DES 算法的加密与解密

1611532 刘一静 信息安全

实验要求：

(1) 分别实现 DES 的加密和解密，提交程序代码和执行结果。

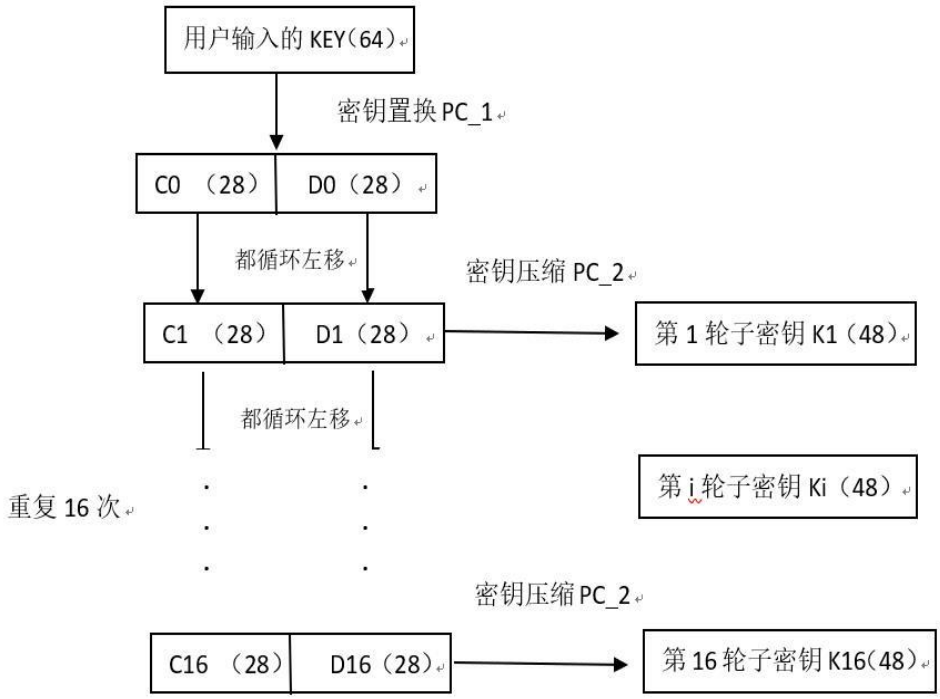
(2) 在检验雪崩效应中，要求至少改变明文和密文中各八位，给出统计结果并计算出平均值。

