**实验二 分组密码算法DES**

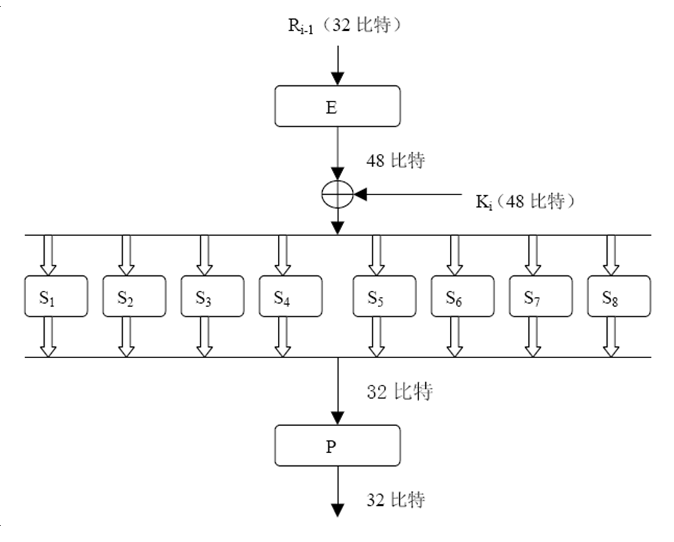
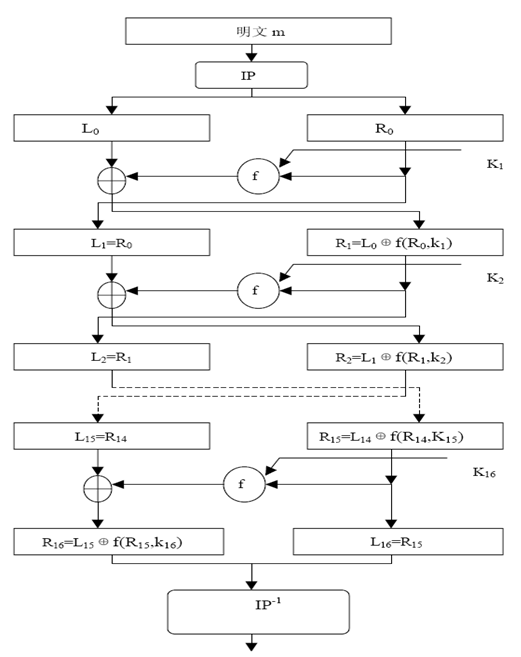
姓名：范毓哲 学号：1910378

1. **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。

1. **设计流程**

移位密码的加解密：



1. **实验环境**

Windows10操作系统+VS2019(VC17)。

同级目录下，des.cpp为DES分组加解密算法的实现，密钥雪崩.cpp为修改密钥时密文的变化情况统计，明文雪崩.cpp为修改明文时密文的变化情况统计，对应的exe分别为其可执行程序。所有程序需要使用的头文件都是tables.h

1. **程序实现**

1. DES加解密

如下的代码实现了DES分组加解密（tables.h当中只列了已经给出的置换表等基础数据，没有需要进行程序设计的地方，故报告中不再分析）。

#include"tables.h"

**using** **namespace** std;

//定义全局变量：置换前的明文，置换后的明文，操作码，密钥，56位密钥

**int** plaintext\_64[64], plaintext\_64\_IP[64], mode = 0, key[64], key\_56[56];

//将指定字段转换为二进制数组

**void** hex2bin(**int** x,**int** txt[],**int** tag)

{

**for** (**int** i = 0; i < 8; i++)

    {

**int** a = cases[x].txt[i],j = 0;

**if** (tag == 1) a = cases[x].key[i];

**while** (a > 0) {txt[8\*i+7-j]=a%2;a/=2;j++;}

    }

}

//生成子密钥

**int** mkidkey[16][56],tp[4]={26,27,54,55},kidkey[16][48];

**void** shiftkey()

{

**for** (**int** j = 0; j < 27; j++)

    {

        mkidkey[0][j] = key\_56[j + 1];

        mkidkey[0][j + 28] = key\_56[j + 29];

    }

    mkidkey[0][27] = key\_56[0]; mkidkey[0][55] = key\_56[28];

