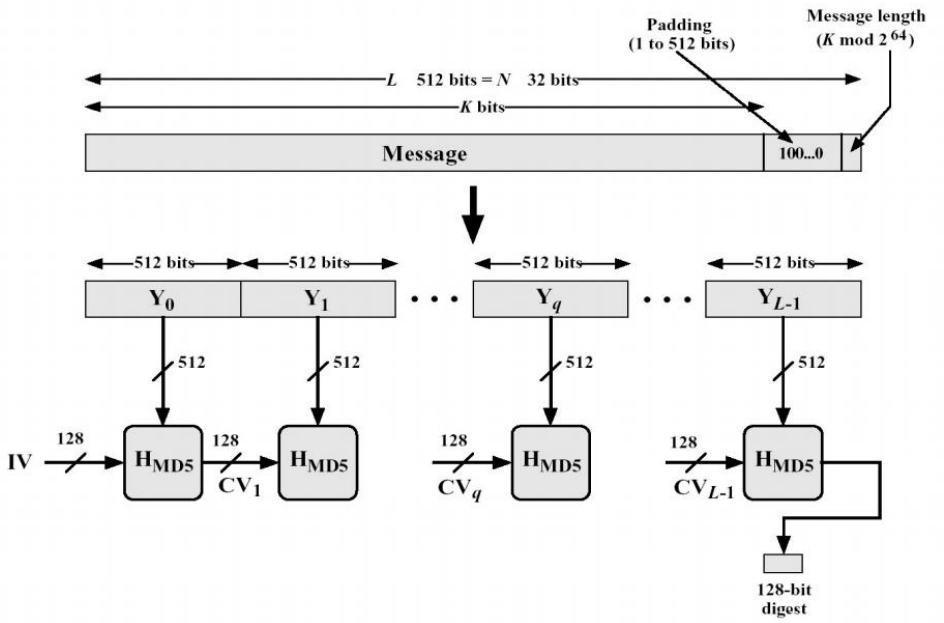
HMAC-MD5 算法 C 语言实现

一、 MD5 算法原理概述

MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5)，由 Ron Rivest 发明，是广泛使用的 Hash 算法，用于确保信息传输的完整性和一致性。MD5 使用 little-endian (小端模式)，输入任意不定长度信息，以 512-bit 进行分组，生成四个 32-bit 数据，最后联合输出固定 128-bit 的信息摘要。MD5 算法的基本过程为：填充、分块、缓冲区初始化、循环压缩、得出结果。

基本流程图如下：



二、 HMAC 算法原理概述

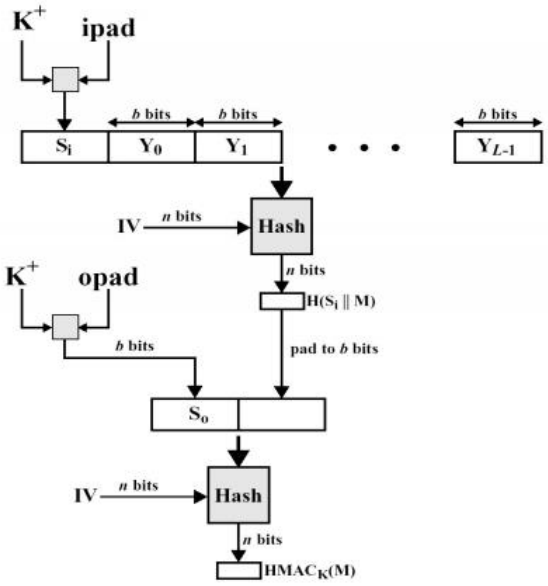
HMAC 算法是一种基于密钥的报文完整性的验证方法，其安全性是建立在 Hash 加密算法基础上的。它要求通信双方共享密钥、约定算法、对报文进行 Hash 运算，形成固定长度的认证码。通信双方通过认证码的校验来确定报文的合法性。HMAC 算法可以用来作加密、数字签名、报文验证等。

一句话总结：HMAC 是密钥相关的哈希运算消息认证码（Hash-based Message Authentication Code），HMAC 运算利用哈希算法，以一个密钥和一个消息为输入，生成一个消息摘要作为输出。

基本步骤如下：

- 对共享密钥 k 左边补 0，生成一个 b 位的数据块 K+；
- K^+ 与 ipad 作 XOR，生成 b 位的 S_i；
- 对 $(S_i||M)$ 进行 hash 压缩（例如 MD5），得到 $H(S_i||M)$；
- K^+ 与 opad 作 XOR，生成 b 位的 S_o；
- 对 $S_o||H(S_i||M)$进行 hash 压缩（例如 MD5），得到 $HMAC_k = H(S_o||H(S_i||M))$.

