密码学第二次实验报告

学号：2111033 姓名：艾明旭 专业：信息安全

分组密码算法des

⼀、实验内容说明

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。

2、实验要求

. 分别实现DES的加密和解密，提交程序代码和执⾏结果。

\_x0019\_. 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。

3、实验步骤

. 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。

\_x0019\_. 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。

. 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。

. 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。

⼆、实验环境

操作系统：windows11

软件系统：Visual studio

编译⼯具：Visual studio2022

编程语⾔：C++

三、实验过程

本次实验⾸先翻阅课本，对理论课上的知识进⾏回顾，然后设计整个实验的流程图以及各个结构体和函数的

⼤致思路，然后进⾏具体代码的编写实现，以下为具体过程：

1、流程分析

DES*分析*

可以将其分为以下⼏步：

1.⼦秘钥⽣成

2初始置换 IP

3密码函数 f

4.尾置换 IP-1

考虑到我们的存储格式，由于我们进⾏的⼤部分是位运算，为了更加⽅便的进⾏代码编程，我们⾸先考虑将

数据格式转为bit格式，这⾥我们使⽤bitset<64>。

⽤户输⼊的秘钥（

key）、明⽂（

origin）、密⽂（

secret）均设定为变量类型为bitset<64>全局变量。

*第⼀步*.*数据处理*

⾸先我们编写处理函数，void getbit(string a, bitset<64>& temp)能将我们输⼊的⼗六进制字符串转换为

bitset<64>,⽽void getHex(bitset<64> b,string &s)则起到相反的作⽤，我们将数据处理完之后，存⼊全局变

量中。

*第⼆步*.*⼦秘钥⽣成*

⾸先在我们进⾏加解密之前，我们要⽣成好16组48位秘钥。

接下来我们已经获取到了秘钥key，我们通过函数 void getKeys()将⽣成的⼦秘钥存到bitset<48> sonKey[16]

中。

getKeys()函数流程如下：

去掉奇偶校验位64->56，并根据压缩置换表pc\_1进⾏置换，但我们可以注意到，压缩置换表中已经省

去了奇偶校验位，所以我们可以跳过去掉奇偶校验位的过程，直接按照pc\_1表进⾏压缩置换，置换过

程如下：

for (int i = 0; i < 56; i++)

{

firstKey[55 - i] = key[64 - PC\_1[i]];

}

由于以后置换全部采取这种⽅法，所以后续置换就不详细展示

循环⼗六次

从sonKey[16]中取出⼀组，将该组56位⼦秘钥分为⾼低位两部分，分别根据移位表进⾏左移调

⽤leftShift(bitset<28> k, int shift 左移位数)，再将位移后的两部分重新拼接，再对照着PC\_2

表，压缩置换为48位，我们将压缩后的秘钥重新放回sonKey[16]

*第三步*.DES*加密*

. IP初始置换，按照IP表对明⽂进⾏置换

\_x0019\_. 将64位数据分为⾼低位，调⽤bitset<32> f(bitset<32> R, bitset<48> k)对低位进⾏处

理，得到的结果与⾼位进⾏异或，然后互换新的⾼低位位置，进⾏⼗六次迭代

. 进⾏合并，注意合并时原本⾼低位位置发⽣调换

. 逆初始置换IP－1

重要函数bitset<32> f(bitset<32> R, bitset<48> k ⼦秘钥)流程:

