西安电子科技大学

**应用密码学与网络安全 课程实验报告**

**实验名称 计算机与网络安全实验**

学院 班

成 绩

姓名 学号

同作者 无

实验日期 年 月 日

实验地点

|  |
| --- |
| 指导教师评语：  指导教师：  年 月 日 |
| **实验报告内容基本要求及参考格式**  一、实验目的  二、实验所用仪器（或实验环境）  三、实验基本原理及步骤（或方案设计及理论计算）  四、实验数据记录（或仿真及软件设计）  五、实验结果分析及回答问题（或测试环境及测试结果） |

# 一、实验报告

## 实验一

### 古典密码算法-Kaiser密码

##### 一、实验目的

1. 熟悉VC++6.0系统，设计实现一个Caesar加密工具，完成对文件的加解密。
2. 对实验结果进行分析和评价。

##### 二、实验所用仪器（或实验环境）

​​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

**基本原理**: 凯撒密码是一种简单的密码算法，也被称为移位密码。它的原理很简单：将明文中的每个字母按照固定的偏移量（例如向右移动三个位置）进行移位，得到密文。解密过程也很简单：将密文中的每个字母反向移动同样的偏移量，即可还原出明文。

**代码解析**:

在 main 函数中首先要求用户输入执行加密或解密操作的选项，然后再根据用户选择的操作类型分别执行对应的操作。在执行操作之前，用户需要输入密钥值 k，该值会作为加密解密算法的参数。在加密操作中，程序会读取指定文件中的明文，并将加密结果写入到另一个指定文件中；在解密操作中，程序会读取指定文件中的密文，并将解密结果写入到另一个指定文件中。

加密算法encrypt如下

//k为密钥,len为长度,fp为储存的加密文件

void encrypt(char c[],int len,int k,char e[],FILE \*fp){

int i = 0;

for(i=0;i<len;i++){

if(c[i]>='a'&&c[i]<='z'-k){

e[i] = c[i] + k;

}

else if(c[i]>='a'&&c[i]>'z'-k){

e[i] = c[i] + k - 26;

}

else if(c[i]>='A'&&c[i]<='Z'-k){

e[i] = c[i] + k;

}

else if(c[i]>='A'&&c[i]>'Z'-k){

e[i] = c[i] + k - 26;

}

else{

e[i] = c[i];

}

}

fputs(e,fp);

printf("加密方成功进行加密！\n");

}

encrypt 函数实现了加密过程，接收4个参数：原始消息字符串、字符串长度、密钥值以及用于存储加密结果的字符串数组，最后将结果写入指定文件中。加密算法中使用了一种简单的移位方法，将每个字符的ASCII码增加密钥值 k，如果超出字母 'z' 或 'Z' 的范围则需要回到字母表开头处。否则，不做任何更改。

解密算法decrypt如下所示

void decrypt(char c[],int len,int k,char d[],FILE \*fp)

{

int i = 0;

for(i=0;i<len;i++){

if(c[i]>='a'+k&&c[i]<='z'){

d[i] = c[i] - k;

}

else if(c[i]>='a'&&c[i]<'a'+k){

d[i] = c[i] - k + 26;

}

else if(c[i]>='A'+k&&c[i]<='Z'){

d[i] = c[i] - k;

}

else if(c[i]>='A'&&c[i]<'A'+k){

d[i] = c[i] - k + 26;

}

else{

d[i] = c[i];

}

}

fputs(d,fp);

