



南開大學  
Nankai University

# 南开大学网络空间安全学院

## 《密码学》实验报告

学 号：1813540

姓 名：陈鸿运

年 级：2018 级

专 业：信息安全

完成日期：2020 年 12 月 23 日

目录

1 MD5 原理 .....	1
1.1 MD5 简介 .....	1
1.2 MD5 算法描述 .....	1
1.2.1 MD5 的分组处理 .....	2
1.2.2 MD5 的压缩函数 .....	3
2 MD5 实现 .....	5
2.1 定义 MD5 类 .....	5
2.2 实现 MD5 类 .....	6
2.2.1 $CLR_s$ 循环左移位数 .....	6
2.2.2 基本逻辑函数 F、G、H、I .....	7
2.2.3 每轮的处理函数 .....	7
3 实验过程 .....	9
3.1 测试数据 .....	9
3.2 雪崩效应验证 .....	9

# 第一章 MD5 原理

## 1.1 MD5 简介

MD5 讯息摘要演算法（英语：MD5 Message-Digest Algorithm），一种被广泛使用的密码杂凑函数，可以产生出一个 128 位元（16 位元组）的散列值（hash value），用于确保信息传输完整一致。

其前身 MD4，由 Ron Rivest 于 1990 年 10 月作为 RFC 提出，1992 年 4 月公布的 MD4 的改进 (RFC 1320,1321) 称为 MD5。

## 1.2 MD5 算法描述

MD5 算法采用的是迭代性哈希算法函数，其一般结构如图1.1所示：

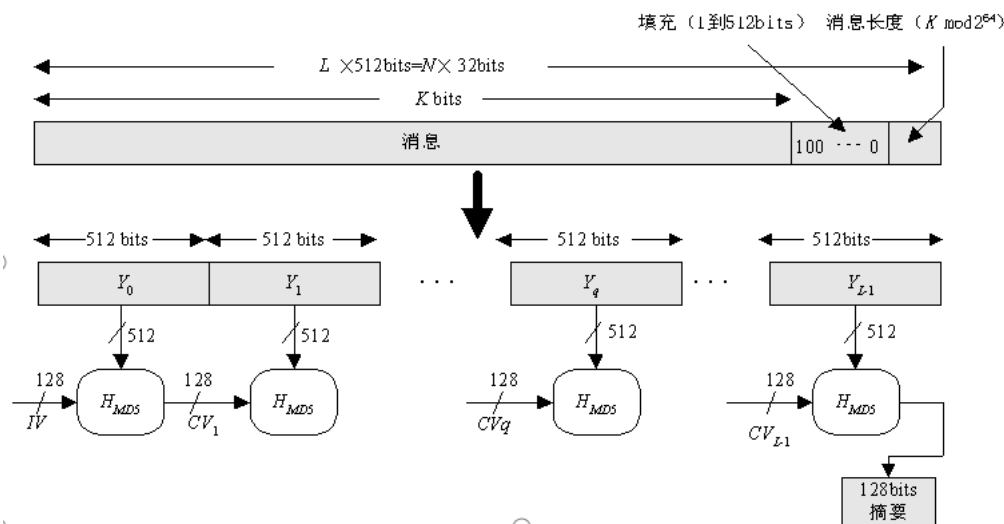


图 1.1: MD5 算法的框图

其主要处理过程有以下几步：

- 消息填充

对消息填充，使得其比特长在模 512 下为 448，即填充后消息的长度为 512 的某一倍数减 64，留出的 64 比特备第 2 步使用。步骤 1 是必需的，即使消息长度已

满足要求，仍需填充。例如，消息长为 448 比特，则需填充 512 比特，使其长度变为 960，因此填充的比特数大于等于 1 而小于等于 512。填充方式是固定的：第 1 位为 1，其后各位皆为 0。

- 附加消息的长度

使用前一个步骤中留出的 64 比特以小端方式来表示消息被填充前的长度。如果消息大于  $2^{64}$ ，则以  $2^{64}$  为模数取模。

- MD5 缓冲区初始化

算法使用 128 比特长的缓冲区以存储中间结果和最终哈希值，缓冲区可表示为四个 32 比特长的寄存器 (A, B, C, D)，每个寄存器都以小端方式存储数据，其初值取为（以存储方式）A=01234567, B=89ABCDEF, C= FEDCBA98, D=76543210，实际上为 67452301, EFCDAB89, 98BADCFE, 10325476。