**for** (**int** i = 1; i < 16; i++)

    {

**if** (shift[i] == 1)  //shift是密钥的移位位数

        {

**for** (**int** j = 0; j < 27; j++)

            {

                mkidkey[i][j] = mkidkey[i - 1][j + 1];

                mkidkey[i][j + 28] = mkidkey[i - 1][j + 29];

            }

            mkidkey[i][27] = mkidkey[i - 1][0];

            mkidkey[i][55] = mkidkey[i - 1][28];

        }

**if** (shift[i] == 2)

        {

**for** (**int** j = 0; j < 26; j++)

            {

                mkidkey[i][j] = mkidkey[i - 1][j + 2];

                mkidkey[i][j + 28] = mkidkey[i - 1][j + 30];

            }

**for** (**int** j = 0; j<4; j++) mkidkey[i][tp[j]] = mkidkey[i-1][tp[j] - 26];

        }

    }

}

**void** lun(**int** temp[])  //DES的核心轮操作

{

**int** sum = 0, Lturn[17][64],Eres[16][48], S\_res[16][32], P\_res[16][32];

    memset(S\_res,0, **sizeof**(S\_res));

**for** (**int** i = 0; i < 64; i++) Lturn[0][i] = plaintext\_64\_IP[i];

**for** (**int** i = 1; i < 17; i++)

    {

**for** (**int** j = 0; j < 32; j++) Lturn[i][j] = Lturn[i - 1][j + 32];

        //E扩展+异或

**for** (**int** j = 0; j < 48; j++)

        {

**int** te= Lturn[i - 1][E[j] - 1 + 32];

**if** (mode == 0) Eres[i - 1][j] = te ^ kidkey[i - 1][j];

**if** (mode == 1) Eres[i - 1][j] = te ^ kidkey[16 - i][j];

        }

**for** (**int** j = 0; j < 8; j++)  //S盒置换

        {

**int** a = Eres[i - 1][6 \* j] \* 2 + Eres[i - 1][6 \* j + 5];

**int** b = Eres[i - 1][6 \* j + 1] \* 8 + Eres[i - 1][6 \* j + 2] \* 4 + Eres[i - 1][6 \* j + 3] \* 2 + Eres[i - 1][6 \* j + 4];

**int** c = S[j][a][b]; **int** ji = 0;

**while** (c > 0) { S\_res[i - 1][4 \* j + 3 - ji] = c % 2; c /= 2; ji++; }

        }

        //P置换

**for** (**int** j = 0; j < 32; j++) P\_res[i - 1][j] = S\_res[i - 1][P[j] - 1];

**for** (**int** j = 0; j < 32; j++) Lturn[i][j + 32] = Lturn[i - 1][j] ^ P\_res[i - 1][j];

    }//最后一次对换

**for** (**int** i = 0; i < 32; i++) { temp[i] = Lturn[16][i + 32]; temp[i + 32] = Lturn[16][i]; }

}

**int** main()

{

**int** idx = 0,cipher[64], temp[64];

    cout <<"请输入加解密的mode和测试组别，用空格分开(mode为0表示加密，1解密)." << endl;

    cout << "(输负数则可直接退出程序)\n"; **int** ii = 0;

    //cout << "二十组加解密测试结果如下：\n";

**while** (ii<20)

    {

        cout << "输入mode和组别："; cin >> mode >> idx;

        //idx = ii; ii++; if (idx > 9) mode = 1;

        memset(plaintext\_64,0,**sizeof**(plaintext\_64));memset(plaintext\_64\_IP,0,**sizeof**(plaintext\_64\_IP));

        memset(key, 0, **sizeof**(key)); memset(key\_56, 0, **sizeof**(key\_56));

        idx=10 \* mode+idx; **if**(mode<0||idx<0) {cout<<"【退出】程序结束\n";**return** 0; }

        hex2bin(idx, plaintext\_64, 0);//得到IP置换后的文本

**for** (**int** i = 0; i < 64; i++) plaintext\_64\_IP[i] = plaintext\_64[IP[i] - 1];

        hex2bin(idx, key, 1); //得到密钥PC-1置换后的文本

**for** (**int** i = 0; i < 56; i++) key\_56[i] = key[PC\_1[i] - 1];

        shiftkey();//子秘钥移位

**for** (**int** i = 0; i < 16; i++) **for** (**int** j = 0; j < 48; j++) kidkey[i][j] = mkidkey[i][PC\_2[j] - 1];

        lun(temp);

