南開大學
Nankai University

网络空间安全学院

密码学实验报告

第二次实验：分组密码算法 DES

姓名：魏伯繁

学号：2011395

专业：信息安全

2022 年 11 月 23 日

目录

1 实验内容

1.1 实验目的

通过用 DES 算法对实际的数据进行加密和解密来深刻了解 DES 的运行原理。

1.2 实验内容

(1) 分别实现 DES 的加密和解密，提交程序代码和执行结果。
(2) 在检验雪崩效应中，要求至少改变明文和密文中各八位，给出统计结果并计算出平均值。

2 实验原理

2.1 分组密码

分组密码是一种对称密码体制，其特点是在明文加密和密文解密的过程中，信息都是按照固定长度分组后进行处理的。在分组密码的发展历史中，曾出现了许多优秀的算法，包括 DES，IDEA，AES，Safer ++等等。下面以 DES 算法为例介绍分组密码算法的实现机制。

2.2 DES 密码原理

2.2.1 DES 原理概述

DES 算法将明文分成 64 位大小的众多数据块，即分组长度为 64 位。同时用 56 位密钥对 64 位明文信息加密，最终形成 64 位的密文。如果明文长度不足 64 位，即将其扩展为 64 位（如补零等方法)。具体加密过程首先是将输入的数据进行初始置换（IP)，即将明文 M 中数据的排列顺序按一定的规则重新排列，生成新的数据序列，以打乱原来的次序。然后将变换后的数据平分成左右两部分，左边记为 L0，右边记为 R0，然后对 R0 实行在子密钥（由加密密钥产生）控制下的变换 f，结果记为 f(R0，K1)，再与 L0 做逐位异或运算，其结果记为 R1，R0 则作为下一轮的 L1。如此循环 16 轮，最后得到 L16、R16，再对 L16、R16 实行逆初始置换 IP - 1，即可得到加密数据。解密过程与此类似，不同之处仅在于子密钥的使用顺序正好相反。

2.2.2 DES 加密函数

DES 的加密算法包括 3 个基本函数:

1. 初始置换 IP

它的作用是把输入的 64 位数据块的排列顺序打乱，每位数据按照下面的置换规则重新排列，即将第 58 位换到第一位，第 50 位换打第 2 位，…，依次类推。置换后的 64 位输出分为 L0 、R0（左、右）两部分，每部分分别为 32 位。

R0 和 K1 经过 f（R0，K1）变换后的输出结果，再和 L0 进行异或运算，输出结果位 R1，R0 则赋给 L1。L1 和 R1 同样再做类似运算生成 L2 和 R2，…，经过 16 次运算后生成 L16 和 R16。

2. f 函数

f 函数是多个置换函数和替代函数的组合函数，它将 32 位比特的输入变换为 32 位的输出，如图 1 - 2 所示。Ri 经过扩展运算 E 变换后扩展为 48 位的 E（Ri)，与进行异或运算后输出的结果分成 8 组，每组 6 比特。每一组再经过一个 S 盒（共 8 个 S 盒）运算转换为 4 位，8 个 4 位合并为 32 位后再经过 P 变换输出为 32 位的。其中，扩展运算 E 与置换 P 主要作用是增加算法的扩散效果。

3. 逆初始置换 IP - 1 它将 L16 和 R16 作为输入，进行逆初始置换得到密文输出。逆初始置换是初始置换的逆运算。

DES 的加密算法中除了上面介绍的 3 个基本函数，还有一个非常重要的功能模块，即子密钥的生成模块，具体子密钥的产生流程图如图 1 - 3 所示。输入的初始密钥值为 64 位，但 DES 算法规定，其中第 8、16、…、64 位为奇偶校验位，不参予 DES 的运算。所以，实际可用位数只有 56 位，经过缩小选择位表 1（表 1 - 2）即密钥置换 PC-1 的变换后，初始密钥的位数由 64 位变成了 56 位，将其平分位两部分 C0，D0。然后分别进行第一次循环左移，得到 C1 和 D1，将 C1（28 位）、D1（28 位）合并后得到 56 位的输出结果，再经过压缩置换 PC-2(表 1 - 3)，从而得到了密钥 K1（48 位）。依次类推，便可得到 K2、…、K16。

