

RSA 算法报告

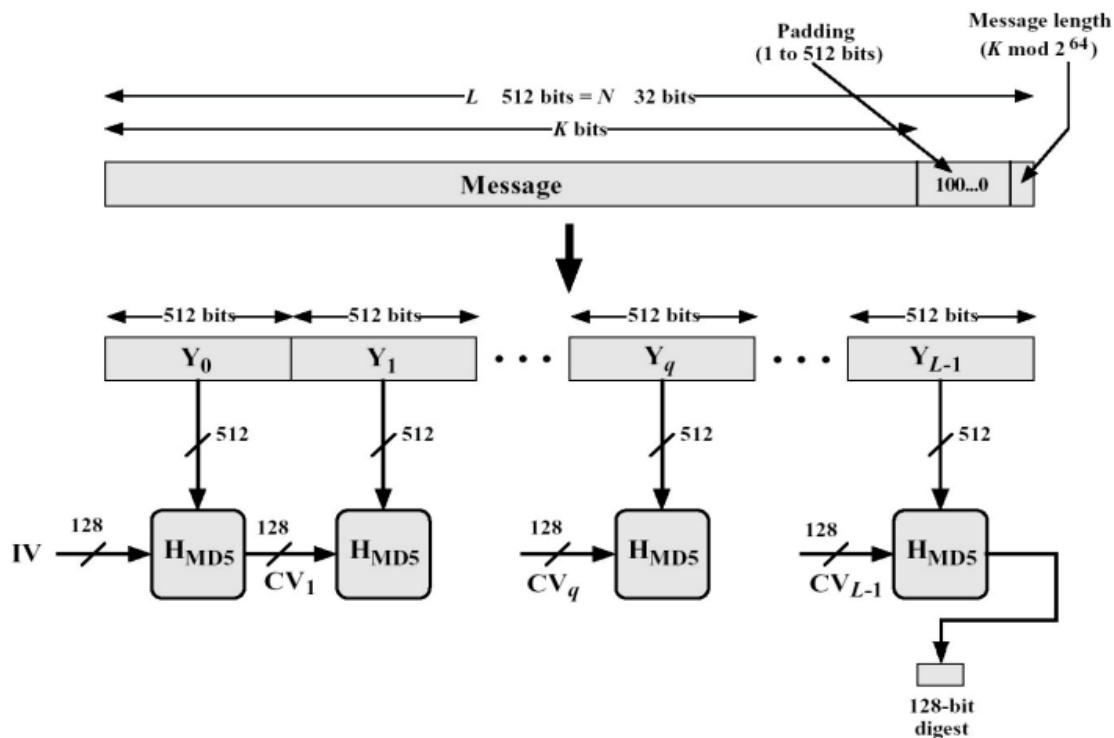
- 米家龙
- 18342075
- 数据科学与计算机学院
- RSA 算法报告
 - 原理概述
 - MD5
 - 总控流程
 - 压缩函数 H_{MD5}
 - 轮函数 F, G, H, I
 - X表
 - T表
 - S表
 - HMAC 算法
 - 总体结构设计
 - 数据结构设计
 - MD5
 - HMAC
 - 模块分解
 - MD5
 - HMAC
 - C语言代码
 - md5.c
 - hmac-md5.c
 - md5test.c
 - 编译运行结果
 - 验证用例

