```

md5.c

```
#include <memory.h>
#include "md5.h"

unsigned char PADDING[] = { 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 };

void MD5Init(MD5_CTX *context)
{
    context->count[0] = 0;
    context->count[1] = 0;
    context->state[0] = 0x67452301;
    context->state[1] = 0xEFCDAB89;
    context->state[2] = 0x98BADCFE;
```

```
    context->state[3] = 0x10325476;
}
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int
inputlen)
{
    unsigned int i = 0, index = 0, partlen = 0;
    index = (context->count[0] >> 3) & 0x3F;
    partlen = 64 - index;
    context->count[0] += inputlen << 3;
    if (context->count[0] < (inputlen << 3))
        context->count[1]++;
    context->count[1] += inputlen >> 29;

    if (inputlen >= partlen)
    {
        memcpy(&context->buffer[index], input, partlen);
        MD5Transform(context->state, context->buffer);
        for (i = partlen; i + 64 <= inputlen; i += 64)
            MD5Transform(context->state, &input[i]);
        index = 0;
    }
    else
    {
        i = 0;
    }
    memcpy(&context->buffer[index], &input[i], inputlen - i);
}
void MD5Final(MD5_CTX *context, unsigned char digest[16])
{
    unsigned int index = 0, padlen = 0;
    unsigned char bits[8];
    index = (context->count[0] >> 3) & 0x3F;
    padlen = (index < 56) ? (56 - index) : (120 - index);
    MD5Encode(bits, context->count, 8);
    MD5Update(context, PADDING, padlen);
    MD5Update(context, bits, 8);
    MD5Encode(digest, context->state, 16);
}
void MD5Encode(unsigned char *output, unsigned int *input, unsigned int
len)
{
    unsigned int i = 0, j = 0;
    while (j < len)
    {
        output[j] = input[i] & 0xFF;
        output[j + 1] = (input[i] >> 8) & 0xFF;
        output[j + 2] = (input[i] >> 16) & 0xFF;
        output[j + 3] = (input[i] >> 24) & 0xFF;
        i++;
        j += 4;
    }
}
void MD5Decode(unsigned int *output, unsigned char *input, unsigned int
len)
{
    unsigned int i = 0, j = 0;
    while (j < len)
    {
```

```
        output[i] = (input[j]) |
            (input[j + 1] << 8) |
            (input[j + 2] << 16) |
            (input[j + 3] << 24);
        i++;
        j += 4;
    }
}
void MD5Transform(unsigned int state[4], unsigned char block[64])
{
    unsigned int a = state[0];
    unsigned int b = state[1];
    unsigned int c = state[2];
    unsigned int d = state[3];
    unsigned int x[64];
    MD5Decode(x, block, 64);
    FF(a, b, c, d, x[0], 7, 0xd76aa478); /* 1 */
    FF(d, a, b, c, x[1], 12, 0xe8c7b756); /* 2 */
    FF(c, d, a, b, x[2], 17, 0x242070db); /* 3 */
    FF(b, c, d, a, x[3], 22, 0xc1bdceee); /* 4 */
    FF(a, b, c, d, x[4], 7, 0xf57c0faf); /* 5 */
    FF(d, a, b, c, x[5], 12, 0x4787c62a); /* 6 */
    FF(c, d, a, b, x[6], 17, 0xa8304613); /* 7 */
    FF(b, c, d, a, x[7], 22, 0xfd469501); /* 8 */
    FF(a, b, c, d, x[8], 7, 0x698098d8); /* 9 */
    FF(d, a, b, c, x[9], 12, 0x8b44f7af); /* 10 */
    FF(c, d, a, b, x[10], 17, 0xffff5bb1); /* 11 */
    FF(b, c, d, a, x[11], 22, 0x895cd7be); /* 12 */
    FF(a, b, c, d, x[12], 7, 0x6b901122); /* 13 */
    FF(d, a, b, c, x[13], 12, 0xfd987193); /* 14 */
    FF(c, d, a, b, x[14], 17, 0xa679438e); /* 15 */
    FF(b, c, d, a, x[15], 22, 0x49b40821); /* 16 */