2.3 DES 流程

在了解了几个重要的加密函数之后，我们对 DES 六层进行完整叙述，并对其关键步骤予以说明。

首先,给定明文,通过一个固定的初始置换 IP 来重排输入明文块 P 中的比特,得到比特串 P0=IP(P)=L0R0，这里 L0 和 R0 分别是 P0 的前 32 比特和后 32 比特，下图展示了具体的 ip 置换表

```
int IP[] = { 58, 50, 42, 34, 26, 18, 10, 2,
             60, 52, 44, 36, 28, 20, 12, 4,
             62, 54, 46, 38, 30, 22, 14, 6,
             64, 56, 48, 40, 32, 24, 16, 8,
             57, 49, 41, 33, 25, 17, 9,  1,
             59, 51, 43, 35, 27, 19, 11, 3,
             61, 53, 45, 37, 29, 21, 13, 5,
             63, 55, 47, 39, 31, 23, 15, 7 };//初始置换IP
```

图 2.1: ip 置换表

接下来，保存右侧的 32 比特以备他用，然后进入轮函数，下图展示了轮函数的示意图，首先先对右侧的 32 比特做一个 E 扩展将 32 位的密码扩展为 48 位。随后与密钥生成部分提供的密钥进行异或运算。

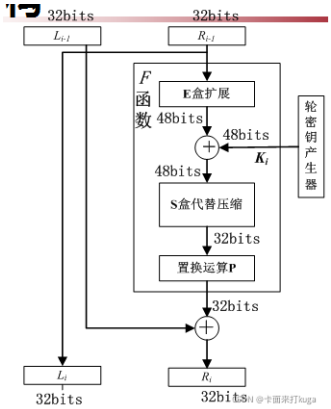

图 2.2: 轮函数示意图

E 扩展表如下图所示

```
int E[] = { 32,  1,  2,  3,  4,  5,
            4,  5,  6,  7,  8,  9,
            8,  9, 10, 11, 12, 13,
           12, 13, 14, 15, 16, 17,
           16, 17, 18, 19, 20, 21,
           20, 21, 22, 23, 24, 25,
           24, 25, 26, 27, 28, 29,
           28, 29, 30, 31, 32,  1 };//扩展置换表E
```

图 2.3: E 扩展表

下图为 E 盒的效果图，完成 32 位到 48 位的映射

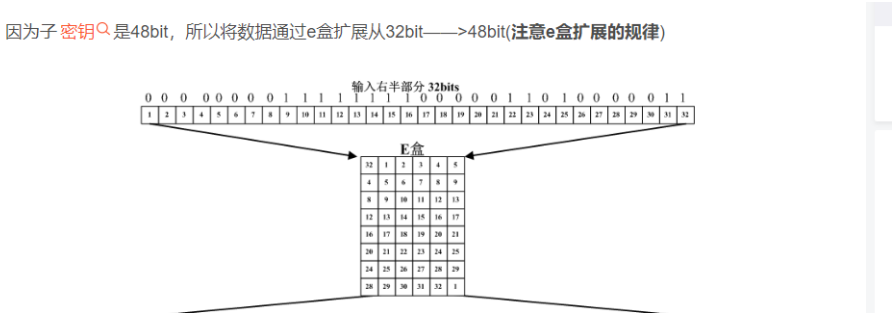

图 2.4: 密钥产生

异或后的 48 位数字将经过 S 盒重新变成 32 位，具体方式是将 48 位数分成 8 组，每组 6 位数，其中第一位和第六位用来选择行，中间四位用来选择列来精准定位 S 盒的一个转换结果并将转换结果编程 4 位二进制数。