目录

一、DES 加密

步骤 1：创建 16 个子密钥，每个子密钥长度为 48 位。

流程图如下：



文字表示为：

1、用户输出的密钥是 64 位的，根据密钥置换表 PC-1，将 64 位变成 56 位密钥。（去掉了奇偶校验位）

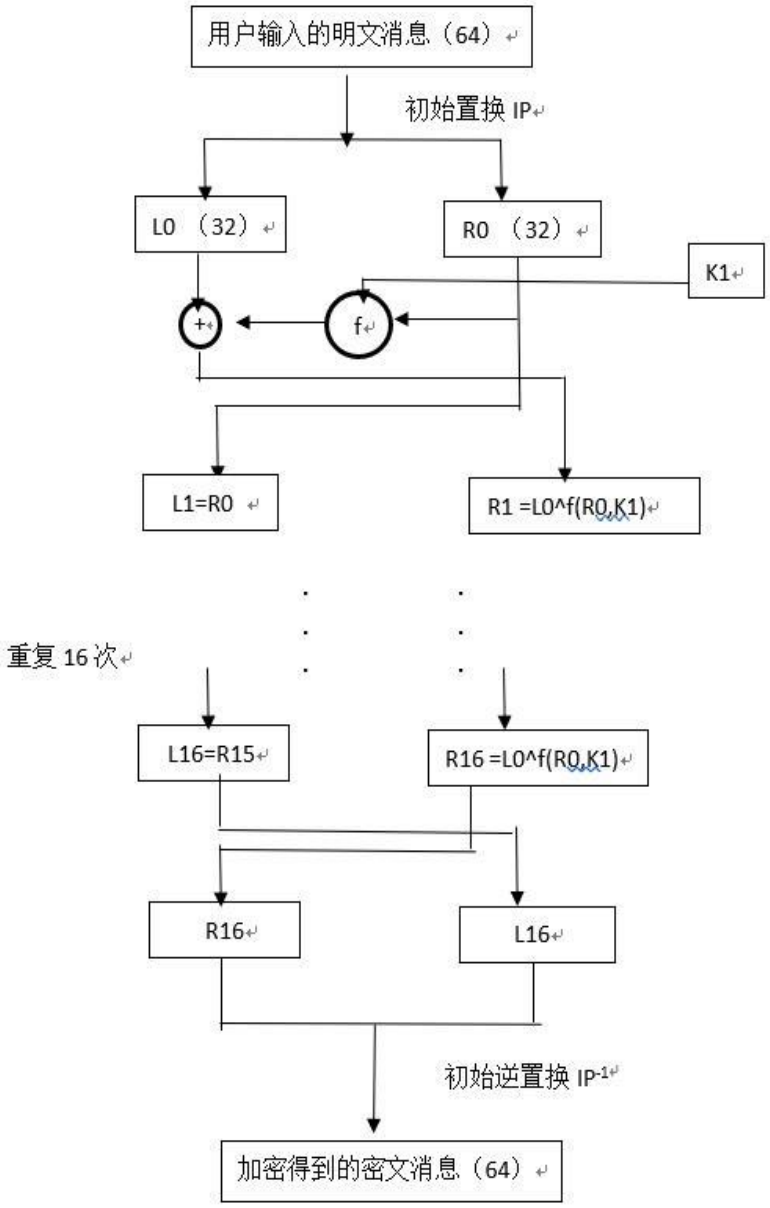
2、将 PC-1 置换得到的 56 位密钥，分为前 28 位 C0 和后 28 位 D0，分别对它们进行循环左移，C0 左移得到 C1，D0 左移得到 D1。

3、将 C1 和 D1 合并成 56 位，然后通过 PC-2 表进行压缩置换，得到当前这一轮的 48 位子密钥 K1。

4、然后对 C1 和 D1 进行左移和压缩置换，获取下一轮的子密钥……一共进行 16 轮，得到 16 个 48 位的子密钥。

步骤 2：对每个 64 位数据块进行编码。

流程图如下：



文字描述如下：

1、存在消息数据 M 的 64 位的初始置换 IP_1。

2、接下来将置换块 IP 划分为 32 位的左半部分 L0 和 32 位的右半部分 R0。

3、现在进行 16 次迭代：一个 32 位的数据块（Ri）和一个 48 位的密钥 Ki ，在 f 函数作用下产生一个 32 位的块。这个块再和另个初始的 32 位数据块（Li）进行异或作为下一轮的 Li+1，Ri+1 为 Li。公式表示为：