                                           /* Round 2 */
    GG(a, b, c, d, x[1], 5, 0xf61e2562); /* 17 */
    GG(d, a, b, c, x[6], 9, 0xc040b340); /* 18 */
    GG(c, d, a, b, x[11], 14, 0x265e5a51); /* 19 */
    GG(b, c, d, a, x[0], 20, 0xe9b6c7aa); /* 20 */
    GG(a, b, c, d, x[5], 5, 0xd62f105d); /* 21 */
    GG(d, a, b, c, x[10], 9, 0x2441453); /* 22 */
    GG(c, d, a, b, x[15], 14, 0xd8a1e681); /* 23 */
    GG(b, c, d, a, x[4], 20, 0xe7d3fbc8); /* 24 */
    GG(a, b, c, d, x[9], 5, 0x21e1cde6); /* 25 */
    GG(d, a, b, c, x[14], 9, 0xc33707d6); /* 26 */
    GG(c, d, a, b, x[3], 14, 0xf4d50d87); /* 27 */
    GG(b, c, d, a, x[8], 20, 0x455a14ed); /* 28 */
    GG(a, b, c, d, x[13], 5, 0xa9e3e905); /* 29 */
    GG(d, a, b, c, x[2], 9, 0xfcefa3f8); /* 30 */
    GG(c, d, a, b, x[7], 14, 0x676f02d9); /* 31 */
    GG(b, c, d, a, x[12], 20, 0x8d2a4c8a); /* 32 */

                                           /* Round 3 */
    HH(a, b, c, d, x[5], 4, 0xfffa3942); /* 33 */
    HH(d, a, b, c, x[8], 11, 0x8771f681); /* 34 */
    HH(c, d, a, b, x[11], 16, 0x6d9d6122); /* 35 */
    HH(b, c, d, a, x[14], 23, 0xfde5380c); /* 36 */
    HH(a, b, c, d, x[1], 4, 0xa4beea44); /* 37 */
    HH(d, a, b, c, x[4], 11, 0x4bdecfa9); /* 38 */
```

```
    HH(c, d, a, b, x[7], 16, 0xf6bb4b60); /* 39 */
    HH(b, c, d, a, x[10], 23, 0xbebfbc70); /* 40 */
    HH(a, b, c, d, x[13], 4, 0x289b7ec6); /* 41 */
    HH(d, a, b, c, x[0], 11, 0xeaa127fa); /* 42 */
    HH(c, d, a, b, x[3], 16, 0xd4ef3085); /* 43 */
    HH(b, c, d, a, x[6], 23, 0x4881d05); /* 44 */
    HH(a, b, c, d, x[9], 4, 0xd9d4d039); /* 45 */
    HH(d, a, b, c, x[12], 11, 0xe6db99e5); /* 46 */
    HH(c, d, a, b, x[15], 16, 0x1fa27cf8); /* 47 */
    HH(b, c, d, a, x[2], 23, 0xc4ac5665); /* 48 */

                                          /* Round 4 */
    II(a, b, c, d, x[0], 6, 0xf4292244); /* 49 */
    II(d, a, b, c, x[7], 10, 0x432aff97); /* 50 */
    II(c, d, a, b, x[14], 15, 0xab9423a7); /* 51 */
    II(b, c, d, a, x[5], 21, 0xfc93a039); /* 52 */
    II(a, b, c, d, x[12], 6, 0x655b59c3); /* 53 */
    II(d, a, b, c, x[3], 10, 0x8f0ccc92); /* 54 */
    II(c, d, a, b, x[10], 15, 0xffeff47d); /* 55 */
    II(b, c, d, a, x[1], 21, 0x85845dd1); /* 56 */
    II(a, b, c, d, x[8], 6, 0x6fa87e4f); /* 57 */
    II(d, a, b, c, x[15], 10, 0xfe2ce6e0); /* 58 */
    II(c, d, a, b, x[6], 15, 0xa3014314); /* 59 */
    II(b, c, d, a, x[13], 21, 0x4e0811a1); /* 60 */
    II(a, b, c, d, x[4], 6, 0xf7537e82); /* 61 */
    II(d, a, b, c, x[11], 10, 0xbd3af235); /* 62 */
    II(c, d, a, b, x[2], 15, 0x2ad7d2bb); /* 63 */
    II(b, c, d, a, x[9], 21, 0xeb86d391); /* 64 */
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
}
```

分析

MD5加密分三步：

`MD5Init(MD5_CTX *context)`

`MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inputlen)`

`MD5Final(MD5_CTX *context, unsigned char digest[16])`

以下分别对三个函数进行分析：

首先MD5的数据存储结构是一个`MD5_CTX`类型的结构体：