原理概述

MD5

基本流程如下图：

■ MD5 算法的基本流程



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

$$CV_0 = IV$$

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

其中初始向量 IV 为：

$$A = 0x67452301$$

$$B = 0xEFCDAB89$$

$$C = 0x98BADCFE$$

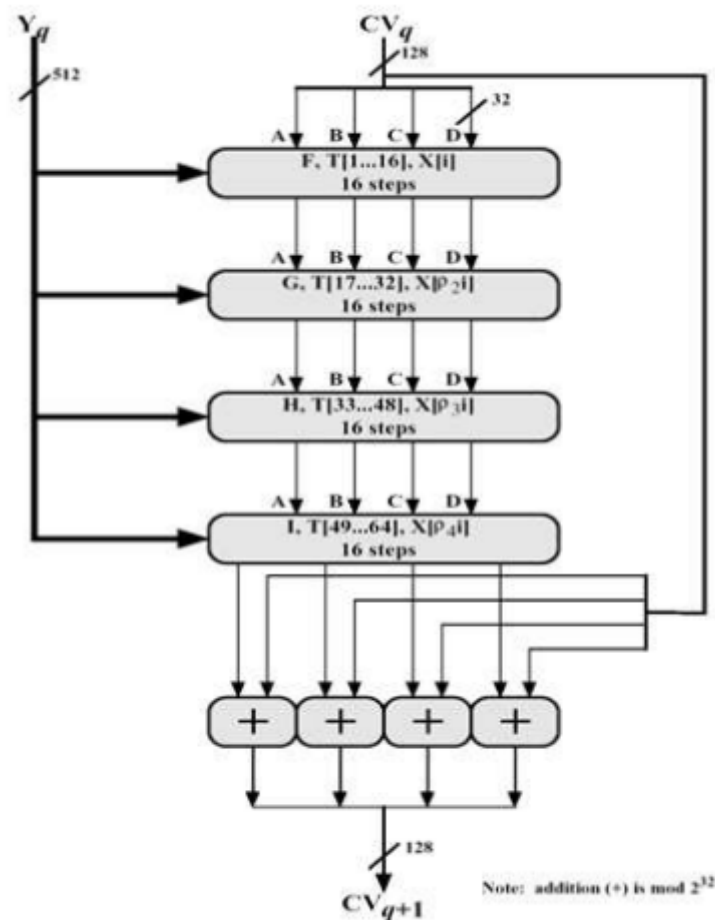
$$D = 0x10325476$$

- 最终结果为： $MD = CV_L$

压缩函数 H_{MD5}

- H_{MD5} 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生成函数 F, G, H, I ，结合指定的 T 表元素 $T[]$ 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。

流程如下图：



MD5 第 q 分组的4轮循环逻辑 (压缩函数)

轮函数 F, G, H, I

4轮循环中使用的生成函数 g (也称**轮函数**) 是一个32位非线性逻辑函数。同一轮循环的所有迭代使用相同的 g 函数，而各轮循环对应的 g 函数具有不同的定义，具体如下：

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

每轮循环中的一次迭代运算逻辑

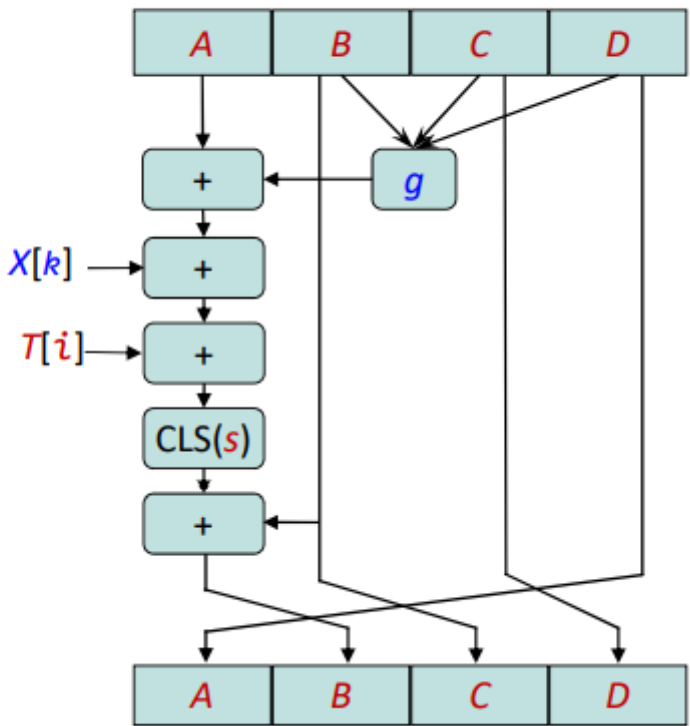
1. 对A迭代: $a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i]) \lll s)$
2. 对缓冲区: $(B, C, D, A) \leftarrow (A, B, C, D)$

- 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
- $\lll s$: 将32位输入循环左移 (CLS) s 位; s 为规定值
- $X[k]$: 当前处理消息分组 q 的第 k 个 ($k = 0 \dots 15$) 32位字。如果消息 M 按 32-bit 编址, 即为 $M_{q \times 16 + k}$

- $T[i]$: T表的第*i*个元素, 32位字; T表总共有64个元素, 也称为加法常数
- $+$: 模 2^{32} 加法

一次迭代流程如图:

每轮循环中的一次迭代运算逻辑示意图



X表

四轮循环, 16次迭代, 合计64次

在每轮循环第*i*次迭代($i = 0 \dots 15$)中, $X[k]$ 的选择如下:

- 第1轮迭代: $k = i$
- 第2轮迭代: $k = (1 + 5i) \bmod 16$
- 第3轮迭代: $k = (5 + 3i) \bmod 16$
- 第4轮迭代: $k = 7j \bmod 16$