$$Ln = Rn-1$$

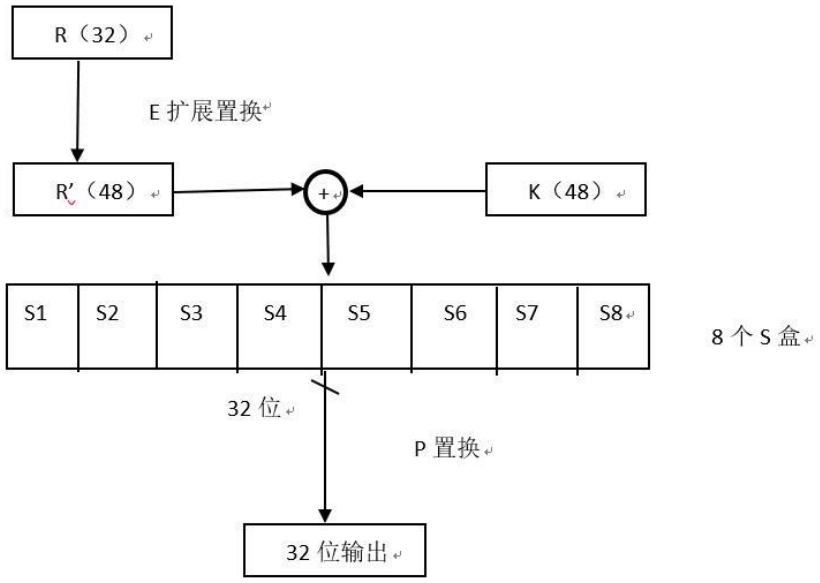
$$Rn = Ln-1 + f(Rn-1,Kn)$$

4、将最后一轮得到的 L16，R16 数据块左右反转，组合为 64 位。

5、对这 64 位数据进行初始置换的逆置换 IP_2,得到最终加密之后的结果。

下面具体分析 f 函数的实现：

流程图如下:



文字描述如下：

f 函数接受两个输入：32 位的数据和 48 位的子密钥。

1、通过表 E 进行扩展置换，将输入的 32 位数据扩展为 48 位。

2、将扩展后的 48 位数据与 48 位的子密钥进行异或运算；

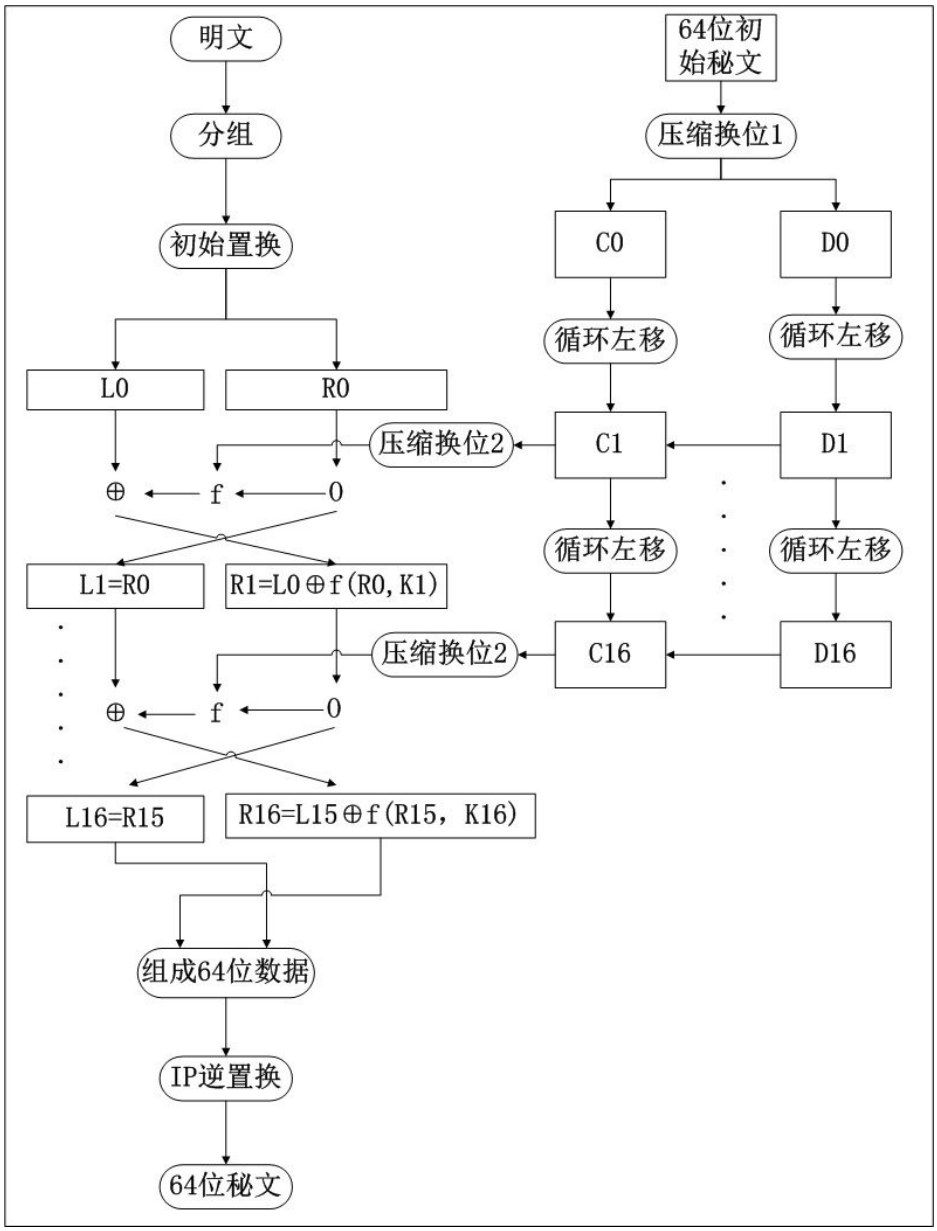
3、将异或得到的 48 位数据分成 8 个 6 位的块，每一个块通过对应的一个 S 表产生一个 4 位的输出。

