实验五 Hash 函数 MD5

一、 实验目的

通过实际编程了解 MD5 算法的过程，加深对 Hash 函数的认识。

二、 实验原理

Hash 函数是将任意长的数字串转换成一个较短的定长输出数字串的函数，输出的结果称为 Hash 值。Hash 函数具有如下特点：

（1） 快速性：对于任意一个输入值 x，由 Hash 函数 $H(x)$，计算 Hash 值 y，即 $y = H(x)$ 是非常容易的。

（2） 单向性：对于任意一个输出值 y，希望反向推出输入值 x，使得 $y = H(x)$，是非常困难的。

（3） 无碰撞性：包括强无碰撞性和弱无碰撞性，一个好的 Hash 函数应该满足强无碰撞性，即找到两个不同的数字串 x 和 y，满足 $H(x) = H(y)$，在计算上是不可能的。

Hash 函数可用于数字签名、消息的完整性检验。消息的来源认证检测等。现在常用的 Hash 算法有 MD5、SHA－1 等。下面从 MD5 入手来介绍 Hash 算法的实现机制。

MD 系列单向散列函数是由 Ron Rivest 设计的，MD5 算法对任意长度的输入值处理后产生 128 位的 Hash 值。MD5 算法的实现步骤如下（见图 5－1）：

在 MD5 算法中，首先需要对信息进行填充，使其字节长度与 448 模 512 同余，即信息的字节长度扩展至 $n*512+448$，n 为一个正整数。填充的方法如下：在信息的后面填充第一位为 1，其余各位均为 0，直到满足上面的条件时才停止用 0 对信息的填充。然后，再在这个结果后面附加一个以 64 位二进制表示的填充前信息长度。经过这两步的处理，现在的信息字节长度为 $n*512+448+64=(n+1)*512$，即长度恰好是 512 的整数倍，这样做的目的是为了满足后面处理中对信息长度的要求。

MD5 中有 A、B、C、D，4 个 32 位被称为链接变量的整数参数，它们的

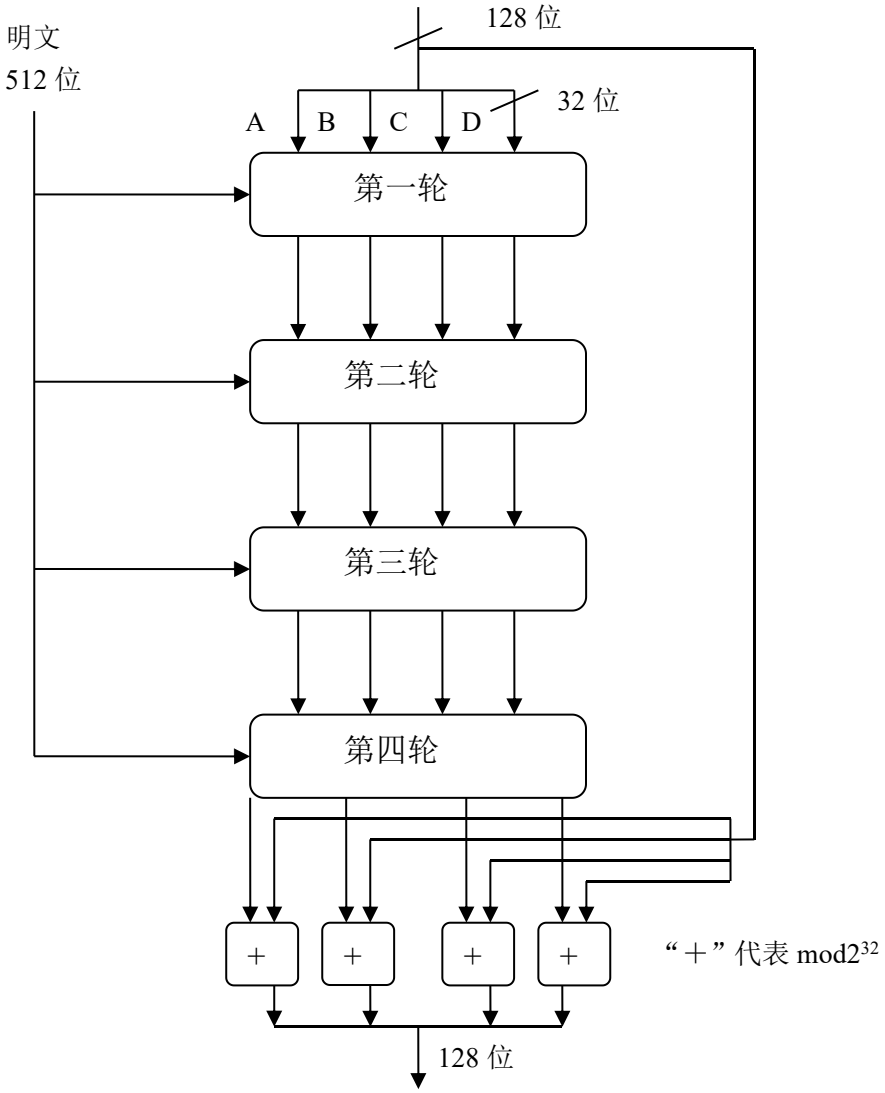

图 5－1 MD5 算法的实现步骤

初始值分别为：

A_0＝0x01234567，B_0＝0x89abcdef,C_0=0xfedcba98,D_0=0x76543210

当设置好这 4 个链接变量后，就开始进入算法的 4 轮循环运算。循环的次数是信息中 512 位信息分组数目。

首先将上面 4 个链接变量复制到变量 A、B、C、D 中，以备后面进行处理。

然后进入主循环，主循环有 4 轮，每轮循环都很相似。第一轮进行 16 次操作，每次操作对 A、B、C、D 中的 3 个做一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组（32 位）和一个常数。再将所得结果向左循环移 S 位，并加上 A、B、C、D 其中之一。最后用该结果取代 A、B、C、D 其中

