实验目的与要求：

1、 理解对称密码体制和分组密码算法的基本思想
2、 理解分组密码 3DES 的基本原理
3、 实现 3DES 的加解密过程，可以对各种文件（word、txt、mp3、jpg）进行加解密
4、 实现分组密码的密码分组链接工作模式与计算器工作模式

实验环境：

MacOS Visual Studio 2019；C++语言

实验原理：

1、 什么是数据加密标准（DES）

DES 是最广泛使用的对称加密方案，由原美国国家标准局于 1977 年采用。其设置明文分组长度为 64 bit，密钥有效长度为 56 bit，在基于 Feistel 网络的基础上，采用 16 轮迭代，从原始 56 bit 密钥产生 16 组子密钥，每一轮迭代使用一个子密钥。

2、 DES 流程

笔者以其输入输出的比特流为主线进行了简单梳理，可以看到 DES 涉及以下操作。关于每个操作的具体例子笔者也进行了总结，由于实验报告篇幅，笔者发布在了个人网站供进一步查阅：【网络安全】数据加密标准（DES 算法）详细介绍（ 分组密码、Feistel 密码结构、轮函数、子密钥生成算法）。

（1）初始置换（通过置换矩阵，输入 64bit，输出 64bit）

（2）轮函数操作（F 操作）

- 先将输入的 64bit 分为左右两半部分，每个部分 32bit，下面只对右半部分进行操作：
  - 扩展运算 E （输入 32bit，输出 48bit）
  - 与子密钥进行异或操作（输入 48bit，子密钥也是 48bit，输出 48bit）
  - 压缩运算 S（输入 48bit，输出 32bit）（唯一不可逆的操作）
  - 置换运算 P （输入 32bit，输出 32bit）
  - 与上一轮的左半部分进行异或运算（输入 32bit，输出 32bit），得到这一轮的右半部分
- 这一轮的左半部分等于上一轮的左半部分（32bit）。这一轮的右半部分（32bit）通过上面的操作已经得到了。

（3）子密钥产生算法

- 置换选择 1（输入 64bit，输出 56bit）
- 将 56bit 分为 2 个 28bit 的部分（输入 56bit，输出 2*28bit）
- 对 2 个 28bit 的部分进行循环左移操作（输入 2*28bit，输出 2*28bit）
- 将 2 个 28bit 部分合并成一个 56bit（输入 2*28bit，输出 56bit）
- 置换选择 2（输入 56bit，输出 48bit），得到这一轮的子密钥
- （利用这个子密钥跟轮函数扩展运算后的的 48bit 进行异或操作）

（4）逆初始操作（通过逆置换矩阵，输入 64bit，输出 64bit）

对上述流程进行总结，DES 的加密流程如下图 1 所示：

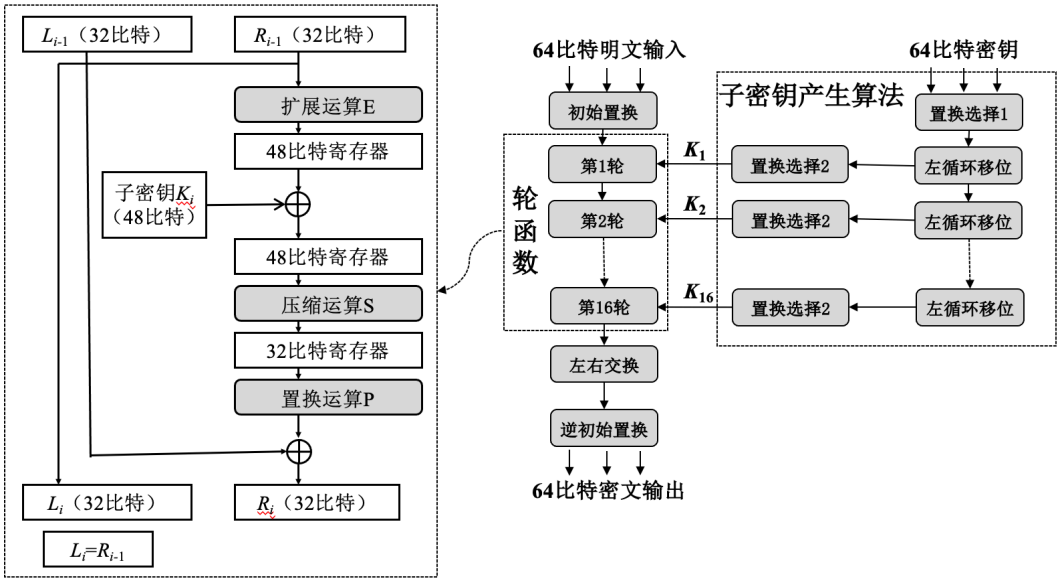

图 1：DES 加密流程图

3、3DES

3DES 是在是三重数据加密算法（TDEA，Triple Data Encryption Algorithm）块密码的通称，它相当于是对每个数据块应用三次 DES 加密算法。设$E_K()$和$D_K()$代表 DES 算法的加密和解密过程，K 代表 DES 算法使用的密钥，M 代表明文，C 代表密文。则

3DES 加密过程为：$C = E_{K3}\big(D_{K2}(E_{K1}(P))\big)$

3DES 解密过程为：$P = D_{K1}\big(E_{K2}(D_{K3}(C))\big)$

具体的 3DES 流程图下如图 2 所示：

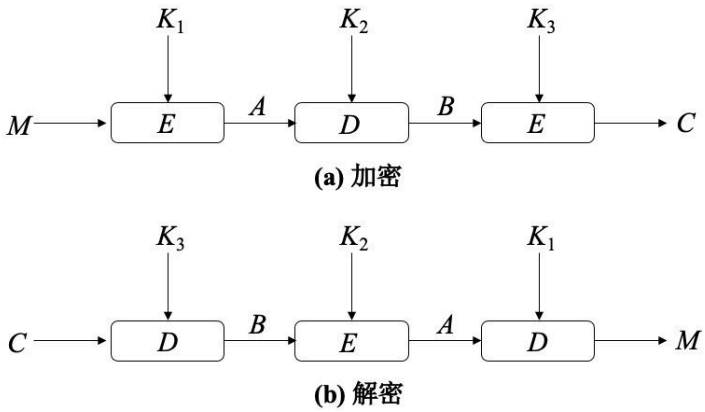

图 2：3DES 加密解密流程图

实验内容：

1、 熟悉 3DES 的加解密过程
2、 采用自己熟悉的编程语言实现 3DES 加密算法
3、 将编写的 3DES 用于加密各种文件（word、txt、mp3、jpg），并能成功解密
4、 实现分组密码的密码分组链接工作模式与计算器工作模式