1.根据表E将32扩展为48

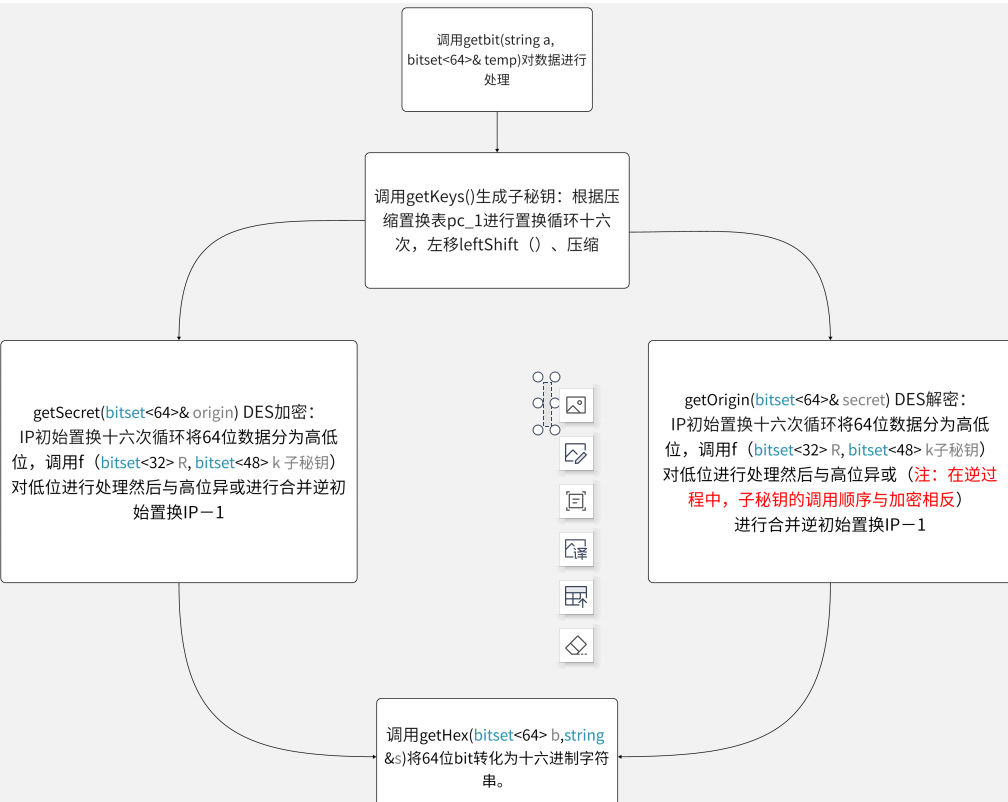
2.与⼦秘钥异或运算

3.查找S\_BOX置换表，分成8组，每个6位，其中最⾼位与最低位组成⾏号，剩下4位组成列号

4.P-1置换

流程图

DES密码流程图如下图所示：



2、代码实现

各类的结构体（即书本上出现的置换表以及各种扩展和置换运算）已经预先定义

各类预处理函数

⾸先是⼆进制与字符的互相转换

// 将string类型转换为bits类型

void getbit(string a, bitset<64>& temp)

{

int num = 63;

for (int i = 2; i < a.length(); i++)

{

if (a[i] <= '9')

{

for (int j = 0; j < 4; j++)

{

temp[num--] = HexToBit[a[i] - 48][j];

}

}

else

{

for (int j = 0; j < 4; j++)

{

temp[num--] = HexToBit[a[i] - 65 + 10][j];

}

}

}

}

// 将bits类型转换为string类型

void getHex(bitset<64> b, string& s)

{

s[0] = '0';

s[1] = 'x';

int num = 2;

for (int i = 63; i >= 0; i -= 4)

{

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

if (a > 9)

s[num++] = 'A' + a - 10;

else

s[num++] = '0' + a;

}

}

接下来是左循环移位函数

// 左循环移位

bitset<28> leftShift(bitset<28> k, int shift)

{

bitset<28> tmp = k;

for (int i = 27; i >= 0; --i)

{

if (i - shift < 0)

k[i] = tmp[i - shift + 28];

else

k[i] = tmp[i - shift];

}

return k;

}

然后是F函数的运算，包括S盒代换、扩展运算、⼦密钥的异或以及P置换操作

// 轮函数的计算并返回一个32bits的值

bitset<32> f(bitset<32> R, bitset<48> k)