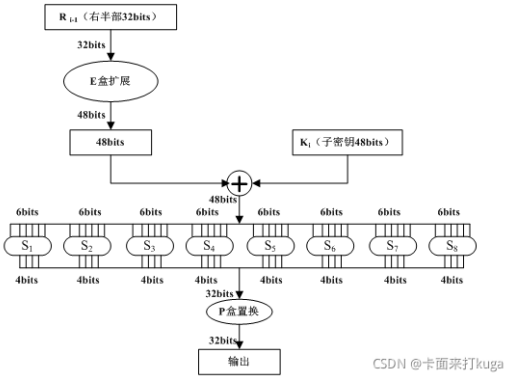

图 2.5: S 盒

S 盒运算之后再进行 P 置换，P 置换表如下图所示

```
//P置换表
int P[] = { 16,  7, 20, 21,
           29, 12, 28, 17,
            1, 15, 23, 26,
            5, 18, 31, 10,
            2,  8, 24, 14,
           32, 27,  3,  9,
           19, 13, 30,  6,
           22, 11,  4, 25 };
```

图 2.6: P 置换表

轮函数的最后一步是将左侧的 32 位与经过轮函数的 32 位进行异或运行形成新的 32 位数，而新的左侧的 32 位数则由一开始没有进入轮函数而被保存下来的 32 位右侧的数来决定的

这样的流程将被执行 16 轮，每一轮密钥产生部分都会提供新的密钥，当 16 轮轮函数执行完毕后会退出，将左右 32 位交换再进行一个初始置换的逆置换。

至此，明文的部分就完成了，下面我们来阐述密钥是如何做转换的，首先输入的密钥是 64 位密钥，其中包含 8 位奇偶校验位，经过一个 PC1 置换变为 56 位并分为左右 28 位，之后分别根据移位表进行左移位，移位后的 56 位密钥进入 PC2 置换选择 48 位与明文中的右侧 48 位进行异或运算，具体的流程如下图所示

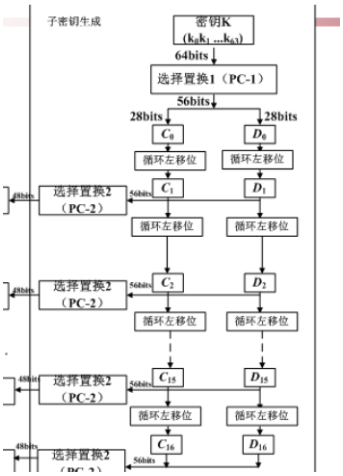

图 2.7: 密钥产生

左循环移位的效果图如下图所示，每一轮根据轮数的不同进行不同的移位位数

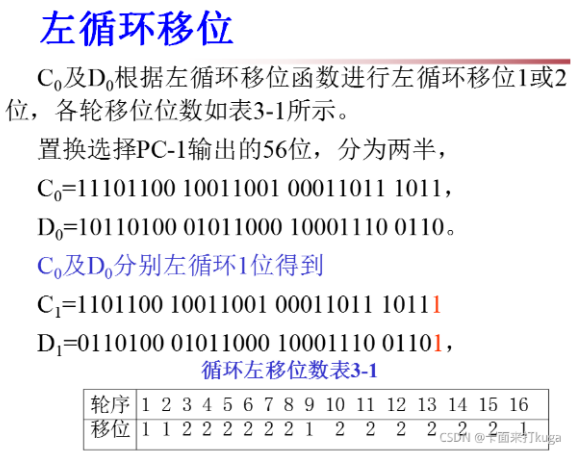

图 2.8: P 置换表

并且，DES 算法有一个非常重要的性质就是他的加密和解密的顺序是完全一致的，唯一有区别的地方就是使用子密钥的顺序是相反的，也就是说，当我们加密时可以顺序的生成 16 个子密钥，但是当我们解密时则需要首先算出 16 个子密钥并以逆序使用它们。