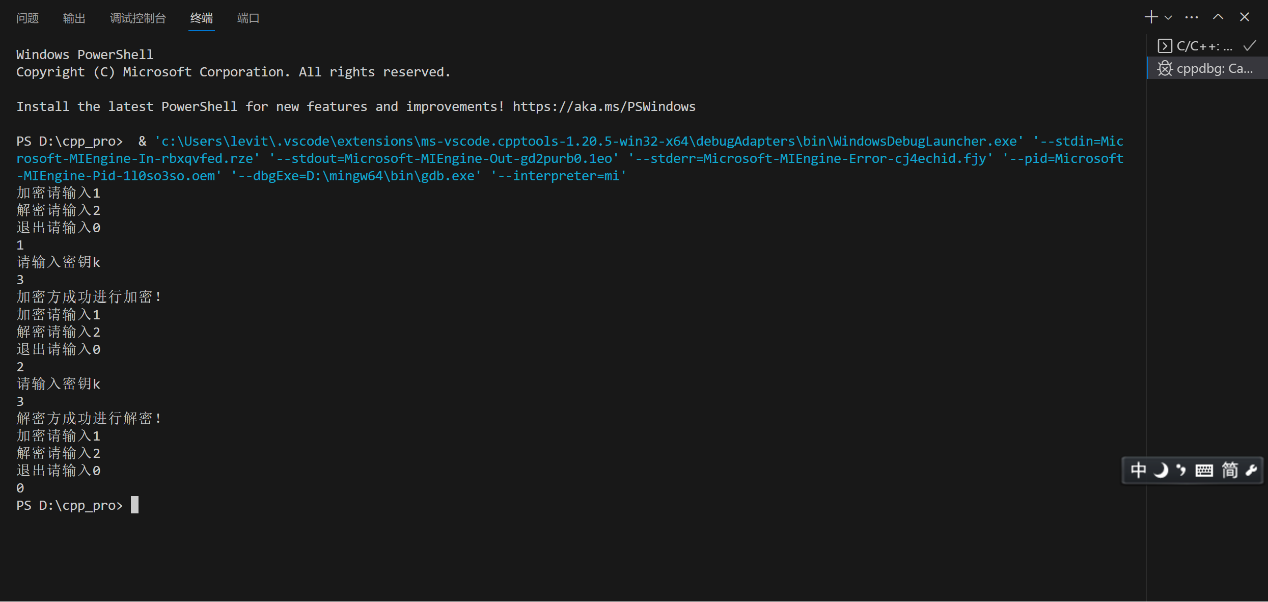
printf("解密方成功进行解密！\n");