实验步骤与结果：

1、3DES 加密算法的实现及介绍[1]

1.1 算法流程结构设计

3DES 算法加密过程为：$C = E_{K3}\left(D_{K2}(E_{K1}(P))\right)$，要完成 3DES 加密算法，首先需要完成 DES 加密算法的结构设计，具体过程如下图 3 所示：（解密过程原理一致，只是使用了$P = D_{K1}\left(E_{K2}(D_{K3}(C))\right)$这个解密公式，此处不再赘述。）

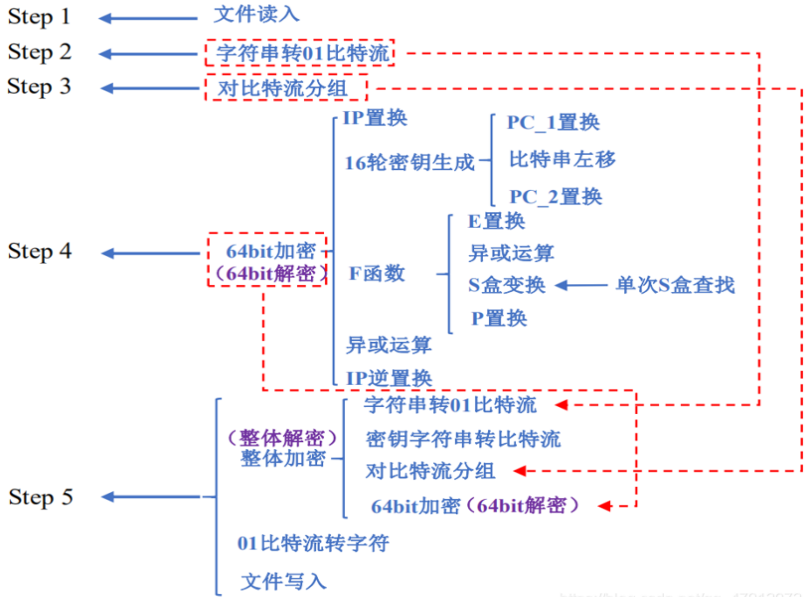

图 3 DES 加密算法实现步骤

笔者是使用 C++来编写此次实验。在开始进行算法前，笔者首先定义一个 Table_DES 类，将后面会使用的函数及其功能编写清楚。Class Table_DES 的结构如图 4 所示：

```
typedef char type; //定义数据类
class Table_DES
{
    int IP[64];      //初始置换表IP
    int IP_1[64];    //逆初始置换表IP_1
    int E[48];       //扩充置换表E
    int P[32];       //置换函数P
    int S[8][4][16]; // S盒
    int PC_1[56];    //置换选择1
    int PC_2[48];    //置换选择2
    int left[16];    //对循环左移次数的规定

    int ByteToBit(type ch, type bit[8]);        //字节转换成二进制
    int BitToByte(type bit[8], type *ch);       //二进制转换成字节
    int Char8ToBit64(type ch[8], type bit[64]); //将长度为8的字符串转为二进制位串
    int Bit64ToChar8(type bit[64], type ch[8]); //将二进制位串转为长度为8的字符串

    int DES_MakeSubKeys(type key[64], type subKeys[16][48]); //生成子密钥
    int DES_PC1_Transform(type key[64], type tempbts[56]);   //密钥置换pc1
    int DES_PC2_Transform(type key[56], type tempbts[48]);   //密钥置换pc2
    int DES_ROL(type data[56], int time);                    //循环左移

    int DES_IP_Transform(type data[64]);   // IP置换
    int DES_IP_1_Transform(type data[64]); // IP逆置换

    int DES_E_Transform(type data[48]);            //扩展置换
    int DES_P_Transform(type data[32]);            // P置换
    int DES_XOR(type R[48], type L[48], int count); //异或

    int DES_SBOX(type data[48]);                // S盒置换
    int DES_Swap(type left[32], type right[32]); //交换

    int DES_EncryptBlock(type plainBlock[8], type subKeys[16][48], type cipherBlock[8]);             //加密单个分组
    int DES_DecryptBlock(type cipherBlock[8], type subKeys[16][48], type plainBlock[8]);             //解密单个分组
    int treble_DES_Encrypt(char *plainFile, type *key_1, type *key_2, type *key_3, char *cipherFile); //加密文件
    int treble_DES_Decrypt(char *cipherFile, type *key_1, type *key_2, type *key_3, char *plainFile); //解密文件
};
```

[1] 参考：https://blog.csdn.net/blowfire123/article/details/78566312

图 4：Table_DES 类结构

1.2 定义标准表变量

首先将后面会使用的标准表先定义清楚，方面后面函数调用。主要包括：初始置换表、逆初始置换表、扩充置换表、置换函数表、S 盒、在子密钥生成中需要的置换选择 1、置换选择 2 以及循环左移操作列表。（示例如图 5 所示，这里由于表太过多了，就不在实验报告里面一一展示）

```
class Table_DES
{
    //初始置换表IP
    int IP[64] = {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,
                  56, 48, 40, 32, 24, 16, 8, 0,
                  58, 50, 42, 34, 26, 18, 10, 2,
                  60, 52, 44, 36, 28, 20, 12, 4,
                  62, 54, 46, 38, 30, 22, 14, 6};

    //逆初始置换表IP_1
    int IP_1[64] = {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,
                    32, 0, 40, 8, 48, 16, 56, 24};
```

图 5:定义标准表变量（部分）

1.3 字符、字节和二进制之间的转换

在将文件加密的过程当中，需要将文件转化为二进制的形式，在解码还原文件的过程中，也需要将二进制还原成字节。所以这里笔者设置两个函数：ByteToBit()将字节转换成二进制形式，BitToByte()将二进制转化为字节形式。

考虑到 DES 加密是将明文分解为 64 bit 一组一组来进行操作的，所以还需要定义两个函数：Char8ToBit64()，Bit64ToChar8()来分别将长度为 8 的字符串转化为二进制的字符串以及将二进制位串转化位长度位 8 的字符串。