{

// S盒代换时需要用到的计数器

int temp = 0;

// 预定义扩展运算的结果

bitset<48> expandR;

// 进行E扩展运算

for (int i = 0; i < 48; ++i)

{

expandR[47 - i] = R[32 - E[i]];

}

// 与子密钥进行异或运算

expandR = expandR ^ k;

// 进行S盒代换操作

bitset<32> output;

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);

output[31 - temp] = binary[3];

output[31 - temp - 1] = binary[2];

output[31 - temp - 2] = binary[1];

output[31 - temp - 3] = binary[0];

temp += 4;

}

// 进行P置换操作

bitset<32> tmp = output;

for (int i = 0; i < 32; ++i)

{

output[31 - i] = tmp[32 - P[i]];

}

return output;

}

加密过程

加密函数⾸先进⾏IP初始置换，按照IP表对明⽂进⾏置换

然后将64位数据分为⾼低位，调⽤bitset<32> f(bitset<32> R, bitset<48> k)对低位进⾏

处理，得到的结果与⾼位进⾏异或，然后互换新的⾼低位位置，进⾏⼗六次迭代

接下来进⾏合并，注意合并时原本⾼低位位置发⽣调换

最后进⾏逆初始置换IP－1，具体函数如下：

// 加密函数

bitset<64> getSecret(bitset<64>& origin)

{

bitset<64> secret;

bitset<64> firstInput;

bitset<32> high;

bitset<32> low;

// 初始置换IP

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

{

firstInput[63 - i] = origin[64 - IP[i]];

}

// 将其分为左右两部分

for (int i = 32; i < 64; i++)

{

high[i - 32] = firstInput[i];

}

for (int i = 0; i < 32; i++)

{

low[i] = firstInput[i];

}

// 进行16轮的循环

for (int round = 0; round < 16; round++)

{

bitset<32> nextHigh;

nextHigh = low;

low = high ^ f(low, sonKey[round]);

high = nextHigh;

}

// 左右交换

for (int i = 0; i < 32; i++)

{

secret[i] = high[i];

secret[i + 32] = low[i];

}

// 逆初始置换

firstInput = secret;