2.4 DES 流程图

根据上述流程的叙述，我们基本可以给出 DES 的流程图以方便理解、展示 DES 的整体过程

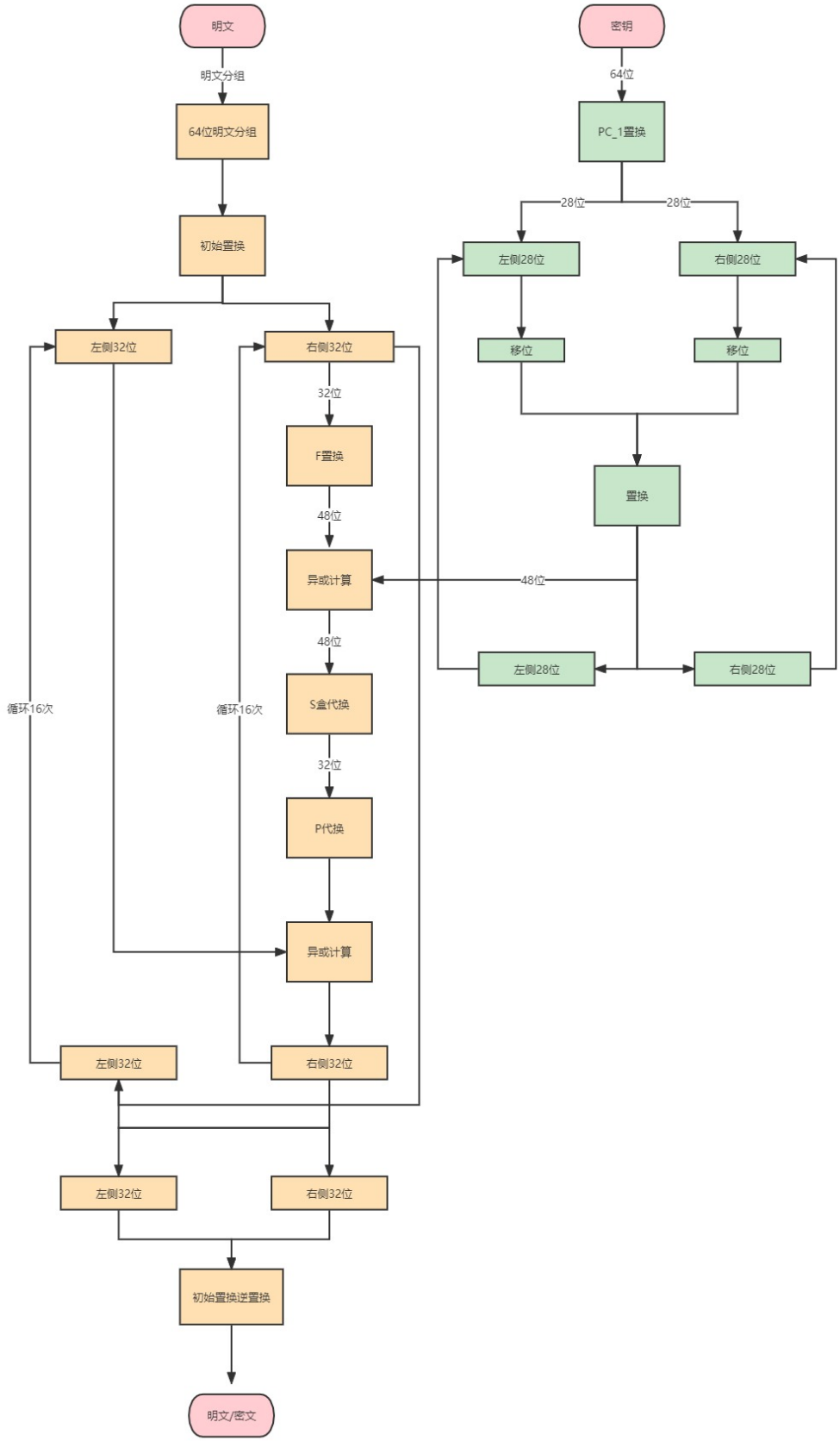
明文
明文分组
64位明文分组
初始置换
左侧32位
右侧32位
32位
F置换
48位
异或计算
48位
S盒代换
32位
P代换
异或计算
左侧32位
右侧32位
左侧32位
右侧32位
初始置换逆置换
明文/密文
循环16次
循环16次
密钥
64位
PC_1置换
28位
28位
左侧28位
右侧28位
移位
移位
置换
48位
左侧28位
右侧28位

