南开大学
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

密码学 (1091) 第二次实验

分组密码算法 DES

苗发生

年级：2020 级

专业：信息安全 & 法学双学位

指导教师：古力

2022 年 11 月 18 日

目录

一、 实验内容

(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) 实验环境

运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境

(三) 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

(四) DES 算法介绍

DES 是一个分组加密算法，就是将明文分组进行加密，每次按顺序取明文一部分，一个典型的 DES 以 64 位为分组，加密解密用算法相同。它的密钥长度为 56 位，因为每组第 8 位是用来做奇偶校验，密钥可以是任意 56 位的数，保密性依赖于密钥。 DES 算法的核心主要包括两部分，第一部分是 Feistel 结构，第二部分是 16 个子密钥的生成。

(五) 实验原理

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 L_0，右边记为 R_0，然后对 R_0 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（R_0，K_1)，再与 L_0 做逐位异或运算，其结果记为 R_1,R_0 则作为下一轮的 L_1。如此循环 16 轮，最后得到 L_{16}、R_{16}，再对 L_{16}、R_{16} 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。

(六) 实验报告和要求

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

NKU

二、 实验具体流程

(一) 算法分析

1. 初始替换

DES 算法使用 64 位的密钥 key 将 64 位的明文输入块变为 64 位的密文输出块, 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 L_0 R_0（左、右）两部分，每部分分别为 32 位, 替换表如下:

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

注意：这里的数字表示的是原数据的位置，不是数据本身

2. E 扩展置换

经过初始置换的 64 位数据被分为左右 32 位数据，其中右边的 32 位数据经过 E 扩展置换变成 48 位数据，扩展置换表如下:

扩展置换表

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

https://blog.csdn.net/ybzzz

图 1: E 扩展置换表

E 扩展置换本质就是在某些位置进行了填位操作，将其中的 16 个比特各重复一次

3. S 盒代换

在密码学中，S 盒 (Substitution-box) 是对称密钥算法执行置换计算的基本结构。多用在分组密码算法中，是唯一的非线性结构

S 盒将 48 比特压缩成 32 比特，S 盒接受特定数量的输入 48 比特，经过 8 个盒将其转换为 32 比特输出。在 DES 算法中替代由 8 个不同的 S 盒完成，每个 S 盒有 6 位输入 4 位输出，S 盒的具体过程如下:

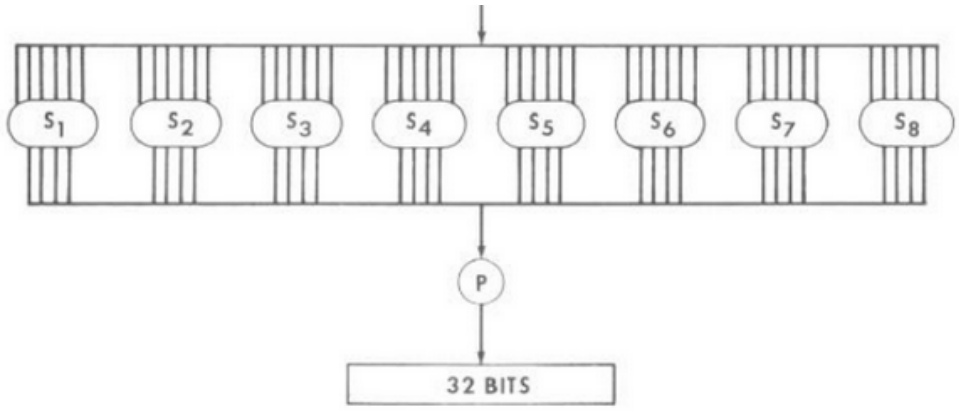

图 2: S 盒的具体过程

具体的八个盒中代换的数据如下:

S1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

图 3: DES 的 S 盒定义

具体代换的过程如下，例如需要代换的第一组数据输入为 011001，则第一位 0 和最后一位 1 组合成的 01 即为行号，中间的 1101 为列号，第一组数据对应 S1，01 转化成 10 进制为 1，1101

转化成 10 进制为 13，因此 S1 中的第 1 行第 13 列就为对应的输出，查表得 5，转化成 2 进制为 0101。因此 0101 就为最终的 4 位输出.

4. 轮结构

DES 加密算法的轮结构如下图所示:

