南开大学

Nankai University

南　开　大　学

网络空间安全学院

密码学课程报告

第二次实验报告

——分组密码算法 DES

学号：1611519

姓名：周子祎

年级：2016 级

专业：信息安全-法学

2018 年 12 月 13 日

密码学第二次实验报告

——分组密码算法 DES

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. 整体流程

明文分组长度为 64 位，秘钥长度为 56 位，最终形成的密文长度为 64 位。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：

(1) 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；

(2) 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；

(3) 如此循环 16 轮，最后得到 L16、R16；

(4) 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；

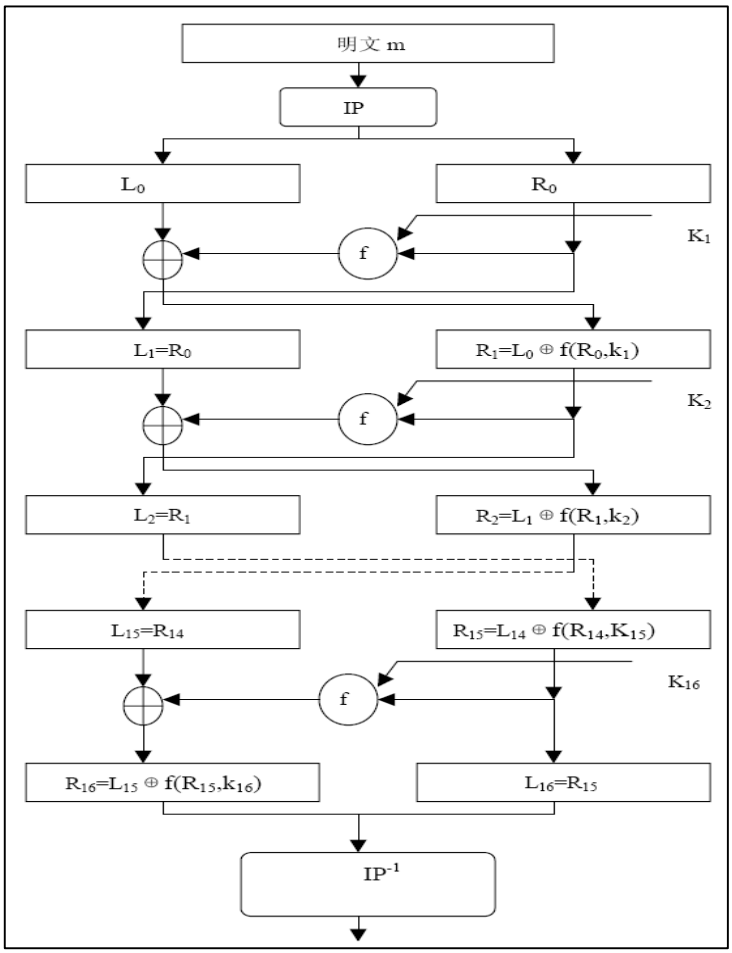


图 1－1 DES 加密/解密流程

2. 基本函数

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 位换到第 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。

58 50 42 34 26 18 10 02

60 52 44 36 28 20 12 04

62 54 46 38 30 22 14 06

64 56 48 40 32 24 16 08

57 49 41 33 25 17 09 01

59 51 43 35 27 19 11 03

61 53 45 37 29 21 13 05

63 55 47 39 31 23 15 07

R_0 和 K_1 经过 f（R_0，K_1）变换后的输出结果，再和 L_0 进行异或运算，输出结果位 R_1，R_0 则赋给 L_1。L_1 和 R_1 同样再做类似运算生成 L_2 和 R_2，…，**经过 16 次运算后生成 L_{16} 和 R_{16}**。

(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。R_i 经过**扩展运算 E** 变换后扩展为 48 位的 E（R_i），与 K_{i+1} 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** $f(R_i, K_{i+1})$。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。

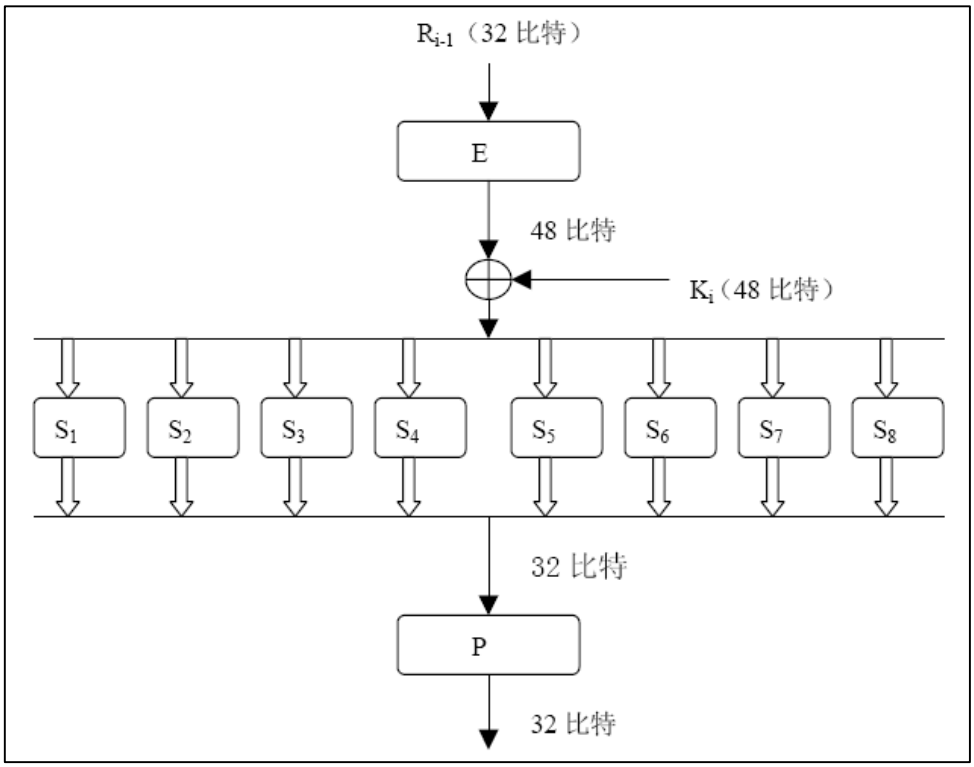


图 1—2 F 函数原理图

(3) 逆初始置换 IP^{-1}

它将 L_{16} 和 R_{16} 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**，置换规则如下所列：

40，8，48，16，56，24，64，32，39，7，47，15，55，23，63，31

38，6，46，14，54，22，62，30，37，5，45，13，53，21，61，29

36，4，44，12，52，20，60，28，35，3，43，11，51，19，59，27

34，2，42，10，50，18，58，26，33，1，41，9，49，17，57，25

(4) 子秘钥生成模块

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**，不参与 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**平分为两部分 C_0，D_0**。然后**分别进行第一次循环左移**，得到 C_1 和 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**（表 1－3），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表 1－1 的规则进行**。