主要注意的是，对于字节转为二进制，笔者这里使用的方法是将字节中的每个二进制 bit 位分别写入 bit[8]数组中，并进行 8 次循环。二进制转换成字节类似。而将长度为 8 的字符串转为二进制位串，需要使用到字节转为二进制的函数，同样是八轮循环，将二进制 bit 位分别写入 bit 数组中。

```
//字节转换成二进制
int ByteToBit(type ch, type bit[8])
{
    int cnt;
    for (cnt = 0; cnt < 8; cnt++)
    {
        *(bit + cnt) = (ch >> cnt) & 1;
    }
    return 0;
}
//二进制转换成字节
int BitToByte(type bit[8], type *ch)
{
    int cnt;
    for (cnt = 0; cnt < 8; cnt++)
    {
        *ch |= *(bit + cnt) << cnt;
    }
    return 0;
}
```

```
//将长度为8的字符串转为二进制位串
int Char8ToBit64(type ch[8], type bit[64])
{
    int cnt;
    for (cnt = 0; cnt < 8; cnt++)
    {
        ByteToBit(*(ch + cnt), bit + (cnt << 3));
    }
    return 0;
}
//将二进制位串转为长度为8的字符串
int Bit64ToChar8(type bit[64], type ch[8])
{
    int cnt;
    memset(ch, 0, 8);
    for (cnt = 0; cnt < 8; cnt++)
    {
        BitToByte(bit + (cnt << 3), ch + cnt);
    }
    return 0;
}
```

图 6：字符、字节和二进制之间的转换

1.4 生成子密钥

子密钥生成是指将 64 位的密钥通过置换选择和循环位移后得到 48 位的子密钥。生成子密钥的流程图如图 7 所示：

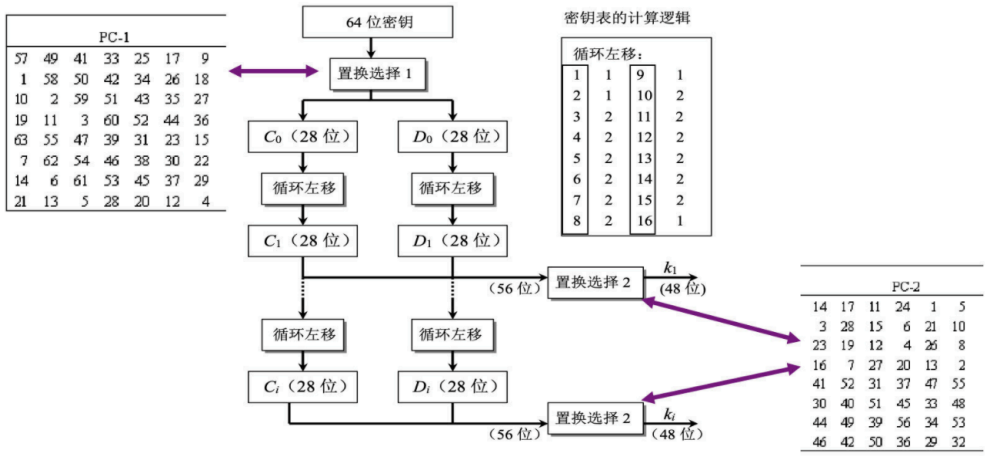

图 7:子密钥生成流程图

具体过程是将密钥通过置换选择 1，转换成前后两部分（分别为 28 位），然后对这两部分分别进行循环左移，然后通过置换选择 2 得到最终的子密钥。这里有一个迭代阶段，在每一次的迭代的过程当中，生成一个密钥，进过 16 次的迭代，就生成了 16 个密钥了。

```
///生成子密钥
int DES_MakeSubKeys(type key[64], type subKeys[16][48])
{
    type temp[56];
    int cnt;
    DES_PC1_Transform(key, temp);              // PC1置换
    for (cnt = 0; cnt < 16; cnt++)
    {                                          // 16轮跌代，产生16个子密钥
        DES_ROL(temp, left[cnt]);              //循环左移
        DES_PC2_Transform(temp, subKeys[cnt]); // PC2置换，产生子密钥
    }
    return 0;
}
```

图 8:子密钥生成函数

在生成子密钥的过程中，涉及到两个密钥置换操作（根据置换选择表进行 56 次转换）以及循环左移操作（因为置换后的数据依然有 56 bit，这里分别对前 28 位的Ci和后 28 位的Di进行循环位移，过程中要先保存将要循环移动到右边的位，然后分别对Ci 和 Di 进行移动），这三个函数如下图 9 所示：

```
//密钥置换pc1
int DES_PC1_Transform(type key[64], type tempbts[56])
{
    int cnt;
    for (cnt = 0; cnt < 56; cnt++)
    {
        tempbts[cnt] = key[PC_1[cnt]];
    }
    return 0;
}
//密钥置换pc2
int DES_PC2_Transform(type key[56], type tempbts[48])
{
    int cnt;
    for (cnt = 0; cnt < 48; cnt++)
    {
        tempbts[cnt] = key[PC_2[cnt]];
    }
    return 0;
}
```

```
//循环左移
int DES_ROL(type data[56], int time)
{
    type temp[56];
    //保存将要循环移动到右边的位
    memcpy(temp, data, time);
    memcpy(temp + time, data + 28, time);
    //前28位移动
    memcpy(data, data + time, 28 - time);
    memcpy(data + 28 - time, temp, time);
    //后28位移动
    memcpy(data + 28, data + 28 + time, 28 - time);
    memcpy(data + 56 - time, temp + time, time);

    return 0;
}
```

图 9:子密钥生成涉及的三个函数

1.5 初始置换和逆置换

在进行轮函数之前，还需要进行初始置换，置换是指将数码中的某一位的值根据置换表的规定，用另一位代替。逆置换虽然是最后一个步骤，但由于和初始置换的原理是一致的，只是变化了一下逆置换矩阵的大小，所以这里一起进行讲解。

```c
// IP初始置换
int DES_IP_Transform(type data[64])
{
    int cnt;
    type temp[64];
    for (cnt = 0; cnt < 64; cnt++)
    {
        temp[cnt] = data[IP[cnt]];
    }
    memcpy(data, temp, 64);
    return 0;
}
```

```c
//IP逆置换
int DES_IP_1_Transform(type data[64])
{
    int cnt;
    type temp[64];
    for (cnt = 0; cnt < 64; cnt++)
    {
        temp[cnt] = data[IP_1[cnt]];
    }
    memcpy(data, temp, 64);
    return 0;
}
```

图 9:初始置换和逆置换矩阵

1.6 轮函数

轮函数的操作流程请参见图 1 左半部分，这里不再赘述。整个过程涉及到以下操作（1）扩展运算（2）与子密钥的异或操作（3）S 盒压缩运算（4）置换运算 P（5）进行交换。接下来我们一一介绍：

