密码学综合设计实验

实验 5：MD5 算法实现

学号：苏煜程

姓名：031803108

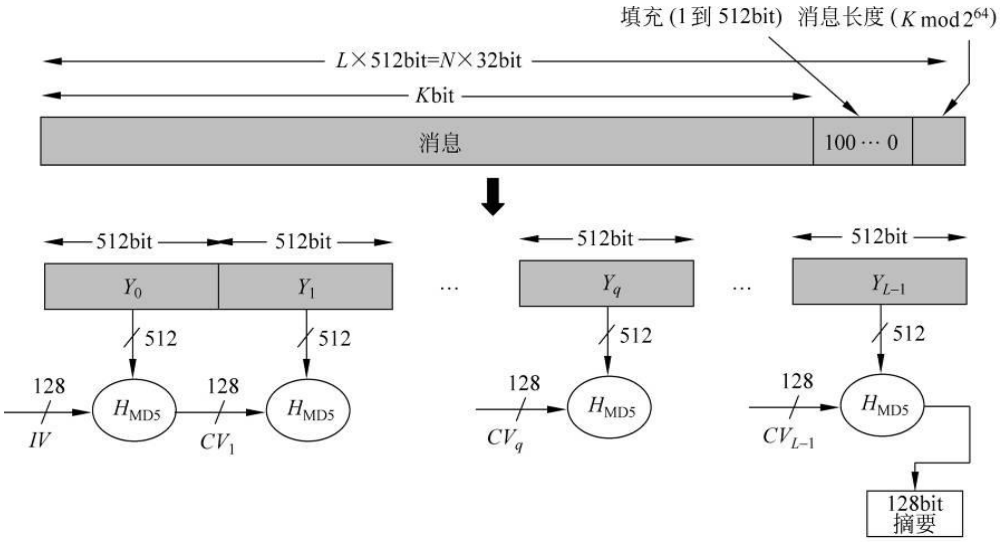
2019 年　10 月　28 日

一、 实验要求

1. 实现 MD5 压缩函数（课本 p169）
   a) 寄存器 ABCD 用的是小端存储方式，建议使用 uint32_t 类型定义
   b) 输入分组 512 比特长，建议用长为 16 的数组表示，成员是联合体(unsigned char[4] 与 uint32_t 联合）
   c) 输出 uint32_t[4]数组
2. 实现 MD5 的分组处理
   a) 算法参见课本 p168
3. 实现消息分组和消息填充(课本 p167)
   a) 消息：不定长 unsigned char[]数组
      分组：512 比特分组，分组格式在第 1 步已给出建议。
   b) 注意，消息长度是小端表示
   c) 分组填充算法见 p167
   d) 输入：消息
      输出：分组数组
4. 消息的 MD5 值计算实现
   a) 输入明文
   b) 输出 MD5 值
   c) 明文是长字符串(超过 80 个字符)
   d) 注意寄存器 ABCD 初值(课本 p167)
5. 附加内容
   a) 实现对一个文件的 MD5 值计算
   b) 输入：文件路径及文件名
      输出：文件的 MD5 值
   c) 注意，文件可能很大，不能把整个文件都载入到内存再运算。需要读一个分组长度，执行一次 MD5 分组处理，直到文件读完。