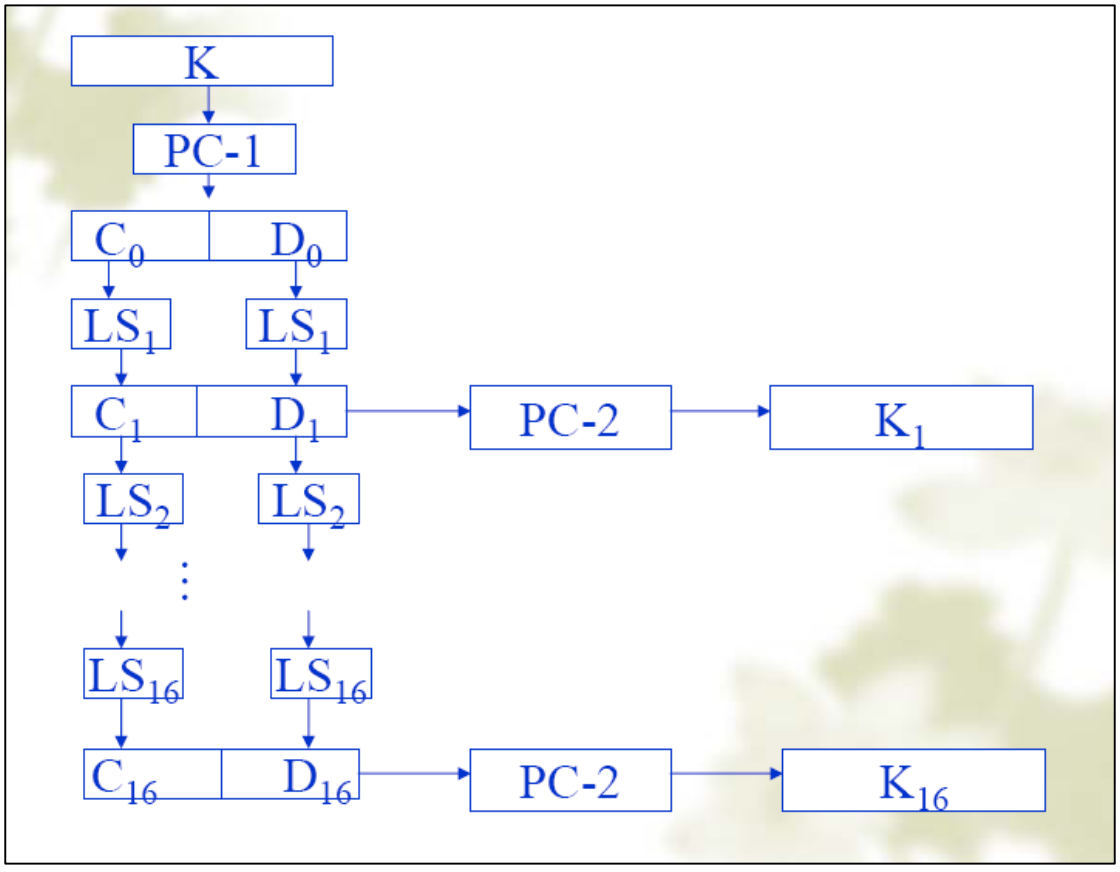


图 1－3 子密钥的生成流程

表 1－1 左移位数规则

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LS_i	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

表 1－2 压缩置换 PC－1

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 1－3 压缩置换 PC－2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	3	48
44	49	39	56	34	53
46	42	50	36	29	32

三、 实验要求

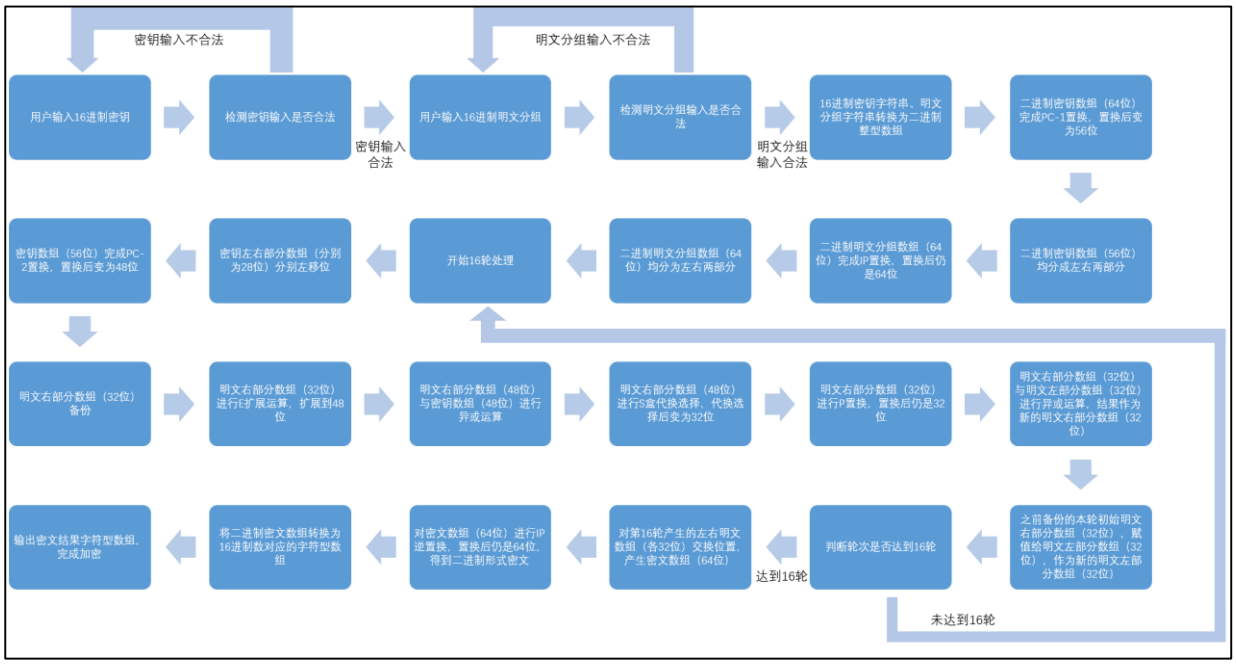
1. DES 实现程序的总体设计：在第一步的基础上，对整个 **DES 加密函数**的实现进行**总体设计**，考虑数据的存储格式，参数的传递格式，程序实现的总体层次等，**画出程序实现的流程图**。
2. 在总体设计完成后，开始**具体的编码**，在编码过程中，注意要尽量使用高效的编码方式。