T表

T表生成函数:

$$T[i] = \text{int}(2^{32} \times |\sin(i)|)$$

- *int* 取整函数
- *sin* 正弦函数
- *i* 弧度

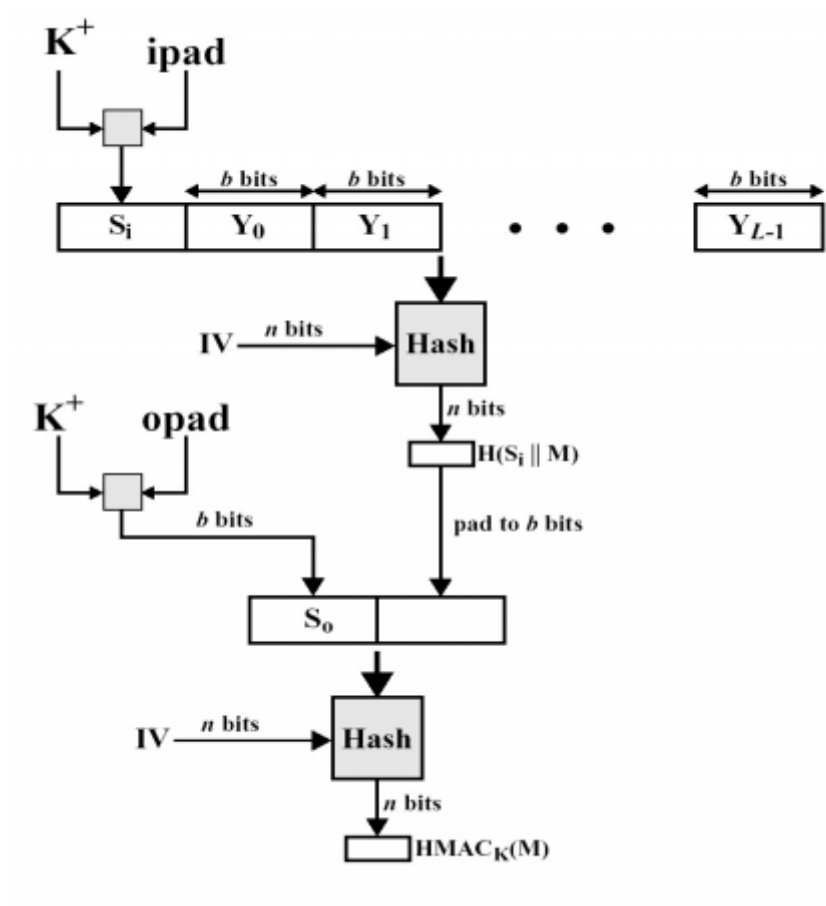
S表

每次迭代($i = 0 \dots 63$) 采用的左循环移位的位数 *s* 值

$S[0 \dots 15] = \{7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22\}$
 $S[16 \dots 31] = \{5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20\}$
 $S[32 \dots 47] = \{4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23\}$
 $S[48 \dots 63] = \{6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21\}$

HMAC 算法

总体流程如图：



- M – message input to HMAC
- H – embedded hash function
- b – length (bits) of input block
- k – secrete key, $|k| < b$
- n – length of hash code
- *ipad* – 00110110 (0x36) 重复 b/8 次
- *opad* – 01011100 (0x5c) 重复 b/8 次

1. 对共享密钥 k 右边进行补0，生成一个64字节的数据块 K^+
2. K^+ 与 *ipad* 进行异或，生成64字节的 S_i
3. $(S_i || M)$ 进行 hash 压缩 (例如 MD5)，得到 $H(S_i || M)$
4. K^+ 与 *opad* 进行异或，生成64字节的 S_o
5. 对 $S_o || H(S_i || M)$ 进行 hash 压缩 (例如 MD5)，得到 $HMAC_K = H(S_o || H(S_i || M))$

|| 代表字符串拼接

总体结构设计

该项目分成三个文件，如下：

- md5.c：MD5 算法文件
- hamc-md5.c：HMAC_MD5 算法文件
- md5test.c：用于测试 MD5 算法的测试文件

数据结构设计

MD5

函数和数据结构设计如下：

```
1  #define F(b, c, d) ((b & c) | (~b & d))          // 第一轮
2  #define G(b, c, d) ((b & d) | (c & ~d))          // 第二轮
3  #define H(b, c, d) (b ^ c ^ d)                  // 第三轮
4  #define I(b, c, d) (c ^ (b | ~d))                // 第四轮
5  #define CLS(x, s) ((x << s) | (x >> (32 - s)))    // 循环左移
6
7  typedef struct
8  {
9      unsigned int content[4];
10 } MD5_CTX;
11
12 // 初始向量，小端
13 const unsigned int IV[4] = {
14     0x67452301,
15     0xEFCDAB89,
16     0x98BADCFE,
17     0x10325476};
18
19 unsigned int CV[4] = {
20     0x67452301,
21     0xEFCDAB89,
22     0x98BADCFE,
23     0x10325476};
24
25 // 各轮各次迭代运算采用的 T 值
26 const int T_TABLE[] = {
27     0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
28     0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
29     0x698098d8, 0x8b44f7af, 0xfffff5bb1, 0x895cd7be,
30     0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
31     0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
32     0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
33     0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
34     0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
35     0xfffffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
36     0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbcb70,
37     0x289b7ec6, 0xeee127fa, 0xd4ef3085, 0x04881d05,
38     0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
39     0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
40     0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
41     0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
42     0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};
43
```

```

44 // 各轮各次迭代运算 (1 .. 64) 采用的左循环移位的位数 s 值
45 const int S_TABLE[] = {
46     7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
47     5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
48     4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
49     6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21};
50
51 // 暂时没用上
52 const int X_TABLE[4][16] = {
53     {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}, // k = j
54     {1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12}, // k = (1 + 5 * j) %
16
55     {5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2}, // k = (5 + 3 * j) %
16
56     {0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9}}; // k = (7 * j) % 16
57
58 unsigned int *MessagePadded; // 由字符串转化而成的32位块
59 unsigned long blockLength; // 512位块的长度
60 unsigned char *messagePaddingTmp;