之一。

以下是每次操作中用到的 4 个非线性函数（每轮一个）。

$$F(B,C,D) = (B \wedge C) \vee (\overline{B} \wedge D)$$

$$G(B,C,D) = (B \wedge D) \vee (C \wedge \overline{D})$$

$$H(B,C,D) = B \oplus C \oplus D$$

$$I(B,C,D) = C \oplus (B \vee \overline{D})$$

MD5 轮主要操作为：

$$a = b + ((a + f(b,c,d) + M + t) <<< s)$$

对应于四轮操作，f 分别取 F，G，H，I；对每一轮的 16 次运算，M 分别取 M_1，M_2，…，M_{16}。对于 4 轮共 64 次运算，t 为给定的一些常数，另外一个常数 $s(i)$ 是 $2^{32} * abs(\sin(i))$ 的整数部分，其中 i＝1，2，…，64。在 $\sin(i)$ 中，i 的单位是弧度，由此构成了 32 位的随机数源 $s(i)$，它消除了输入数据中任何规律性的特征。

对于 4 轮 64 次操作的具体运算，可查阅课本的内容。所有这些操作完成之后，将 A，B，C，D 分别加上 A_0，B_0，C_0，D_0。然后用下一分组数据继续进行运算，最后得到一组 A，B，C，D。把这组数据级联起来，即得到 128 比特的 Hash 结果。

需要说明的是,2004 年 8 月，在 Crypto2004 国际密码学会议上，山东大学王小云教授发现了一种找到 MD5 散列函数的碰撞的方法,即可以找到两个不同的输入 x 和 y，得到相同的 Hash 结果。这一发现意味着采用 MD5 算法的数字签名、完整性检验等信息安全应用系统将不在安全了,这就促使信息安全系统的设计者尽快去寻找和探索新的 Hash 算法。

三、 实验环境

运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境

四、 实验内容和步骤，

1、算法分析

请参照教材内容，分析 MD5 算法实现的每一步原理。

2、算法实现：

利用 Visual C＋＋语言，自己编写 MD5 的实现代码，并检验代码实现的正确性。

3、雪崩效应检验：

尝试对一个长字符串进行 Hash 运算，并获得其运算结果。对该字符串进行轻微的改动，比如增加一个空格或标点，比较 Hash 结果值的改变位数。进行 8 次这样的测试。

五、 实验报告和要求

1、自己编写完整的 MD5 实现代码，并提交程序和程序流程图。

2、对编好的 MD5 算法，测试其雪崩效应，要求给出文本改变前和改变后的 Hash 值，并计算出改变的位数。写出 8 次测试的结果，并计算出平均改变的位数。

附：测试数据

```
tests[] = {
    { "",
      { 0xd4, 0x1d, 0x8c, 0xd9, 0x8f, 0x00, 0xb2, 0x04,
        0xe9, 0x80, 0x09, 0x98, 0xec, 0xf8, 0x42, 0x7e } },
    { "a",
      {0x0c, 0xc1, 0x75, 0xb9, 0xc0, 0xf1, 0xb6, 0xa8,
       0x31, 0xc3, 0x99, 0xe2, 0x69, 0x77, 0x26, 0x61 } },
    { "abc",
      { 0x90, 0x01, 0x50, 0x98, 0x3c, 0xd2, 0x4f, 0xb0,
        0xd6, 0x96, 0x3f, 0x7d, 0x28, 0xe1, 0x7f, 0x72 } },
    { "message digest",
      { 0xf9, 0x6b, 0x69, 0x7d, 0x7c, 0xb7, 0x93, 0x8d,
        0x52, 0x5a, 0x2f, 0x31, 0xaa, 0xf1, 0x61, 0xd0 } },
    { "abcdefghijklmnopqrstuvwxyz",
      { 0xc3, 0xfc, 0xd3, 0xd7, 0x61, 0x92, 0xe4, 0x00,
        0x7d, 0xfb, 0x49, 0x6c, 0xca, 0x67, 0xe1, 0x3b } },
    { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
      { 0xd1, 0x74, 0xab, 0x98, 0xd2, 0x77, 0xd9, 0xf5,
        0xa5, 0x61, 0x1c, 0x2c, 0x9f, 0x41, 0x9d, 0x9f } },
```

```
{ "12345678901234567890123456789012345678901234567890123456789012345678901234567890",
      { 0x57, 0xed, 0xf4, 0xa2, 0x2b, 0xe3, 0xc9, 0x55,
        0xac, 0x49, 0xda, 0x2e, 0x21, 0x07, 0xb6, 0x7a } },

};
```

上面给出的数据是一组测试 MD5 算法是否正确的数据，其本质是一个结构数组，数组中的每一个元素就是一组测试数据，数据结构的定义方式如下：

```
  static const struct {
      char *msg;
      unsigned char hash[16];
  }
```

这个结构表示，每一个数据结构包含两部分的内容
其中第一部分是要进行 HASH 运算的原始明文，第二部分则是其对应的 HASH 值。

大家可以利用这种数据来测试程序编写正确与否。