基本流程图如下：



三、 MD5 总体结构设计与模块分解

1. 数据结构设计

- 输入明文为 char 字符串，输出为 32 个 16 进制数符号（128 bits）；
- 32 位寄存器使用 unsigned int 类型的变量来实现
- 64 位原始明文的位数 count 采用 unsigned long long 来存储。

2. 填充（padding）

以 512 位数据为一个块，不足 512 位时需要填充二进制序列 1000…0，留下末尾 64 位来填充原始消息的位数 count。count 是一个 64 位无符号整形变量，填充时将其看成 2 个 32 位的字，按小端模式进行填充。

```
// 填充函数
void padding(char *message, unsigned long long messageLen)
{
    blockLen = (messageLen / 64) + ((messageLen * 8) % 512 >= 448 ? 2 : 1);
    unsigned char temp[blockLen * 64];
    for (int i = 0; i < messageLen; ++i)
        temp[i] = message[i];
    temp[messageLen] = 0x80; // 0x80 -> 1000 0000
    for (int i = messageLen + 1; i < blockLen * 64; ++i)
        temp[i] = 0x00;

    // 动态分配二维数组的内存空间
```

```
    paddedMessage = (unsigned int **)malloc(blockLen * sizeof(unsigned int *));
    for (int i = 0; i < blockLen; ++i)
        paddedMessage[i] = (unsigned int *)malloc(16 * sizeof(unsigned int));

    Byte2int(temp);
    unsigned int left = ((messageLen * 8) >> 32) & 0x00000000ffffffff;
    unsigned int right = (messageLen * 8) & 0x00000000ffffffff;
    paddedMessage[blockLen - 1][15] = left;
    paddedMessage[blockLen - 1][14] = right;
}
```

3. 初始化

每个带 4 个字节 (32-bit) 的 4 个寄存器构成向量 (A, B, C, D)，也称 MD 缓冲区。以下 16 进制初值作为 MD 缓冲区的初始向量 IV，并采用小端存储 (little-endian) 的存储结构：

- *A* = 0x67452301
- *B* = 0xEFCDAB89
- *C* = 0x98BADCFE
- *D* = 0x10325476

Word *A*	01	23	45	67
Word *B*	89	AB	CD	EF
Word *C*	FE	DC	BA	98
Word *D*	76	54	32	10

→ 行编址方向 →

将其存储为全局变量：

```
// MD 缓冲区的初始向量 IV(A, B, C, D)
const unsigned int IV[4] = {
    0x67452301,
    0xEFCDAB89,
    0x98BADCFE,
    0x10325476};
```

4. 轮函数

压缩函数中需要使用的 4 个不同的轮函数，在压缩函数中同一轮循环的所有迭代使用相同的轮函数，而各轮循环对应的轮函数具有不同的定义：

轮次	Function g	$g(b, c, d)$
1	$F(b, c, d)$	$(b \wedge c) \vee (\neg b \wedge d)$
2	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \neg d)$
3	$H(b, c, d)$	$b \oplus c \oplus d$
4	$I(b, c, d)$	$c \oplus (b \vee \neg d)$

将其定义为宏：

```
// 四个轮函数
#define F(b, c, d) ((b & c) | (~b & d))
#define G(b, c, d) ((b & d) | (c & ~d))
#define H(b, c, d) (b ^ c ^ d)
#define I(b, c, d) (c ^ (b | ~d))
```

5. X 表

压缩函数中各轮循环中第 i 次迭代 (i = 1 … 16) 使用的 X[k] 的确定，设 j = i -1:
第 1 轮迭代：k= j
第 2 轮迭代：k = (1 + 5j) mod 16
第 3 轮迭代：k = (5 + 3j) mod 16
第 4 轮迭代：k = 7j mod 16
将其定义为宏：

```
// 各轮循环中第 i 次迭代 (i = 1 .. 16) 使用的 X[k]
const int X_index[4][16] = {
    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15},
    {1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12},
    {5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2},
    {0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9}};
```