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

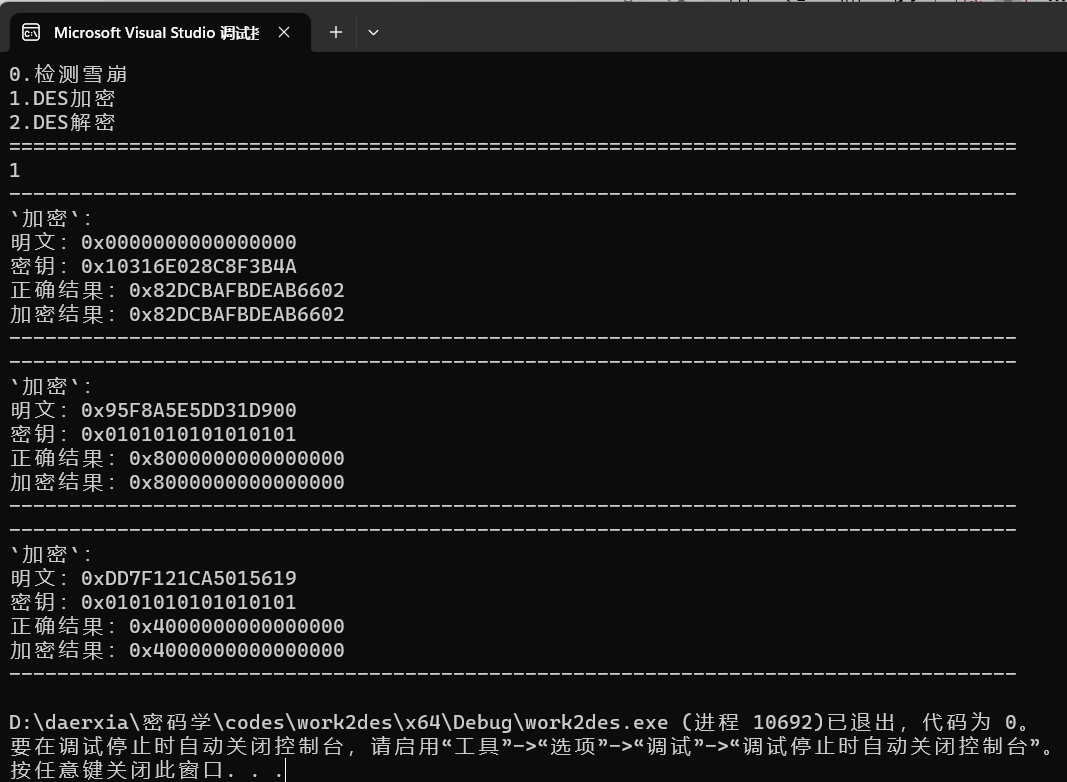
{

secret[63 - i] = firstInput[64 - IP\_1[i]];

}

return secret;

}



可以看到加密结果与正确结果⼀样

解密过程

解密函数与加密过程类似，只是⼦秘钥的调⽤顺序与加密相反，具体函数如下：

// 解密函数

bitset<64> getOrigin(bitset<64>& secret)

{

bitset<64> origin;

bitset<64> firstInput;

bitset<32> high;

bitset<32> low;

// 初始置换IP

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

firstInput[63 - i] = secret[64 - IP[i]];

// 将其分为左右两部分

for (int i = 32; i < 64; i++)

high[i - 32] = firstInput[i];

for (int i = 0; i < 32; i++)

low[i] = firstInput[i];

// 进行16轮的循环

bitset<32> nextHigh;

for (int round = 0; round < 16; round++)

{

nextHigh = low;

low = high ^ f(low, sonKey[15 - round]);

high = nextHigh;

}

// 左右交换

for (int i = 0; i < 32; i++)

{

origin[i] = high[i];

origin[i + 32] = low[i];

}

// 逆初始置换

firstInput = origin;

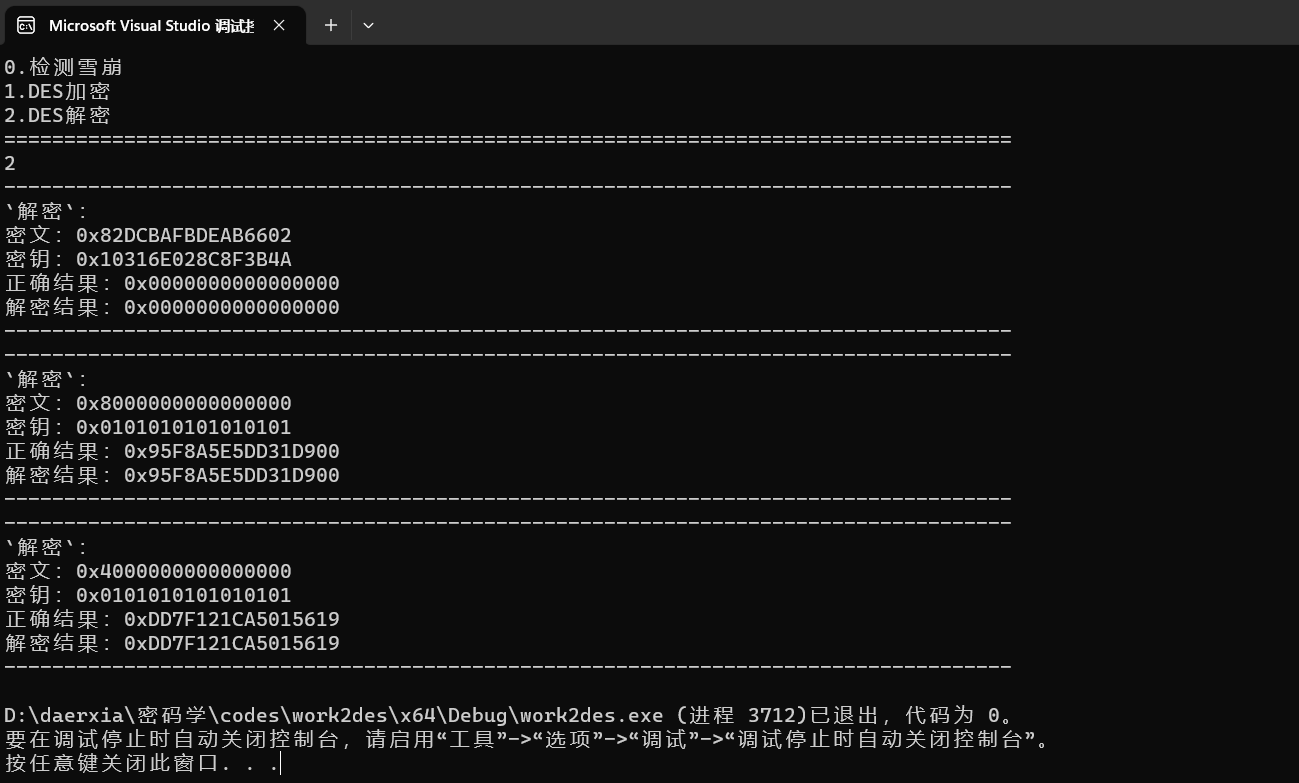
for (int i = 0; i < 64; i++)