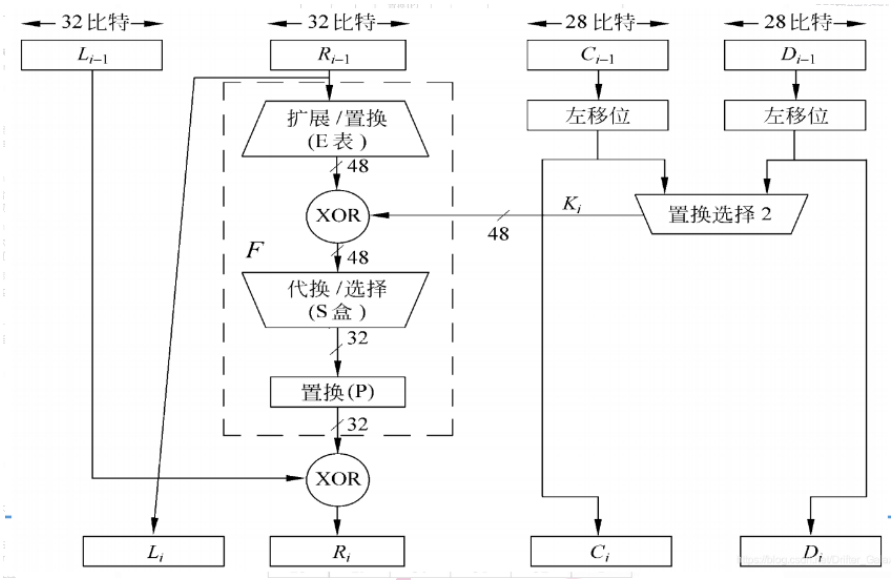

图 4: DES 加密算法的轮结构

轮结构的左半部分将 64 比特的轮输入分为各 32 比特的左、右两半，分别记为 L 和 R，和 Feistel 网络一样，每轮变换可由以下公式表示

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \bigoplus F(R_{i-1}, K_i)$$

注意：无论 F 函数如何选取，DES 的轮函数是一个对合变换

5. 密钥生成

初始输入 64 位的初始密钥，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，将其平分成两部分 C_0 D_0。然后分别进行第一次循环左移，得到 C_1 D_1，将 C_1（28 位）、D_1（28 位）合并后得到 56 位的输出结果，再经过压缩置换 PC-2，从而得到了密钥 K_1（48 位）。依次类推，便可得到 K_2 ... K_{16}。需要注意的是，16 次循环左移对应的左移位数要依据移位次数表进行的。

(二) DES 算法整体设计

1. 数据的存储格式

由于 DES 算法加密和解密的过程中，需要用到初始置换、逆初始置换、扩展运算 E、置换运算 P、S 盒以及做循环移位产生子密钥等多种定义，而这些定义都需要以表的形式存储数据，算法中根据这些预先给定的数据进行具体操作，考虑到程序的可读性以及可扩展性，将这些表用静

态常量数组的形式（const static int）放在头文件中，DES 算法用到这些表的时候，只需要引一下头文件

2. 参数的传递格式

考虑到编码的效率，我们在整个 DES 加密和解密的过程中，参数均以 bit 的形式传递，故我们首先需要将输入的密钥以及明文（十六进制）转为 bit 的形式，另外，考虑到最终的结果（加密的结果是密文，解密的结果是明文）仍需要为 16 进制的形式，故我们编写 Hextobit () 和 BittoHex () 两个函数，进行 16 进制和 2 进制之间的切换，另外，由于进制的切换函数对于 DES 算法整体没有任何影响，只是为了提高编码的效率，故我们将编写好的进制转换函数仍旧放到头文件中，需要的时候引入头文件进行调用即可

3. 子秘钥的生成

虽然 DES 算法加解密过程中需要经过 16 轮的迭代，并且每轮迭代需要的子密钥均不相同，但是在 DES 算法执行输入密钥之后，16 个子密钥均已经能够确定，不会随着每轮算法的执行而发生变化，故为了提高算法的执行效率以及代码的可读性，在算法执行之前，先将 16 轮所需的子密钥均计算出来，保存到全局变量之中，算法每轮的执行过程中，只需要调用这一轮对应的子密钥即可，**第一轮置换的过程中会自动将 8 个校验位去除**，故在计算子密钥的过程中，不需要手动去除无效位

密钥生成的具体流程如下图所示：

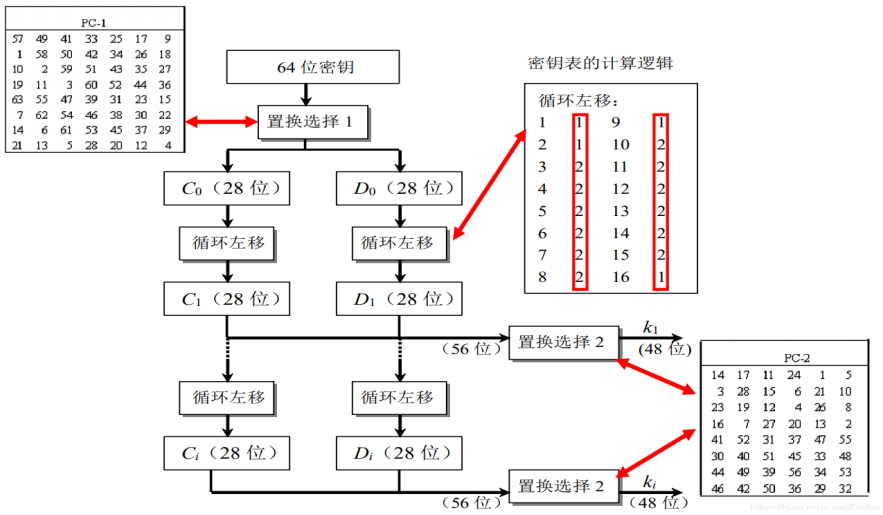

图 5: 子密钥的生成流程

4. DES 算法加解密的整过程

DES 算法加解密的核心过程是 16 轮的代换，但由于每一轮的代换步骤完全相同，仅仅是参数不同，故我们可以将轮结构中的每一个过程单独封装为一个函数，不同轮数的代换只需要给函数传入不同的参数即可，16 轮的代换步流程如下：