6. T 表

T 表的生成函数：$int(2^{32} \times |\sin(i)|)$
Int 为取整函数，sin 为正弦函数，以 i 作为弧度输入。
将各轮各次迭代运算采用的 T 值的计算结果定义为宏：

```
// 各轮各次迭代运算采用的 T 值
const unsigned int T[4][16] = {
    {0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf,
0x4787c62a, 0xa8304613, 0xfd469501,
     0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be, 0x6b901122,
0xfd987193, 0xa679438e, 0x49b40821},
```

```
5.
6.     {0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa, 0xd62f105d,
   0x02441453, 0xd8a1e681, 0xe7d3fbc8,
7.      0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed, 0xa9e3e905,
   0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a},
8.
9.     {0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c, 0xa4beea44,
   0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
10.     0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05, 0xd9d4d039,
   0xe6db99e5, 0x1fa27cf8, 0xc4ac5665},
11.
12.    {0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039, 0x655b59c3,
   0x8f0ccc92, 0xffeff47d, 0x85845dd1,
13.     0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1, 0xf7537e82,
   0xbd3af235, 0x2ad7d2bb, 0xeb86d391}};
```

7. S 表

各轮各次迭代运算 (1 .. 64) 采用的左循环移位的位数 s 值:

```
1. // 各轮各次迭代运算 (1 .. 64) 采用的左循环移位的位数 s 值
2. const int S[4][16] = {
3.     {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22},
4.     {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20},
5.     {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23},
6.     {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}}
   ;
```

8. 循环置换

将 4 个寄存器向量循环右移，ABCD -> DABC

```
1. // 循环置换：ABCD -> DABC
2. void buffer_iteration(unsigned int MD[4])
3. {
4.     unsigned int temp = MD[0];
5.     MD[0] = MD[3];
6.     MD[3] = MD[2];
7.     MD[2] = MD[1];
8.     MD[1] = temp;
9. }
```