3. 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、 实验内容

1. DES 加密程序整体流程图

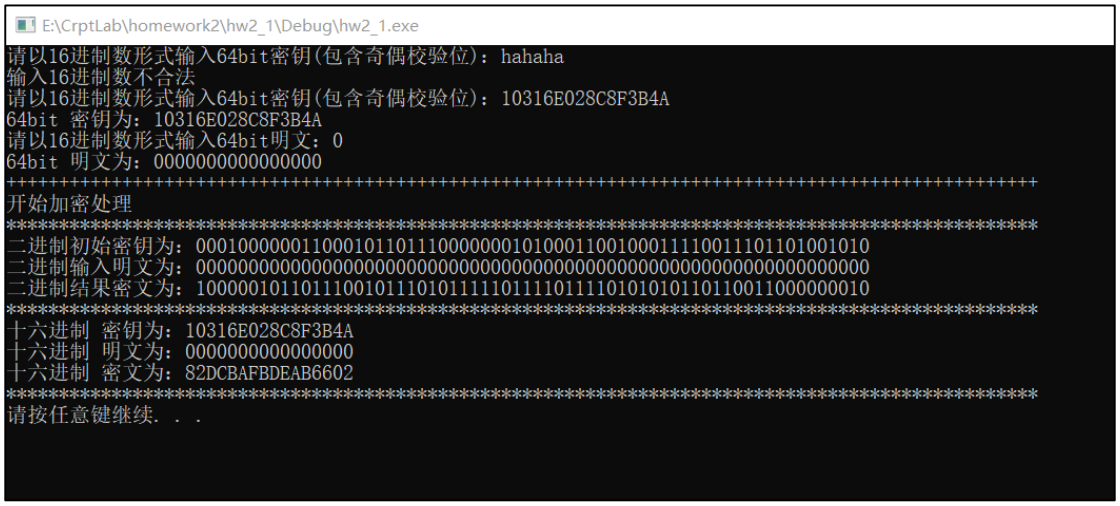
（**图片 png 文件已经单独附在压缩包中上交**，如查看本图，建议放大查看）



2. DES 程序实现结果

(1) 加密过程程序实现结果

（相关程序文件见工程文件 hw2_1）

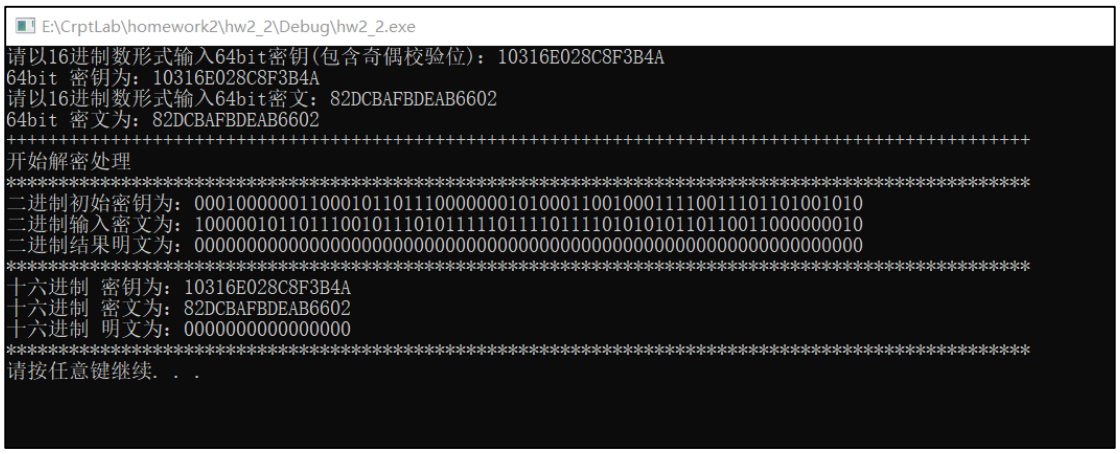


与所给定的测试数据结果相吻合：

```
{ 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A },
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
{ 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02 }
```

(2) 解密过程程序实现结果

（相关程序文件见工程文件 hw2_2）



与所给定的测试数据结果相吻合：

```
{ 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A },
{ 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02 },
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
```

3. DES 程序实现代码

(1) 程序所使用的常量数据表

相关代码见 hw2_1 和 hw2_2 工程文件中的 hw2_1_data.h 和 hw2_2_data.h 头文件，此处不再展示。

其中定义了程序中要使用的**明文初始置换 IP 表**、**明文逆初始置换 IP^{-1} 表**、**明文 E 扩展选择表**、**明文 P 换位表**、**密钥 PC1 选位表**、**密钥 PC2 选位表**、**左循环移位位数表**以及 **S 盒**。

(2) 相关功能函数实现

相关代码见 hw2_1 和 hw2_2 工程文件中的 hw2_1.h 和 hw2_2.h 头文件。

以 hw2_1 工程文件中的 hw2_1.h 头文件为例：

```
#include <iostream>
#include <string>
#include "hw2_1_data.h"
using namespace std;

char * get_Key(string String_Key_Input); //将用户输入 String 转换为 char 数组 Char_Key

char * get_PlainText(string String_PlainText_Input); //将用户输入 String 转换为 char 数组 Char_PlainText

bool is_input_valid(char * str);//检查用户输入 16 进制数是否合法

void Hex_to_Bin(char * Hex, int * Bin);//十六进制 char 数组转换为二进制 int 数组

void Bin_to_Hex(int * Bin, char * Hex);//二进制 int 数组转换为十六进制 char 数组

//将用户输入 String 转换为 char 数组 Char_Key
char * get_Key(string String_Key_Input)
{
    //截取前 16 位十六进制数作为密钥
    string String_Key = String_Key_Input.substr(0,16);

    //不足 16 位十六进制数则末尾补 0 至 16 位十六进制数
    if(String_Key.size()<16)
    {
        int temp_String_Key_size = String_Key.size();
        for(int i=1;i<=16 - temp_String_Key_size;i++)
        {
```

```
            String_Key = String_Key + '0';
        }
    }

    char* Char_Key = new char[17];
    String_Key.copy(Char_Key, 16, 0);
    Char_Key[16] = '\0';

    return Char_Key;
}

//将用户输入 String 转换为 char 数组 Char_PlainText
char * get_PlainText(string String_PlainText_Input)
{
    //截取前 16 位十六进制数作为明文
    string String_PlainText = String_PlainText_Input.substr(0,16);

    //不足 16 位十六进制数则末尾补 0 至 16 位十六进制数
    if(String_PlainText.size()<16)
    {
        int temp_String_PlainText_size = String_PlainText.size();
        for(int i=1;i<=16 - temp_String_PlainText_size;i++)
        {
            String_PlainText = String_PlainText + '0';
        }
    }

    char* Char_PlainText = new char[17];
    String_PlainText.copy(Char_PlainText, 16, 0);
    Char_PlainText[16] = '\0';

    return Char_PlainText;
}

//检查用户输入 16 进制数是否合法
bool is_input_valid(char * str)
{
    for(int i=0;i<=15;i++)
    {
        if(!( (str[i]>='A'&&str[i]<='F') || (str[i]>='a'&&str[i]<='f')
||(str[i]>='0'&&str[i]<='9') ))
        {
            return 0;
        }
    }
    return 1;
}

//十六进制 char 数组转换为二进制 int 数组
void Hex_to_Bin(char * Hex, int * Bin)
{
    int i=0,k=0,j=0;

    for(i=0;i<16;i++)
    {
```

```
        int val = 0;
        if(Hex[i] >= '0' && Hex[i] <= '9')
            val = Hex[i]-'0';
        else if (Hex[i] >= 'A' && Hex[i] <= 'F')
            val = Hex[i]-'A'+10;
        else if (Hex[i] >= 'a' && Hex[i] <= 'f')
            val = Hex[i]-'a'+10;

        for(int k=4*i+3;k>=4*i;k--)
        {
            Bin[k] = val % 2;
            val/=2;
        }
    }
}

//二进制 int 数组转换为十六进制 char 数组
void Bin_to_Hex(int * Bin, char * Hex)
{
    int i=0,k=0,j=0;

    for(i=0;i<16;i++)
    {
        int val = 0;

        for(int k=4*i;k<=4*i+3;k++)
        {
            val *= 2;
            val += Bin[k];
        }

        if( val<10 )
            Hex[i] = val+'0';
        else if (val >=10 && val <=15 )
            Hex[i] = val-10+'A';
    }

    Hex[16]='\0';
}
```