（1)扩展运算：首先需要使用到扩展置换 E 来将 32bit 的右半部分数据扩展成 48bit，存储在寄存器中，这里根据 E 表进行 48 次循环置换。

（2）异或操作：异或是一种二进制布尔代数运算，可以理解为，参与异或运算的两数位如相等，则结果为 0，不等则为 1。在得到扩展后的右半部分数据后，将其与子密钥进行异或操作，将结果存储在 48 位的寄存器中。

```c
//扩展运算E，将32bit的数据变为48bit
int DES_E_Transform(type data[48])
{
    int cnt;
    type temp[48];
    for (cnt = 0; cnt < 48; cnt++)
    {
        //根据扩展矩阵E进行置换
        temp[cnt] = data[E[cnt]];
    }
    memcpy(data, temp, 48);
    return 0;
}
```

```c
//异或操作，将原来的48位数据根据子密钥替代成新的48位数据
int DES_XOR(type R[48], type L[48], int count)
{
    int cnt;
    for (cnt = 0; cnt < count; cnt++)
    {
        R[cnt] ^= L[cnt]; //位运算异或^
    }
    return 0;
}
```

图 10:扩展运算和异或操作

（3）压缩运算：在子密钥与右半部分数据进行异或操作后，需要通过压缩运算将 48bit 的结果压缩为 32bit。压缩运算 S 是指把异或操作的 48 位结果划分为 8 组，每组 6 位，进而将每组的 6 位输入一个 S 盒，获得长度为 4 位的输出。S 盒共有 8 个，互不相同，以 S1~S8 标识，8 个 S 盒的输出连在一起可以得到 32 位的输出。

```c
//压缩运算S-BOX
int DES_SBOX(type data[48])
{
    int cnt;
    int line, row, output;
    int cur1, cur2;
    //把异或操作的48位结果划分为8组。每组6位，进而将每组的6位输入一个S盒
    for (cnt = 0; cnt < 8; cnt++)
    {
        cur1 = cnt * 6;
        cur2 = cnt << 2;

        //计算在S盒中的行与列
        // S盒6位数输入的第一位和第六位构成一个两位的二进制数，对应于s盒中的某一行。
        // S盒6位输入的第二至五位构成一个四位的二进制数，对应于s盒中的某一列。
        line = (data[cur1] << 1) + data[cur1 + 5];
        row = (data[cur1 + 1] << 3) + (data[cur1 + 2] << 2) + (data[cur1 + 3] << 1) + data[cur1 + 4];
        output = S[cnt][line][row];

        //通过确定的行和列在s盒中定位一个十进制数
        //将其转化为二进制的四位数输出。
        data[cur2] = (output & 0X08) >> 3;
        data[cur2 + 1] = (output & 0X04) >> 2;
        data[cur2 + 2] = (output & 0X02) >> 1;
        data[cur2 + 3] = output & 0x01;
    }
    return 0;
}
```

图 11:压缩运算

（4）置换运算 P：这一步能把 S 盒输出的 32 位数据打乱重排,得到 32 位的加密函数输出。置换 P 与 S 盒的互相配合提高了 DES 的安全性。

（5）交换操作：将原来的右部分与处理得到的结果交换

```c
//置换运算P
int DES_P_Transform(type data[32])
{
    int cnt;
    type temp[32];
    for (cnt = 0; cnt < 32; cnt++)
    {
        temp[cnt] = data[P[cnt]];
    }
    memcpy(data, temp, 32);
    return 0;
}
```

```c
//交换：将原来的右部分与处理得到的结果交换
int DES_Swap(type left[32], type right[32])
{
    type temp[32];
    memcpy(temp, left, 32);
    memcpy(left, right, 32);
    memcpy(right, temp, 32);
    return 0;
}
```

图 12:置换运算 P 和交换操作

1.7 对单个明文分组进行加解密

接着就本次实验最为关键的，加密单个分组的操作了。整个流程如下：先将字符串长度为 8 的输入转化为 64 字节的形式，然后开始初始置换，接着进行轮函数的 16 次迭代：

1. 将右半部分数据 Ri-1 通过选择扩展运算 E 扩展成 48-bit 数据
2. 与子密钥 Ki 异或生成新的 48-bit 数据
3. 经过压缩运算 S 变成 32-bit 数据
4. 进行置换运算 P
5. 与左半部分数据进行异或

6.将第 5 步的结果与左部分进行交换

经过 16 轮迭代之后，做逆初始置换处理，再将 bit 形式还原为 char 形式，这样就完成了加密单个分组的操作。

```c
//加密单个分组
int DES_EncryptBlock(type plainBlock[8], type subKeys[16][48], type cipherBlock[8])
{
    type plainBits[64];
    type copyRight[48];
    int cnt;
    //转换字节
    Char8ToBit64(plainBlock, plainBits);
    //初始置换（IP置换）
    DES_IP_Transform(plainBits);
    // 16轮迭代
    for (cnt = 0; cnt < 16; cnt++)
    {
        memcpy(copyRight, plainBits + 32, 32);
        //将右半部分进行扩展置换，从32位扩展到48位
        DES_E_Transform(copyRight);
        //将右半部分与子密钥进行异或操作
        DES_XOR(copyRight, subKeys[cnt], 48);
        //异或结果进入S盒，输出32位结果
        DES_SBOX(copyRight);
        // P置换
        DES_P_Transform(copyRight);
        //将明文左半部分与右半部分进行异或
        DES_XOR(plainBits, copyRight, 32);
        if (cnt != 15)
        {
            //最终完成左右部的交换
            DES_Swap(plainBits, plainBits + 32);
        }
    }
    //逆初始置换（IP^1置换）
    DES_IP_1_Transform(plainBits);
    Bit64ToChar8(plainBits, cipherBlock);
    return 0;
}
```

图 13:加密单个分组函数

解密单个明文分组的流程和加密一致，只是密钥的次序相反，此处不再赘述。

1.8 对整个文件进行加解密

接着就是加密整个文件的操作了，将要加密的文件，三个密钥和生成的文件放进函数里面，然后进行如下操作：生成子密钥、读取文件、对文件分组并加密单个分组、加密后的结果写入新建的目标文件中。

对于文件的操作，首先要定义文件的地址，然后进行读写操作，通过判断 fopen() 的返回值是否和 NULL 相等来判断是否读写失败。如果无法打开原文件则返回-1。对于密钥，首先要设置三个密钥，接着对每一个密钥转换为二进制流，同时生成子密钥。