- 以分组为单位对消息进行处理

每一分组  $Y_q (q = 0, \dots, L - 1)$  都经一压缩函数  $H_{MD5}$  处理。 $H_{MD5}$  是算法的核心，其中又有 4 轮处理过程（见下文）。

- 输出

消息的所有分组都被处理完后，最后一个  $H_{MD5}$  的输出即为产生的消息摘要。

### 1.2.1 MD5 的分组处理

四轮处理过程详见图1.2。

$H_{MD5}$  的 4 轮处理过程结构一样，但所用的逻辑函数不同，分别表示为 F、G、H、I。每轮的输入为当前处理的消息分组和缓冲区的当前值 A、B、C、D，输出仍放在缓冲区中以产生新的 A、B、C、D。每轮处理过程还需加上常数表 T 中四分之一元素，分别为 T[1..16], T[17..32], T[33..48], T[49..64]。表 T 有 64 个元素，在后续代码中展示。第 i 个元素 T[i] 为  $2^{32} * \text{abs}(\sin(i))$  的整数部分，其中 sin 为正弦函数，i 以弧度为单位。由于  $\text{abs}(\sin(i))$  大于 0 小于 1，所以 T[i] 可由 32 比特的字表示。第 4 轮的输出再与第 1 轮的输入  $CV_q$  相加，相加时将  $CV_q$  看作 4 个 32 比特的字，每个字与第 4 轮输出的对应的字按模相加，相加的结果即为压缩函数  $H_{MD5}$  的输出。