图 2.9: DES 流程图

3 核心代码展示

3.1 置换操作

在 DES 中有很多置换操作，其实这些操作的流程是差不多的，所以在此处只给出明文的初始置换代码

```

void initialChange() {
    char temp[65];
    for (int i = 1; i < 65; i++) {
        temp[i] = message[IP[i-1]];
    }
    for (int i = 1; i < 65; i++) {
        message[i] = temp[i];
    }
}

}
```

另外一个比较重要的操作就是 S 盒代换，在这里也是给出 S 盒代换的代码

```

void SBoxChange() {
    for (int i = 1; i <= 8; i++) {
        char temp[7];
        for (int j = 1; j <= 6; j++) {
            temp[j] = Right48[(i - 1) * 6 + j];
        }
        int r = getRow(temp[1], temp[6]);
        int c = getColumn(temp[2], temp[3], temp[4], temp[5]);
        int result = S_BOX[i - 1][r][c];
        string s = int2string(result);
        for (int k = 1; k <= 4; k++) {
            Right[(i - 1) * 4 + k] = s[k-1];
        }
    }
}

```

至此，对明文的操作中两个最重要的函数就已经实现完成了，还有一个比较有特色的函数就是在密钥加密过程中使用的移位函数，移位函数的具体实现如下：

```
1
2       //密钥左移函数
3   void shiftKey(int n) {
4       char temp[29];
5       for (int i = 1; i < 29; i++) {
6           temp[i] = LeftKey[i];
7       }
8       for (int i = 1; i < 29; i++) {
9           if (i + n <= 28) {
10              LeftKey[i] = temp[i + n];
11          }
12          else {
13              LeftKey[i] = temp[i + n - 28];
14          }
15      }
16
17      for (int i = 1; i < 29; i++) {
18          temp[i] = RightKey[i];
19      }
20      for (int i = 1; i < 29; i++) {
21          if (i + n <= 28) {
22              RightKey[i] = temp[i + n];
23          }
24          else {
25              RightKey[i] = temp[i + n - 28];
26          }
27      }
28  }
29
```