如果文件非空，读写正常，那就对其单个分组进行加密，将文件分解为每 8 个字节来进行操作。在这个过程当中就可以使用 3DES 了，先是用K_1加密，然后用K_2进行解密，最后再用K_3进行加密处理，最后直到加密文件为空，如果不足一组数据则需要进行补充处理。最后将加密结果写入目标文件中，关闭文件。

```c
//如果文件非空
while (!feof(plain))
{
    //每次读8个字节，并返回成功读取的字节数
    if ((count = fread(plainBlock, sizeof(char), 8, plain)) == 8)
    {
        // 3DES加密单组文件
        DES_EncryptBlock(plainBlock, subKeys1, cipherBlock1);
        DES_DecryptBlock(cipherBlock1, subKeys2, cipherBlock2);
        DES_EncryptBlock(cipherBlock2, subKeys3, cipherBlock3);
        //写入
        fwrite(cipherBlock3, sizeof(char), 8, cipher);
    }
}
```

图 14:加密整个文件函数（部分代码）

对整个文件进行解密过程与加密过程类似，不再赘述。

1.9 主函数

对 3DES 的各个过程函数定义结束后，笔者进行一下测试。首先输入 3 个密钥，然后对 testP.docx（图 16 所示）进行加密，根据之前的函数定义，如果加密函数调用成功会返回参数 1，得到加密后的文件 testC.docx 保存到当前路径文件夹下。接着对加密后的文件 testC.docx 进行解密，根据之前的函数定义，如果解密函数调用成功也会返回参数 1，并将解密后的文件保存为 testP_2.docx。

```
int main()
{
    type key_1[64], key_2[64], key_3[64];
    cout << "请输入密钥1" << endl;
    cin >> key_1;
    cout << "请输入密钥2" << endl;
    cin >> key_2;
    cout << "请输入密钥3" << endl;
    cin >> key_3;

    Table_DES d;
    cout << "开始加密" << endl;
    cout << d.treble_DES_Encrypt("testP.docx", key_1, key_2, key_3, "testC.docx");
    cout << endl;
    cout << "加密结束，开始解密" << endl;
    cout << d.treble_DES_Decrypt("testC.docx", key_1, key_2, key_3, "testP_2.docx");
    cout << endl;
    cout << "解密完成" << endl;
    getchar();

    return 0;
}
```

图 15:主函数进行测试（部分代码）

图 16:要进行加密的 testP.docx 文件内容

运行函数，得到结果如下图 17 所示：可以看到都运行成功了。

图 17:主函数运行结果

接着验证生成的各个文件，从图 18 可以看到，加密后的 testC.docx 文件用 word 程序是打不开的，使用电脑的文本编辑器打开，可以看到是乱码的格式，这说明了加密成功了。接着看使用 testC.docx 解密生成的 testP_2.docx 文件，可以发现跟最开始的原文件 testP.docx 文件内容是一致的，说明了解密也成功了。

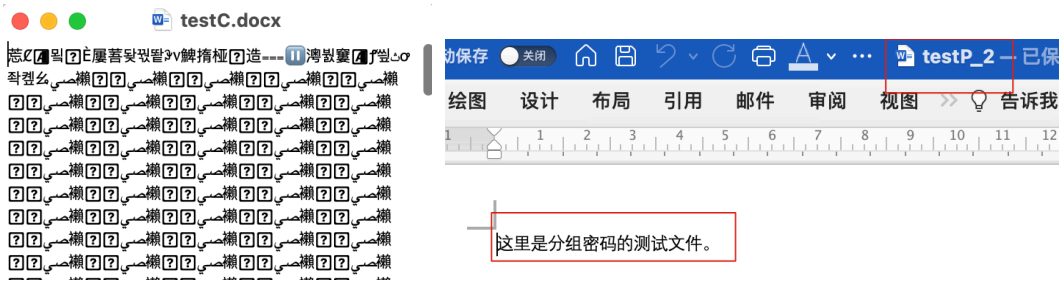

图 18: testC.docx 文件（左）和 testP_2.docx 文件（右）

2、3DES 用于加密各种文件（word、txt、mp3、jpg）

2.1 对 word 进行加密

对 word 的加密和解密测试如上 1.9 所示，成功进行。这里不再赘述。

2.2 对 txt 进行加密

对 txt 文件的测试函数如下图所示，对 testTXT_org.txt 这个原文件进行加密，保存到 testTXT_Encode.txt 文件。再通过对 testTXT_Encode.txt 文件进行解密，保存到 testTXT_Decode.txt 文件。

```
Table_DES d;
cout << "开始加密" << endl;
cout << d.treble_DES_Encrypt("testTXT_org.txt", key_1, key_2, key_3, "testTXT_Encode.txt");
cout << endl;
cout << "加密结束，开始解密" << endl;
cout << d.treble_DES_Decrypt("testTXT_Encode.txt", key_1, key_2, key_3, "testTXT_Decode.txt");
cout << endl;
cout << "解密完成" << endl;
getchar();
```

图 19:对 TXT 文件的测试函数

运行程序后，结果如下图 20 所示，可以看到加密解密都成功了。

```
请输入密钥1
shadjhsjdh
请输入密钥2
87483278
请输入密钥3
67dsfdf
开始加密
1
加密结束，开始解密
1
解密完成
fubo@fubodeMacBook-Pro 谢晓锋_2018031275_EXP1_分组密码 %
```

图 20:TXT 文件测试函数运行结果

接下来我们查看各个文件，如下图 21、22、23 所示。说明对 txt 文件的加密和解密都成功了。

图 21: testTXT_org.txt 文件

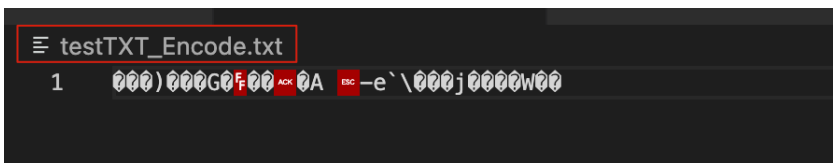

图 22: testTXT_Encode.txt 文件

图 21: testTXT_Decode.txt 文件

2.3 对 mp3 进行加密

对 MP3 文件的测试函数如下图所示，对 testMP3_org.mp3 这个原文件进行加密，保存到 testMP3_Encode.mp3 文件。再通过对 testMP3_Encode.mp3 文件进行解密，保存到 testMP3_Decode.mp3 文件。

```
Table_DES d;
cout << "开始加密" << endl;
cout << d.treble_DES_Encrypt("testMP3_org.mp3", key_1, key_2, key_3, "testMP3_Encode.mp3");
cout << endl;
cout << "加密结束，开始解密" << endl;
cout << d.treble_DES_Decrypt("testMP3_Encode.mp3", key_1, key_2, key_3, "testMP3_Decode.mp3");
cout << endl;
cout << "解密完成" << endl;
getchar();
```