}

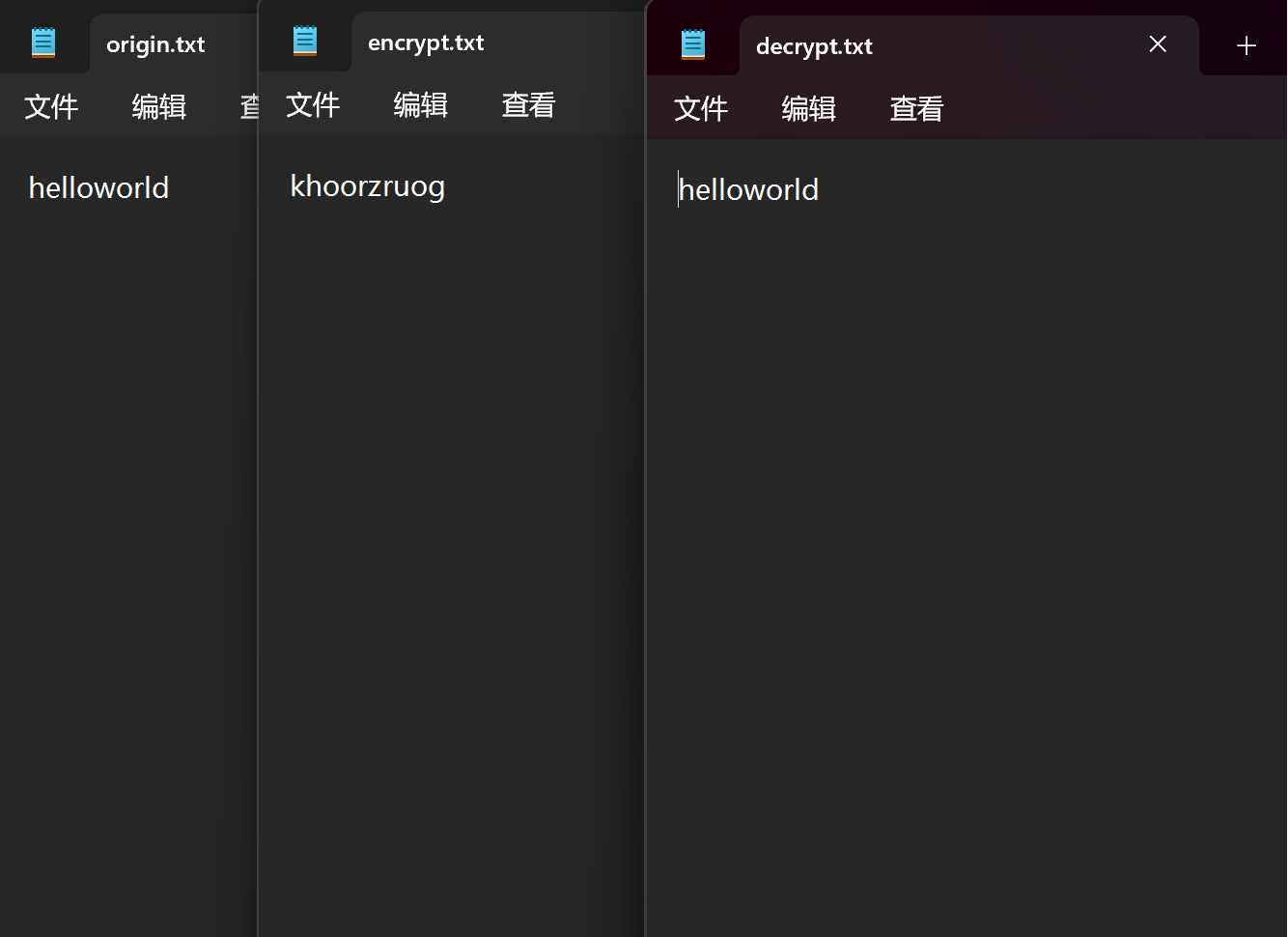
decrypt 函数实现了解密过程，接收4个参数：密文字符串、字符串长度、密钥值以及用于存储解密结果的字符串数组，最后将结果写入指定文件中。解密算法中同样采用了移位方法，但是需要将增加的密钥值 k 换成减少的密钥值 k。

##### 四、实验数据记录（或仿真及软件设计）

先对明文文件p.txt进行加密，密钥k为3，加密后的内容存储到c.txt中。再对密文文件c.txt进行解密，密钥值为3，解密后的结果存放到d.txt中



明文文件、密文文件以及解密文件内容如下所示



利用给的小工具对加密解密流程进行测试，可以看出加解密流程正确



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1. 在手动完成Caesar密码实验中，密钥k=3，试着画出这时的Caesar置换表？

| **明文** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **G** | **K** | **L** | **M** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **密文** | D | E | F | G | H | I | J | K | L | M | N | O | P |
| **明文** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z** |
| **密文** | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

1. 古典密码学曾经被广泛应用，它可以分为代替密码和置换密码两种，请查找相关资料，列举出几种属于代替密码和置换密码的古典密码算法？

代替密码：

* + 凯撒密码：将明文中的每个字母向右移动一个固定数量的位置，比如向右移动3个位置。
  + 维吉尼亚密码：结合了多个凯撒密码，通过使用不同的移动量来加密每个字母。
  + Hill密码：使用矩阵乘法来加密明文中的每个字母。
  + RC4：一种流密码，使用伪随机数生成器生成密钥流，再将其与明文异或运算得到密文。

置换密码：

* + 栅栏密码：将明文按照一定规则分成若干组，然后将每组中的字母重新排列。
  + 简单替换密码：使用一个简单的固定字母表来加密明文，即将明文中的每个字母替换为另一个字母。
  + 频率分析密码：利用字母出现频率的统计规律来猜测密文中的字母，从而解密消息。
  + Playfair密码：将明文中的每个字母映射到一个二维矩阵中的两个字母，然后将这两个字母替换为矩阵中的另外两个字母。