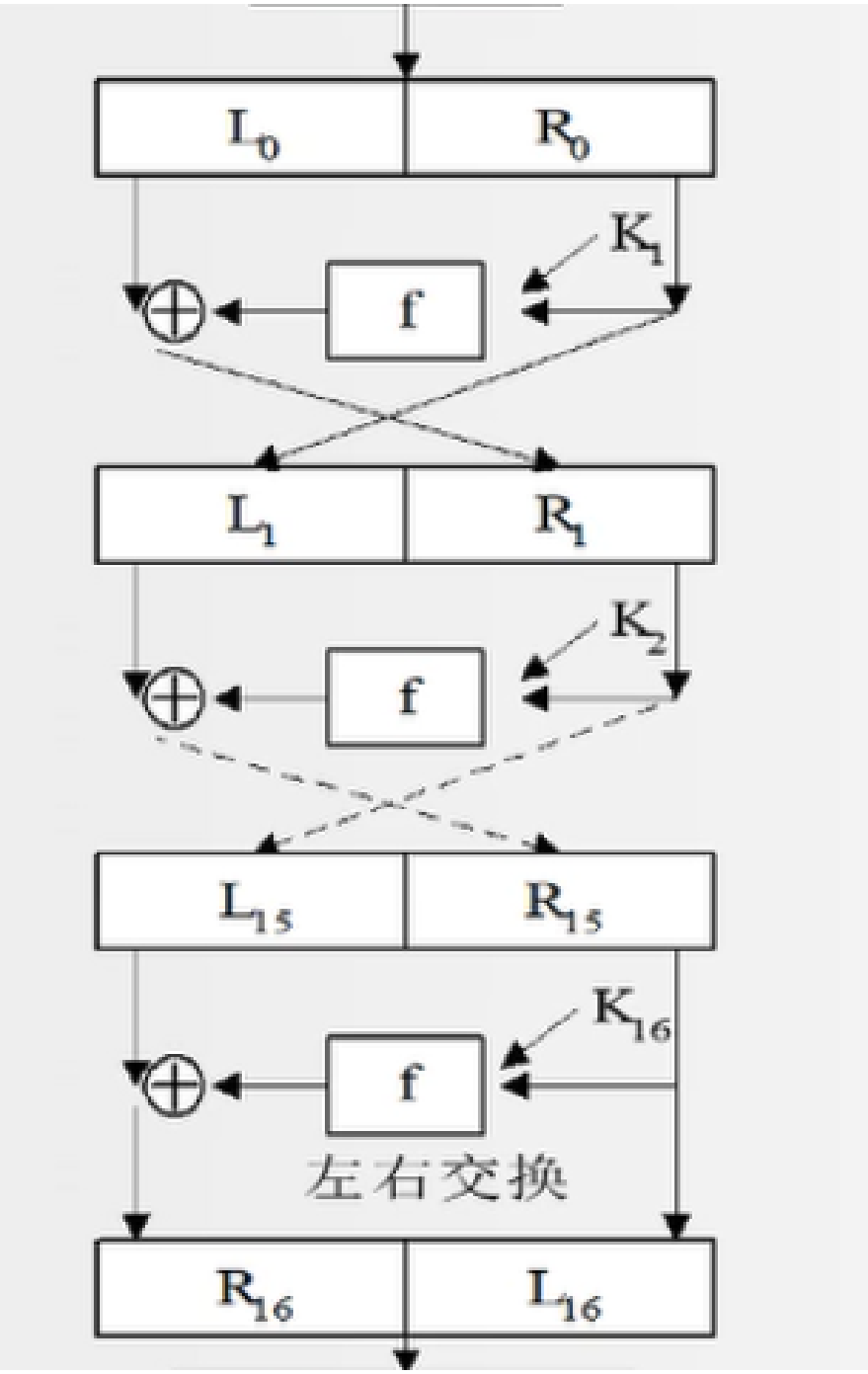

图 6: DES 的十六轮代换

5. 实验的整体流程

本实验的整体流程为如下：

加密：首先输入 16 进制的明文和密钥，并进行合法性检查，之后将其转换为 2 进制的形式，接着利用输入的密钥预先算好所有的子密钥并存储起来，方便后续调用。接着将 2 进制形式的明文进行初始替换，并分为左右两部分，之后进行 16 轮的迭代，之后将左右两部分合并，即密文的 2 进制形式，再将其转为 16 进制的形式，加密完毕，由于 DES 加密和解密使用同一算法，只需要将子密钥的顺序颠倒即可，加密整体流程如下：