(3) 加密过程函数实现

相关代码见 hw2_1 工程文件中的 hw2_1.cpp 源文件。

```
#include "hw2_1.h"

int main()
{
    //用户输入密钥
    string String_Key_Input;
    cout <<"请以 16 进制数形式输入 64bit 密钥(包含奇偶校验位)： " ;
    cin >> String_Key_Input;
```

```
    //根据用户输入的 string 获得 char 数组
    char * Char_Key = get_Key(String_Key_Input);

    //检查密钥输入是否合法
    while(true)
    {
        if(is_input_valid(Char_Key))
        {
            cout<<"64bit 密钥为："<< Char_Key<<endl;
            break;
        }
        else
        {
            cout<<"输入 16 进制数不合法"<<endl;
            cout <<"请以 16 进制数形式输入 64bit 密钥(包含奇偶校验位)：" ;
            cin >> String_Key_Input;
            Char_Key = get_Key(String_Key_Input);
        }
    }

    //用户输入明文
    string String_PlainText_Input;
    cout <<"请以 16 进制数形式输入 64bit 明文：" ;
    cin >> String_PlainText_Input;

    //根据用户输入的 string 获得 char 数组
    char * Char_PlainText = get_PlainText(String_PlainText_Input);

    //检查明文输入是否合法
    while(true)
    {
        if(is_input_valid(Char_PlainText))
        {
            cout<<"64bit 明文为："<< Char_PlainText<<endl;
            break;
        }
        else
        {
            cout<<"输入 16 进制数不合法"<<endl;
            cout <<"请以 16 进制数形式输入 64bit 明文：" ;
            cin >> String_PlainText_Input;
            Char_PlainText = get_PlainText(String_PlainText_Input);
        }
    }

    cout<<"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++"<<endl;
    cout<<"开始加密处理"<<endl;
    cout<<"**********************************************************
**********************************"<<endl;

    //用于保存用户输入的 64bit 二进制密钥
    int Bit_Key_Init[64];
```

```
//将输入的十六进制密钥 char 数组转换为二进制 int 数组
Hex_to_Bin(Char_Key, Bit_Key_Init);
cout<<"二进制初始密钥为：";
for(int i=0;i<64;i++)
    cout<<Bit_Key_Init[i];
cout<<endl;

//用于保存用户输入的 64bit 二进制明文
int Bit_PlainText_Init[64];

//将输入的十六进制明文 char 数组转换为二进制 int 数组
Hex_to_Bin(Char_PlainText, Bit_PlainText_Init);
cout<<"二进制输入明文为：";
for(int i=0;i<64;i++)
    cout<<Bit_PlainText_Init[i];
cout<<endl;

//PC-1 置换后的二进制密钥数组(56bit)
int Bit_Key_aft_PC1[56];

//密钥完成初始置换 PC-1
for(int m=0;m<56;m++)
{
    Bit_Key_aft_PC1[m]=Bit_Key_Init[PC1_Table[m]-1];
}

//把密钥分为左右两半
int Bit_Key_l1[28],Bit_Key_r1[28];
for(int m=0;m<56;m++)
{
    if(m<28)
        Bit_Key_l1[m]=Bit_Key_aft_PC1[m];
    else
        Bit_Key_r1[m-28]=Bit_Key_aft_PC1[m];
}

//IP 置换后的二进制明文数组(64bit)
int Bit_PlainText_aft_IP[64];

//明文完成初始 IP 置换
for(int m=0;m<64;m++)
{
    Bit_PlainText_aft_IP[m]=Bit_PlainText_Init[IP_Table[m]-1];
}

//把明文分为左右两半
int Bit_PlainText_l1[32],Bit_PlainText_r1[32];
for(int m=0;m<64;m++)
{
    if(m<32)
        Bit_PlainText_l1[m]=Bit_PlainText_aft_IP[m];
    else
        Bit_PlainText_r1[m-32]=Bit_PlainText_aft_IP[m];
}
```

```
    //存储每轮子密钥用于解密
    int Round_Key_Store[16][48];

    //存储每轮移位后的左右密钥
    int Bit_Key_l2[28],Bit_Key_r2[28];

    //存储左右明文
    int Bit_PlainText_l2[32],Bit_PlainText_r2[32];
    int Bit_PlainText_l3[32],Bit_PlainText_r3[48];

    //16 轮次结构
    for(int round=0;round<16;round++)
    {
        //密钥左移位
        for(int m=0;m<28;m++)
        {
            Bit_Key_l2[m]=Bit_Key_l1[(m+LOOP_Table[round])%28];
            Bit_Key_r2[m]=Bit_Key_r1[(m+LOOP_Table[round])%28];
        }

        //更新左右密钥，用于下次循环使用
        for(int m=0;m<28;m++)
        {
            Bit_Key_l1[m]=Bit_Key_l2[m];
            Bit_Key_r1[m]=Bit_Key_r2[m];
        }

        //密钥完成 PC-2 置换
        for(int m=0;m<48;m++)
        {
            if(PC2_Table[m]<29)
            {
                Round_Key_Store[round][m]=Bit_Key_l2[PC2_Table[m]-1];
            }
            else
            {
                Round_Key_Store[round][m]=Bit_Key_r2[PC2_Table[m]-29];
            }
        }

        //对右明文备份
        for(int m=0;m<32;m++)
        {
            Bit_PlainText_r2[m]=Bit_PlainText_r1[m];
        }

        //对右明文进行 E 扩展运算
        for(int m=0;m<48;m++)
        {
            Bit_PlainText_r3[m]=Bit_PlainText_r1[E_Table[m]-1];
        }

        //右明文与密钥做异或运算
        for(int m=0;m<48;m++)
        {
```

```
            Bit_PlainText_r3[m]=Bit_PlainText_r3[m]^Round_Key_Store[rou
nd][m];
        }

        //右明文 S 盒代换
        for(int m=0;m<8;m++)
        {
            //根据右明文的值从 S_Box 中选出值
            int s;
            s=S_Box[(Bit_PlainText_r3[m*6]*2+Bit_PlainText_r3[5+m*6])+m
*4][Bit_PlainText_r3[1+m*6]*8+Bit_PlainText_r3[2+m*6]*4+Bit_PlainText_r
3[3+m*6]*2+Bit_PlainText_r3[4+m*6]];

            //将 S_Box 中选中的值表示为二进制 并 赋值到 Bit_PlainText_r3 中
            for(int n=1;n<=4;n++)
            {
                Bit_PlainText_r3[(4-n)+4*m]=s%2;
                s=s/2;
            }
        }

        //右明文完成 P 置换,与左明文异或,并将备份的本轮原始右明文传给左明文,
用于下次循环使用
        for(int m=0;m<32;m++)
        {
            Bit_PlainText_r1[m]=Bit_PlainText_r3[P_Table[m]-
1]^Bit_PlainText_l1[m];
            Bit_PlainText_l1[m]=Bit_PlainText_r2[m];
        }
    }

    //逆初始置换 IPR 置换前的二进制密文数组(64bit)
    int Bit_CipherText_bfr_IPR[64];

    //16 轮后明文左右交换形成密文
    for(int m=0;m<64;m++)
    {
        if(m<32)
            Bit_CipherText_bfr_IPR[m]=Bit_PlainText_r1[m];
        else
            Bit_CipherText_bfr_IPR[m]=Bit_PlainText_l1[m-32];
    }

    //逆初始置换 IPR 置换后的二进制密文数组(64bit)
    int Bit_CipherText_aft_IPR[64];

    //密文完成逆初始置换
    for(int m=0;m<64;m++)
    {
        Bit_CipherText_aft_IPR[m]=Bit_CipherText_bfr_IPR[IPR_Table[m]-
1];
    }

    cout<<"二进制结果密文为: ";
    for(int m=0;m<64;m++)
```

```
    {
        cout<<Bit_CipherText_aft_IPR[m];
    }
    cout<<endl;

    //存储密文十六进制 char 数组
    char Char_CipherText[17];

    //将得到的二进制密文 int 数组转化为十六进制 char 数组
    Bin_to_Hex(Bit_CipherText_aft_IPR, Char_CipherText);

    //输出 16 进制密钥、明文、密文，方便对比结果
    cout<<"**************************************************************
************************************"<<endl;

    //输出密钥
    cout<<"十六进制 密钥为："<<Char_Key<<endl;

    //输出明文
    cout<<"十六进制 明文为："<<Char_PlainText<<endl;

    //输出密文
    cout<<"十六进制 密文为："<<Char_CipherText<<endl;

    cout<<"**************************************************************
************************************"<<endl;
    system("pause");
    return 0;
}
```