## 实验二

### 古典密码算法-单表置换密码

##### 一、实验目的

1. 设计一个单表置换加密工具，利用单表置换加密算法对文件进行加密。
2. 对实验结果进行分析和评价。

##### 二、实验所用仪器（或实验环境）

​​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

单表置换加密（单字母替换密码）在加密时将明文中的每个字符按照预先定义好的顺序替换成另一个字符，生成密文。解密时则根据相同的替换顺序进行反向替换，恢复原文。

主函数 main() 的代码如下

int main(){

FILE \*fp1,\*fp2;

int m;

do{

printf("加密请输入1\n解密请输入2\n退出请输入0\n");

scanf("%d",&m);

if(m==0)

break;

else if(m!=0&&m!=1&&m!=2)

printf("输出格式有误，请重新输入！\n");

else

{

char k[26]={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'},

\*c,\*s;

c = (char\*)malloc(sizeof(char));

s = (char\*)malloc(sizeof(char));

getchar();

printf("请输入密钥：\n");

scanf("%[^\n]",c);

int len = strlen(c);

int i,j;

for(i=0,j=0;i<len;i++,j++){

if(isupper(c[i]))

s[j] = c[i];

else if(islower(c[i]))

s[j] = toupper(c[i]);

else

j--;

}

len = strlen(s);

int flag = 0;

int min = 26;

for(i=0;i<len;i++){

int temp = 1;

for(j=0;j<flag;j++){

if(s[i]==k[j]){

temp = 0;

break;

}

}

if(temp){

if(s[i]!=k[flag]){

int n = (s[i]-'A'>min)?s[i]-'A':flag+s[i]-'A';

for(n;n>flag;n--){

k[n] = k[n-1];

}

k[flag] = s[i];

}

flag++;

if(s[i]-'A'<min)

min = s[i]-'A';

}

}

if(m==1)

{

encrypt(k);

printf("加密方成功进行加密！\n");

}

else if(m==2)

{

decrypt(k);

printf("解密方成功进行解密！\n");

}

}

}while(m!=0);

return 0;

}

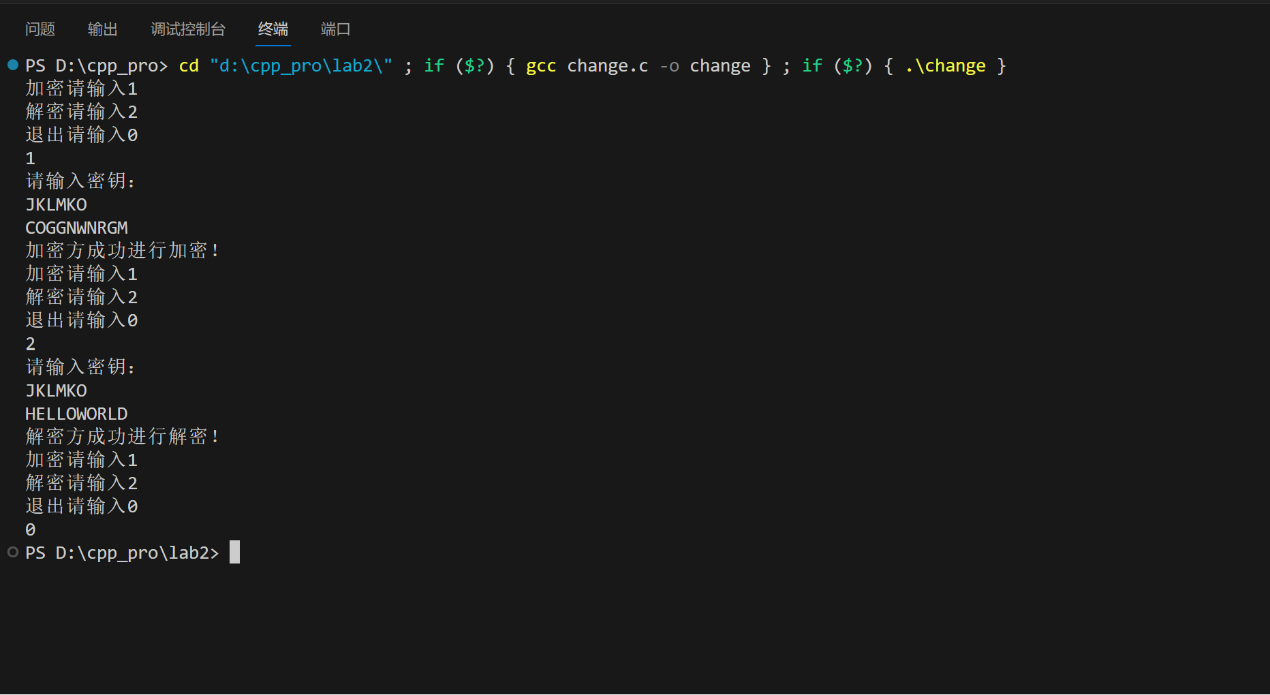
主函数 main() 中，首先定义了两个文件指针 fp1 和 fp2，然后通过 do-while 循环不断接受用户输入，直到用户输入 0 退出程序。在循环内部，先让用户选择加密还是解密，如果输入有误则重新输入。如果用户选择加密或解密，程序会要求用户输入一个密钥。这里通过 scanf() 函数读取密钥字符串，然后使用 isupper() 和 toupper() 函数将其转换为大写字母，并将所有非字母字符删除。接下来程序会遍历密钥字符串，对于每个字母，如果它还没有出现在密钥表中，就将它插入到密钥表中。密钥表中的字母按照它们在密钥字符串中出现的顺序排序，如果有相同字母，则以第一次出现的位置为准。最后，程序会遍历密钥表，将其中未出现在密钥字符串中的字母按顺序填充到密钥表中，并生成加密表（数组k）。最后根据用户的选择调用 encrypt() 或 decrypt() 函数进行加密或解密，并输出相应的提示信息。