```

HMAC

相关声明如下：

```

1  #define BLOCKSIZE 64
2
3  unsigned long b = 0; // length (bits) of input block
4  unsigned long kLength = 0; // length (bits) of input key
5  char *M; // Message
6  char *k; // 密钥
7  char *KPlus; // 根据输入内容和密钥生成的数据块
8  char *Si; // K+ ^ ipad
9  char *So; // K+ ^ opad
10
11 #define ipad 0x36 // 00110110
12 #define opad 0x5c // 01011100

```

模块分解

MD5

具体函数声明如下：

```

1  /**
2   * 初始化向量
3   */
4  void CVInit(unsigned int CV[4]);
5
6  /**
7   * 填充步骤
8   * @param originMessage char* 原始消息
9   * @param messageLength unsigned long long 为了避免特殊情况，将明文长度（字节数）作为输入
10  */
11 void padMessage(char *originMessage, unsigned long messageLength);
12

```

```

13  /**
14   * 主要函数
15   * @param originMessage char* 原始消息
16   * @param messageLength unsigned long long 为了避免特殊情况，将明文长度（字节数）作为输入
17   */
18  MD5_CTX MD5(char *originMessage, unsigned long messageLength);
19
20  /**
21   * 将字符串转换为数字
22   * @param src unsigned char* 源字符串
23   * @param charLength unsigned long 字符串长度
24   * @return 一个 unsigned int 数组指针
25   */
26  unsigned int *MD5_Decode(unsigned char *src, unsigned int *dst, unsigned long
charLength);
27
28  /**
29   * 将数字转换为字符串
30   * @param src unsigned int* 源数组
31   * @param intLength unsigned long 数组长度
32   */
33  unsigned char *MD5_Encode(unsigned int *src, unsigned char *dst, unsigned long
intLength);
34
35  /**
36   * MD5 压缩函数
37   * @param thisCV char* 输入向量，128位
38   * @param Y int* 分组，每个分组为一个块，512位
39   * @param res int* 返回结果
40   */
41  void H_MD5(int *Y, unsigned int *res);
42
43  /**
44   * 释放内存
45   */
46  void clear();

```

其中主要函数 MD5() 具体代码如下：

```

1  MD5_CTX MD5(char *originMessage, unsigned long messageLength)
2  {
3      MD5_CTX res;
4      padMessage(originMessage, messageLength); // 消息填充
5      CVInit(CV); // 初始化 CV
6
7      for (int i = 0; i < blockLength; i++)
8      {
9          unsigned int tmp[4] = {0, 0, 0, 0};
10         H_MD5(MessagePadded + i * 16, tmp);
11         for (int j = 0; j < 4; j++)
12         {
13             CV[j] += tmp[j];
14         }
15     }
16
17     clear();
18     return res;
19 }

```


HMAC

相关函数声明如下：

```
1  /**
2   * 获取文件的大小，并设置初始的相关值
3   * @param inputFilename char* 输入文件名
4   * @param keyFilename char* 密钥文件名
5   * @return 1 | 0，其中0代表密钥不符合要求
6   */
7  int getFileSize(char *inputFilename, char *keyFilename);
8
9  /**
10   * 生成 K+，顺带生成 Si 和 So
11   */
12  void generateKPlus();
13
14  /**
15   * 哈希函数
16   * @param S unsigned char* 第一个字符串
17   * @param SLength unsigned long 第一个字符串的长度
18   * @param M unsigned char* 第二个字符串
19   * @param MLength unsigned long 第二个字符串的长度
20   * @return MD5_CTX 结构体
21   */
22  MD5_CTX Hash(unsigned char *S, unsigned long SLength, unsigned char *M, unsigned
long MLength);
23
24  /**
25   * 释放内存
26   */
27  void freeAll();
```