origin[63 - i] = firstInput[64 - IP\_1[i]];

return origin;

}

最后的解密输出结果如下：



可以看到解密结果与正确结果⼀样

雪崩效应

在输⼊为0时进⾏雪崩效应

先固定密钥，每次改变⼀个明⽂，进⾏⼋次，观察更改的结果的不同位数

然后固定明⽂，每次改变⼀个密钥，进⾏⼋次，观察更改的结果的不同位数，具体函数如下：

if (a == 0)

{

cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++" << endl;

string a\_pre = "0x95F8A5E5DD31D900";

string b\_pre = "0x0101010101010101";

int num[8] = { 0,0,0,0,0,0,0,0 };

int num\_pre = 0;

// 需要改变的明文

string a[8] = { "0x95F8A5E5DD31D901","0x95F8A5E5DD31D902","0x95F8A5E5DD31D904","0x95F8A5E5DD31D908",

"0x95F8A5E5DD31D910","0x95F8A5E5DD31D920","0x95F8A5E5DD31D940","0x95F8A5E5DD31D980"

};

string b[8] = { "0x0101010101010111","0x0101010101011101","0x0101010101110101","0x0101010111010101",

"0x0101011101010101","0x0101110101010101","0x0111010101010101","0x1101010101010101"

};

cout << "接下来每次改变一位明文，观察结果" << endl;

for (int w = 0; w < 8; w++)

{

// 正确的密文

string c = "0x8000000000000000";

getbit(a[w], origin);

getbit(c, first);

getbit(b\_pre, key);

// 获取密钥

getKeys();

// 进行加密

result = getSecret(origin);

cout << "原始明文：" << a\_pre << endl << "改后明文：" << a[w] << endl;

cout << "原始结果：" << first << endl << "改后结果：" << result << endl;

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

{

if (first[i] != result[i])

{

num[w]++;

}

}

cout << "不同的数字位数：" << num[w] << endl;

}

for (int w = 0; w < 8; w++)

{

num\_pre += num[w];

}

cout << "改变明文获得不同的数字位数的平均数为：" << double(num\_pre) / 8 << endl;

// 重新初始化，开始更改密钥

num\_pre = 0;

for (int w = 0; w < 8; w++)