函数 encrypt 中，首先打开名为 "origin.txt" 的文件并将其读入到字符数组 a 中，然后遍历该数组中的每个字符，如果是小写字母则将其转换为大写字母；接着，将置换表 k 中对应位置的字符替换原来在 a 中相同位置的字符；最后将修改后的字符串写入名为 "encrypt.txt" 的文件中。

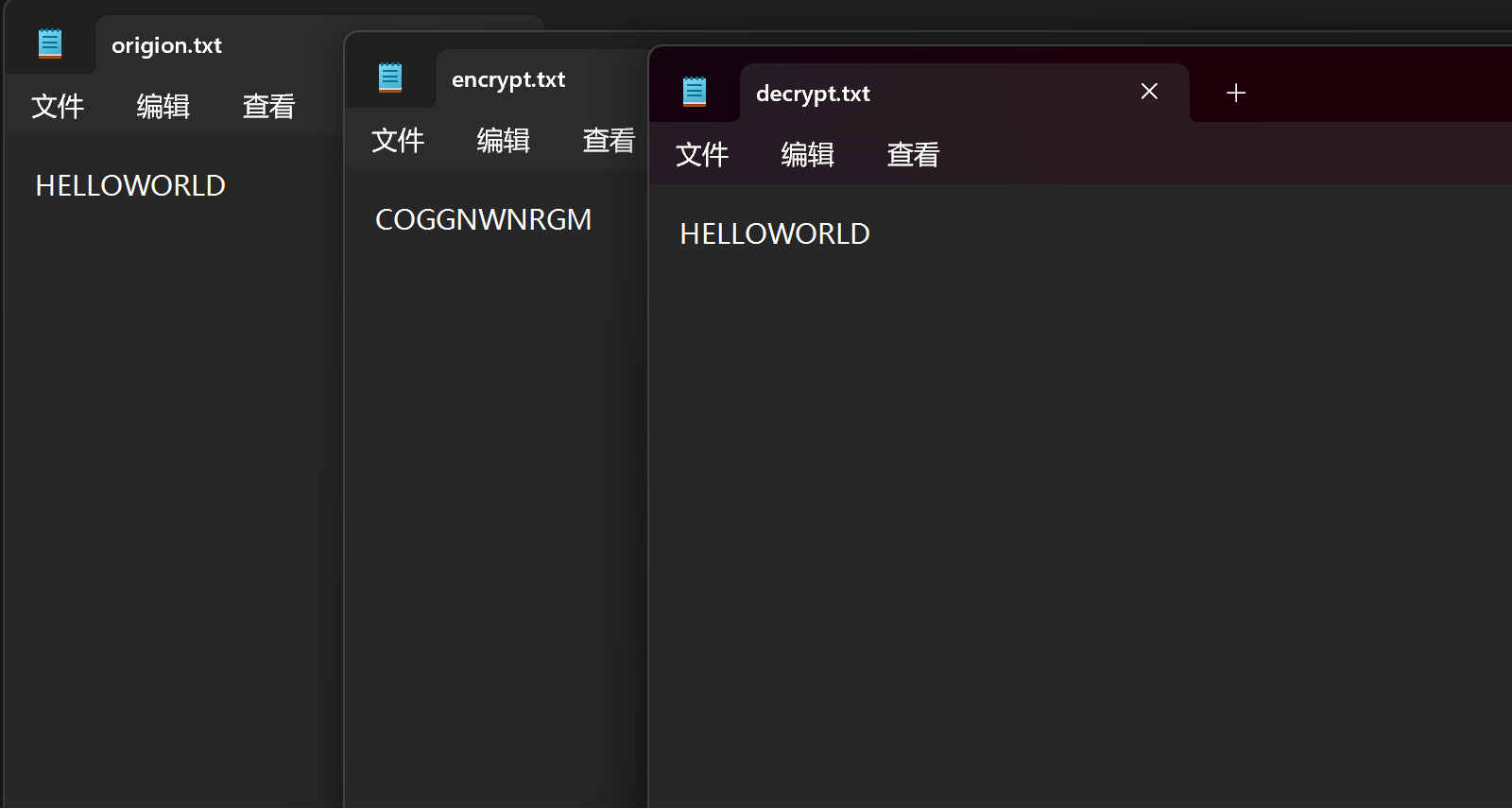
函数 decrypt 中，首先打开名为 "encrypt.txt" 的文件并将其读入到字符数组 b 中，然后遍历该数组中的每个字符，将 