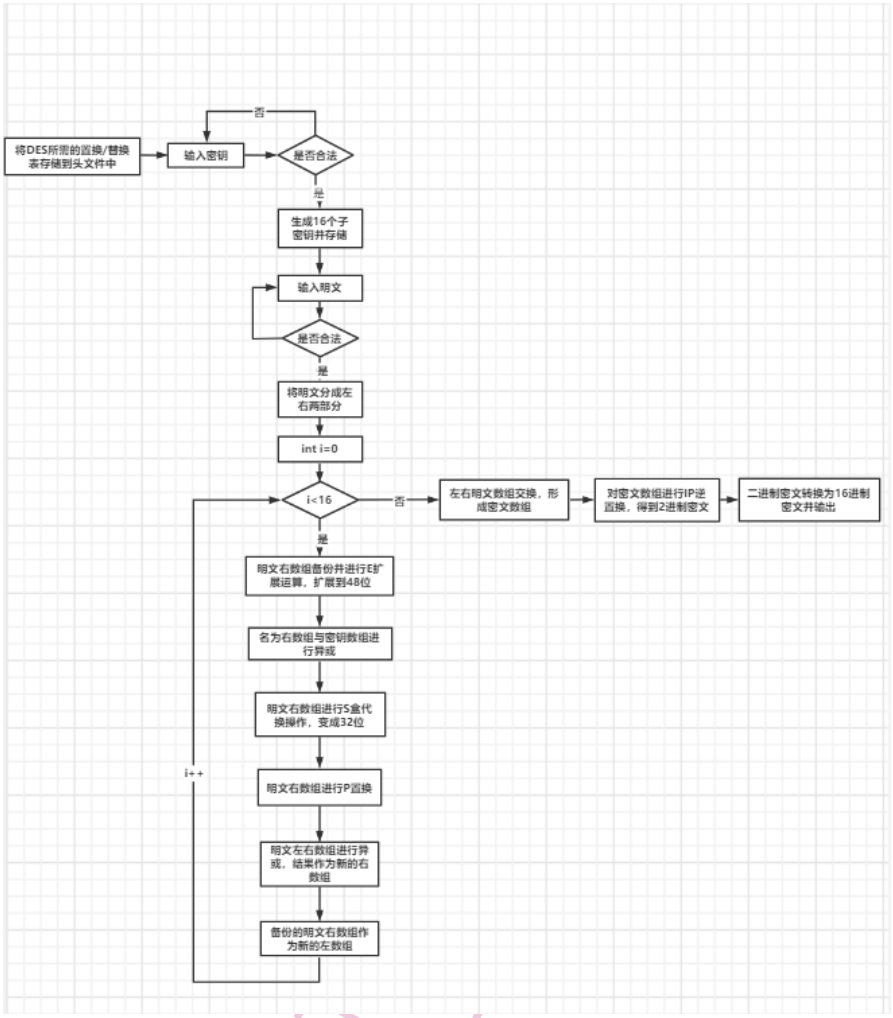

图 7: 加密流程图

由于 DES 解密流程和加密流程完全相同，只是子密钥的使用顺序恰好相反，故完全可以参考 DES 加密流程图

(三) 关键代码解析

1. 生成子密钥

由于子密钥和每一轮的迭代无关，输入密钥后，16 个子密钥就已经确定，故在进行 16 轮迭代之前，先计算出所有子密钥，并存储起来，加密或解密的迭代过程中，只需要调用即可，本次实验中，我们将生成子密钥封装为 GenSubkey() 函数，生成子密钥的 C++ 代码如下:

生成子密钥

```
void GenSubkey()
{
        bitset<28> left;
        bitset<28> right;
        //经过PC1_table的置换选择后，已经去除了奇偶校验位
        bitset<56> firstKey;
        for (int i = 0; i < 56; i++)
```

```
        {
                firstKey[55 - i] = key[64 - PC1_table[i]];
        }
        // 生成子密钥
        for (int index = 0; index < 16; index++)
        {
                // 分为左右两部分
                // 前28位与后28位
                for (int i = 28; i < 56; i++)
                {
                        left[i - 28] = firstKey[i];
                }
                for (int i = 0; i < 28; i++)
                {
                        right[i] = firstKey[i];
                }
                // 左移
                left = leftShift(left, LeftShift[index]);
                right = leftShift(right, LeftShift[index]);
                for (int i = 28; i < 56; i++)
                {
                        firstKey[i] = left[i - 28];
                }
                for (int i = 0; i < 28; i++)
                {
                        firstKey[i] = right[i];
                }
                bitset<48> zipKey;
                // 压缩置换，56位到48位
                for (int i = 0; i < 48; i++)
                {
                        zipKey[47 - i] = firstKey[56 - PC2_table[i]];
                }
                subKey[index] = zipKey;
        }
}
```

另外，在生成子密钥的过程中，我们必须根据左循环移位表进行子密钥的生成，循环左移位的代码如下：

循环左移位

```
bitset<28> leftShift(bitset<28> key, int shift)
{
        bitset<28> temp = key;
        for (int i = 27; i >= 0; --i)//左右两部分密钥各位28位
        {
                if (i - shift < 0)
                        key[i] = temp[i - shift + 28];
                else
```

NKU

```
                key[i] = temp[i - shift];
        }
        return key;
}
```