4、把通过 S 表置换得到的 8 个 4 位连在一起，形成一个 32 位的数据。

5、将该 32 位数据通过表 P 进行置换置换后得到一个仍然是 32 位的结果数据，这就是 f(R, K)函数的输出。

至此 DES 的加密逻辑解释完毕。

完整流程图如下：



二、DES 加密算法的实现

使用 visual studio2015 进行 C++编程。使用 bitset 容器（bool 型数组）存储过程中的 01 比特串（明文，密钥，密文等）。

（一）关键函数分析

1、置换函数

原理：根据置换表进行置换。以密钥生成过程中的初始置换 PC（64->56）为例：PC 置换表为：

```
int PC_1[56] = {
 57,  49,  41,  33,  25,  17,  9,
 1,   58,  50,  42,  34,  26,  18,
 10,  2,   59,  51,  43,  35,  27,
 19,  11,  3,   60,  52,  44,  36,
 63,  55,  47,  39,  31,  23,  15,
 7,   62,  54,  46,  38,  30,  22,
 14,  6,   61,  53,  45,  37,  29,
 21,  13,  5,   28,  20,  12,  4
};
```

表中的第一个数是“57”，意思是原始密钥 K 的第 57 位成为置换密钥 K +的第一个比特。原始密钥的第 49 位成为置换密钥的第二位。原始密钥的第 4 位是置换密钥的最后一位。

下面是代码实现：

使用 template 模板函数，P 位数据根据 table 表置换为 Q 位数据，可扩展性大大增强。

```
template<int P, int Q>
inline bitset<Q> DES::Reverse(bitset<P> k, int table[Q])
{
    bitset<Q> temp;

    for (int i = 0; i < Q; i++)
    {
        temp[Q - i - 1] = k[P - table[i]];
    }

    return temp;
}
```

2、移位函数

该函数实现循环左移 shift 位：

```
template<int P>
inline bitset<P> DES::movemove(bitset<P> k, int shift)
{
    bitset<P> temp;
    temp =( k << shift)|(k >> P-shift);
    return temp;
}
```

3、S 盒置换

S 盒输入 48 位（8*6）。对对每组的六位作用 S 盒进行置换。用 S 盒中的 4 位数取代原来的 6 位。最终结果是 8 组 6 比特被转换成 8 组 4 比特，总共 32 比特。输入的第一位和最后一位组合为一个两位二进制数，用来确定 i，中间 4 位（二进制 0000 到 1111）用来确定 j。在表中查找第 i 行和第 j 列中的数字，即为一个 S 盒的输出。

代码实现如下：

```
int x = 0;
for (int i = 0; i<48; i = i + 6)
{
    int row = num_xor[47 - i] * 2 + num_xor[47 - i - 5];
    int col = num_xor[47 - i - 1] * 8 + num_xor[47 - i - 2] * 4 + num_xor[47 - i - 3] * 2 + num_xor[47 - i - 4];
    int num = S_Box[i / 6][row][col];
    bitset<4> binary(num);
    S[31 - x] = binary[3];
    S[31 - x - 1] = binary[2];
    S[31 - x - 2] = binary[1];
    S[31 - x - 3] = binary[0];
    x += 4;
}
```

