实验二 分组密码算法DES

学号： 姓名： 专业

一、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。

二、实验原理

1. 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）。
2. 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。

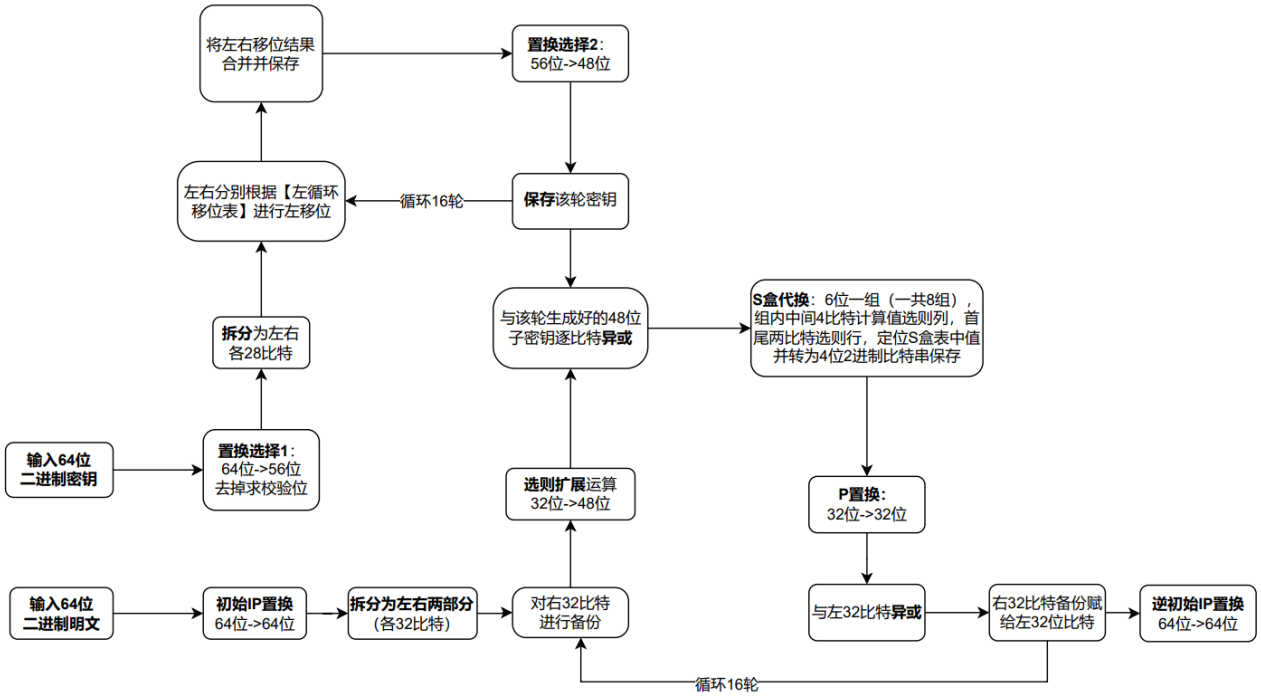
三、实验环境

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：对课本中DES算法进行深入分析，对初始置换、E扩展置换，S盒代换、轮函数、密钥生成等环节要有清晰的了解，并考虑其每一个环节的实现过程。
2. DES实现程序的总体设计：在第一步的基础上，对整个DES加密函数的实现进行总体设计，考虑数据的存储格式，参数的传递格式，程序实现的总体层次等，画出程序实现的流程图。
3. 在总体设计完成后，开始具体的编码，在编码过程中，注意要尽量使用高效的编码方式。
4. 利用3中实现的程序，对DES的密文进行雪崩效应检验。即固定密钥，仅改变明文中的一位，统计密文改变的位数；固定明文，仅改变密钥中的一位，统计密文改变的位数。