图 22:对 MP3 文件的测试函数

运行程序后，结果如下图 23 所示，可以看到加密解密都成功了。由于 mp3 文件比较大，可以看到对 mp3 的加密花费了几秒钟的时间，比较久。

```
请输入密钥1
4234jhfdsfs
请输入密钥2
787jdsjhn
请输入密钥3
dhsnfshfns
开始加密
1
加密结束，开始解密
1
解密完成
fubo@fubodeMacBook-Pro 谢晓锋_2018031275_EXP1_分组密码 %
```

图 23:MP3 文件测试函数运行结果

接下来我们查看各个文件，如下图 24 所示。打开解密后的文件也可以听到是王菲的传奇，说明对 MP3 文件的加密和解密都成功了。

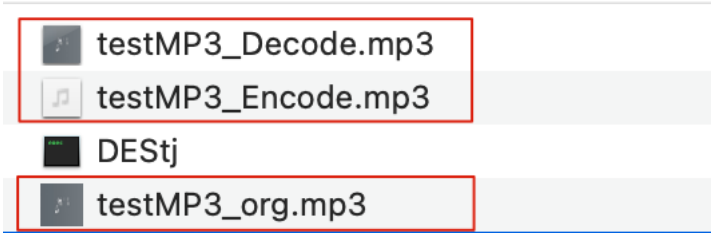

图 24 : mp3 文件

2.4 对 jpg 进行加密

同理，对 jpg 文件的测试函数如下图所示，对 testJPG_org.jpg 这个原文件进行加密，保存到 testJPG_Encode.jpg 文件。再通过对 testJPG_Encode.jpg 文件进行解密，保存到 testJPG_Decode.jpg 文件。

```
Table_DES d;
cout << "开始加密" << endl;
cout << d.treble_DES_Encrypt("testJPG_org.jpg", key_1, key_2, key_3, "testJPG_Encode.jpg");
cout << endl;
cout << "加密结束，开始解密" << endl;
cout << d.treble_DES_Decrypt("testJPG_Encode.jpg", key_1, key_2, key_3, "testJPG_Decode.jpg");
cout << endl;
cout << "解密完成" << endl;
getchar();
```

图 25:对 JPG 文件的测试函数

运行程序后，结果如下图 26 所示，可以看到加密解密都成功了。

图 26:JPG 文件测试函数运行结果

接下来我们查看各个文件，如下图 27、28、29 所示。说明对 jpg 文件的加密和解密都成功了。

图 27: testJPG_org.jpg 文件

图 28: testJPG_Encode.jpg 文件（打不开）

图 29: testJPG_Decode.jpg 文件

3、各种分组密码多种工作模式关键代码及简单介绍

本部分的代码实现笔者专注于分组密码工作模式的具体实现，使用 16 位的二进制演示加密过程。明白了这些原理之后，既可以将这些工作模式应用于 3DES，也可以应用于现在常用的 AES。由于篇幅有限，这里只放核心代码截图，具体注释可查阅代码。关于分组密码工作模式的总结笔者放到了个人网站可供查阅：【网络安全 02】对称加密和消息机密性

3.1 电子密码本模式 (ECB)

- 消息被独立分成分组进行加密

- 每一个分组是一个值，将会被替换，就像一个密码本一样，因此命名为电子密码本模式
- 每一个分组都是独立其它分组进行编码$C_i = E_K(P_i)$
- 使用：单一数值是可以安全传输，对于过长的消息，ECB 模式可能不安全

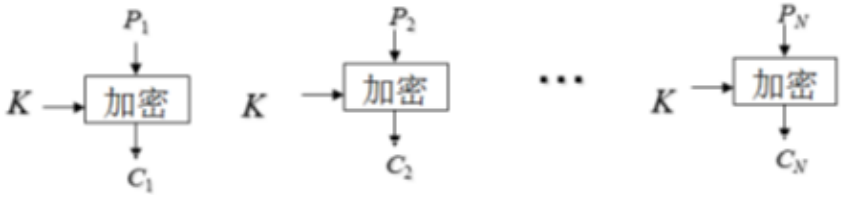

图 30：电子密码本模式（ECB)

```
dataCount = 0; //重置位置变量
for (int i = 0; i < dataLen; i = i + encLen)
{
    int r = i / encLen; //行
    int l = 0;          //列
    int encQue[4];      //编码片段
    for (int j = 0; j < encLen; j++)
    {
        encQue[j] = a[r][l];
        l++;
    }
    encode(encQue); //切片加密
    //添加到密文表中
    for (int p = 0; p < encLen; p++)
    {
        ciphertext[dataCount] = encQue[p];
        dataCount++;
    }
}
```

图 31: ECB 核心代码示例

```
int init[4] = {1, 1, 0, 0}; //初始异或运算输入
//初始异或运算
for (int i = 0; i < dataLen; i = i + encLen)
{
    int r = i / encLen; //行
    int l = 0;          //列
    int encQue[4];      //编码片段
    //初始化异或运算
    for (int k = 0; k < encLen; k++)
    {
        a[r][k] = a[r][k] ^ init[k];
    }
    //与Key加密的单切片
    for (int j = 0; j < encLen; j++)
    {
        encQue[j] = a[r][j];
    }
    encode(encQue); //切片加密
    //添加到密文表中
    for (int p = 0; p < encLen; p++)
    {
        ciphertext[dataCount] = encQue[p];
        dataCount++;
    }
    //变换初始输入
    for (int t = 0; t < encLen; t++)
    {
        init[t] = encQue[t];
    }
}
```

图 32:CBC 核心代码示例

3.2 密码分组链接模式（CBC)

- 分组密码链接模式主要是用来解决电子密码本模式中的消息重复在密文出现的问题。
- 消息被分成分组，在加密操作时链接在一起
- 每一个现在的明文分组与前面密文分组链接（做异或操作）
- 采用初始向量去开始处理$C_i = E_K(P_i XOR C_{i-1}), C_{-1} = IV$（通过异或操作之后，即使跟前面是重复的信息，加密后也不会重复。）
- 使用：批量数据加密，认证

