实验四 Hash 函数 MD5

1611531-信息安全-刘新慧

摘要

通过实际编程了解 MD5 算法的过程，加深对 Hash 函数的认识。

目录

1 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，计算 Hash 值 y，即是非常容易的。

（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，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，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（见表 4 － 1）：

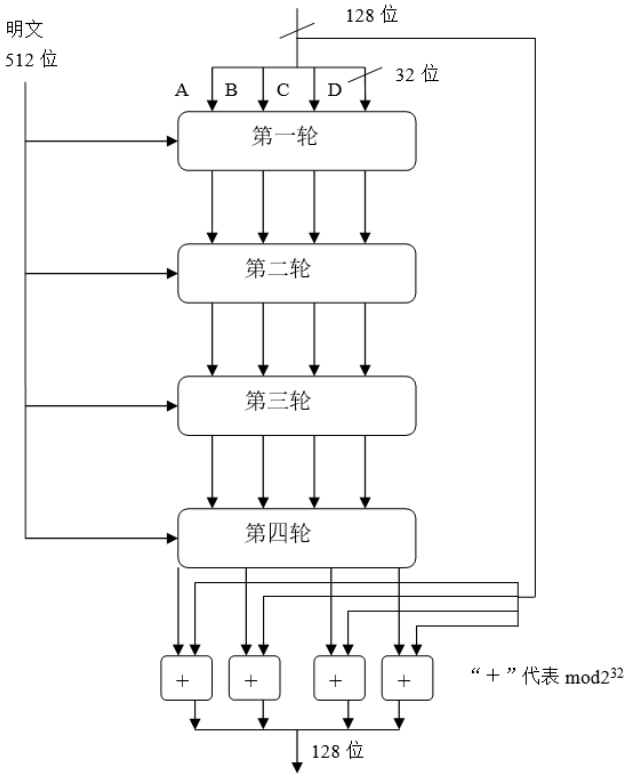


图 1: 表 4-1

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 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，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，它们的初始值分别为：

A0 ＝ 0x01234567，B0 ＝ 0x89abcdef, C0=0xfedcba98, D0=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 M1，M2，…，M16。对于 4 轮共 64 次运算，t 为给定的一些常数，另外一个常数是的整数部分，其中 i = 1，2，…，64。在中，i 的单位是弧度，由此构成了 32 位的随机数源，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 A0，B0，C0，D0。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。

2 实验内容和步骤

1、算法分析

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。

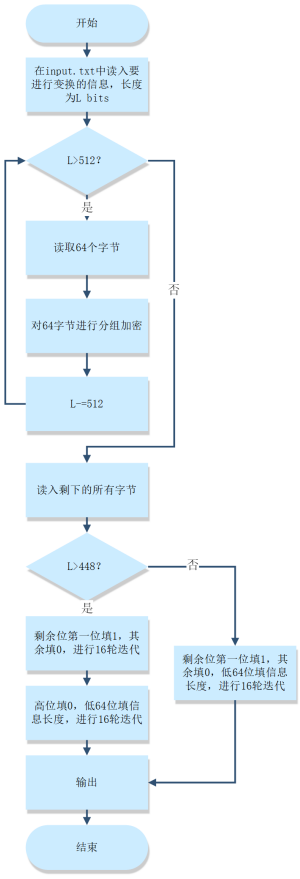
3、雪崩效应检验：

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。

3 实验结果

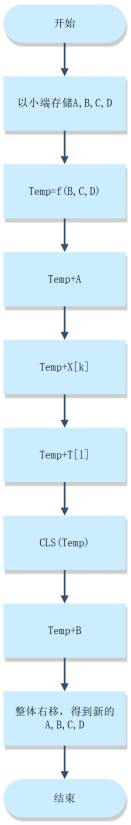
3.1 程序框图

程序框图如下：



整体框图

每一组512bits



f/g/h/i函数

3.2 实现 MD5 编程

运行代码以下为两个结果示例：
当输入的内容为”a” 时，结果如下：

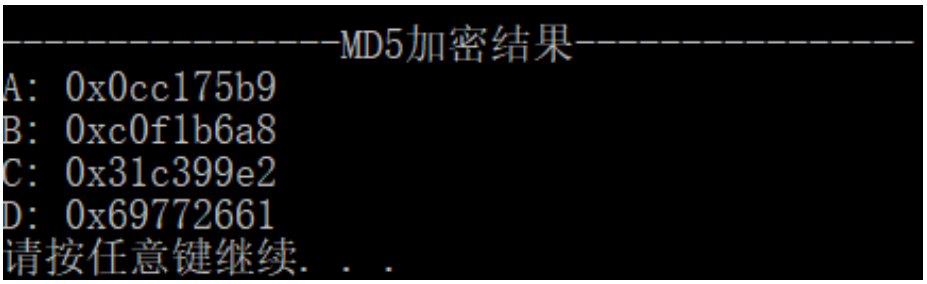


图 2: 结果 1

运行代码,当输入的内容为”ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789” 时，结果如下：

图 3: 结果 2

3.3 雪崩效应

最开始进行 hash 的输入：Hello I am Lethe
hash 结果：0x03319a22f0416ae301a35743bf5d6ba6
(1) 输入修改为：H ello I am Lethe
hash 结果为：0x6947163a0a45ff14decaa73b8efd81b0
改变位数为：64
(2) 输入修改为：He llo I am Lethe
hash 结果为：0x033ab41e2d0c5d8fcf65d214eaa0c836
改变位数为：64
(3) 输入修改为：Hel lo I am Lethe
hash 结果为：0xfab7e9a9cc2b32a14e416748547a1dca
改变位数为：64
(4) 输入修改为：Hell o I am Lethe
hash 结果为：0xf04239ab5c01b72d38eb5222051ae388
改变位数为：60
(5) 输入修改为：Hello I a m Lethe
hash 结果为：0xdb03fd0a6fa73873b63476b23b91c479
改变位数为：67
(6) 输入修改为：Hello I am L ethe
hash 结果为：0x9a128057a978d32be86476a0b64f0157
改变位数为：61
(7) 输入修改为：Hello I am Le the
hash 结果为：0xacb71a03f8e71d32510221fe90f2d556
改变位数为：64
(8) 输入修改为：Hello I am Let he
hash 结果为：0x2c6f9f06bd86f0d1e951ce8c816ca7e8
改变位数为：65
平均结果为：(64+64+64+60+67+61+64+65)/8=63.625≈64 位