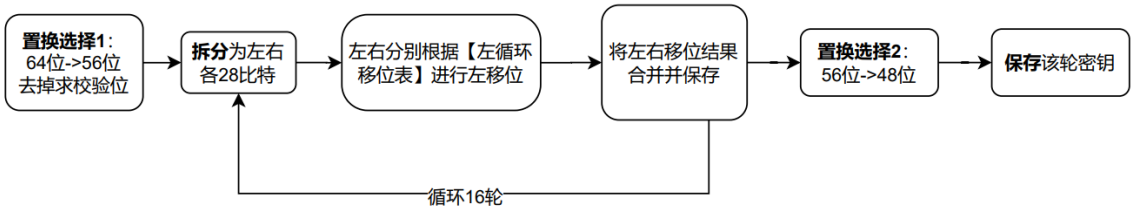
五. 整体流程图



六. DES加密与解密

1. 十六轮子密钥生成

◦ 流程图



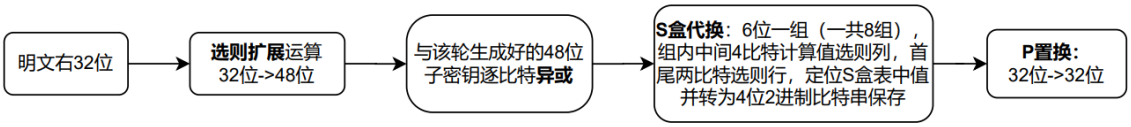
◦ 代码

```
void Gen_Subkeys(bitset<64> key, bitset<48>* subKey)
{
    bitset<28> left_key;    //子密钥左半部分
    bitset<28> right_key;   //子密钥右半部分
    bitset<56> new_key;  //保存更新的密钥

    // 密钥完成置换选则1：64位 -> 56位（去掉奇偶标记位）
    for (int i = 0; i < 56; ++i)
        new_key[55 - i] = key[64 - PC_1[i]];

    // 通过16轮迭代，得到每一次迭代的48位用作和明文F函数加密的子密钥
    for (int cycle = 0; cycle < 16; cycle++)
    {
        //把密钥分为左右两半
        for (int i = 0; i < 28; ++i)
            right_key[i] = new_key[i];
        for (int i = 28; i < 56; ++i)
            left_key[i - 28] = new_key[i];

```

```
        // 左右分别根据做循环移位表进行左移
        left_key = leftShift(left_key, Left_Shift[cycle]);
        right_key = leftShift(right_key, Left_Shift[cycle]);

        // 密钥完成置换选则2：56位->48位
        for (int i = 28; i < 56; ++i)
            new_key[i] = left_key[i - 28];
        for (int i = 0; i < 28; ++i)
            new_key[i] = right_key[i];
        for (int i = 0; i < 48; ++i)
            subKey[cycle][47 - i] = new_key[56 - PC_2[i]];
    }
}
```

2. F函数

流程图



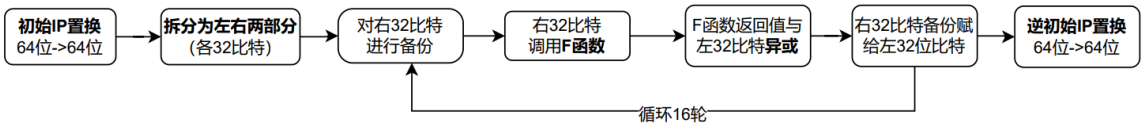
代码

```
bitset<32> F(bitset<32> R, bitset<48> K)
{
    bitset<48> Right_Expand;  //保存选则扩展运算E后的48位
    bitset<32> res;       //保存P置换之后结果

    // 选则扩展运算E：32位 -> 48位
    for (int i = 0; i < 48; ++i)
        Right_Expand[47 - i] = R[32 - Extend_Table[i]];

    // 与子密钥进行异或运算
    Right_Expand = Right_Expand ^ K;

    // S盒代换：48位 -> 32位
    int index = 31;
    for (int i = 0; i < 48; i = i + 6)   //S1~S8 一组六个值
    {
        int row = Right_Expand[47 - i] * 2 + Right_Expand[47 - i - 5];
        int col = 0;
        for (int j = 1; j <= 4; j++)
        {
            col += Right_Expand[47 - i - j];
            if (j != 4)
                col *= 2;
        }
        int num = S_Box[i / 6][row][col];
        bitset<4> binary(num);
        for (int j = 0; j < 4; j++)
            res[index--] = binary[3 - j];
    }

```

```
    // P置换：32位 -> 32位
    bitset<32> tmp = res;
    for (int i = 0; i < 32; ++i)
        res[31 - i] = tmp[32 - P_Tabe[i]];

    return res;
}
```