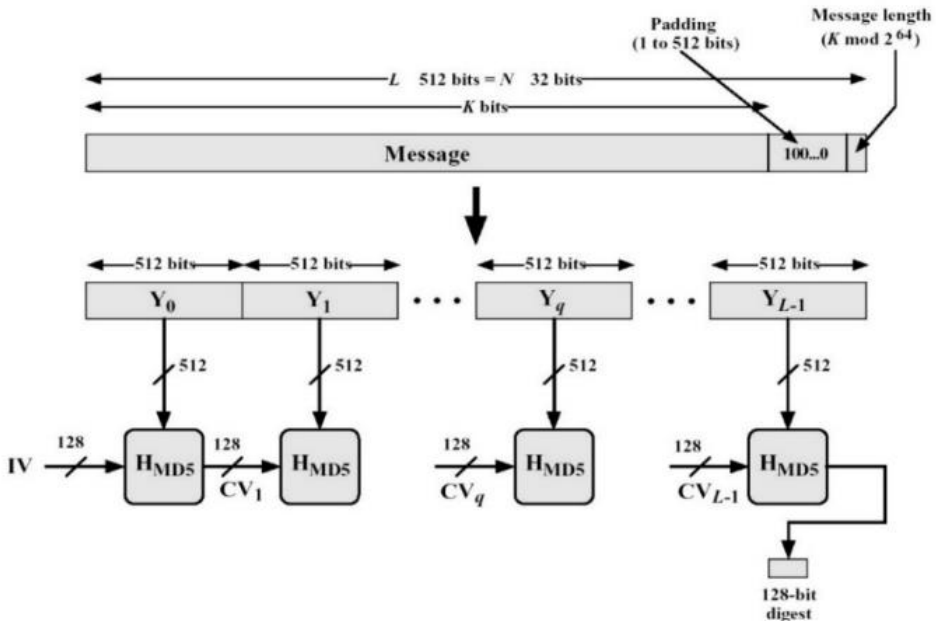
9. 总控流程

以 512-bit 消息分组为单位，每一分组$y_q(q = 0,1,\dots,L-1)$经过 4 个循环的压缩算法，表示为：

$$CV_0 = IV$$

$$CV_i = H_{MD5}(CV_{i-1}, Y_{i-1}), i = 1,\dots,L$$

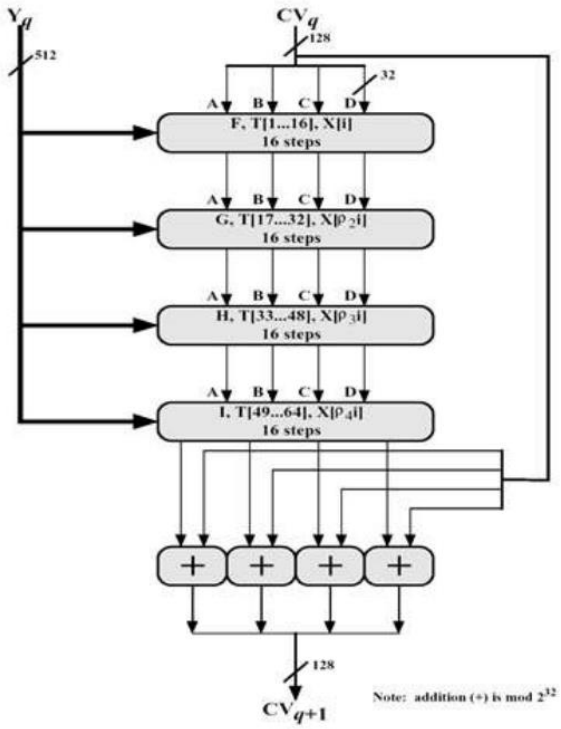
输出结果哈希值：$MD = CV_L$



10. MD5 压缩函数H_{MD5}

HMD5 从 CV 输入 128 位，从消息分组输入 512 位，完成 4 轮循环后，输出 128 位，作为用于下一轮输入的 CV 值。每轮循环分别固定不同的轮函数 F, G, H, I，结合指定的 T 表元素和消息分组的不同部分 X 做 16 次迭代运算，生成下一轮循环的输入，4 轮循环共有 64 次迭代运算。

流程图如下：



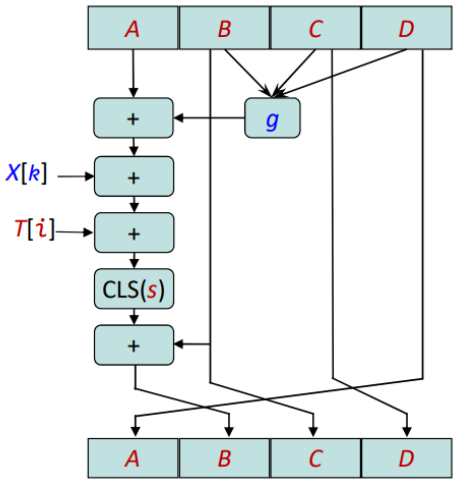
每轮循环中的一次迭代运算逻辑：

1) 对 A 迭代：a ← b + ((a + g(b, c, d) + X[k] + T[i]) <<< s)
2) 对缓冲区(A, B, C, D)作循环置换：(B, C, D, A) ← (A, B, C, D)

说明：

- a, b, c, d : MD 缓冲区 (A, B, C, D) 的各个寄存器的当前值；
- g：轮函数 F, G, H, I 中的一个；
- <<< s：将 32 位输入循环左移 (CLS) s 位；查询 S 表可知；
- X[k] :当前处理消息分组 q 的第 k 个 32 位字，查询 X 表可知；
- T[i] : T 表的第 i 个元素，查询 T 表可知；
- +：模 2^{32}加法。

示意图如下：



```
// MD5 压缩函数 a = b + ((a + g(b, c, d) + X[k] + T[i]) <<< s)
void HMD5(char *message, char * output, unsigned long long messageLen)
{
    padding(message, messageLen);
    unsigned int CV[4];
    //初始化 CV
    for (int i = 0; i < 4; ++i)
        CV[i] = IV[i];

    for (int i = 0; i < blockLen; ++i)
    {
        // 首先记录下 64 轮迭代前的 CV 值
        unsigned int temp[4];
        for(int j = 0; j < 4; ++j)
            temp[j] = CV[j];

        // 第一轮循环 F
        for (int j = 0; j < 16; ++j)
        {
            CV[0] = CV[1] + CLS((CV[0] + F(CV[1], CV[2], CV[3]) + paddedMessage[i][X_index[0][j]] + T[0][j]), S[0][j]);
            buffer_iteration(CV);
        }
        // 第二轮循环 G
        for (int j = 0; j < 16; ++j)
        {
            CV[0] = CV[1] + CLS((CV[0] + G(CV[1], CV[2], CV[3]) + paddedMessage[i][X_index[1][j]] + T[1][j]), S[1][j]);
            buffer_iteration(CV);
        }
        // 第三轮循环 H
        for (int j = 0; j < 16; ++j)
        {
            CV[0] = CV[1] + CLS((CV[0] + H(CV[1], CV[2], CV[3]) + paddedMessage[i][X_index[2][j]] + T[2][j]), S[2][j]);
            buffer_iteration(CV);
        }
        // 第四轮循环 I
        for (int j = 0; j < 16; ++j)
        {
            CV[0] = CV[1] + CLS((CV[0] + I(CV[1], CV[2], CV[3]) + paddedMessage[i][X_index[3][j]] + T[3][j]), S[3][j]);
            buffer_iteration(CV);
```

```
        }

        for(int j = 0; j < 4; ++j)
            CV[j] += temp[j];
    }

    int2char(CV, output, 4);
}
```