（二）密钥生成部分

每轮生成的密钥存储在 finalKey 中，finalKey 声明为全局的，方便在 f 函数中使用：

```
DES::DES(b64 k) : k(k)//密钥生成
{
    b56 k_1;
    b28 k_c,k_d;
    b56 Key;
    k_1=Reverse<64, 56>(k, PC_1);
    cout << k_1 << endl;
    k_c = (k_1 >> 28).to_ullong() ;
    k_d = (k_1 << 28 >> 28).to_ullong();
    cout <<"c0:"<< k_c<<endl;
    cout <<"d0:"<<k_d<<endl;
    for (int round = 1; round <= 16; round++)
    {
        k_c = movemove(k_c, movetimes[round]);
        k_d = movemove(k_d, movetimes[round]);
        cout <<"c"<<round<<":"<< k_c << endl;
        cout <<"d"<<round<<":"<< k_d << endl;

        for (int i = 28; i<56; ++i)
            Key[i] = k_c[i - 28];
        for (int i = 0; i<28; ++i)
            Key[i] = k_d[i];
        finalKey[round] = Reverse<56, 48>(Key, PC_2);
        cout << "第"<<round<<"轮子秘钥："<<finalKey[round] << endl;
    }
}
```

（三）f 函数的实现

```
b32 DES::f(b32 R, int E[48], b48 Key, int S_Box[8][4][16])   //F函数
{
    b32 temp;
    b48 large = Reverse<32, 48>(R, E);
    b48 num_xor = large^Key;
    b32 S;
    int x = 0;
    for (int i = 0; i<48; i = i + 6)
    {
        int row = num_xor[47 - i] * 2 + num_xor[47 - i - 5];
        int col = num_xor[47 - i - 1] * 8 + num_xor[47 - i - 2] * 4 + num_xor[47 - i - 3] * 2 + num_xor[47 - i - 4];
        int num = S_Box[i / 6][row][col];
        bitset<4> binary(num);
        S[31 - x] = binary[3];
        S[31 - x - 1] = binary[2];
        S[31 - x - 2] = binary[1];
        S[31 - x - 3] = binary[0];
        x += 4;
    }
    temp = Reverse<32, 32>(S, P);
    //cout<<"f函数输出："<<temp<<endl;
    return temp;
}
```

（四）加密部分的实现

就是上述函数的组合：

```
b64 DES::E_DES(b64 m)   //DES加密
{
    b64 m_1;
    b32 m_L, m_R;
    b64 result;
    b64 finalresult;
    m_1= Reverse<64, 64>(m, IP_1);
    m_L = (m_1 >> 32).to_ullong();
    m_R = (m_1 << 32 >> 32).to_ullong();
    for (int round = 1; round <= 16; round++)
    {
        b32 temp;
        temp=m_L^f(m_R, E, finalKey[round], S_Box);
        m_L = m_R;
        m_R = temp;
    }
    for (int i = 0; i<32; ++i)
        result[i] = m_L[i];
    for (int i = 32; i<64; ++i)
        result[i] = m_R[i - 32];
    finalresult = Reverse<64, 64>(result, IP_2);
    return finalresult;
}
```

三、DES 解密算法及实现

DES 解密与 DES 加密算法相同，唯一的不同就是密钥使用的顺序相反，也就是解密过程的第一轮使用加密过程第 16 轮的子密钥。代码如下：

```
b64 DES::D_DES(b64 c)    //DES解密
{
    b64 c_1;
    b32 c_L, c_R;
    b64 result;
    b64 finalresult;
    c_1 = Reverse<64, 64>(c, IP_1);
    c_L = (c_1 >> 32).to_ullong();
    c_R = (c_1 << 32 >> 32).to_ullong();
    for (int round = 1; round <= 16; round++)
    {
        b32 temp;
        temp = c_L^f(c_R, E, finalKey[17-round], S_Box);
        c_L = c_R;
        c_R = temp;
    }
    for (int i = 0; i<32; ++i)
        result[i] = c_L[i];
    for (int i = 32; i<64; ++i)
        result[i] = c_R[i - 32];
    finalresult = Reverse<64, 64>(result, IP_2);
    return finalresult;
}
```