k 中对应位置的字符替换原来在 b 中相同位置的字符；最后将修改后的字符串写入名为 "decrypt.txt" 的文件中，解密时并没有进行大小写转换。

**四、实验数据记录（或仿真及软件设计）**

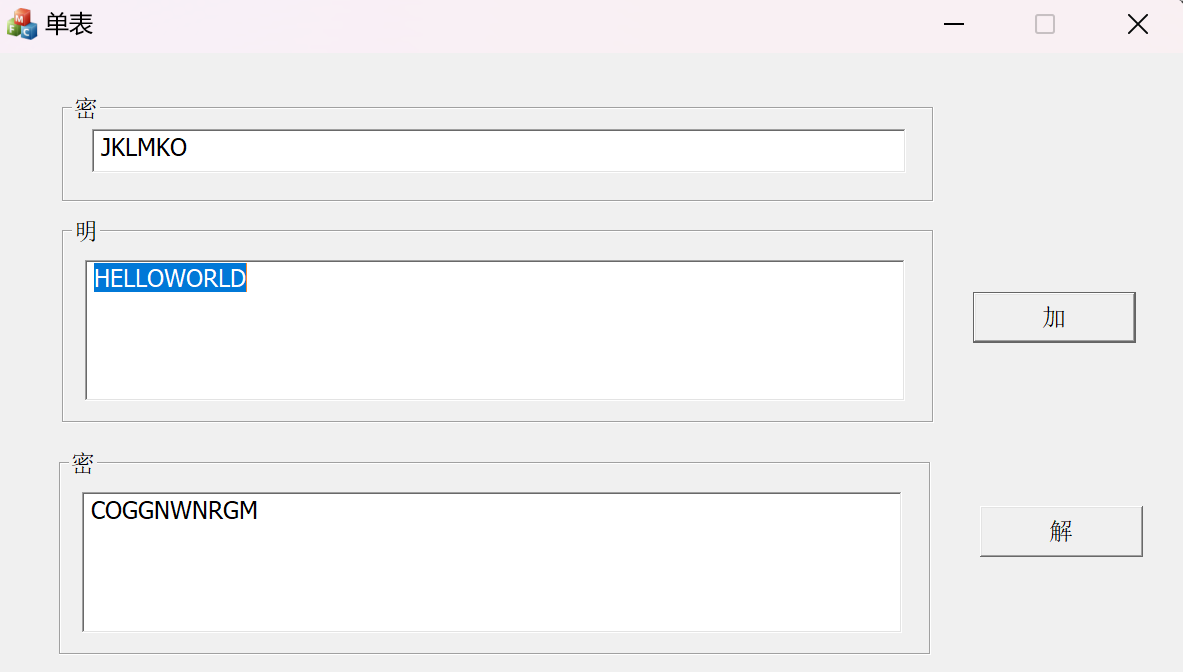
先对明文文件origin.txt进行加密，密钥为JKLMKO，加密后的内容存储到encrypt.txt中。再对密文文件encrypt.txt进行解密，密钥为JKLMKO，解密后的结果存放到decrypt.txt中