3. 加密总函数

流程图

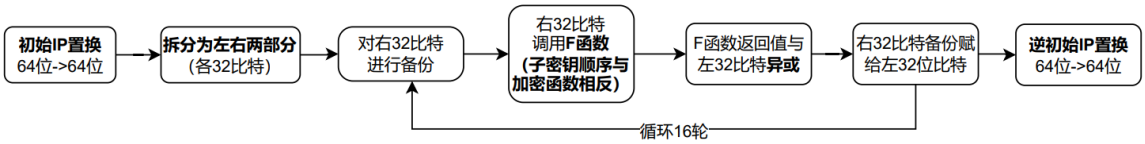


代码

```
bitset<64> encrypt(bitset<64>& initial, bitset<48>* subKey)
{
    bitset<64> result;          //保存最终结果
    bitset<64> combine;     //保存初始置换后的64位输入/左右合并的加密结果
    bitset<32> left;            //左边的32位
    bitset<32> right;           //右边的32位
    bitset<32> temp;         //对右明文进行备份

    // 初始置换IP过程
    for (int i = 0; i < 64; ++i)
        combine[63 - i] = initial[64 - IP_Table[i]];

    // 分为左右两部分
    for (int i = 32; i < 64; ++i)
        left[i - 32] = combine[i];
    for (int i = 0; i < 32; ++i)
        right[i] = combine[i];

    // 进行迭代（共16轮）
    for (int cycle = 0; cycle < 16; cycle++)
    {
        temp = right;
        right = left ^ F(right, subKey[cycle]);
        left = temp;
    }

    // 左右交换
    for (int i = 0; i < 32; ++i)
        combine[i] = left[i];         //把left的放在高位
    for (int i = 32; i < 64; ++i)
        combine[i] = right[i - 32];  //把right的放在低位

    // 逆初始置换
    for (int i = 0; i < 64; ++i)
        result[63 - i] = combine[64 - Rev_IP[i]];
    return result;
}
```

4. 解密总函数

◦ 流程图



◦ 代码

唯一与加密总函数不同的地方在迭代16轮的时候，子密钥和加密时使用的顺序相反

```
bitset<64> decrypt(bitset<64>& cipher, bitset<48>* subKey)
{
    bitset<64> result;    //保存最终结果
    bitset<64> combine;   //保存初始置换后的64位输入/左右合并的解密结果
    bitset<32> left;      //左边的32位
    bitset<32> right;     //右边的32位
    bitset<32> temp;     //对右明文进行备份
    // 初始置换IP
    for (int i = 0; i < 64; ++i)
        combine[63 - i] = cipher[64 - IP_Table[i]];

    // 拆分为左右两部分
    for (int i = 0; i < 32; ++i)
        right[i] = combine[i];
    for (int i = 32; i < 64; ++i)
        left[i - 32] = combine[i];

    for (int cycle = 0; cycle < 16; cycle++){
        temp = right;
        right = left ^ F(right, subKey[15 - cycle]);
        left = temp;
    }

    // 左右交换
    for (int i = 0; i < 32; ++i)
        combine[i] = left[i];
    for (int i = 32; i < 64; ++i)
        combine[i] = right[i - 32];

    // 逆初始置换
    for (int i = 0; i < 64; ++i)
        result[63 - i] = combine[64 - Rev_IP[i]];
    // 返回明文
    return result;
}
```