C语言代码

md5.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define F(b, c, d) ((b & c) | (~b & d))          // 第一轮
5  #define G(b, c, d) ((b & d) | (c & ~d))          // 第二轮
6  #define H(b, c, d) (b ^ c ^ d)                  // 第三轮
7  #define I(b, c, d) (c ^ (b | ~d))                // 第四轮
8  #define CLS(x, s) ((x << s) | (x >> (32 - s))) // 循环左移
9
10 typedef struct
11 {
12     unsigned int content[4];
13 } MD5_CTX;
14
15 // 初始向量，小端
16 const unsigned int IV[4] = {
17     0x67452301,
```

```

18     0xEFCDA89,
19     0x98BADCFE,
20     0x10325476};
21
22 unsigned int CV[4] = {
23     0x67452301,
24     0xEFCDA89,
25     0x98BADCFE,
26     0x10325476};
27
28 // 各轮各次迭代运算采用的 T 值
29 const int T_TABLE[] = {
30     0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
31     0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
32     0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
33     0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
34     0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
35     0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
36     0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
37     0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
38     0xffffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
39     0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfb70,
40     0x289b7ec6, 0xeea127fa, 0xd4ef3085, 0x04881d05,
41     0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
42     0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
43     0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
44     0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
45     0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391};
46
47 // 各轮各次迭代运算 (1 .. 64) 采用的左循环移位的位数 s 值
48 const int S_TABLE[] = {
49     7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
50     5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
51     4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
52     6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21};
53
54 // 暂时没用上
55 const int X_TABLE[4][16] = {
56     {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}, // k = j
57     {1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12}, // k = (1 + 5 * j)
58     {5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2}, // k = (5 + 3 * j)
59     {0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9}}; // k = (7 * j) % 16
60
61 unsigned int *MessagePadded; // 由字符串转化而成的32位块
62 unsigned long blockLength; // 512位块的长度
63 unsigned char *messagePaddingTmp;
64
65 /**
66  * 初始化向量
67  */
68 void CVInit(unsigned int CV[4]);
69
70 /**
71  * 填充步骤
72  * @param originMessage char* 原始消息

```

```

73     * @param messageLength unsigned long long 为了避免特殊情况，将明文长度（字节数）作为输入
74     */
75     void padMessage(char *originMessage, unsigned long messageLength);
76
77     /**
78     * 主要函数
79     * @param originMessage char* 原始消息
80     * @param messageLength unsigned long long 为了避免特殊情况，将明文长度（字节数）作为输入
81     */
82     MD5_CTX MD5(char *originMessage, unsigned long messageLength);
83
84     /**
85     * 将字符串转换为数字
86     * @param src unsigned char* 源字符串
87     * @param charLength unsigned long 字符串长度
88     * @return 一个 unsigned int 数组指针
89     */
90     unsigned int *MD5_Decode(unsigned char *src, unsigned int *dst, unsigned long charLength);
91
92     /**
93     * 将数字转换为字符串
94     * @param src unsigned int* 源数组
95     * @param intLength unsigned long 数组长度
96     */
97     unsigned char *MD5_Encode(unsigned int *src, unsigned char *dst, unsigned long intLength);
98
99     /**
100    * MD5 压缩函数
101    * @param thisCV char* 输入向量，128位
102    * @param Y int* 分组，每个分组为一个块，512位
103    * @param res int* 返回结果
104    */
105    void H_MD5(int *Y, unsigned int *res);
106
107    /**
108    * 释放内存
109    */
110    void clear();
111
112    void clear()
113    {
114        free(MessagePadded);
115        free(messagePaddingTmp);
116    }
117
118    MD5_CTX MD5(char *originMessage, unsigned long messageLength)
119    {
120        MD5_CTX res;
121        padMessage(originMessage, messageLength); // 消息填充
122        CVInit(CV); // 初始化 CV
123
124        for (int i = 0; i < blockLength; i++)
125        {
126            unsigned int tmp[4] = {0, 0, 0, 0};

```

```

127     H_MD5(MessagePadded + i * 16, tmp);
128     for (int j = 0; j < 4; j++)
129     {
130         CV[j] += tmp[j];
131     }
132 }
133
134 // 放到结果中
135 for (int i = 0; i < 4; i++)
136 {
137     res.content[i] = CV[i];
138 }
139
140 clear();
141 return res;
142 }
143
144 void CVInit(unsigned int thisCV[4])
145 {
146     for (int i = 0; i < 4; i++)
147     {
148         thisCV[i] = IV[i];
149     }
150 }
151
152 void padMessage(char *originMessage, unsigned long messageLength)
153 {
154     blockLength = messageLength / 64 + (((messageLength * 8) % 512) >= 448 ? 2 :
155     1);
156     messagePaddingTmp = (unsigned char *)malloc(blockLength * 64);
157     for (int i = 0; i < messageLength; i++)
158     {
159         messagePaddingTmp[i] = originMessage[i];
160     }
161
162     // 后续填充为0
163     for (int i = messageLength; i < blockLength * 64; i++)
164     {
165         messagePaddingTmp[i] = 0;
166     }
167
168     messagePaddingTmp[messageLength] = 0x80; // 结束的第一位为1
169
170     MessagePadded = malloc(blockLength * 16);
171     MD5_Decode(messagePaddingTmp, MessagePadded, blockLength * 64);
172     unsigned int front32 = ((messageLength * 8) >> 32) & 0x00000000ffffffff; // 前
173     32位，但是需要倒序放在最后
174     unsigned int behind32 = (messageLength * 8) & 0x00000000ffffffff; // 后
175     32位，倒序放在最前
176     MessagePadded[blockLength * 16 - 2] = behind32;
177     MessagePadded[blockLength * 16 - 1] = front32;
178
179     return;
180 }
181
182 unsigned char *MD5_Encode(unsigned int *src, unsigned char *dst, unsigned long
183 intLength)
184 {

```