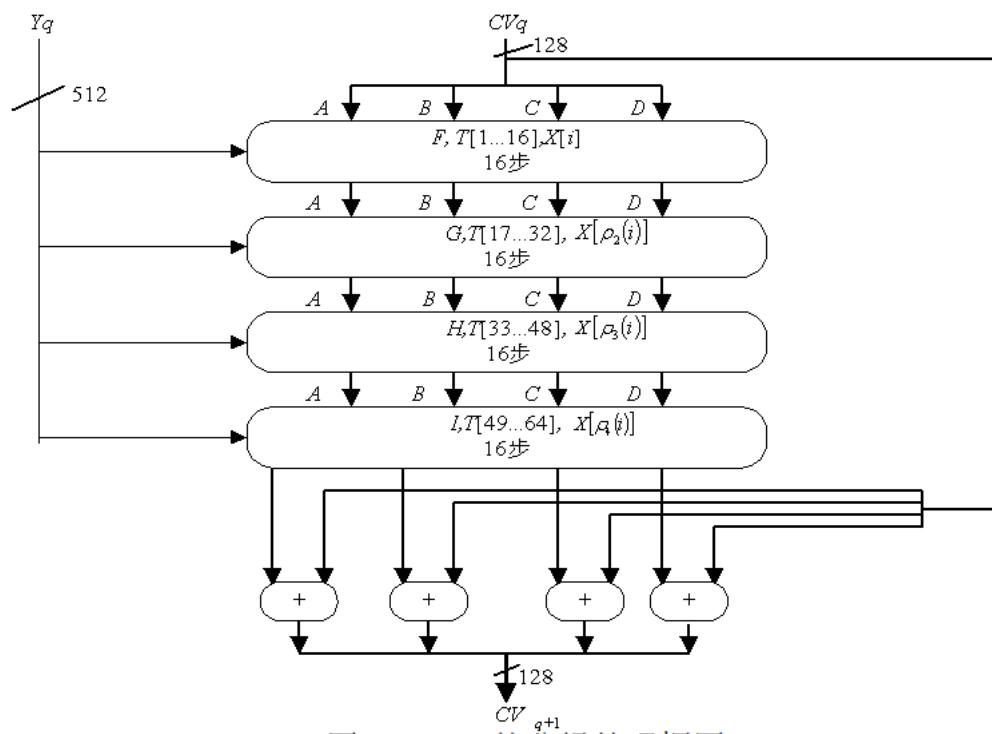


图 1.2: MD5 的分组处理框图

### 1.2.2 MD5 的压缩函数

压缩函数  $H_{MD5}$  中有 4 轮处理过程，每轮又对缓冲区 ABCD 进行 16 步迭代运算，每一步的运算形式为（见图1.3）：

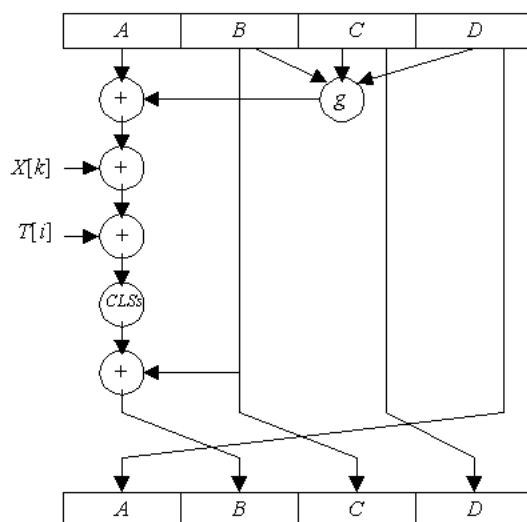


图 1.3: 压缩函数中的一部迭代示意图

其中 a、b、c、d 为缓冲区的四个字，运算完成后再右循环一个字，即得这一步迭代的输出。g 是基本逻辑函数 F、G、H、I 之一。 $CLS_s$  是 32 位存数左循环移 s 位，s 的取值将

在后续的代码中展示。 $T[i]$  为表  $T$  中的第  $i$  个字,  $+$  为模 232 加法。 $X[k] = M[q*16+k]$ , 即消息第  $q$  个分组中的第  $k$  个字 ( $k = 1, \dots, 16$ )。四轮处理过程中, 每轮以不同的次序使用 16 个字, 其中在第一轮以字的初始次序使用。第二轮到第四轮, 分别对字的次序  $i$  做置换后得到一个新次序, 然后以新次序使用 16 个字。三个置换分别为:

$$\rho_2(i) = (1 + 5i) \bmod 16$$

$$\rho_3(i) = (5 + 3i) \bmod 16$$

$$\rho_4(i) = 7i \bmod 16$$

4 轮处理过程分别使用不同的基本逻辑函数  $F$ 、 $G$ 、 $H$ 、 $I$ , 每个逻辑函数的输入为 3 个 32 比特的字, 输出是一个 32 比特的字, 其中的运算为逐比特的逻辑运算, 即输出的第  $n$  个比特是三个输入的第  $n$  个比特的函数, 函数的定义在后续代码中展示。

## 第二章 MD5 实现

### 2.1 定义 MD5 类

这里我们定义一个 MD5 类以便于实现我们的加解密操作，代码如下：

```
1 // 定义 MD5 类
2 class MD5
3 {
4 public:
5     MD5();
6     MD5(const void* input, size_t length);
7     MD5(const string& str);
8     void update(const void* input, size_t length);
9     void update(const string& str);
10    const byte* digest();
11    string to_string();
12    void reset();
13
14 private:
15     void update(const byte* input, size_t length);
16     void final();
17     void transform(const byte block[64]);
18     void encode(const uint32* input, byte* output, size_t
19 length);
20     void decode(const byte* input, uint32* output, size_t
21 length);
22     string bytes_to_string(const byte* input, size_t length);
23
24     MD5(const MD5&);
25     MD5& operator = (const MD5&);
```

```
25 private:
26     uint32 _state[4];           // 四个32bit寄存器缓冲区
27     uint32 _count[2];          // 消息被填充前的长度，小端存储，共
    64bit
28     byte _buffer[64];          // 输入数组
29     byte _digest[16];           // 摘要（输出结果），固定为128比特
30     bool _finished;             // 判断运算是否结束
31
32     static const byte PADDING[64];
33     static const char HEX[16];
34     enum { BUFFER_SIZE = 1024 };
35
36 };
```

## 2.2 实现 MD5 类

实现 MD5 类是我们代码的主体部分，这里我们分成几部分进行说明：

### 2.2.1 $CLR_s$ 循环左移位数

在压缩函数中需要进行循环左移  $s$  位的操作，这里我们使用宏定义表示了其在对应轮数  $s$  的值：

```
1 // 定义压缩函数每轮循环左移的位数
2 // 每一轮循环左移16次，这16次按照重复的4个数字选取左移位数
3 // 如第一轮为
4 // 7 12 17 22
5 // 7 12 17 22
6 // 7 12 17 22
7 // 7 12 17 22
8
9 #define S_11 7
10 #define S_12 12
11 #define S_13 17
12 #define S_14 22
13
```



```
14 #define S_21 5
15 #define S_22 9
16 #define S_23 14
17 #define S_24 20
18
19 #define S_31 4
20 #define S_32 11
21 #define S_33 16
22 #define S_34 23
23
24 #define S_41 6
25 #define S_42 10
26 #define S_43 15
27 #define S_44 21
```