{

num[w] = 0;

}

cout << "------------------------------------------------------------------------------------" << endl;

cout << "接下来每次改变一位密钥，观察结果" << endl;

for (int w = 0; w < 8; w++)

{

// 正确的密文

string c = "0x8000000000000000";

getbit(a\_pre, origin);

getbit(c, first);

getbit(b[w], key);

// 获取密钥

getKeys();

// 进行加密

result = getSecret(origin);

cout << "原始密钥：" << b\_pre << endl << "改后密钥：" << b[w] << endl;

cout << "原始结果：" << first << endl << "改后结果：" << result << endl;

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

{

if (first[i] != result[i])

{

num[w]++;

}

}

cout << "不同的数字位数：" << num[w] << endl;

}

for (int w = 0; w < 8; w++)

{

num\_pre += num[w];

}

cout << "改变明文获得不同的数字位数的平均数为：" << double(num\_pre) / 8 << endl;

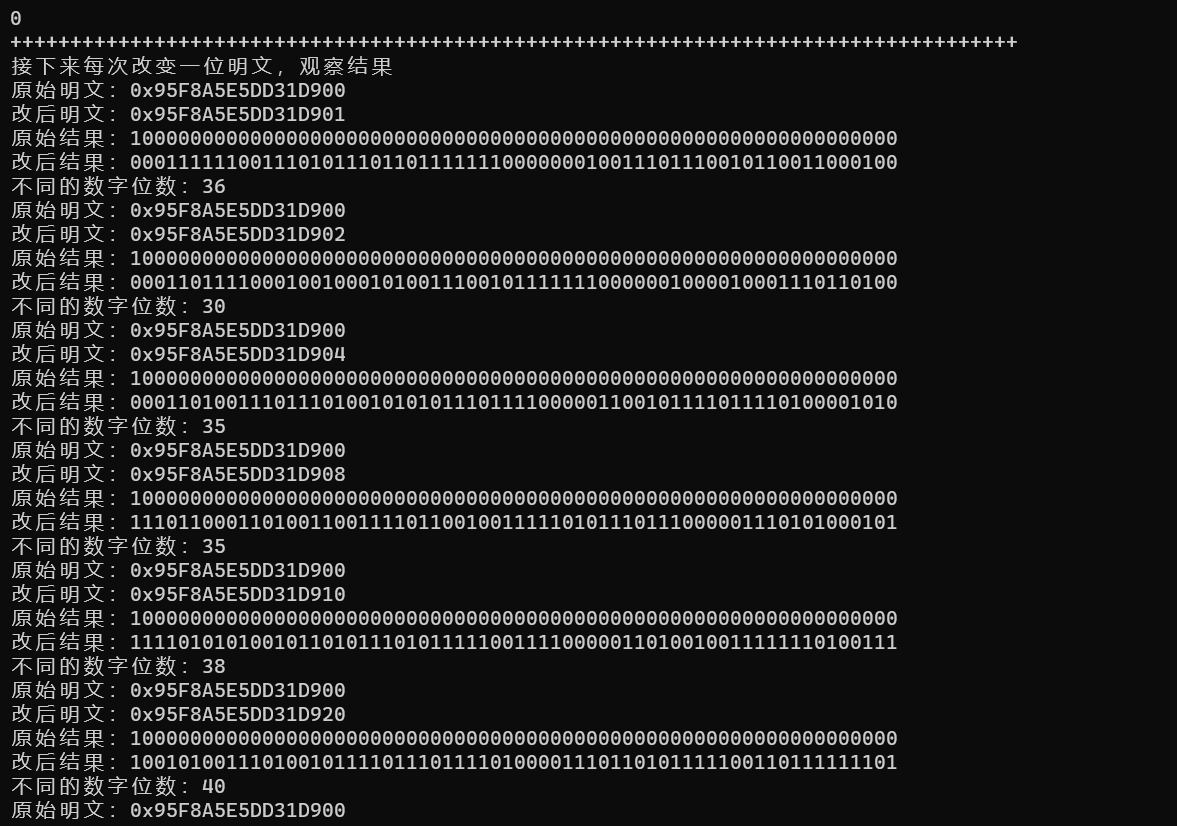
cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++" << endl;