二、 实验原理

算法的输入为任意长的消息（图中为 K 比特），分为 512 比特长的分组，输出为 128 比特的消息摘要。



处理过程有以下几步：

(1) 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，即第 1 位为 1，其后各位皆为 0。

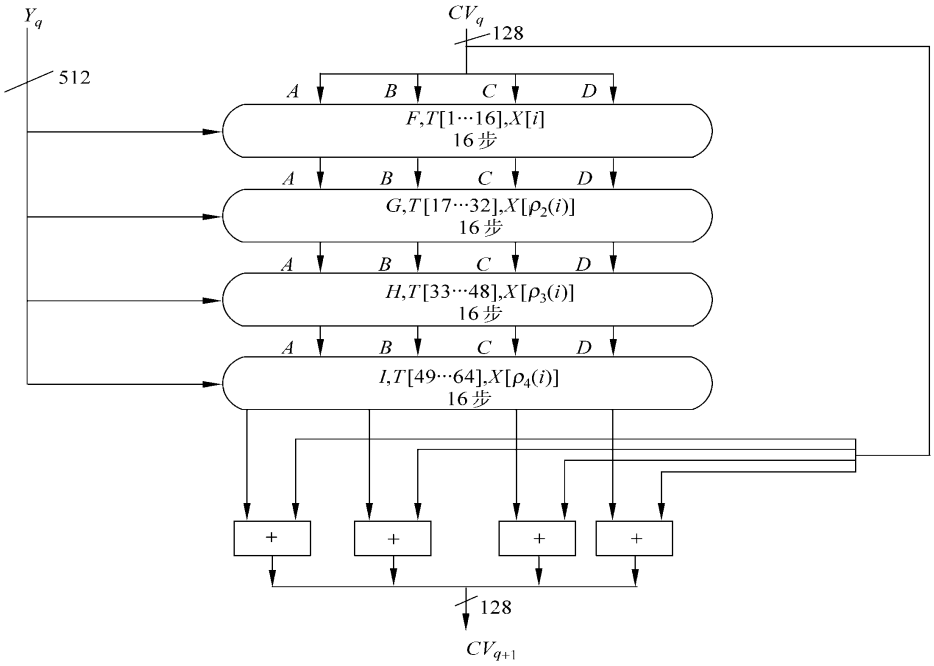
(2) 附加消息的长度。用步骤(1)留出的 64 比特以小端（little-endian）方式来表示消息被填充前的长度。如果消息长度大于2^{64}，则以2^{64}为模数取模。

小端方式是指按数据的最低有效字节（byte）（或最低有效位）优先的顺序存储数据，即将最低有效字节（或最低有效位）存于低地址字节（或位）。相反的存储方式称为大端（big-endian）方式。

前两步执行完后，消息的长度为 512 的倍数（设为 L 倍），则可将消息表示为分组长为 512 的一系列分组Y_0，Y_1，…，Y_{L-1}，而每一分组又可表示为 16 个 32 比特长的字，这样消息中的总字数为 N=L×16，因此消息又可按字表示为 M[0,…,N-1]。

(3) 对 MD 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杂凑值，缓冲区可表示为 4 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。

(4) 以分组为单位对消息进行处理。每一分组Y_q (q=0,…,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：



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Y_q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 中四分之一个元素，分别为 T[1…16]，T[17…32]，T[33…48]，T[49…64]。表 T 有 64 个元素，见表 6-1，第 i 个元素 T[i]为 232×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 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 2^{32} 相加，相加的结果即为压缩函数 HMD5 的输出。

表 6-1　常数表 T

T[1]=D76AA478	T[17]=F61E2562	T[33]=FFFA3942	T[49]=F4292244
T[2]=E8C7B756	T[18]=C040B340	T[34]=8771F681	T[50]=432AFF97
T[3]=242070DB	T[19]=265E5A51	T[35]=699D6122	T[51]=AB9423A7
T[4]=C1BDCEEE	T[20]=E9B6C7AA	T[36]=FDE5380C	T[52]=FC93A039
T[5]=F57C0FAF	T[21]=D62F105D	T[37]=A4BEEA44	T[53]=655B59C3
T[6]=4787C62A	T[22]=02441453	T[38]=4BDECFA9	T[54]=8F0CCC92
T[7]=A8304613	T[23]=D8A1E681	T[39]=F6BB4B60	T[55]=FFEFF47D
T[8]=FD469501	T[24]=E7D3FBC8	T[40]=BEBFBC70	T[56]=85845DD1
T[9]=698098D8	T[25]=21E1CDE6	T[41]=289B7EC6	T[57]=6FA87E4F
T[10]=8B44F7AF	T[26]=C33707D6	T[42]=EAA127FA	T[58]=FE2CE6E0
T[11]=FFFF5BB1	T[27]=F4D50D87	T[43]=D4EF3085	T[59]=A3014314
T[12]=895CD7BE	T[28]=455A14ED	T[44]=04881D05	T[60]=4E0811A1
T[13]=6B901122	T[29]=A9E3E905	T[45]=D9D4D039	T[61]=F7537E82
T[14]=FD987193	T[30]=FCEFA3F8	T[46]=E6DB99E5	T[62]=BD3AF235
T[15]=A679438E	T[31]=676F02D9	T[47]=1FA27CF8	T[63]=2AD7D2BB
T[16]=49B40821	T[32]=8D2A4C8A	T[48]=C4AC5665	T[64]=EB86D391