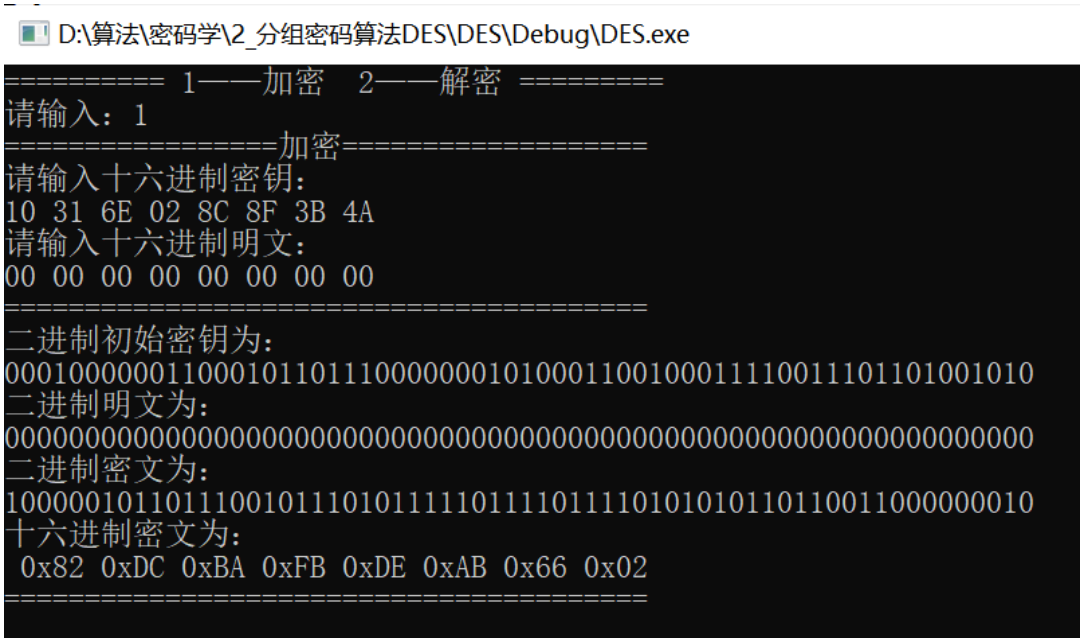
七. 运行结果

1. 加密

十六进制密钥：0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

十六进制明文：0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

十六进制密文：0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

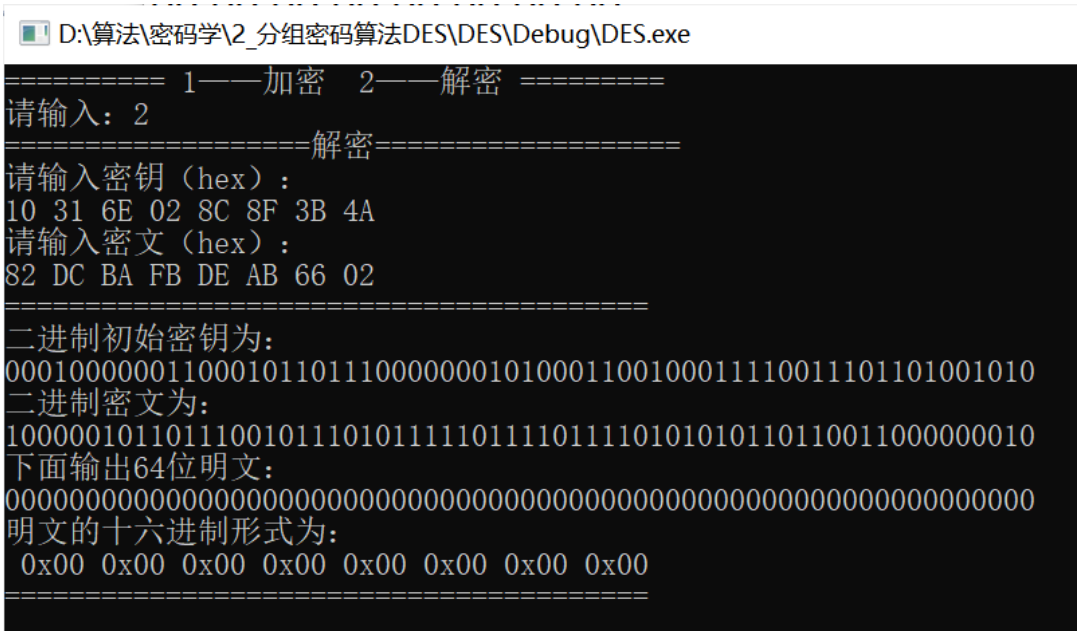


2. 解密

十六进制密钥：0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

十六进制密文：0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

十六进制明文：0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00



八. 检验雪崩效应

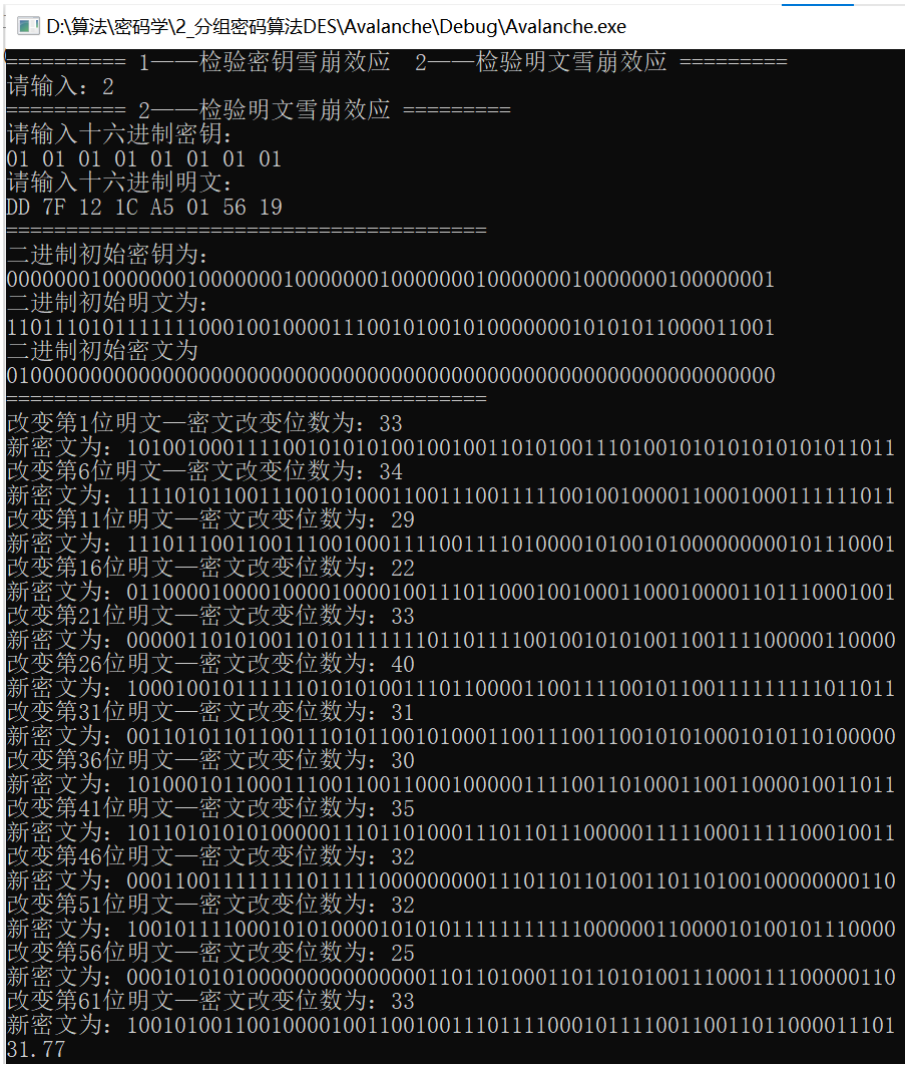
1. 明文雪崩效应

◦ 核心主函数代码

```
cout << "========== 2--检验明文雪崩效应 =========" << endl;
// 输入十六进制密钥并保存为64位二进制
cout << "请输入十六进制密钥：" << endl;
for (int i = 0; i <= 7; i++)
    cin >> (hex) >> hex_key[i];
bin_key_original = Hex_Bin(hex_key);

// 输入十六进制明文并保存为64位二进制
cout << "请输入十六进制明文：" << endl;
for (int i = 0; i <= 7; i++)
    cin >> (hex) >> hex_text[i];
bin_text_original = Hex_Bin(hex_text);
// 得到十六轮密钥
Gen_Subkeys(bin_key_original, subKey);

// 加密过程
encrypt_text_original = encrypt(bin_text_original, subKey);

cout << "=======================================" << endl;

cout << "二进制初始密钥为：" << endl;
cout << bin_key_original << endl;
cout << "二进制初始明文为：" << endl;
cout << bin_text_original << endl;
cout << "二进制初始密文为" << endl;
for (int i = 0; i <= 63; i++)
    cout << encrypt_text_original[63 - i];
cout << endl;
cout << "=======================================" << endl;

// 分别改变密钥每一位检验雪崩效应
for (int i = 0; i < 64; i++)
{
    bitset<64> encrypt_text;  //保存获得的密文
    bitset<64> bin_key = bin_key_original;
    bitset<64> bin_text = bin_text_original;
    bin_text[63 - i] = 1 - bin_text[63 - i];

    // 得到十六轮密钥
    Gen_Subkeys(bin_key, subKey);

    // 加密过程
    encrypt_text = encrypt(bin_text, subKey);

    //统计密文不同位数
    int num = 0;
    for (int m = 0; m < 64; m++)
        if (encrypt_text_original[m] != encrypt_text[m])
            num++;
    if (i % 5 == 0)
    {
```