}

最后的雪崩效应的输出结果如下：

改变明⽂：

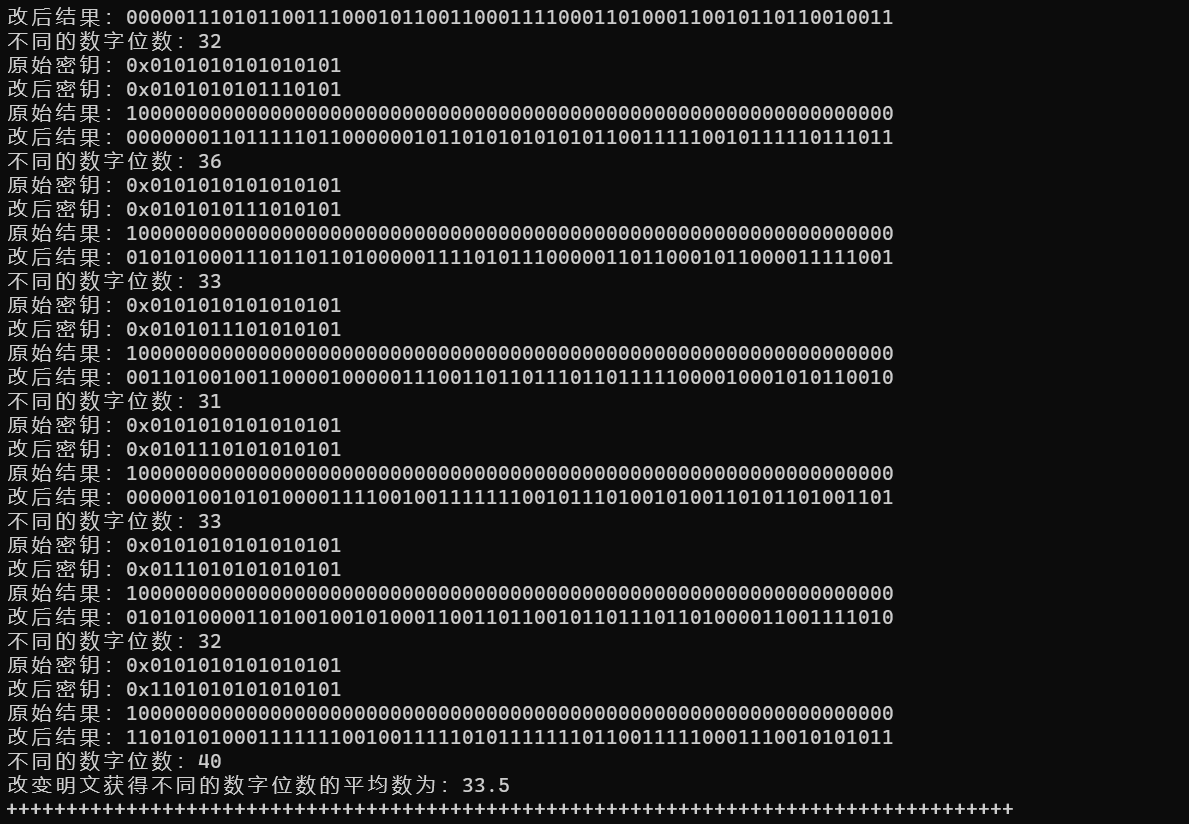
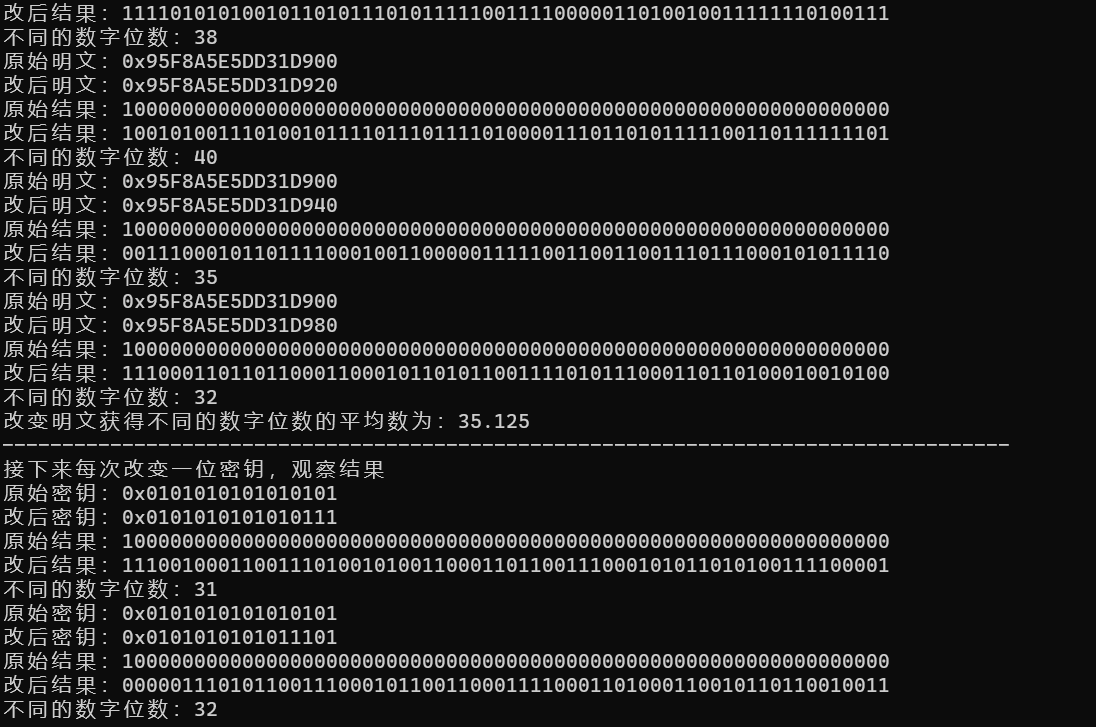
使秘钥不变，每次仅更改⼀位明⽂，重复⼋次，平均每次有35.125位发⽣变化