**for** (**int** i = 0; i < 64; i++) cipher[i] = temp[IP\_1[i] - 1];//逆置换

**for** (**int** i = 0; i < 16; i++)  //输出十六进制形式的结果

        {

**int** resi = cipher[4\*i]\*8+cipher[4\*i+1]\*4+cipher[4\*i+2]\*2+cipher[4\*i+3];

**if** (i % 2 == 0) { printf("0x%X", resi); **continue**; }

            printf("%X", resi); **if** (i != 15)  cout << ",";

        }

        cout << endl;

    }

}

2. 明文雪崩

程序的整体和原先的加解密没有任何区别，只是添加了对比密文变化的函数：

**int** sum = 0;  //记录每一组的总位数变化

**void** compare(**int** x,**int** cipher[])

{

**int** standard[64]; memset(standard, 0, **sizeof**(standard));

**for** (**int** i = 0; i < 8; i++)

    {

**int** a = cases[x].out[i], j = 0;

**while** (a > 0)    //取出原结果

        { standard[8 \* i + 7 - j] = a % 2; a /= 2; j++; }

    }

**for** (**int** i = 0; i < 64; i++)   //进行逐位比较

    {**if** (cipher[i] != standard[i]) sum++;}

}

然后是main函数当中：

**int** ii = 0; srand(time(0));

**int** num[10]; **for** (**int** i = 0; i < 8; i++) num[i]=rand() % 64;

**while** (ii<10)

    {

        idx = ii; ii++; **if** (idx > 9) mode = 1;

        sum = 0;

**for** (**int** k = 0; k < 8; k++)   //进行八次明文变换

        {

            memset(plaintext\_64, 0, **sizeof**(plaintext\_64)); memset(plaintext\_64\_IP, 0, **sizeof**(plaintext\_64\_IP));  memset(key, 0, **sizeof**(key)); memset(key\_56, 0, **sizeof**(key\_56)); memset(temp, 0, **sizeof**(temp));  memset(kidkey, 0, **sizeof**(kidkey)); memset(mkidkey, 0, **sizeof**(mkidkey));

            hex2bin(idx, plaintext\_64, 0);//得到IP置换后的文本

            plaintext\_64[num[k]] = 1 - plaintext\_64[num[k]];

            cout <<"第"<<idx<< "组的明文第" << num[k] << "位取反。" << endl;

  //其余内容不变，此处省略

        }

        cout << "第" << idx << "组密文改变的平均位数是" << sum / 8; cout << "\n\n";

    }

    getchar();

}

3. 密钥雪崩

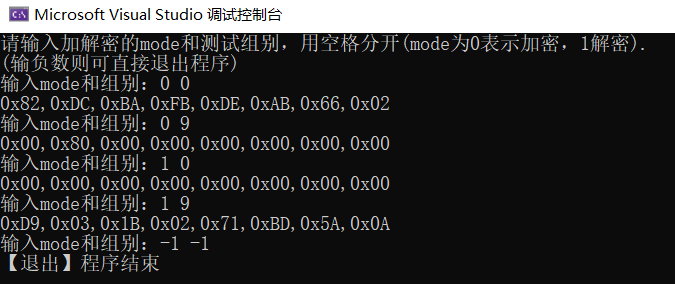
只要把明文雪崩效应测试代码当中对明文的修改，改成对密钥的修改即可，其他内容不变：

key[num[k]] = 1 - key[num[k]];

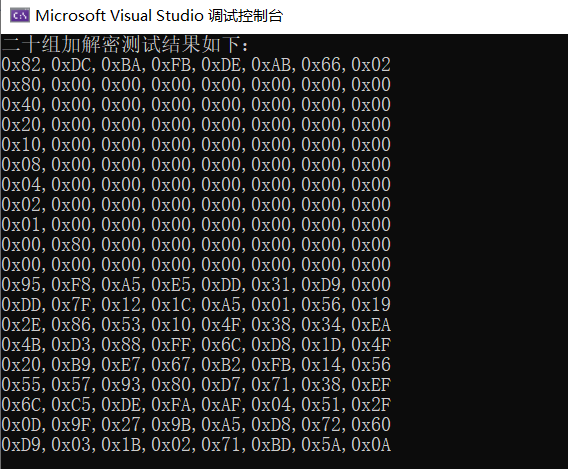
cout <<"第"<<idx<< "组的密钥第" << num[k] << "位取反。" << endl;