说明：由于 DES 加解密过程中存在大量对 bit 的操作，故函数的返回值我们用 bitset<28> 的类型

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。f 函数的步骤为首先根据表对应关系，将其扩展为 48 位，接着再和密钥进行异或操作，之后根据 S 盒将其恢复到 32 位，之后进行 P-置换，详细的 C++ 代码如下：

f 函数

```
bitset<28> leftShift(bitset<28> key, int shift)
//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，
bitset<32> f(bitset<32> R, bitset<48> k)
{
        int temp = 0;
        bitset<48> expandR;
        for (int i = 0; i < 48; ++i)
                expandR[47 - i] = R[32 - Extend_table[i]];
        expandR = expandR ^ k;
        bitset<32> result;
        for (int i = 0; i < 48; i = i + 6)
        {
                int row = expandR[47 - i] * 2 + expandR[47 - i - 5];
                int col = expandR[47 - i - 1] * 8 + expandR[47 - i - 2] * 4 +
                     expandR[47 - i - 3] * 2 + expandR[47 - i - 4];
                int num = S_BOX[i / 6][row][col];
                bitset<4> binary(num);
                result[31 - temp] = binary[3];
                result[31 - temp - 1] = binary[2];
                result[31 - temp - 2] = binary[1];
                result[31 - temp - 3] = binary[0];
                temp += 4;
        }
        // 第四步：P–置换

        bitset<32> tmp = result;
        for (int i = 0; i < 32; ++i)
                result[31 - i] = tmp[32 - Perm_table[i]];
        return result;
}
```

说明：S 盒的具体对应关系可由如下图中所示方式进行：（图片摘自 CSDN）

NKU

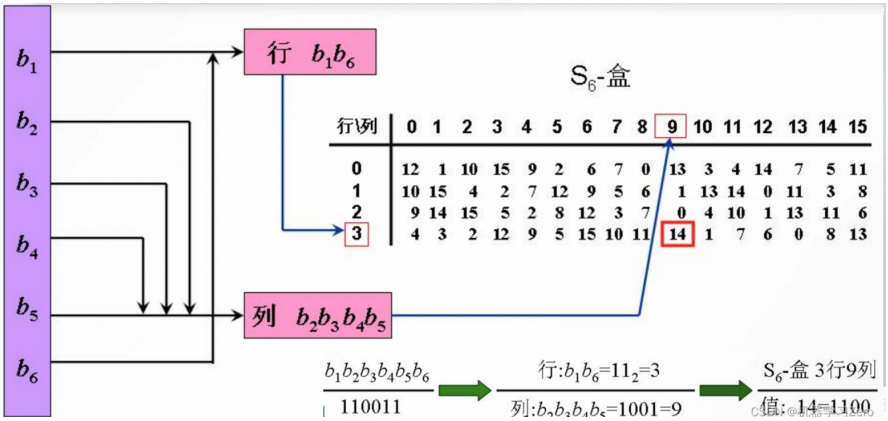

图 8: S 盒对应关系图

3. DES 加密

根据上图 7 所示的 DES 加密流程图，加密是 DES 算法的核心步骤，主要可以分为以下几步

1) 输入明文，进行初始 IP 置换
2) 将明文分为左右两部分
3) 进行 16 轮的轮函数迭代
4) 高 32 位和低 32 位交换位置
5) 结尾置换 IP-1

根据这些步骤，实现的 C++ 代码如下:

DES 加密

```
bitset<64> Encryption(bitset<64>& plaintext)
{
        bitset<64> ciphertext;
        bitset<64> firstInput;
        bitset<32> left;
        bitset<32> right;

        // 初始置换IP
        for (int i = 0; i < 64; i++)
                firstInput[63 - i] = plaintext[64 - InitPerm[i]];
        // 一分为二
        for (int i = 32; i < 64; i++)
                left[i - 32] = firstInput[i];
        for (int i = 0; i < 32; i++)
                right[i] = firstInput[i];

        // 迭代
        for (int round = 0; round < 16; round++)
        {
```

NKU

```
                bitset<32> nextHigh;
                nextHigh = right;
                right = left ^ f(right, subKey[round]);
                left = nextHigh;
        }
        // 合并注意高位和低位发生了调换
        for (int i = 0; i < 32; i++)
        {
                ciphertext[i] = left[i];
                ciphertext[i + 32] = right[i];
        }
        // 结尾置换IP-1
        firstInput = ciphertext;
        for (int i = 0; i < 64; i++)
                ciphertext[63 - i] = firstInput[64 - InvPerm[i]];
        return ciphertext;
}
```

4. DES 解密

由于 DES 算法属于对称密钥体制，故 DES 的解密和加密步骤完全相同，只是加密和解密使用的子密钥顺序不同，对 DES 解密的实现可以复用加密所用的代码，只需更换子密钥的调用顺序即可