改变密钥：使明⽂不变，每次仅更改⼀位密钥，重复⼋次，平均每次有33.5位发⽣变化

综上可以看出即使⼩⼩的改动，加密结果也会发⽣很⼤的变化

接下来输出改变密钥的结果

四、总结与展望

在本次DES密码学实验中，我们学习了DES算法的基本原理及其加密过程。通过实验，我们深刻理解了DES算法的加密过程，并掌握了其加密与解密的方法。

在实验过程中，我们使用了Python语言编写程序，通过将明文输入程序中，经过DES算法加密后输出密文，再将密文输入程序中，经过DES算法解密后输出明文。通过程序的运行结果，我们可以验证DES算法的加密与解密过程的正确性，并且可以看到密文和明文的变化过程。

此外，我们还探究了DES算法的密钥长度对加密强度的影响，发现密钥长度越长，加密强度越高，但同时也会导致加密和解密的速度变慢。

展望：

随着计算机技术的不断发展，密码学的应用越来越广泛，而DES算法作为一种经典的加密算法，仍然具有重要的研究价值。未来，我们可以进一步探索DES算法的加密强度、密钥长度等方面的研究，并且可以考虑结合其他加密算法，进一步提高加密的安全性。同时，我们也可以将DES算法应用到实际的数据加密中，保护个人隐私和机密信息的安全。