## 2.2.2 基本逻辑函数 F、G、H、I

四个消息分组中的 B、C、D 分组需要使用基本逻辑函数：

```
1 //MD5 分组处理过程中所用的函数
2 #define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
3 #define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
4 #define H(x, y, z) ((x) ^ (y) ^ (z))
5 #define I(x, y, z) ((y) ^ ((x) | (~z)))
```

## 2.2.3 每轮的处理函数

这里我们定义了四轮中每一轮的基本处理函数，因为每一轮都要经过 16 次压缩函数的处理，因此，这里的没一个处理函数都要操作 16 次。

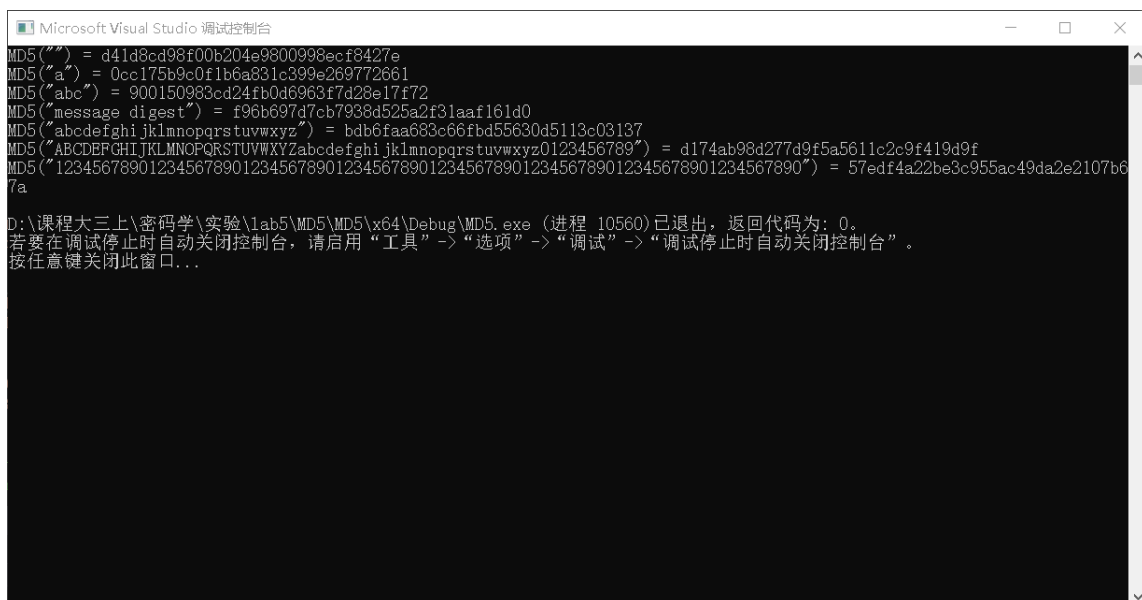
```
1 #define FF(a, b, c, d, x, s, ac) { \
2     (a) += F((b), (c), (d)) + (x) + ac; \
3     (a) = ROTATE_LEFT((a), (s)); \
4     (a) += (b); \
5 }
6 #define GG(a, b, c, d, x, s, ac) { \
```

```
7   (a) += G ((b), (c), (d)) + (x) + ac; \
8   (a) = ROTATE_LEFT ((a), (s)); \
9   (a) += (b); \
10  }
11  #define HH(a, b, c, d, x, s, ac) { \
12      (a) += H ((b), (c), (d)) + (x) + ac; \
13      (a) = ROTATE_LEFT ((a), (s)); \
14      (a) += (b); \
15  }
16  #define II(a, b, c, d, x, s, ac) { \
17      (a) += I ((b), (c), (d)) + (x) + ac; \
18      (a) = ROTATE_LEFT ((a), (s)); \
19      (a) += (b); \
20  }
```