4 结果展示

4.1 加密结果展示

下图展示了输入、输出以及加密结果

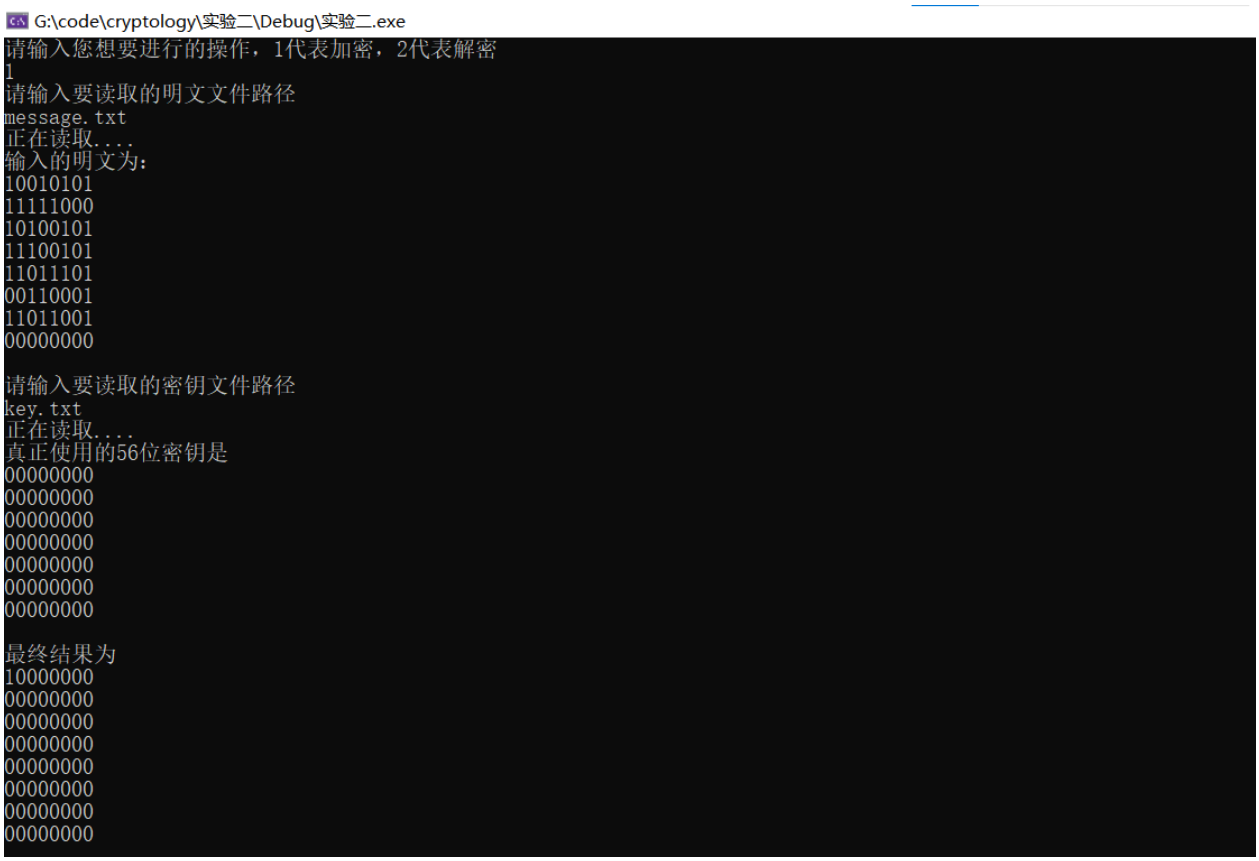

图 4.10: 加密效果

下图展示了测试样例中的加密结果，可以发现与预期结果完全一致，代表加密成功

```
{ 2, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
            { 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00 },
            { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
```

图 4.11: 加密效果

4.2 解密结果展示

下面验证解密操作，下图是解密的效果图

```
请输入您想要进行的操作，1代表加密，2代表解密
2
请输入要读取的明文文件路径
message.txt
正在读取....
输入的密文为：
10000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000

请输入要读取的密钥文件路径
key.txt
正在读取....
真正使用的56位密钥是
00000000
00000000
00000000
00000000
00000000
00000000
00000000

最终结果为
10010101
11111000
10100101
11100101
11011101
00110001
11011001
00000000
```

图 4.12: 加密效果

与测试样例中所给的结果比较发现一致，解密操作成功

```
{ 2, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
            { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
            { 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00 } },
```

图 4.13: 加密效果

5 雪崩效应

雪崩效应（avalanche effect），密码学术语，指加密算法（尤其是块密码和加密散列函数）的一种理想属性。雪崩效应是指当输入发生最微小的改变（例如，反转一个二进制位）时，也会导致输出的不可区分性改变（输出中每个二进制位有 50% 的概率发生反转）。

使用测试样例中加密组的第二个测试样例作为明文原本，每次改变其 1 比特，统计加密结果的比特变换位数，最终统计结果如下表所示：

	A	B	C	D	E
1	改变明文	密文改变数位	改变密钥	密文改变位数	
2	第一次	36	第一次	40	
3	第二次	25	第二次	40	
4	第三次	33	第三次	32	
5	第四次	35	第四次	33	
6	第五次	37	第五次	32	
7	第六次	40	第六次	35	
8	第七次	34	第七次	32	
9	第八次	32	第八次	35	
10	平均	34	平均	34.875	

图 5.14: 雪崩效应统计表

根据统计结果，我们可以较为清晰的看出，雪崩效应在 DES 加密算法中是十分成功改的，64 位的明文平均每次有 34 位左右在明文或者密钥改变 1 比特的情况下改变，这说明大约每一比特改变的概率均在 50% 上下，这让破解明文的难度大大提升了。