四、 HMAC-MD5 总体结构设计与模块分解

1. 数据结构设计

- 块大小为 512 位，64 个字节，即大小为 64 的字符数组
- 输入为 char 类型的字符串（明文和密钥），输出为 32 个 16 进制数符号（128 bits）
- 定义 ipad 和 opad:

```
#define ipad 0x36     // 00110110
#define opad 0x5c     // 01011100
```

2. 填充（padding）

对共享密钥 k 右边补 0，生成一个大小为 64 的字符数组 K^+

```
void getKplus()
{
    Kplus = (char *)malloc(sizeof(char) * BLOCKSIZE);
    for(int i = 0; i < strlen(key); ++i)
        Kplus[i] = key[i];
    for(int i = strlen(key); i < BLOCKSIZE; ++i)
        Kplus[i] = 0;
}
```

3. 生成S_i和S_o

K^+与 ipad 作 XOR，生成 b 位的 S_i
K^+与 opad 作 XOR，生成 b 位的 S_o

```
void getSi_So()
{
    Si = (char *)malloc(sizeof(char) * BLOCKSIZE);
```

```
    for(int i = 0; i < BLOCKSIZE; ++i)
        Si[i] = Kplus[i] ^ ipad;
    So = (char *)malloc(sizeof(char) * BLOCKSIZE);
    for(int i = 0; i < BLOCKSIZE; ++i)
        So[i] = Kplus[i] ^ opad;
}
```

4. Hash 压缩

首先调用 add_str 函数将S_i和 M 拼接起来得到$S_i||M$，add_str 函数实现如下：

```
// 字符串拼接函数
void add_str(char * str1, int len1, char * str2, int len2, char *
 res)
{
    for(int i = 0; i < len1; ++i)
        res[i] = str1[i];
    for(int i = 0; i < len2; ++i)
        res[len1 + i] = str2[i];
}
```

然后调用前文实现的 MD5 算法中的 HMD5 函数，得到 $H(S_i||M)$。再调用 add_str 函数将S_o和$H(S_i||M)$拼接得到$S_o||H(S_i||M)$，调用 HMD5 函数得到$HMAC_k = H(S_o||H(S_i||M))$。

```
void hmac(unsigned char * M, unsigned long long Mlen, unsigned ch
ar * key, unsigned char * HMAC)
{
    getKplus();
    getSi_So();
    char SM[BLOCKSIZE + Mlen];
    add_str(Si, BLOCKSIZE, M, Mlen, SM);
    unsigned char hash1[16];
    HMD5(SM, hash1, BLOCKSIZE + Mlen);
    char SH[BLOCKSIZE + 16];
    add_str(So, BLOCKSIZE, hash1, 16, SH);
    HMD5(SH, HMAC, BLOCKSIZE + 16);
}
```

五、 编译运行结果

运行环境：Windows10（64 位）

1. MD5 算法单独运行

   注：将 MD5.c 中注释掉的 main 函数恢复

   输入："I love you, you are my sunshine!"

   输出结果如下：

```
PS C:\Users\18052\Desktop\HMAC-MD5> gcc MD5.c
PS C:\Users\18052\Desktop\HMAC-MD5> ./a
f468bee4bc7c607a576be72ad1fa9d72
```

   代码使用说明：如果要换别的测试输入，需修改 main 函数中 s 变量的值以及传入 HMD5 中的明文长度!

2. Hmac 算法运行

   注：将 MD5.c 中的 main 函数再次注释

   输入明文："I love you, you are my sunshine!"