(4) 解密过程函数实现

相关代码见 hw2_2 工程文件中的 hw2_2.cpp 源文件。

```
#include "hw2_2.h"

int main()
{
    //用户输入密钥
    string String_Key_Input;
    cout <<"请以 16 进制数形式输入 64bit 密钥(包含奇偶校验位)： " ;
    cin >> String_Key_Input;

    //根据用户输入的 string 获得 char 数组
    char * Char_Key = get_Key(String_Key_Input);

    //检查密钥输入是否合法
    while(true)
    {
        if(is_input_valid(Char_Key))
```

```
        {
            cout<<"64bit 密钥为："<< Char_Key<<endl;
            break;
        }
        else
        {
            cout<<"输入 16 进制数不合法"<<endl;
            cout <<"请以 16 进制数形式输入 64bit 密钥(包含奇偶校验位)： " ;
            cin >> String_Key_Input;
            Char_Key = get_Key(String_Key_Input);
        }
    }

    //用户输入密文
    string String_CipherText_Input;
    cout <<"请以 16 进制数形式输入 64bit 密文： " ;
    cin >> String_CipherText_Input;

    //根据用户输入的 string 获得 char 数组
    char * Char_CipherText = get_CipherText(String_CipherText_Input);

    //检查密文输入是否合法
    while(true)
    {
        if(is_input_valid(Char_CipherText))
        {
            cout<<"64bit 密文为："<< Char_CipherText<<endl;
            break;
        }
        else
        {
            cout<<"输入 16 进制数不合法"<<endl;
            cout <<"请以 16 进制数形式输入 64bit 密文： " ;
            cin >> String_CipherText_Input;
            Char_CipherText = get_CipherText(String_CipherText_Input);
        }
    }

    cout<<"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++"<<endl;
    cout<<"开始解密处理"<<endl;
    cout<<"**********************************************************
**********************************"<<endl;

    //用于保存用户输入的 64bit 二进制密钥
    int Bit_Key_Init[64];

    //将输入的十六进制密钥 char 数组转换为二进制 int 数组
    Hex_to_Bin(Char_Key, Bit_Key_Init);
    cout<<"二进制初始密钥为：";
    for(int i=0;i<64;i++)
        cout<<Bit_Key_Init[i];
    cout<<endl;
```

```
//用于保存用户输入的 64bit 二进制密文
int Bit_CipherText_Init[64];

//将输入的十六进制密文 char 数组转换为二进制 int 数组
Hex_to_Bin(Char_CipherText, Bit_CipherText_Init);
cout<<"二进制输入密文为：";
for(int i=0;i<64;i++)
    cout<<Bit_CipherText_Init[i];
cout<<endl;

//-----------------子密钥获取------------------------------
//先要获得 16 轮子密钥

//PC-1 置换后的二进制密钥数组(56bit)
int Bit_Key_aft_PC1[56];

//密钥完成初始置换 PC-1
for(int m=0;m<56;m++)
{
    Bit_Key_aft_PC1[m]=Bit_Key_Init[PC1_Table[m]-1];
}

//把密钥分为左右两半
int Bit_Key_l1[28],Bit_Key_r1[28];
for(int m=0;m<56;m++)
{
    if(m<28)
        Bit_Key_l1[m]=Bit_Key_aft_PC1[m];
    else
        Bit_Key_r1[m-28]=Bit_Key_aft_PC1[m];
}

//存储每轮子密钥用于解密
int Round_Key_Store[16][48];

//存储每轮移位后的左右密钥
int Bit_Key_l2[28],Bit_Key_r2[28];

//16 轮次结构
for(int round=0;round<16;round++)
{
    //密钥左移位
    for(int m=0;m<28;m++)
    {
        Bit_Key_l2[m]=Bit_Key_l1[(m+LOOP_Table[round])%28];
        Bit_Key_r2[m]=Bit_Key_r1[(m+LOOP_Table[round])%28];
    }

    //更新左右密钥，用于下次循环使用
    for(int m=0;m<28;m++)
    {
        Bit_Key_l1[m]=Bit_Key_l2[m];
        Bit_Key_r1[m]=Bit_Key_r2[m];
    }
```

```
        //密钥完成 PC-2 置换
        for(int m=0;m<48;m++)
        {
            if(PC2_Table[m]<29)
            {
                Round_Key_Store[round][m]=Bit_Key_l2[PC2_Table[m]-1];
            }
            else
            {
                Round_Key_Store[round][m]=Bit_Key_r2[PC2_Table[m]-29];
            }
        }
    }
    //-----------------子密钥获取结束-------------------------

    //IP 置换后的二进制密文数组(64bit)
    int Bit_CipherText_aft_IP[64];

    //密文完成初始 IP 置换
    for(int m=0;m<64;m++)
    {
        Bit_CipherText_aft_IP[m]=Bit_CipherText_Init[IP_Table[m]-1];
    }

    //把密文分为左右两半
    int Bit_CipherText_l1[32],Bit_CipherText_r1[32];
    for(int m=0;m<64;m++)
    {
        if(m<32)
            Bit_CipherText_l1[m]=Bit_CipherText_aft_IP[m];
        else
            Bit_CipherText_r1[m-32]=Bit_CipherText_aft_IP[m];
    }

    //存储左右明文
    int Bit_CipherText_l2[32],Bit_CipherText_r2[32];
    int Bit_CipherText_l3[32],Bit_CipherText_r3[48];

    //16 轮次结构
    for(int round=0;round<16;round++)
    {

        //对右明文备份
        for(int m=0;m<32;m++)
        {
            Bit_CipherText_r2[m]=Bit_CipherText_r1[m];
        }

        //对右明文进行 E 扩展运算
        for(int m=0;m<48;m++)
        {
            Bit_CipherText_r3[m]=Bit_CipherText_r1[E_Table[m]-1];