```

181     for (int i = 0; i < intLength; i++)
182     {
183         dst[i * 4 + 3] = (src[i] >> 24) & 0x000000ff;
184         dst[i * 4 + 2] = (src[i] >> 16) & 0x000000ff;
185         dst[i * 4 + 1] = (src[i] >> 8) & 0x000000ff;
186         dst[i * 4] = src[i] & 0x000000ff;
187     }
188
189     return dst;
190 }
191
192 unsigned int *MD5_Decode(unsigned char *src, unsigned int *dst, unsigned long
charLength)
193 {
194     for (int i = 0; i < charLength / 4; i++)
195     {
196         dst[i] = (src[i * 4]) |
197                 (src[i * 4 + 1] << 8) |
198                 (src[i * 4 + 2] << 16) |
199                 (src[i * 4 + 3] << 24);
200     }
201
202     return dst;
203 }
204
205 void H_MD5(int *Y, unsigned int res[4])
206 {
207     unsigned int thisCV[4];
208     unsigned int nextCV[4];
209
210     for (int i = 0; i < 4; i++)
211     {
212         thisCV[i] = CV[i];
213     }
214     // 四轮循环, 每轮循环16步
215     for (int j = 0; j < 4; j++)
216     {
217         for (int i = 0; i < 16; i++)
218         {
219             // 每次迭代的参数都有变化
220             switch (j)
221             {
222             case 0:
223                 nextCV[1] = thisCV[1] +
224                             CLS((thisCV[0] +
225                                 F(thisCV[1], thisCV[2], thisCV[3]) +
226                                 Y[i] +
227                                 T_TABLE[i]),
228                                 S_TABLE[i]);
229                 break;
230             case 1:
231                 nextCV[1] = thisCV[1] +
232                             CLS((thisCV[0] +
233                                 G(thisCV[1], thisCV[2], thisCV[3]) +
234                                 Y[(1 + 5 * i) % 16] +
235                                 T_TABLE[i + j * 16]),
236                                 S_TABLE[i + j * 16]);
237                 break;

```

```

238     case 2:
239         nextCV[1] = thisCV[1] +
240             CLS((thisCV[0] +
241                 H(thisCV[1], thisCV[2], thisCV[3]) +
242                 Y[(5 + 3 * i) % 16] +
243                 T_TABLE[i + j * 16]),
244                 S_TABLE[i + j * 16]);
245         break;
246     case 3:
247         nextCV[1] = thisCV[1] +
248             CLS((thisCV[0] +
249                 I(thisCV[1], thisCV[2], thisCV[3]) +
250                 Y[(7 * i) % 16] +
251                 T_TABLE[i + j * 16]),
252                 S_TABLE[i + j * 16]);
253         break;
254     default:
255         break;
256 }
257 nextCV[2] = thisCV[1];
258 nextCV[3] = thisCV[2];
259 nextCV[0] = thisCV[3];
260
261 // 迭代
262 for (int i = 0; i < 4; i++)
263 {
264     thisCV[i] = nextCV[i];
265 }
266 }
267 }
268
269 for (int i = 0; i < 4; i++)
270 {
271     res[i] = thisCV[i];
272 }
273 }

```

hmac-md5.c

```

1  #include <stdio.h>
2  #include <sys/stat.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include "md5.c"
6
7  #define BLOCKSIZE 64
8
9  unsigned long b = 0;           // length (bits) of input block
10 unsigned long kLength = 0;     // length (bits) of input key
11 char *M;                       // Message
12 char *k;                       // 密钥
13 char *KPlus;                   // 根据输入内容和密钥生成的数据块
14 char *Si;                      // K+ ^ ipad
15 char *So;                      // K+ ^ opad
16
17 #define ipad 0x36 // 00110110
18 #define opad 0x5c // 01011100

```