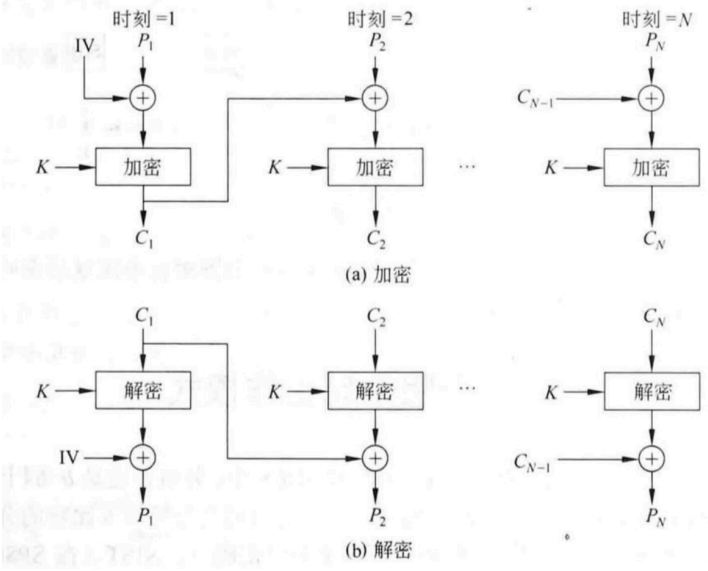

图 33 :CBC 模式流程图

3.3 密码反馈模式 (CFB)

- 消息被当成是比特流（CFB 属于流密码）
- 在分组密码的输出进行增加，结果是在下一阶段进行反馈
- 标准允许任意比特（1,8, 64 or 128 等) 进行反馈 （也就是传到下一轮）
- 最高效率使用分组里的每一比特：不直接对 P_i 加密，而是加密上一轮密文 C_{i-1}，公式为$C_i = P_i XOR E_K(C_{i-1})$，$C_{-1} = IV$
- 使用: 流数据加密，认证

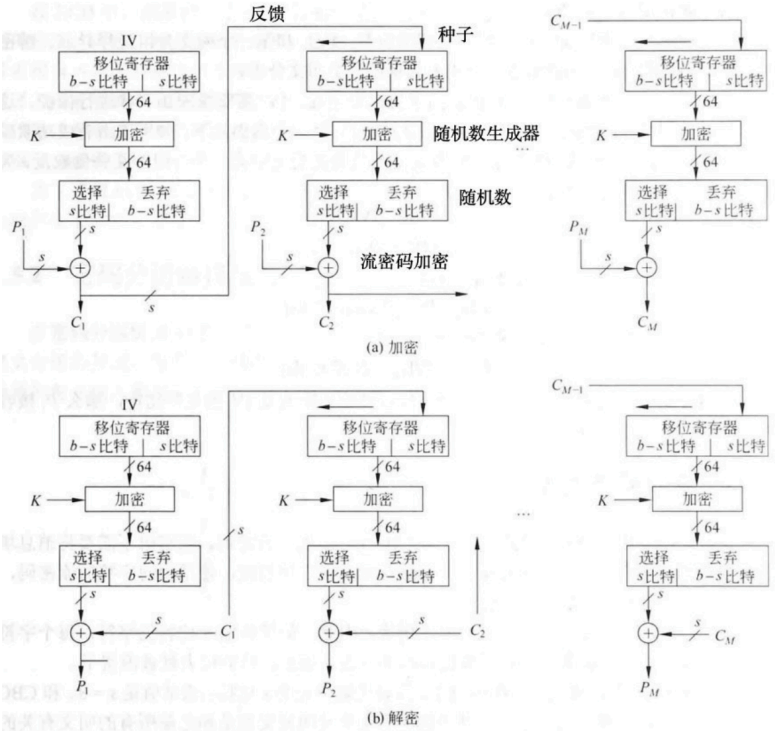

图 34 CFB 模式流程图

3.4 计数器模式(CTR)

- 虽然很早提出，但是是一种 “新” 模式

- 类型于 CFB，但是加密每个计数值，而不是任何反馈值
- 对每个明文分组，必须有不同的密钥和计数值 (从不重复使用)：$O_i = E_K(i)$，$C_i = P_i XOR O_i$
- 使用：快速网络加密

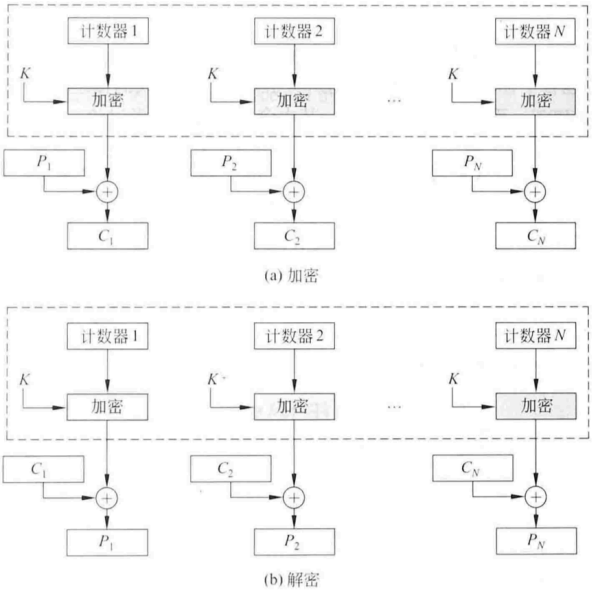

图 35 CTR 模式流程图

```
int lv[4] = {1, 0, 1, 1}; //初始设置的位移变量
int encQue[2];            // K的高两位
int k[4];                 // K
for (int i = 0; i < 2 * encLen; i++) //外层加密循环
{
    //产生K
    for (int vk = 0; vk < encLen; vk++)
    {
        k[vk] = lv[vk];
    }
    encode(k);
    for (int k2 = 0; k2 < 2; k2++)
    {
        encQue[k2] = k[k2];
    }
    // K与数据明文异或产生密文
    for (int j = 0; j < 2; j++)
    {
        ciphertext[dataCount] = a[dataCount / 2][j] ^ encQue[j];
        dataCount++;
    }
    // lv左移变换
    lv[0] = lv[2];
    lv[1] = lv[3];
    lv[2] = ciphertext[dataCount - 2];
    lv[3] = ciphertext[dataCount - 1];
}
```