        }

        //右明文与对应的轮密钥做异或运算
```

```
        for(int m=0;m<48;m++)
        {
            Bit_CipherText_r3[m]=Bit_CipherText_r3[m]^Round_Key_Store[1
5-round][m];
        }

        //右明文 S 盒代换
        for(int m=0;m<8;m++)
        {
            //根据右明文的值从 S_Box 中选出值
            int s;
            s=S_Box[(Bit_CipherText_r3[m*6]*2+Bit_CipherText_r3[5+m*6])
+m*4][Bit_CipherText_r3[1+m*6]*8+Bit_CipherText_r3[2+m*6]*4+Bit_CipherT
ext_r3[3+m*6]*2+Bit_CipherText_r3[4+m*6]];

            //将 S_Box 中选中的值表示为二进制 并 赋值到 Bit_PlainText_r3 中
            for(int n=1;n<=4;n++)
            {
                Bit_CipherText_r3[(4-n)+4*m]=s%2;
                s=s/2;
            }
        }

        //右明文完成 P 置换,与左明文异或,并将备份的本轮原始右明文传给左明文,
用于下次循环使用
        for(int m=0;m<32;m++)
        {
            Bit_CipherText_r1[m]=Bit_CipherText_r3[P_Table[m]-
1]^Bit_CipherText_l1[m];
            Bit_CipherText_l1[m]=Bit_CipherText_r2[m];
        }
    }

    //逆初始置换 IPR 置换前的二进制明文数组(64bit)
    int Bit_PlainText_bfr_IPR[64];

    //16 轮后密文左右交换得到明文
    for(int m=0;m<64;m++)
    {
        if(m<32)
            Bit_PlainText_bfr_IPR[m]=Bit_CipherText_r1[m];
        else
            Bit_PlainText_bfr_IPR[m]=Bit_CipherText_l1[m-32];
    }

    //逆初始置换 IPR 置换后的二进制明文数组(64bit)
    int Bit_PlainText_aft_IPR[64];

    //明文完成逆初始置换
    for(int m=0;m<64;m++)
    {
        Bit_PlainText_aft_IPR[m]=Bit_PlainText_bfr_IPR[IPR_Table[m]-1];
    }

    cout<<"二进制结果明文为: ";
```

```
    for(int m=0;m<64;m++)
    {
        cout<<Bit_PlainText_aft_IPR[m];
    }
    cout<<endl;

    //存储明文十六进制 char 数组
    char Char_PlainText[17];

    //将得到的二进制明文 int 数组转化为十六进制 char 数组
    Bin_to_Hex(Bit_PlainText_aft_IPR, Char_PlainText);

    //输出 16 进制密钥、明文、密文，方便对比结果
    cout<<"**************************************************************
************************************"<<endl;

    //输出密钥
    cout<<"十六进制 密钥为："<<Char_Key<<endl;

    //输出密文
    cout<<"十六进制 密文为："<<Char_CipherText<<endl;

    //输出明文
    cout<<"十六进制 明文为："<<Char_PlainText<<endl;

    cout<<"**************************************************************
************************************"<<endl;
    system("pause");
    return 0;
}

//将用户输入 String 转换为 char 数组 Char_Key
char * get_Key(string String_Key_Input)
{
    //截取前 16 位十六进制数作为密钥
    string String_Key = String_Key_Input.substr(0,16);

    //不足 16 位十六进制数则末尾补 0 至 16 位十六进制数
    if(String_Key.size()<16)
    {
        int temp_String_Key_size = String_Key.size();
        for(int i=1;i<=16 - temp_String_Key_size;i++)
        {
            String_Key = String_Key + '0';
        }
    }

    char* Char_Key = new char[17];
    String_Key.copy(Char_Key, 16, 0);
    Char_Key[16] = '\0';

    return Char_Key;
}

//将用户输入 String 转换为 char 数组 Char_CipherText
```

```
char * get_CipherText(string String_CipherText_Input)
{
    //截取前 16 位十六进制数作为明文
    string String_CipherText = String_CipherText_Input.substr(0,16);

    //不足 16 位十六进制数则末尾补 0 至 16 位十六进制数
    if(String_CipherText.size()<16)
    {
        int temp_String_CipherText_size = String_CipherText.size();
        for(int i=1;i<=16 - temp_String_CipherText_size;i++)
        {
            String_CipherText = String_CipherText + '0';
        }
    }

    char* Char_CipherText = new char[17];
    String_CipherText.copy(Char_CipherText, 16, 0);
    Char_CipherText[16] = '\0';

    return Char_CipherText;
}

//检查用户输入 16 进制数是否合法
bool is_input_valid(char * str)
{
    for(int i=0;i<=15;i++)
    {
        if(!( (str[i]>='A'&&str[i]<='F') || (str[i]>='a'&&str[i]<='f')
||(str[i]>='0'&&str[i]<='9') ))
        {
            return 0;
        }
    }
    return 1;
}

//十六进制 char 数组转换为二进制 int 数组
void Hex_to_Bin(char * Hex, int * Bin)
{
    int i=0,k=0,j=0;

    for(i=0;i<16;i++)
    {
        int val = 0;
        if(Hex[i] >= '0' && Hex[i] <= '9')
            val = Hex[i]-'0';
        else if (Hex[i] >= 'A' && Hex[i] <= 'F')
            val = Hex[i]-'A'+10;
        else if (Hex[i] >= 'a' && Hex[i] <= 'f')
            val = Hex[i]-'a'+10;

        for(int k=4*i+3;k>=4*i;k--)
        {
            Bin[k] = val % 2;
            val/=2;
        }
```