```

19
20  /**
21   * 获取文件的大小，并设置初始的相关值
22   * @param inputFilename char* 输入文件名
23   * @param keyFilename char* 密钥文件名
24   * @return 1 | 0，其中0代表密钥不符合要求
25   */
26 int getFileSize(char *inputFilename, char *keyFilename);
27
28  /**
29   * 生成 K+，顺带生成 Si 和 So
30   */
31 void generateKPlus();
32
33  /**
34   * 哈希函数
35   * @param S unsigned char* 第一个字符串
36   * @param SLength unsigned long 第一个字符串的长度
37   * @param M unsigned char* 第二个字符串
38   * @param MLength unsigned long 第二个字符串的长度
39   * @return MD5_CTX 结构体
40   */
41 MD5_CTX Hash(unsigned char *S, unsigned long SLength, unsigned char *M, unsigned
long MLength);
42
43  /**
44   * 释放内存
45   */
46 void freeAll();
47
48 int main(int argc, char *argv[])
49 {
50     if (argc != 3)
51     {
52         printf("usage: ./a.out inputFile keyFile\n");
53         return 0;
54     }
55     else
56     {
57         FILE *inputFile, *keyFile;
58         if (!getFileSize(argv[1], argv[2]))
59         {
60             printf("密钥和文本不匹配\n");
61             return 0;
62         }
63
64         inputFile = fopen(argv[1], "r");
65         keyFile = fopen(argv[2], "r");
66
67         fread(M, 1, b, inputFile);
68         fread(k, 1, kLength, keyFile);
69
70         generateKPlus();
71
72         printf("消息为: \n%s\n", M);
73
74         // 第一次哈希
75         MD5_CTX firstResult = Hash(Si, BLOCKSIZE, M, b); // 第一次结果

```

```

76     unsigned char firstResultString[16];                // 第一次结果的字符串
77     MD5_Encode(firstResult.content, firstResultString, 4);
78     printf("第一次结果: \n");
79     for (int i = 0; i < 4 * 4; i++)
80     {
81         printf("%02x", firstResultString[i]);
82     }
83     putchar('\n');
84
85     //第二次哈希
86     MD5_CTX secondResult = Hash(So, BLOCKSIZE, firstResultString, 16); // 第二次
结果
87     unsigned char secondResultString[16];                // 第二次
结果的字符串
88     printf("第二次结果: \n");
89     MD5_Encode(secondResult.content, secondResultString, 4);
90     for (int i = 0; i < 4 * 4; i++)
91     {
92         printf("%02x", secondResultString[i]);
93     }
94     putchar('\n');
95
96     fclose(inputFile);
97     fclose(keyFile);
98     freeAll();
99     return 0;
100 }
101 }
102
103 MD5_CTX Hash(unsigned char *S, unsigned long SLength, unsigned char *M, unsigned
long MLength)
104 {
105     /**
106      * 拼接两个字符串
107      * 不敢用 strcat() 怕有 0 的存在
108      */
109     unsigned char SM[MLength + SLength + 1];
110     SM[SLength + MLength] = 0;
111
112     // 第一段
113     for (unsigned long i = 0; i < SLength; i++)
114     {
115         SM[i] = S[i];
116     }
117
118     // 第二段
119     for (unsigned long i = 0; i < MLength; i++)
120     {
121         SM[i + SLength] = M[i];
122     }
123
124     MD5_CTX res = MD5(SM, SLength + MLength);
125
126     return res;
127 }
128
129 void freeAll()
130 {

```



```

131     free(M);
132     free(k);
133     free(KPlus);
134     free(Si);
135     free(So);
136 }
137
138 int getFileSize(char *inputFilename, char *keyFilename)
139 {
140     struct stat inputFileBuffer, keyFileBuffer;
141     stat(inputFilename, &inputFileBuffer);
142     stat(keyFilename, &keyFileBuffer);
143     b = inputFileBuffer.st_size;      // 消息大小
144     kLength = keyFileBuffer.st_size; // 密钥长度
145     printf("文件大小为 %ld 字节\n密钥长度为 %ld 字节\n", b, kLength);
146
147     // 如果密钥不符合要求
148     if (BLOCKSIZE < kLength)
149     {
150         return 0;
151     }
152     else
153     {
154         M = (char *)malloc(b);
155         k = (char *)malloc(BLOCKSIZE);
156         return 1;
157     }
158 }
159
160 void generateKPlus()
161 {
162     KPlus = (char *)malloc(BLOCKSIZE);
163
164     // 右边补位0
165     for (unsigned long i = 0; i < kLength; i++)
166     {
167         KPlus[i] = k[i];
168     }
169
170     for (unsigned long i = kLength; i < BLOCKSIZE; i++)
171     {
172         KPlus[i] = 0;
173     }
174
175     // 顺带获取 Si 和 So
176     Si = (char *)malloc(BLOCKSIZE);
177     So = (char *)malloc(BLOCKSIZE);
178
179     for (unsigned long i = 0; i < BLOCKSIZE; i++)
180     {
181         Si[i] = KPlus[i] ^ ipad;
182         So[i] = KPlus[i] ^ opad;
183     }
184 }

```

md5test.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/stat.h>
4  #include "md5.c"
5
6  int main(int argc, char *argv[])
7  {
8      if (argc != 2)
9      {
10         printf("usage: ./md5 inputFile\n");
11         return 0;
12     }
13     FILE *inputFile;
14     struct stat inputFileBuffer;
15     stat(argv[1], &inputFileBuffer);
16     long inputFileSize = inputFileBuffer.st_size;    // 输入文件的大小
17     char *message = (char *)malloc(inputFileSize + 1); // 消息
18
19     // 读取文件
20     inputFile = fopen(argv[1], "r");
21     inputFileSize = fread(message, 1, inputFileSize, inputFile);
22
23     printf("输入消息为: \n%s\n", message);
24
25     // 获取结果
26     MD5_CTX res = MD5(message, inputFileSize);
27     unsigned char rres[16];
28     MD5_Encode(res.content, rres, 4);
29
30     // 输出结果
31     printf("哈希结果为: \n");
32     for (int i = 0; i < 16; i++)
33     {
34         printf("%02x", rres[i]);
35     }
36     printf("\n");
37
38     return 0;
39 }
```