   输入密钥："abcde"

   输出结果如下：

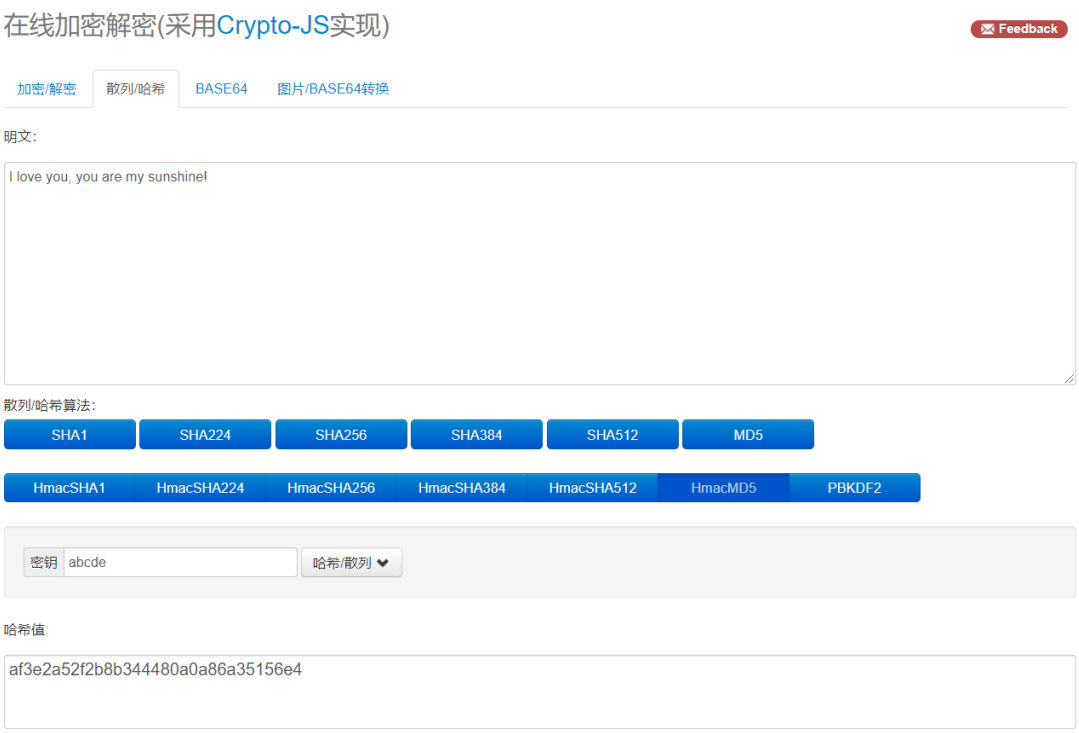
```
PS C:\Users\18052\Desktop\HMAC-MD5> gcc hmac.c
PS C:\Users\18052\Desktop\HMAC-MD5> ./a
af3e2a52f2b8b344480a0a86a35156e4
```

六、 验证用例

利用在线加密解密网站：https://tool.oschina.net/encrypt?type=2 进行结果的验证

MD5 算法：

Hmac 算法：



可以看到与我自己实现的算法运行的结果一致，即为正确。

七、 源代码张贴

MD5.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// 四个轮函数
#define F(b, c, d) ((b & c) | (~b & d))
#define G(b, c, d) ((b & d) | (c & ~d))
#define H(b, c, d) (b ^ c ^ d)
#define I(b, c, d) (c ^ (b | ~d))

/*
循环左移，示例如下：
    10101010 10101010 10101010 10101010
    左移 3 位：
    01010 10101010 10101010 10101010 000
                                     010
    按位或即为正确结果
*/
#define CLS(x, s) ((x << s) | (x >> (32 -s))) //循环左移

```

```
21.
22.// MD 缓冲区的初始向量 IV(A, B, C, D)
23.const unsigned int IV[4] = {
24.    0x67452301,
25.    0xEFCDAB89,
26.    0x98BADCFE,
27.    0x10325476};
28.
29.// 各轮循环中第 i 次迭代 (i = 1 .. 16) 使用的 X[k]
30.const int X_index[4][16] = {
31.    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15},
32.    {1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12},
33.    {5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2},
34.    {0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9}};
35.
36.// 各轮各次迭代运算采用的 T 值
37.const unsigned int T[4][16] = {
38.    {0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf,
   0x4787c62a, 0xa8304613, 0xfd469501,
39.     0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be, 0x6b901122,
   0xfd987193, 0xa679438e, 0x49b40821},
40.
41.    {0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa, 0xd62f105d,
   0x02441453, 0xd8a1e681, 0xe7d3fbc8,
42.     0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed, 0xa9e3e905,
   0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a},
43.
44.    {0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c, 0xa4beea44,
   0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
45.     0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05, 0xd9d4d039,
   0xe6db99e5, 0x1fa27cf8, 0xc4ac5665},
46.
47.    {0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039, 0x655b59c3,
   0x8f0ccc92, 0xffeff47d, 0x85845dd1,
48.     0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1, 0xf7537e82,
   0xbd3af235, 0x2ad7d2bb, 0xeb86d391}};
49.
50.// 各轮各次迭代运算 (1 .. 64) 采用的左循环移位的位数 s 值
51.const int S[4][16] = {
52.    {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22},
53.    {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20},
54.    {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23},
55.    {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}}
   ;
```