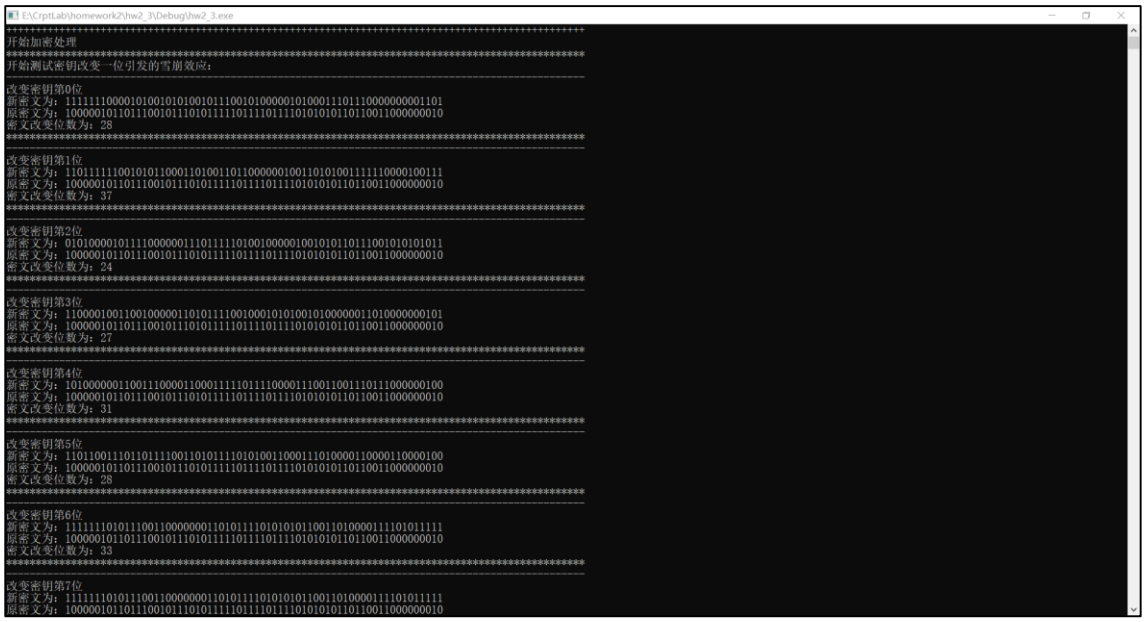
```
    }
}

//二进制 int 数组转换为十六进制 char 数组
void Bin_to_Hex(int * Bin, char * Hex)
{
    int i=0,k=0,j=0;

    for(i=0;i<16;i++)
    {
        int val = 0;

        for(int k=4*i;k<=4*i+3;k++)
        {
            val *= 2;
            val += Bin[k];
        }

        if( val<10 )
            Hex[i] = val+'0';
        else if (val >=10 && val <=15 )
            Hex[i] = val-10+'A';
    }

    Hex[16]='\0';
}
```

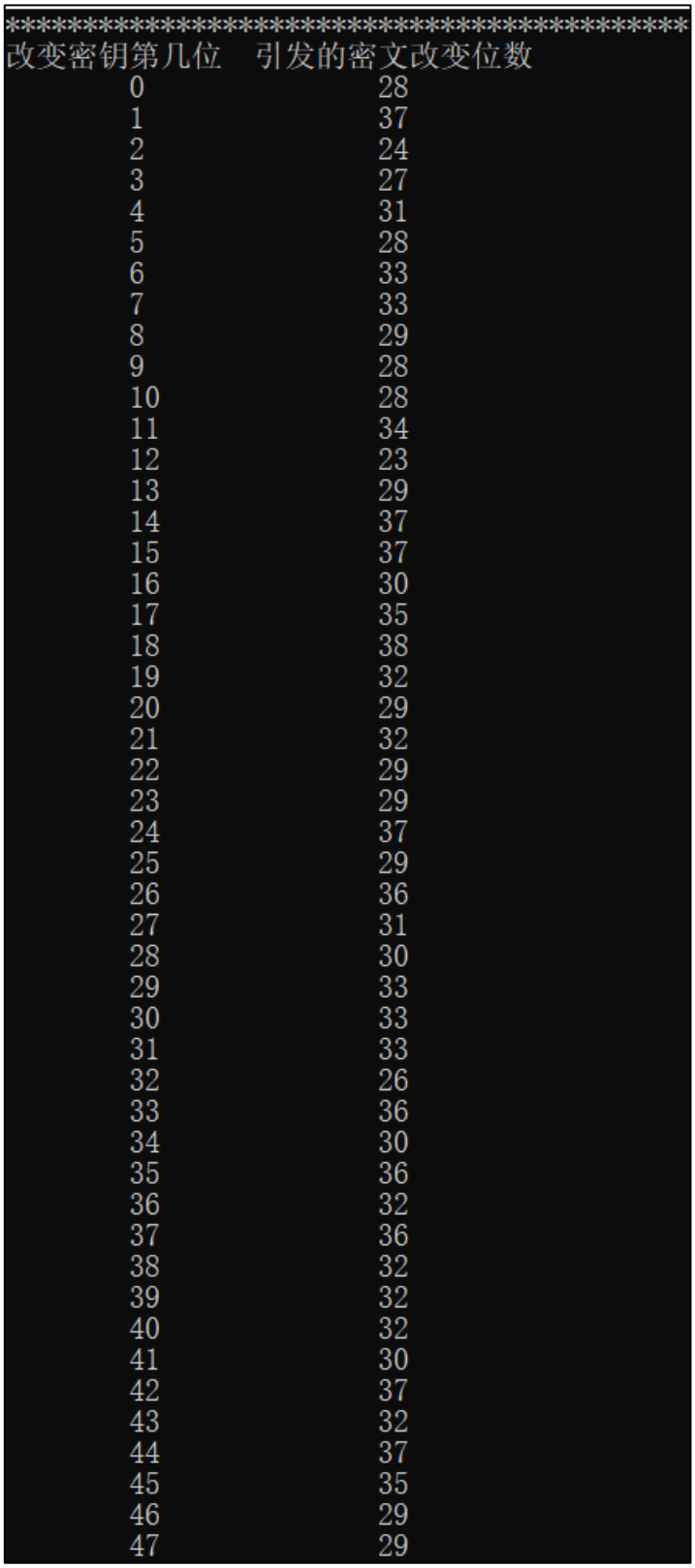
4. 雪崩效应测试代码及结果

(1) 测试密钥改变一位引发的密文改变的位数

（相关程序文件及代码见工程文件 hw2_3）

测试程序运行结果如下：(可以放大查看)