图 36: CFB 核心代码示例

```
int init[4][4] = {{1, 0, 0, 0}, {0, 0, 0, 1},
                  {0, 0, 1, 0}, {0, 1, 0, 0}}; //算子表
int l = 0;   //明文切片表列
//初始异或运算
for (int i = 0; i < dataLen; i = i + encLen)
{
    int r = i / encLen; //行
    int encQue[4];      //编码片段
    //将算子切片
    for (int t = 0; t < encLen; t++)
    {
        encQue[t] = init[r][t];
    }
    encode(encQue); //算子与key加密
    //最后的异或运算
    for (int k = 0; k < encLen; k++)
    {
        encQue[k] = encQue[k] ^ a[l][k];
    }
    l++;

    //添加到密文表中
    for (int p = 0; p < encLen; p++)
    {
        ciphertext[dataCount] = encQue[p];
        dataCount++;
    }
}
```

图 37:CTR 核心代码示例

3.5 输出反馈模式(OFB)

- 在 OFB 模式中，密码算法的输出会反馈到密码算法的输入中
- OFB 模式不是通过密码算法对明文直接加密的，而是通过将明文分组和密码算法的输出进行异或操作来产生密文分组的。
- 具体而言，先用块加密器生成密钥流，然后再将密钥流与明文流异或得到密文流
- 解密是先用块加密器生成密钥流，再将密钥流与密文流异或得到明文，由于异或操作的对称性所以加密和解密的流程是完全一样的

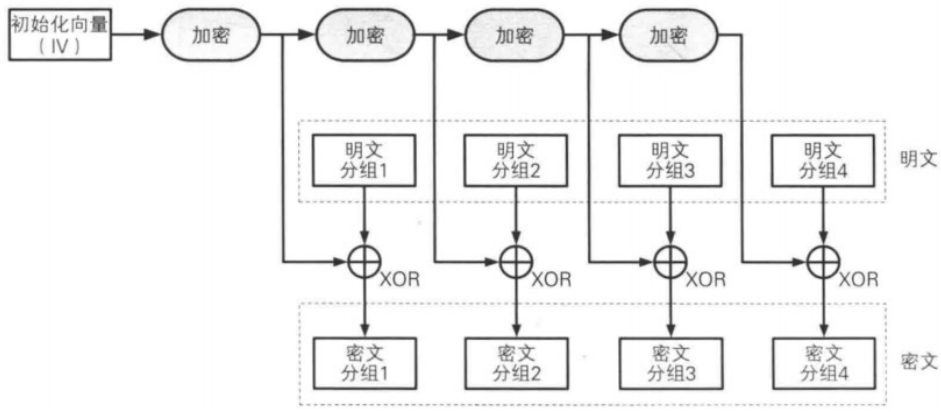

图 38：OFB 模式流程图

接着笔者使用 16 位的二进制对这五种工作模式进行测试，如图 40，可以看到程序的运行结果如下，跟上述各模式的工作原理是一致的。

```
int lv[4] = {1, 0, 1, 1}; //初始设置的位移变量
int encQue[2];            // K的高两位
int k[4];                 // K
for (int i = 0; i < 2 * encLen; i++) //外层加密循环
{
    //产生K
    for (int vk = 0; vk < encLen; vk++)
    {
        k[vk] = lv[vk];
    }
    encode(k);
    for (int k2 = 0; k2 < 2; k2++)
    {
        encQue[k2] = k[k2];
    }
    // K与数据明文异或产生密文
    for (int j = 0; j < 2; j++)
    {
        ciphertext[dataCount] = a[dataCount / 2][j] ^ encQue[j];
        dataCount++;
    }
    // lv左移变换
    lv[0] = lv[2];
    lv[1] = lv[3];
    lv[2] = encQue[0];
    lv[3] = encQue[1];
}
```

图 39: OFB 核心代码示例

```
明文为：
1 0 0 1
0 0 0 1
1 1 1 1
0 0 0 0
ECB加密的密文为：
0 0 1 1
1 0 1 1
0 1 0 1
1 0 1 0
CCB加密的密文为：
1 1 1 1
0 1 0 0
0 0 0 1
1 0 1 1
CTR加密的密文为：
1 0 1 1
1 0 1 0
0 1 1 1
1 1 1 0
CFB加密的密文为：
1 0 0 0
0 0 1 1
0 1 1 0
1 1 0 0
OFB加密的密文为：
1 0 0 0
1 0 1 0
1 1 1 0
1 0 1 1
fubo@fubodeMacBook-Pro 谢晓锋_2018031275_EXP1_分组密码 %
```

图 40：模式运行示例

实验结论：

本次实验笔者收获颇丰，不仅通过编程深刻地理解了 3DES 的加解密逻辑，还通过不断地调试代码形象地理解了各种模式下做位相与操作时分组明文和秘钥之间的关系。再次总结实验逻辑如下：

1、3DES 加密算法的实现及介绍

1.1 算法流程结构设计

1.2 定义标准表变量

1.3 字符、字节和二进制之间的转换

1.4 生成子密钥

1.5 初始置换和逆置换

1.6 轮函数

1.7 对单个明文分组进行加解密
1.8 对整个文件进行加解密
1.9 主函数
2、3DES 用于加密各种文件（word、txt、mp3、jpg）
2.1 对 word 进行加密
2.2 对 txt 进行加密
2.3 对 mp3 进行加密
2.4 对 jpg 进行加密
3、各种分组密码多种工作模式关键代码及简单介绍
3.1 电子密码本模式 (ECB)
3.2 密码分组链接模式 (CBC)
3.3 密码反馈模式 (CFB)
3.4 计数器模式(CTR)
3.5 输出反馈模式(OFB)

另外在编程中遇到的最大困难还是在加密音频文档和图片文档时的格式转换问题，因为在解密之后的二进制流中总是会出现空格，因此无法将数据还原成原来的音频和图片，这可能是在做字符转换时音频和图片的某些字符无法找到对应的转换对象导致的。

表 1. 多工作模式的比较

	ECB	CBC	CFB	CTR
安全强度	较差	较好	较好	较好
运行时间	较快	较快	较慢	较慢
可拓展性	较差	较好	较差	较好
优势	①简单，快速 ②加解密可并行计算 ③误差不传递	①不容易主动攻击 ②适合长报文 ③解密可以并行	①不容易主动攻击 ②分组转变为流模式 ③可加密小于分组数据 ④解密可并行	①加解密可以并行 ②一次一密
劣势	①不能隐藏明文模式 ②容易主动攻击	①加密无法并行	①加密无法并行 ②存在误差传递	①容易主动攻击
用途	①需要并行加密的应用 ②单个数据的安全传输（一个加密密钥）	③长报文传输 ①SSL 和 IPSec	①面向数据流的通用传输认证	①面向分组的通用传输 ②用于高速需求