(5) **MD5 的压缩函数**H_{MD5}。其中有 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 6.7）

$$a \leftarrow b + CLS_s(a + g(b,c,d) + X[k] + T[i])$$

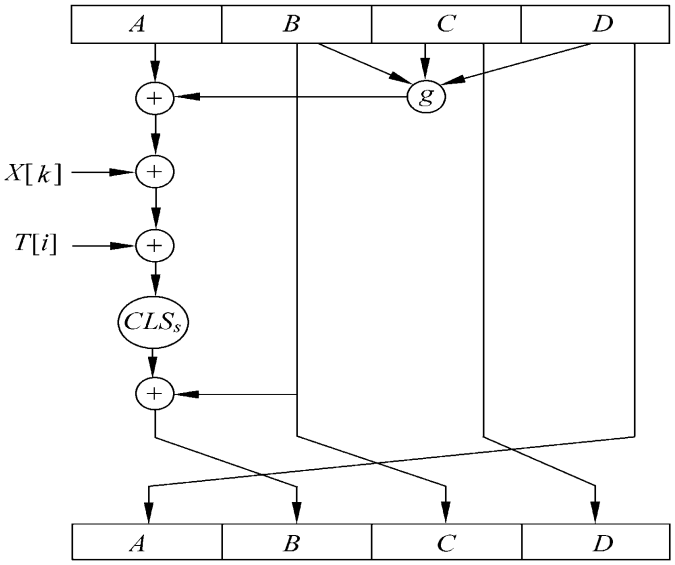


图 6.7 压缩函数中的一步迭代示意图

其中 a、b、c、d 为缓冲区的 4 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是左循环移 s 位，s 的取值由表 6.2 给出。T[i]为表 T 中的第 i 个字，+ 为模 2^{32} 加法。X[k]=M[q×16+k]，即消息第 q 个分组中的第 k 个字（k=1,…,16）。4 轮处理过程中，每轮以不同的次序使用 16 个字，其中在第 1 轮以字的初始次序使用。第 2 轮到第 4 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。3 个置换分别为

$$\rho_2(i) = (1 + 5i)\ mod\ 16$$
$$\rho_3(i) = (5 + 3i)\ mod\ 16$$
$$\rho_4(i) = 7i\ mod\ 16$$

步数 / 轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
2	5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
3	4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
4	6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

表 6.2 压缩函数每步左循环移位的位数

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 3 个输入的第 n 个比特的函数，函数的定义由表 6.3 给出，其中∧，∨，－，⊕分别是逻辑与、逻辑或、逻辑非和异或运算，表 6.4 是四个函数的真值表。

轮数	基本逻辑函数	$g(b,c,d)$
1	$F(b,c,d)$	$(b \wedge c) \vee (\bar{b} \wedge d)$
2	$G(b,c\,d)$	$(b \wedge d) \vee (c \wedge \bar{d})$
3	$H(b,c,d)$	$b \oplus c \oplus d$
4	$I(b,c,d)$	$c \oplus (b \vee \bar{d})$