```
typedef struct
{
    // 存储原始信息的bits数长度,不包括填充的bits
    // 最长为 2^64 bits，因为2^64是一个64位数的最大值
    unsigned int count[2];
    unsigned int state[4];      // 存放每步摘要的变量
    unsigned char buffer[64];   // 存放输入的信息的缓冲区，512bits
}MD5_CTX;
```

```
// 设置存放结果的变量并初始化
void MD5Init(MD5_CTX *context)
{
    context->count[0] = 0;
    context->count[1] = 0;
    context->state[0] = 0x67452301;     // 这里是计算摘要的四个初始化变量
    context->state[1] = 0xEFCDAB89;     // 这四个数可以作为md5加密的标志
    context->state[2] = 0x98BADCFE;
    context->state[3] = 0x10325476;
}
```

以上四个数在内存中的形式为：

`01 23 45 67 89 AB CD EF` `FE DC BA 98 76 54 32 10`

共16个字节。

```
/*将与加密的信息传递给md5结构，可以多次调用
context：初始化过了的md5结构
input：欲加密的信息，可以任意长
inputLen：指定input的长度
*/
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int
inputlen)
{
    unsigned int i = 0, index = 0, partlen = 0;

    // 计算已有信息bits mod 64, 64bytes=512bits。
    index = (context->count[0] >> 3) & 0x3F;

    partlen = 64 - index;
    context->count[0] += inputlen << 3;

    // Update number of bits
    if (context->count[0] < (inputlen << 3))
        context->count[1]++;
    context->count[1] += inputlen >> 29;

    if (inputlen >= partlen)
    {
        // 不够长的填满512bits，放入context->buffer缓冲区
        memcpy(&context->buffer[index], input, partlen);
```

```
        MD5Transform(context->state, context->buffer);
        // 如果还有剩余字节，对剩余字节进行计算
        for (i = partlen; i + 64 <= inputlen; i += 64)
            MD5Transform(context->state, &input[i]);
        index = 0;
    }
    else
    {
        i = 0;
    }
    // 不够长的填满512bits，放入context->buffer缓冲区
    memcpy(&context->buffer[index], &input[i], inputlen - i);
}
```

```
/*获取加密 的最终结果
digest：保存最终的加密串
context：前面初始化并填入了信息的md5结构体
*/
void MD5Final(MD5_CTX *context, unsigned char digest[16])
{
    unsigned int index = 0, padlen = 0;
    unsigned char bits[8];
    // 计算所有的bits长度的字节数 mod 64
    index = (context->count[0] >> 3) & 0x3F;
    // 计算需要填充的字节数，padlen的取值范围在1-64之间
    padlen = (index < 56) ? (56 - index) : (120 - index);
    // 将要被转换的信息(所有的)的bits长度拷贝到bits中
    MD5Encode(bits, context->count, 8);
    MD5Update(context, PADDING, padlen);
    /* Append length (before padding) */
    MD5Update(context, bits, 8);
    // 将最终的结果保存到digest中
    MD5Encode(digest, context->state, 16);
}
```

逆向分析

下面来看看编译过的md5程序的反汇编特征：

未完待续.....