四、样例测试

1、给出了 DES 的测试数据为 des_test_case 类型的 cases 数组，需要将输入的 16 进制转化为 bitset 类型。以密钥的输入为例：

```
for (int j = 0; j < 8; j++)
{
    tempa[j] = bitset<8>((unsigned)cases[i].key[j]);
    a.append(tempa[j].to_string());
}
Key = b64(a);
```

2、不论加密解密都需要使用子密钥，所以首先调用 DES（Key）生成每轮的子密钥。

3、判断 mode，确定是加密还是解密分别调用 E_DES()与 D_DES()函数。

```
if (cases[i].mode == 1)   //加密
{
    cout << "加密" << endl;
    C_C = des.E_DES(M);
    cout << "带加密的明文:     " << M << endl;
    cout << "加密得到的密文为: " << C_C<<endl;
    cout << "期待的密文为:     " << C << endl;
}
if (cases[i].mode == 0)  //解密
{
    cout << "解密" << endl;
    M_M = des.D_DES(C);
    cout << "待解密的密文为:   " << C << endl;
    cout << "解密得到的明文为: " << M_M << endl;
    cout << "期待的明文为:     " << M << endl;
}
```

下面是输出结果：

```
第1个测试数据
初始密钥为:       0001000000110001011011100000001010001100100011110011101101001010
加密
带加密的明文:     0000000000000000000000000000000000000000000000000000000000000000
加密得到的密文为: 1000001011011100101110101111101111011110101010110110011000000010
期待的密文为:     1000001011011100101110101111101111011110101010110110011000000010

第2个测试数据
初始密钥为:       0000000100000001000000010000000100000001000000010000000100000001
加密
带加密的明文:     1001010111111000101001011110010111011101001100011101100100000000
加密得到的密文为: 1000000000000000000000000000000000000000000000000000000000000000
期待的密文为:     1000000000000000000000000000000000000000000000000000000000000000

第3个测试数据
初始密钥为:       0000000100000001000000010000000100000001000000010000000100000001
加密
带加密的明文:     1101110101111111000100100001110010100101000000010101011000011001
加密得到的密文为: 0100000000000000000000000000000000000000000000000000000000000000
期待的密文为:     0100000000000000000000000000000000000000000000000000000000000000
```

```
第18个测试数据
初始密钥为:       0000000100000001000000010000000100000001000000010000000100000001
解密
待解密的密文为:   0110110011000101110111101111101010101111000001000101000100101111
解密得到的明文为: 0000001000000000000000000000000000000000000000000000000000000000
期待的明文为:     0000001000000000000000000000000000000000000000000000000000000000

第19个测试数据
初始密钥为:       0000000100000001000000010000000100000001000000010000000100000001
解密
待解密的密文为:   0000110110011111100100111100110111010010111011000011100100110000
解密得到的明文为: 0000000100000000000000000000000000000000000000000000000000000000
期待的明文为:     0000000100000000000000000000000000000000000000000000000000000000

第20个测试数据
初始密钥为:       0000000100000001000000010000000100000001000000010000000100000001
解密
待解密的密文为:   1101100100000011000110110000001001110001101111010101101000001010
解密得到的明文为: 0000000010000000000000000000000000000000000000000000000000000000
期待的明文为:     0000000010000000000000000000000000000000000000000000000000000000
```

五、雪崩效应及分析

雪崩效应就是一种不稳定的平衡状态也是加密算法的一种特征，它指明文或密钥的少量变化会引起密文的很大变化，就像雪崩前，山上看上去很平静，但是只要有一点问题，就会造成一片大崩溃。