DES 解密

```
bitset<64> Decryption(bitset<64>& ciphertext)
{
        bitset<64> plaintext;
        bitset<64> firstInput;
        bitset<32> left;
        bitset<32> right;

        // 初始置换IP
        for (int i = 0; i < 64; i++)
                firstInput[63 - i] = ciphertext[64 - InitPerm[i]];
        // 一分为二
        for (int i = 32; i < 64; i++)
        {
                left[i - 32] = firstInput[i-32];
                right[i] = firstInput[i];
        }
        // 迭代
        bitset<32> nextHigh;
        for (int round = 0; round < 16; round++)
        {
                nextHigh = right;
                right = left ^ f(right, subKey[15 - round]);
                left = nextHigh;
```

NKU

```
        }
        // 合并注意高位和低位发生了调换
        for (int i = 0; i < 32; i++)
        {
                plaintext[i] = left[i];
                plaintext[i + 32] = right[i];
        }
        // 结尾置换IP-1
        firstInput = plaintext;
        for (int i = 0; i < 64; i++)
                plaintext[63 - i] = firstInput[64 - InvPerm[i]];
        return plaintext;
}
```

（四） 检验雪崩效应

1. 改变明文

由于明文中的每一位均是内容，故我们更改明文的每一位观察雪崩现象，更改明文观察雪崩现象的过程中，密钥并未进行更改可以复用，故我们只需更改明文后重新利用密钥进行加密即可

更改明文观察雪崩

```
int endl1 = 0;
        for (int i = 0; i < 64; i++)
        {
                plaintext.flip(i);//对第i位比特进行翻转
                result2 = Encryption(plaintext);
            for (int j = 0; j < 64; j++)
                {
                        if (result1[j] != result2[j])
                        {
                                ans[i]++;
                        }
                }
                endl1++;
                cout << "修改明文第" << i+1 << "位," << "密文更改" << ans[i]
                    << "位       " ;
                if (endl1 % 2 == 0)
                        cout << endl;
        }
        double num = 0.0;
        for (int i = 0; i < 64; i++)
        {
                num += ans[i];
        }
        cout << "************************************" << endl;
        cout << "单独修改明文的每一个bit，密文平均更改位数为：" << num / 64<<
            endl;
```

2. 改变密钥

由于密钥中的 64 位有 8 位是校验位，并非密钥本身，故在观察雪崩现象的时候，不应该统计校验位的更改带来的雪崩现象，每次更改完密钥后，均需要重新生成子密钥，重新进行解密（即进行 DES 解密步骤），核心代码如下：

更改密钥观察雪崩

```
for (int i = 0; i < 64; i++)
        {

                if ((i+1) % 8 == 0)
                        continue;
                replacekey = temp;
                replacekey.flip(i);//对第i位比特进行翻转
                key = replacekey;

                GenSubkey();
                result2 = Decryption(plaintext);
            for (int j = 0; j < 64; j++)
                {
                        if (result1[j] != result2[j])
                        {
                                ans[i]++;
                        }
                }
                endl1++;
                cout << "修改密钥第" << i+1 << "位," << "解密明文更改" << ans
                    [i] << "位      " ;
                if (endl1 % 2 == 0)
                        cout << endl;
        }
        double num = 0.0;
        for (int i = 0; i < 64; i++)
        {
                num += ans[i];
        }
        cout << "************************************" << endl;
        cout << "单独修改密钥可用的56位，平均更改位数位" << num / 56<<endl;
```