## 第三章 实验过程

### 3.1 测试数据

我们将文档中提供的数据进行测试，得到如下结果3.1：



```
Microsoft Visual Studio 调试控制台
MD5("") = d41d8cd98f00b204e9800998ecf8427e
MD5("a") = 0cc175b9c0f1b6a831c399e269772661
MD5("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5("abcdefghijklmnopqrstuvwxyz") = bdb6faa683c66fbd55630d5113c03137
MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = d174ab98d277d9f5a5611c2c9f419d9f
MD5("123456789012345678901234567890123456789012345678901234567890") = 57edf4a22be3c955ac49da2e2107b67a

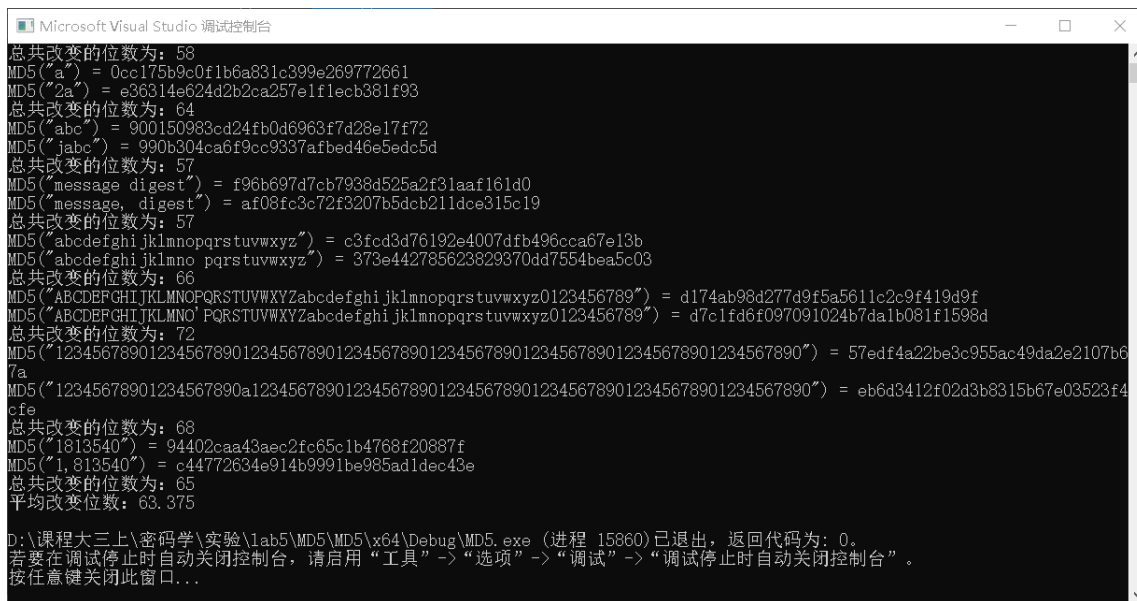
D:\课程大三上\密码学\实验\lab5\MD5\MD5\x64\Debug\MD5.exe (进程 10560) 已退出，返回代码为: 0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

图 3.1: 数据测试结果

可以看到，程序运行成功运行出了正确的结果。

### 3.2 雪崩效应验证

我们对程序稍加修改，尝试实现修改位数的统计，结果如下：



```
Microsoft Visual Studio 调试控制台
总共改变的位数为: 58
MD5("a") = 0cc175b9c0f1b6a831c399e269772661
MD5("2a") = e36314e624d2b2ca257e1f1ecb381f93
总共改变的位数为: 64
MD5("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5("jabc") = 990b304ca6f9cc9337afbed46e5edc5d
总共改变的位数为: 57
MD5("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5("message, digest") = af08fc3c72f3207b5dcb211dce315c19
总共改变的位数为: 57
MD5("abcdefghij klmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5("abcdefghij klmno pqrstuvwxyz") = 373e442785623829370dd7554bea5c03
总共改变的位数为: 66
MD5("ABCDEFGH IJKLMN O PQRSTU VWXYZ abcdefgh i jklmnopq rstuvw xy0123456789") = d174ab98d277d9f5a5611c2c9f419d9f
MD5("ABCDEFGH IJKLMN O PQRSTU VWXYZ abcdefgh i jklmnopq rstuvw xy0123456789") = d7c1fd6f097091024b7da1b081f1598d
总共改变的位数为: 72
MD5("1234567890123456789012345678901234567890123456789012345678901234567890") = 57edf4a22be3c955ac49da2e2107b67a
MD5("12345678901234567890a12345678901234567890123456789012345678901234567890") = eb6d3412f02d3b8315b67e03523f4cfe
总共改变的位数为: 68
MD5("1813540") = 94402caa43aec2fc65c1b4768f20887f
MD5("1,813540") = c44772634e914b9991be985ad1dec43e
总共改变的位数为: 65
平均改变位数: 63.375
D:\课程大三上\密码学\实验\lab5\MD5\MD5\x64\Debug\MD5.exe (进程 15860)已退出, 返回代码为: 0。
若要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

图 3.2: 雪崩效应验证

可以看到, 仅仅是对输入的数据作了微小的修改, 所得到的结果就完全不同。由此可见 MD5 确实具有雪崩效应。