1. **实验结果**

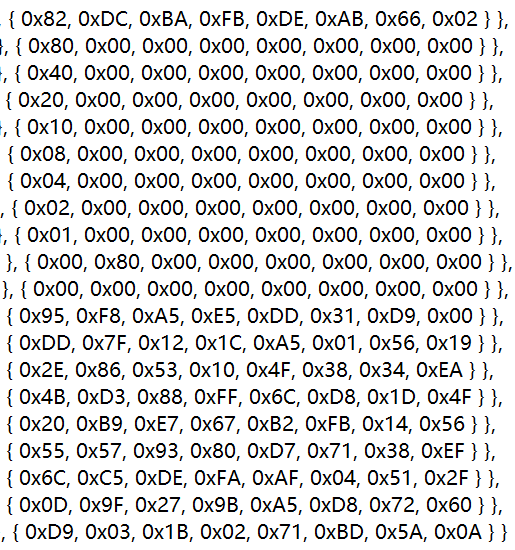
移位密码的加解密结果如下，首先是单组测试的结果：



在注释中也提供了输出全部结果的代码，结果如下：

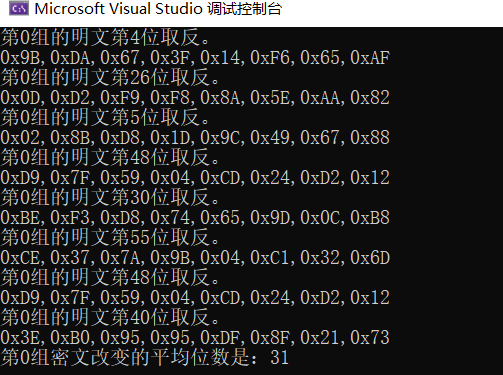


直接将它与实验已经提供的out进行对比，可以看到结果一致，算法实现正确：



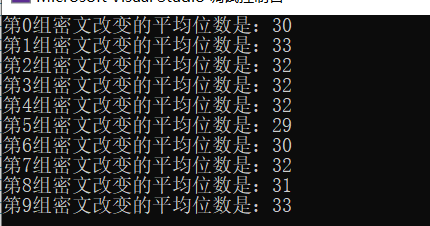
接下来进行雪崩效应的测试验证。

首先是固定密钥、修改明文的密文雪崩效应：



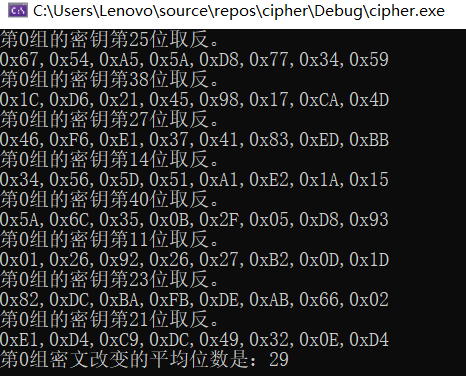
可以看到，这一组在8次测试当中的密文平均改变位数是31。

然后对整个实验提供的所有十个加密组别都进行明文变换测试，一次性输出最终的变化结果：

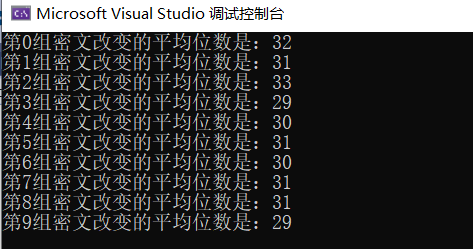


密文改变的平均位数是（30+33+32+32+32+29+30+32+31+33）/10 = 31.4 位

然后是修改密钥的雪崩效应：



然后对全部组别进行验证：



最终的平均密文改变位数是（32+31+33+29+30+31+30+31+31+29）/10 = 30.7位。

从上述实验结果我们可以看到，对明文或密钥只改动一位，就可以使得密文改动32位左右，占其总长度的一半，验证了雪崩效应。