（五） 实验结果演示

1. DES 加密

根据上面的关键函数代码，只需要在 main 函数进行输入输出以及调用即可，现对老师给的样例进行测试，本次展示的测试样例为：

样例一：

密钥： 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

明文： 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

密文： 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

样例二：

密钥： 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01

明文： 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00

密文： 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

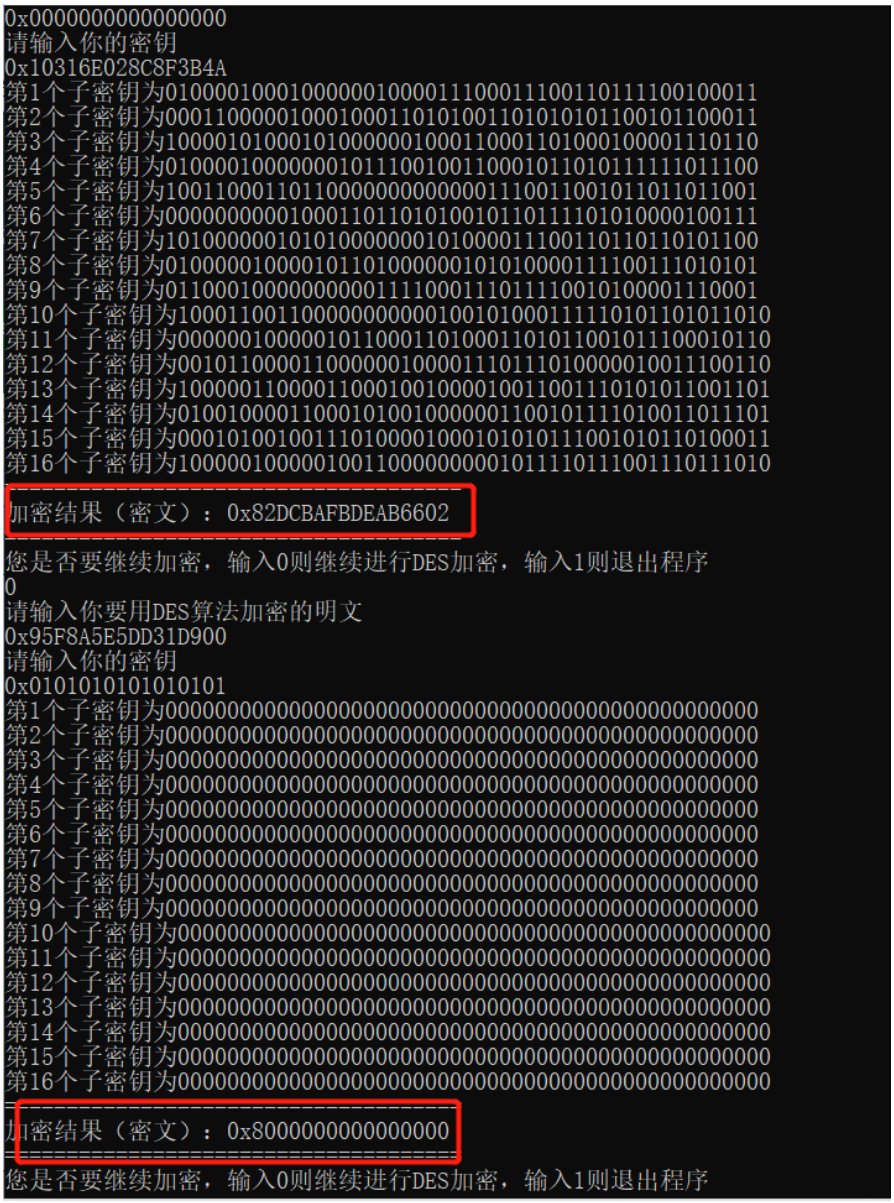

图 9: DES 加密演示

经过检验，加密成功！

2. DES 解密

DES 解密和 DES 加密过程完全相同，将 DES 加密中的加密函数改为解密函数，并对输入输出做修改即可，现对老师给的样例进行测试，本次展示的测试样例为：

样例一：

密钥： 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

明文： 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

密文： 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

样例二：

密钥： 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01

明文： 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00

密文： 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

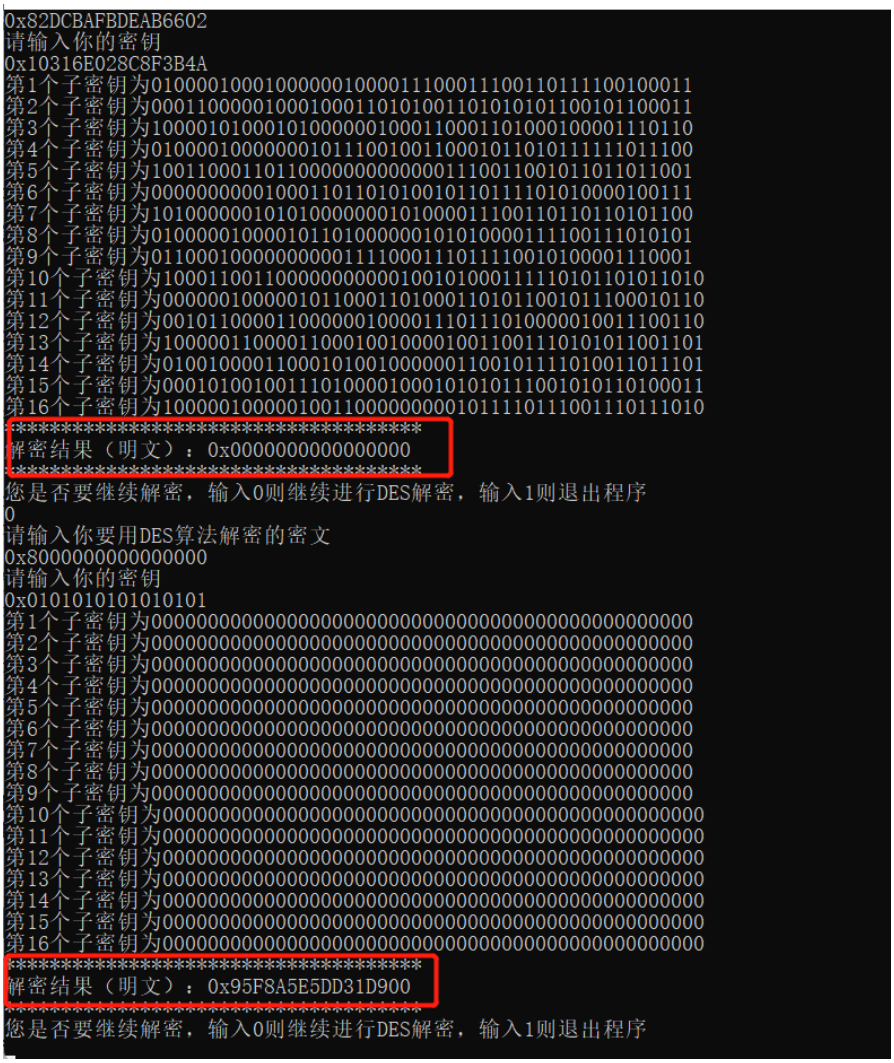

图 10: DES 解密演示

经过检验，解密成功！

3. 改变密钥雪崩效应

由于 64 位的密钥中，有八位作为校验位，没有更改的必要，故我们对其他 56 位密钥，每次改变一个 bit 位，观察雪崩现象，最后记录平均的改变位数，实现截图如下，观测到雪崩现象：