```
56.
57.int blockLen;                  // 512 位长的分组数
58.unsigned int **paddedMessage; // 存放填充后 message 的 32 位无符号整形
   变量二维数组
59.
60.// 取 4 个字节，按 little-endian 转移到 1 个 32 位无符号整型变量
61.void Byte2int(unsigned char *Byte)
62.{
63.    for (int i = 0; i < blockLen; ++i)
64.    {
65.        for (int j = 0; j < 16; ++j)
66.        {
67.            paddedMessage[i][j] = (Byte[i * 64 + j * 4]) |
68.                                  (Byte[i * 64 + j * 4 + 1] << 8)
                                    |(Byte[i * 64 + j * 4 + 2] << 16)
                                    | (Byte[i * 64 + j * 4 + 3] << 24);
69.        }
70.    }
71.}
72.
73.// 将 32 位无符号整型转化成 4 个字符
74.void int2char(unsigned int *src, unsigned char *dst, long length)
75.{
76.    for(int i = 0; i < length; ++i)
77.    {
78.        dst[i * 4 + 3] = (src[i] >> 24) & 0x000000ff;
79.        dst[i * 4 + 2] = (src[i] >> 16) & 0x000000ff;
80.        dst[i * 4 + 1] = (src[i] >> 8) & 0x000000ff;
81.        dst[i * 4] = src[i] & 0x000000ff;
82.    }
83.}
84.
85.// 填充函数
86.void padding(char *message, unsigned long long messageLen)
87.{
88.    blockLen = (messageLen / 64) + ((messageLen * 8) % 512 >= 448
    ? 2 : 1);
89.    unsigned char temp[blockLen * 64];
90.    for (int i = 0; i < messageLen; ++i)
91.        temp[i] = message[i];
92.    temp[messageLen] = 0x80; // 0x80 -> 1000 0000
93.    for (int i = messageLen + 1; i < blockLen * 64; ++i)
94.        temp[i] = 0x00;
95.
```

```
96.     // 动态分配二维数组的内存空间
97.     paddedMessage = (unsigned int **)malloc(blockLen * sizeof(uns
    igned int *));
98.     for (int i = 0; i < blockLen; ++i)
99.         paddedMessage[i] = (unsigned int *)malloc(16 * sizeof(uns
    igned int));
100.
101.     Byte2int(temp);
102.     unsigned int left = ((messageLen * 8) >> 32) & 0x00000000fff
    fffff;
103.     unsigned int right = (messageLen * 8) & 0x00000000ffffffff;
104.     paddedMessage[blockLen - 1][15] = left;
105.     paddedMessage[blockLen - 1][14] = right;
106. }
107.
108. // 循环置换：ABCD -> DABC
109. void buffer_iteration(unsigned int MD[4])
110. {
111.     unsigned int temp = MD[0];
112.     MD[0] = MD[3];
113.     MD[3] = MD[2];
114.     MD[2] = MD[1];
115.     MD[1] = temp;
116. }
117.
118. // MD5 压缩函数 a = b + ((a + g(b, c, d) + X[k] + T[i]) <<< s)
119. void HMD5(char *message, char * output, unsigned long long messa
    geLen)
120. {
121.     padding(message, messageLen);
122.     unsigned int CV[4];
123.     //初始化 CV
124.     for (int i = 0; i < 4; ++i)
125.         CV[i] = IV[i];
126.
127.     for (int i = 0; i < blockLen; ++i)
128.     {
129.         // 首先记录下 64 轮迭代前的 CV 值
130.         unsigned int temp[4];
131.         for(int j = 0; j < 4; ++j)
132.             temp[j] = CV[j];
133.
134.         // 第一轮循环 F
135.         for (int j = 0; j < 16; ++j)
```