E:\CrprtLab\homework2\hw2_3\Debug\hw2_3.exe
开始加密处理
开始测试密钥改变一位引发的雪崩效应：
改变密钥第0位
新密文为：11111110000101001010100101110010100001010001110111000000001101
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：28
改变密钥第1位
新密文为：11011111100101011000110100111011000000100110101001111100001001111
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：37
改变密钥第2位
新密文为：01010000101111000000111011111010010000010010101101100101010101
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：24
改变密钥第3位
新密文为：11000010011001000001101011110010001010100101000001101000000101
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：27
改变密钥第4位
新密文为：10100000011001110000110001111101111000011100110011101100000100
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：31
改变密钥第5位
新密文为：11011001110110111100110101111010100110001110100001100001100100
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：28
改变密钥第6位
新密文为：11111110101110011000000011010111101010101100110100001111010111111
原密文为：10000010110111001011101011111011101111010101011001100000010
密文改变位数为：33
改变密钥第7位
新密文为：11111110101110011000000011010111101010101100110100001111010111111
原密文为：10000010110111001011101011111011101111010101011001100000010



```
**********************************************
改变密钥第几位    引发的密文改变位数
        0                28
        1                37
        2                24
        3                27
        4                31
        5                28
        6                33
        7                33
        8                29
        9                28
        10               28
        11               34
        12               23
        13               29
        14               37
        15               37
        16               30
        17               35
        18               38
        19               32
        20               29
        21               32
        22               29
        23               29
        24               37
        25               29
        26               36
        27               31
        28               30
        29               33
        30               33
        31               33
        32               26
        33               36
        34               30
        35               36
        36               32
        37               36
        38               32
        39               32
        40               32
        41               30
        42               37
        43               32
        44               37
        45               35
        46               29
        47               29
```

测试结果数据如下：

改变密钥中的一位，引发密文改变的位数为：

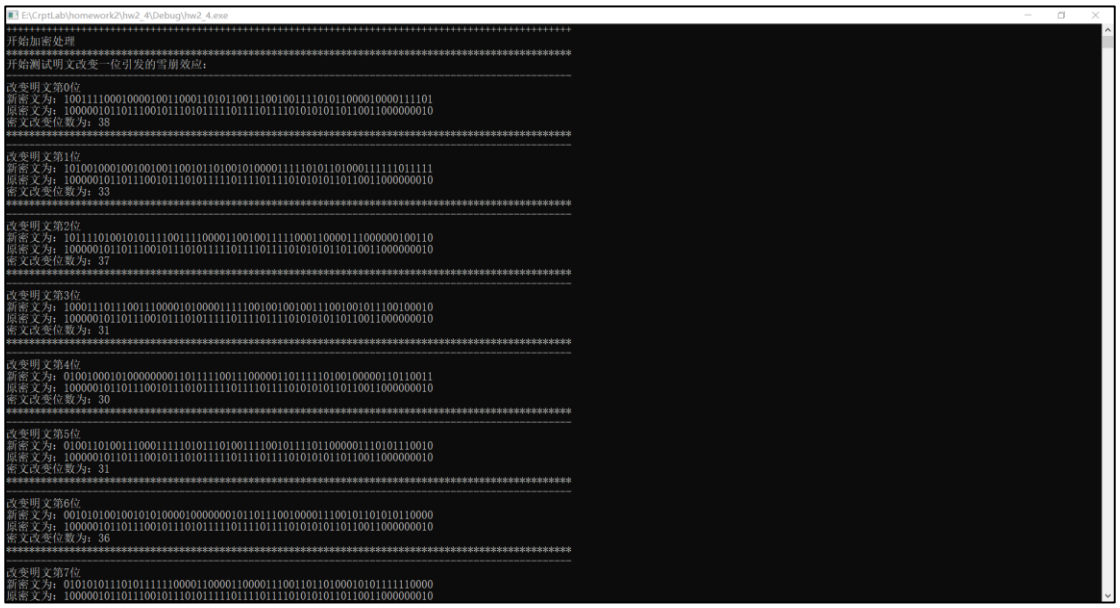
改变密钥第几位	引发的密文改变位数	改变密钥第几位	引发的密文改变位数
0	28	32	26
1	37	33	36
2	24	34	30
3	27	35	36
4	31	36	32
5	28	37	36
6	33	38	32
7	33	39	32
8	29	40	32
9	28	41	30
10	28	42	37
11	34	43	32
12	23	44	37
13	29	45	35
14	37	46	29
15	37	47	29
16	30	48	31
17	35	49	38
18	38	50	26
19	32	51	34
20	29	52	35
21	32	53	37
22	29	54	32
23	29	55	32
24	37	56	34
25	29	57	28
26	36	58	29
27	31	59	31
28	30	60	30
29	33	61	34
30	33	62	30
31	33	63	30

计算平均值，得到密钥改变一位，引发的密文改变的平均位数为：31.7813 位；该值接近 32，说明**密钥中改变一位，将使密文中约一半的位的值发生改变**。

(2) 测试明文改变一位引发的密文改变的位数

（相关程序文件及代码见工程文件 hw2_4）

测试程序运行结果如下：(可以放大查看)



```
**************************************************
改变明文第几位    引发的密文改变位数
        0                38
        1                33
        2                37
        3                31
        4                30
        5                31
        6                36
        7                36
        8                31
        9                31
        10               32
        11               30
        12               35
        13               29
        14               28
        15               35
        16               31
        17               31
        18               34
        19               28
        20               30
        21               33
        22               28
        23               40
        24               39
        25               36
        26               36
        27               37
        28               32
        29               34
        30               36
        31               36
        32               31
        33               35
        34               30
        35               30
        36               32
        37               36
        38               32
        39               34
        40               34
        41               29
        42               36
        43               31
        44               39
        45               33
        46               34
        47               37
```

测试结果数据如下：

改变明文中的一位，引发明文改变的位数为：

改变明文第几位	引发的密文改变位数	改变明文第几位	引发的密文改变位数
0	38	32	31
1	33	33	35
2	37	34	30
3	31	35	30
4	30	36	32
5	31	37	36
6	36	38	32
7	36	39	34
8	31	40	34
9	31	41	29
10	32	42	36
11	30	43	31
12	35	44	39
13	29	45	33
14	28	46	34
15	35	47	37
16	31	48	35
17	31	49	32
18	34	50	33
19	28	51	36
20	30	52	27
21	33	53	35
22	28	54	25
23	40	55	29
24	39	56	27
25	36	57	36
26	36	58	25
27	37	59	34
28	32	60	35
29	34	61	39
30	36	62	31
31	36	63	34

计算平均值，得到明文改变一位，引发的密文改变的平均位数为：32.9688 位；该值接近 32，说明**明文中改变一位，将使密文中约一半的位的值发生改变**。