表 6.3 基本逻辑函数的定义

b	c	d	F	G	H	I	b	c	d	F	G	H	I
0	0	0	0	0	0	1	1	0	0	0	0	1	1
0	0	1	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	1	1	0	1	1	0	0
0	1	1	1	0	0	1	1	1	1	1	1	1	0

表 6.4 基本逻辑函数的真值表

(6) 输出。消息的 L 个分组都被处理完后，最后一个H_{MD5}的输出即为产生的消息摘要。

步骤(3)到步骤(6)的处理过程可总结如下：

CV_0 = IV；

CV_{q+1} = CV_q+ $RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]]$；

MD = CV_L

其中 IV 是步骤(3)所取的缓冲区 ABCD 的初值，Y_q 是消息的第 q 个 512 比特长的分组，L 是消息经过步骤(1)和步骤(2)处理后的分组数，CV_q 为处理消息的第 q 个分组时输入的链接变量（即前一个压缩函数的输出），RF_x 为使用基本逻辑函数 x 的轮函数，+ 为对应字的模 2^{32}加法，MD 为最终的哈希值。

软件系统设计

附加位填充

```
def init_mess(message):
    global A
    global B
    global C
    global D
    A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)
    length = struct.pack('<Q', len(message)*8)
    while len(message) > 64:
        solve(message[:64])
        message = message[64:]
    message += '\x80'
    message += '\x00' * (56 - len(message) % 64)
    message += length
    solve(message[:64])
```

初始化链接变量

```
A, B, C, D = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)
```

循环左移

```
lrot = lambda x,n: (x << n)|(x >> 32- n)
```

四轮 16 步迭代

```
def solve(chunk):
    global A
    global B
    global C
    global D
    w = list(struct.unpack('<' + 'I' * 16, chunk))
    a, b, c, d = A, B, C, D
    for i in range(64):
        if i < 16:
            f = ( b & c)|((~b) & d)
            flag  = i
        elif i < 32:
            f = (b & d)|(c & (~d))
            flag = (5 * i +1) %16
        elif i < 48:
            f = (b ^ c ^ d)
            flag  = (3 * i + 5)% 16
        else:
            f = c ^(b |(~d))
            flag  = (7 * i ) % 16
        tmp = b + lrot((a + f + k[i] + w[flag])& 0xffffffff,r[i])
```

```
        a, b, c, d = d, tmp & 0xffffffff, b, c
    A = (A + a) & 0xffffffff
    B = (B + b) & 0xffffffff
    C = (C + c) & 0xffffffff
    D = (D + d) & 0xffffffff
```

使用正弦函数产生的位随机数

```
k =  [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in range(64)]
```

重要的实现细节

使用正弦函数产生的位随机数

也就是书本上的**T[i]**

```
k =  [int(math.floor(abs(math.sin(i + 1)) * (2 ** 32))) for i in range(64)]
```

四轮运算

```
for i in range(64):
        if i < 16:
            f = ( b & c)|((~b) & d)
            flag  = i
        elif i < 32:
            f = (b & d)|(c & (~d))
            flag = (5 * i +1) %16
        elif i < 48:
            f = (b ^ c ^ d)
            flag  = (3 * i + 5)% 16
        else:
            f = c ^(b |(~d))
            flag  = (7 * i ) % 16
```

即是对一下公式的实现

```
f = ( b & c)|((~b) & d)
f = (b & d)|(c & (~d))
f = (b ^ c ^ d)
f = c ^(b |(~d))
```

实现效果

对多组字符串进行加密

```
C:\Python27\python.exe D:/build/FZU-IS-404/密码学/MD5/mymd5.py
Please input your message :123456
e10adc3949ba59abbe56e057f20f883e
Please input your message :abcdefg
7ac66c0f148de9519b8bd264312c4d64
Please input your message :I love you
e4f58a805a6e1fd0f6bef58c86f9ceb3
Please input your message :1900
9fdb62f932adf55af2c0e09e55861964
```

总结

简单地实现了 md5 加密算法