1、首先固定密钥，改变明文中的一位：

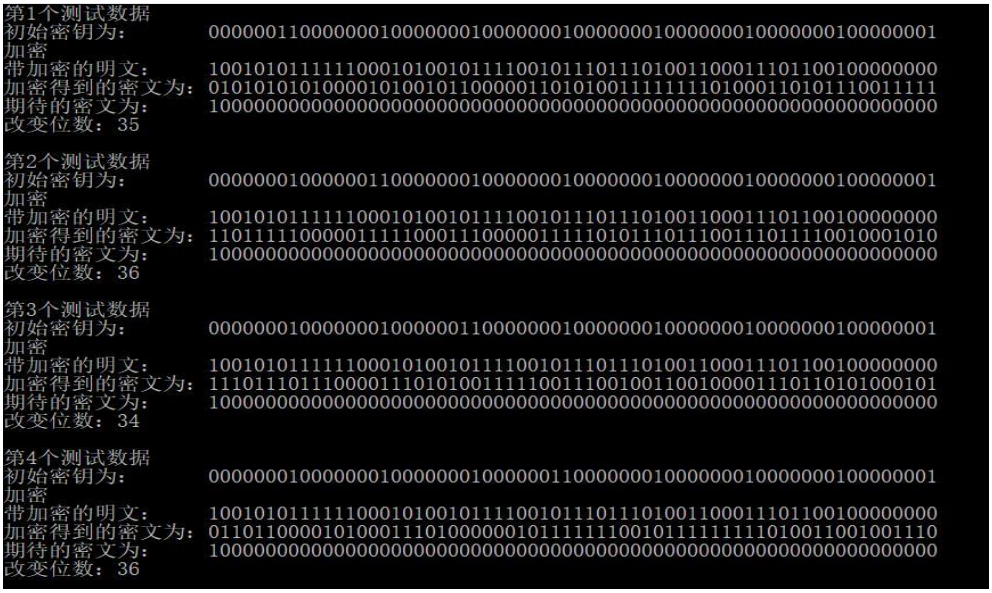
编写程序统计改变位数，结果如下：

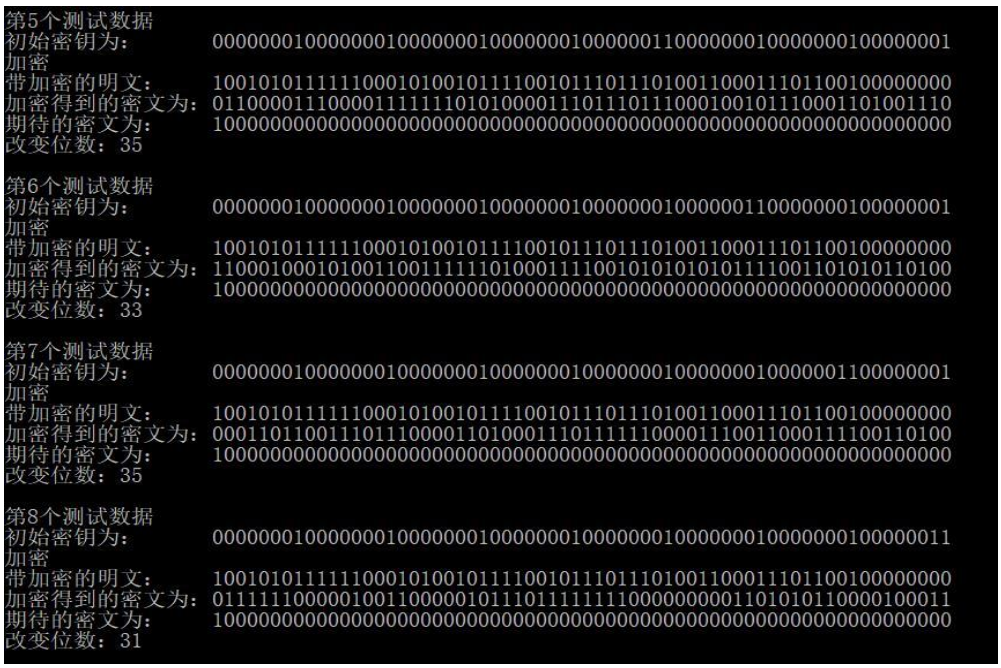
统计密文平均改变位数为：33.125

2、固定明文，改变密钥中的一位：

密钥中一位的改动会导致子密钥的牵连改变。

编写程序分别改动密钥中的一位，统计密文变化位数：





统计密文平均变化位数为：34.375

可以看出 DES 的严格遵循了雪崩效应，且明文或密钥改变一位时，密文改变的位数相对稳定在 30 左右，实现了混淆扩散的目的。