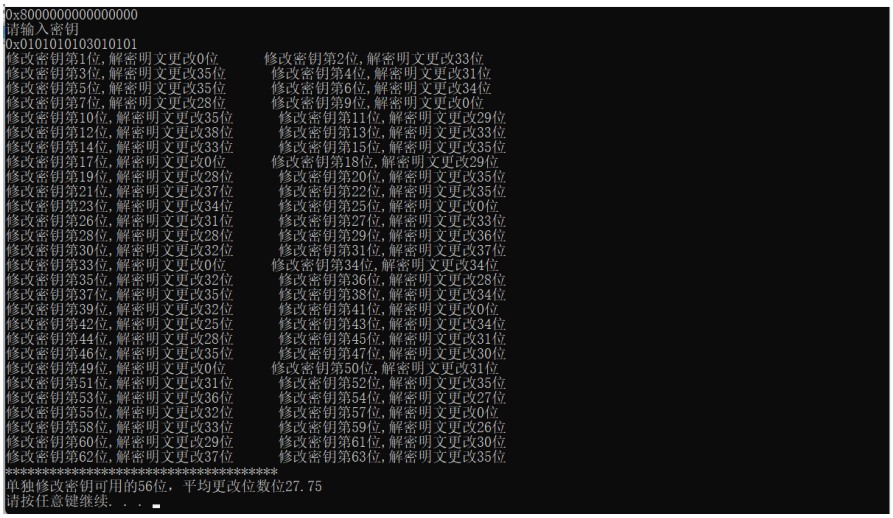

图 11: 修改密钥观察雪崩现象

经过检验，观测到雪崩现象，平均修改 27.75 个 bit！

4. 改变明文雪崩效应

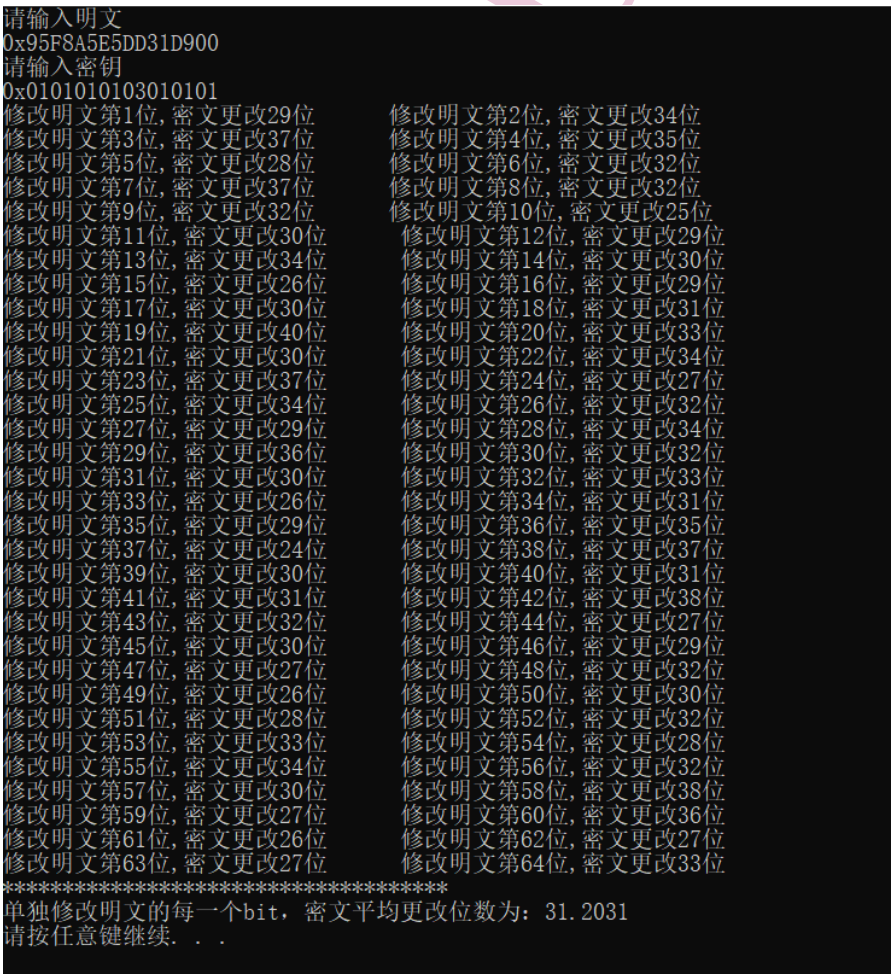

图 12: 修改明文观察雪崩现象

经过检验，观测到雪崩现象，平均修改 31.2031 个 bit！

NKU