```
136.         {
137.             CV[0] = CV[1] + CLS((CV[0] + F(CV[1], CV[2], CV[3])
    + paddedMessage[i][X_index[0][j]] + T[0][j]), S[0][j]);
138.             buffer_iteration(CV);
139.         }
140.         // 第二轮循环G
141.         for (int j = 0; j < 16; ++j)
142.         {
143.             CV[0] = CV[1] + CLS((CV[0] + G(CV[1], CV[2], CV[3])
    + paddedMessage[i][X_index[1][j]] + T[1][j]), S[1][j]);
144.             buffer_iteration(CV);
145.         }
146.         // 第三轮循环H
147.         for (int j = 0; j < 16; ++j)
148.         {
149.             CV[0] = CV[1] + CLS((CV[0] + H(CV[1], CV[2], CV[3])
    + paddedMessage[i][X_index[2][j]] + T[2][j]), S[2][j]);
150.             buffer_iteration(CV);
151.         }
152.         // 第四轮循环I
153.         for (int j = 0; j < 16; ++j)
154.         {
155.             CV[0] = CV[1] + CLS((CV[0] + I(CV[1], CV[2], CV[3])
    + paddedMessage[i][X_index[3][j]] + T[3][j]), S[3][j]);
156.             buffer_iteration(CV);
157.         }
158.
159.         for(int j = 0; j < 4; ++j)
160.             CV[j] += temp[j];
161.     }
162.
163.     int2char(CV, output, 4);
164. }
165.
166. /*
167. int main()
168. {
169.     char *s = "I love you, you are my sunshine!";
170.     unsigned char output[16];
171.     HMD5(s, output, 32);
172.     for(int i = 0; i < 16; ++i)
173.         printf("%02x", output[i]);
174.     free(paddedMessage);
175. }*/
```

hmac.c

```c
#include <stdio.h>
#include <string.h>
#include "MD5.c"

#define BLOCKSIZE 64  // b = 64 * 8 = 512 位
#define ipad 0x36     // 00110110
#define opad 0x5c     // 01011100

char *key;    // secrete key, |k| <= b
char *Kplus;  //对共享密钥 k 左边补0，生成一个 b 位的数据块 K+
char *Si;     // K+ ^ ipad
char *So;     // K+ ^ opad

void getKplus()
{
    Kplus = (char *)malloc(sizeof(char) * BLOCKSIZE);
    for(int i = 0; i < strlen(key); ++i)
        Kplus[i] = key[i];
    for(int i = strlen(key); i < BLOCKSIZE; ++i)
        Kplus[i] = 0;
}

void getSi_So()
{
    Si = (char *)malloc(sizeof(char) * BLOCKSIZE);
    for(int i = 0; i < BLOCKSIZE; ++i)
        Si[i] = Kplus[i] ^ ipad;
    So = (char *)malloc(sizeof(char) * BLOCKSIZE);
    for(int i = 0; i < BLOCKSIZE; ++i)
        So[i] = Kplus[i] ^ opad;
}

// 字符串拼接函数
void add_str(char * str1, int len1, char * str2, int len2, char *
    res)
{
    for(int i = 0; i < len1; ++i)
        res[i] = str1[i];
    for(int i = 0; i < len2; ++i)
        res[len1 + i] = str2[i];
}

```

```
42.void hmac(unsigned char * M, unsigned long long Mlen, unsigned ch
   ar * key, unsigned char * HMAC)
43.{
44.    getKplus();
45.    getSi_So();
46.    char SM[BLOCKSIZE + Mlen];
47.    add_str(Si, BLOCKSIZE, M, Mlen, SM);
48.    unsigned char hash1[16];
49.    HMD5(SM, hash1, BLOCKSIZE + Mlen);
50.    char SH[BLOCKSIZE + 16];
51.    add_str(So, BLOCKSIZE, hash1, 16, SH);
52.    HMD5(SH, HMAC, BLOCKSIZE + 16);
53.}
54.
55.// 将利用malloc 函数动态分配的内存空间统一释放
56.void freeall()
57.{
58.    free(paddedMessage);
59.    free(Kplus);
60.    free(Si);
61.    free(So);
62.}
63.
64.int main()
65.{
66.    char *s = "I love you, you are my sunshine!";
67.    key = "abcde";
68.    unsigned char output[16];
69.    hmac(s, 32, key, output);
70.    for(int i = 0; i < 16; ++i)
71.        printf("%02x", output[i]);
72.    freeall();
73.}
```