编译运行结果

运行环境为 win10 环境下的 wsl (ubuntu 18.04) :

```
1  root@LAPTOP-QTCGESH0:/mnt/d/blog/work/信息安全/003fix1# uname -a
2  Linux LAPTOP-QTCGESH0 4.4.0-19041-Microsoft #488-Microsoft Mon Sep 01 13:43:00 PST
   2020 x86_64 x86_64 x86_64 GNU/Linux
```

采用 makefile 进行简化操作, makefile 具体代码如下:

```
1  GCC := gcc                                # 编译器
2  HMAC-MD5 := ./hmac-md5.c                 # hmac-md5 语言源代码
3  MD5 := ./md5.c                           # md5 算法代码
```

```

4 MD5TEST := ./md5test.c      # md5test 代码
5
6 ORIGINFILE := ./input.txt    # 原始文本文件
7 KEYFILE := ./key.txt        # 密钥储存文件
8 OUTPUTFILE := ./output.txt  # 私钥储存文件
9
10 # hmac-md5 测试
11 htest: hmac-md5
12     @./hmac-md5 ${ORIGINFILE} ${KEYFILE}
13
14 # 生成 hmac-md5 可执行文件
15 hmac-md5: ${HMAC-MD5} ${MD5}
16     @${GCC} ${HMAC-MD5} -o $@
17
18 # md5 测试
19 mtest: md5
20     @./md5 ${ORIGINFILE}
21
22 # 生成 md5test 文件
23 md5: ${MD5TEST} ${MD5}
24     @${GCC} ${MD5TEST} -o $@
25
26 # 清空垃圾文件
27 clean:
28     @rm md5 || exit 0
29     @rm hmac-md5 || exit 0

```

测试文件 (input.txt) 文本为

```

1 My name is mijialong.
2 This is SYSU.
3 We have twenty-first weeks!
4 SYSU is a 985 and 211 university?
5 This is a test file.

```

MD5 测试运行结果为

```

root@LAPTOP-QTCGESHO:/mnt/d/blog/work/信息安全/003fix1# make mtest
输入消息为:
My name is mijialong.
This is SYSU.
We have twenty-first weeks!
SYSU is a 985 and 211 university?
This is a test file.
哈希结果为:
67f34f9a47d8a68d84f280c3ad3d1280

```

密钥文件 (key.txt) 文本为

```

1 this is a key

```

HMAC-MD5 运行结果为:

```
root@LAPTOP-QTCGESHO:/mnt/d/blog/work/信息安全/003fix1# make htest
文件大小为 118 字节
密钥长度为 13 字节
消息为：
My name is mijialong.
This is SYSU.
We have twenty-first weeks!
SYSU is a 985 and 211 university?
This is a test file.
第一次结果：
c7fd68fb7098b2e4eaf148dc1376844c
第二次结果：
8b5ae6e8b175112319954ed6b4a99503
```

验证用例

使用 [在线加密解密网站](#) 进行结果的验证

需要注意复制文本时结尾不是以回车符结尾

1. MD5 验证

结果如下图：



2. HMAC-MD5 验证

结果如下图：

加密/解密 散列/哈希 BASE64 图片/BASE64转换

明文:

My name is mijialong.
This is SYSU.
We have twenty-first weeks!
SYSU is a 985 and 211 university?
This is a test file.

散列/哈希算法:

- SHA1 SHA224 SHA256 SHA384 SHA512 MD5
- HmacSHA1 HmacSHA224 HmacSHA256 HmacSHA384 HmacSHA512 HmacMD5 PBKDF2

密钥 this is a key 哈希/散列

哈希值:

8b5ae6e8b175112319954ed6b4a99503