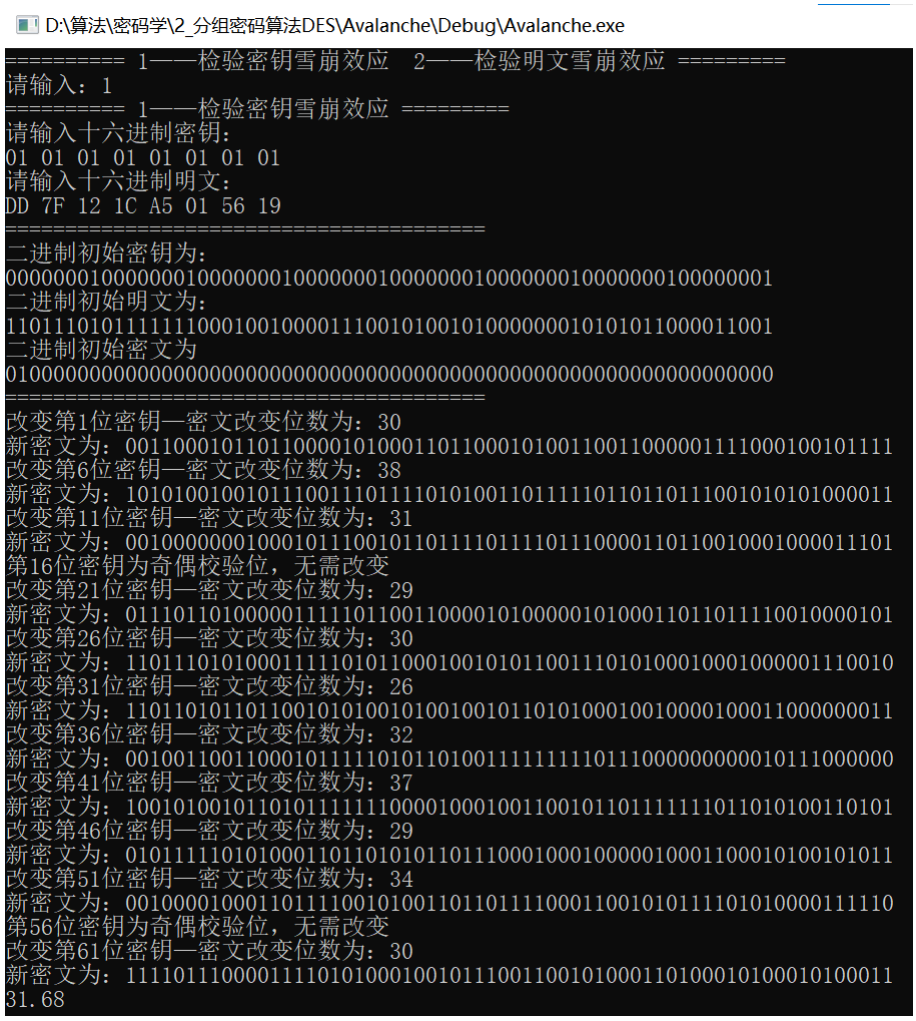
```
52          cout << "改变第" << i + 1 << "位明文-密文改变位数为：" << num
<< endl;
53          cout << "新密文为：";
54          for (int i = 0; i <= 63; i++)
55              cout << encrypt_text[63 - i];
56          cout << endl;
57      }
58      total += num;
59  }
```

- 64位明文逐一改变，下图中每改变5次输出一次，计算得出平均密文改变的位数位31.77



2. 密钥雪崩效应

除去奇偶校验位之外的密钥逐一改变，下图中每改变5次输出一次，计算出平均密文改变位数为31.66



九. 实验中遇到的bug

在检验密钥雪崩效应的时候，初始的实验结果发现有的地方改变密钥之后密文并没有改变，当时思考之后发现是因为我把奇偶校验位也做了改变，但其实奇偶校验位在密钥置换选择一的时候会被扔掉，所以就算改变改位置也不会对最终密文产生影响。