明文文件、密文文件以及解密文件内容如下所示



利用给的小工具对加密解密流程进行测试，可以看出加解密流程正确



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

问：在单表置换密码分析过程中，我们看到破解方法是基于英文字母出现的频率，你能想出一个改进方法使单表置换加密方法能抵抗这种方法的密码分析吗？

答：一种改进单表置换加密方法的方式是使用多个置换表，而不是只使用一个。具体来说，可以为每个字母指定多个置换表中的一个，然后随机选择一个置换表进行加密。这样可以使得破解者无法推断出某个字母在哪个置换表中出现的频率，并且提高了密码的安全性。另外一种改进的方式是引入一些随机性和混淆技术，以增加密码的复杂度和难度。例如，可以使用随机生成的替代字符或随机的字符串来替换明文中的某些字符，以及在加密过程中引入一些随机的步骤，如随机交换字符位置或随机选择加密算法等等。这些技术可以使得密码更难被破解，同时保持单表置换加密方法的简单性和可行性。

## 实验三

### 古典密码算法-仿射密码

##### 一、实验目的

1. 编写调试仿射密码算法，实现对文件的加解密
2. 对实验结果进行分析和评价。

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

基本原理：仿射密码是一种替换密码。它是一个字母对一个字母的。它的加密过程可以表示为下面的函数：e(x)=(ax+b) mod m其中，a和m互质，m是字母的数目。它的解密过程可以表示为下面的函数：d(x)=a^(x-b)\ mod m其中a^是a对m的乘法逆元。

**代码解析：**

主函数main的代码如下

int main(){

int a,b,m;

do{

printf(" 加密请输入1\n 解密请输入2\n 退出请输入0\n");

scanf("%d",&m);

if(m==0)

break;

else if(m!=0&&m!=1&&m!=2)

printf("输入有误，请重新输入！\n");

else{

printf("请输入密钥a、b，其中a与26互素\n");

scanf("%d %d",&a,&b);

if(!(ifisPrime(a,26))){

while(!(ifisPrime(a,26))){

printf("经检测，输入a值与26不满足互素条件，请输入与26互素的a!\n");

scanf("%d",&a);

}

}

if(m==1)

encrypt(a,b);

else if(m==2)

decrypt(a,b);

}

}while(m!=3);

return 0;

}

主函数会根据用户输入的选项进行加密、解密或退出。选择加密时或者解密时提醒用户输入密钥a和b，并检查a和26是否互素。如果用户输入的是加密选项，则调用encrypt函数进行加密；如果用户输入的是解密选项，则调用decrypt函数进行解密。如果用户输入的是退出选项，则程序结束。

判断a和b是否互质的函数 ifisPrime以及求a和b的最大公约数函数gcd

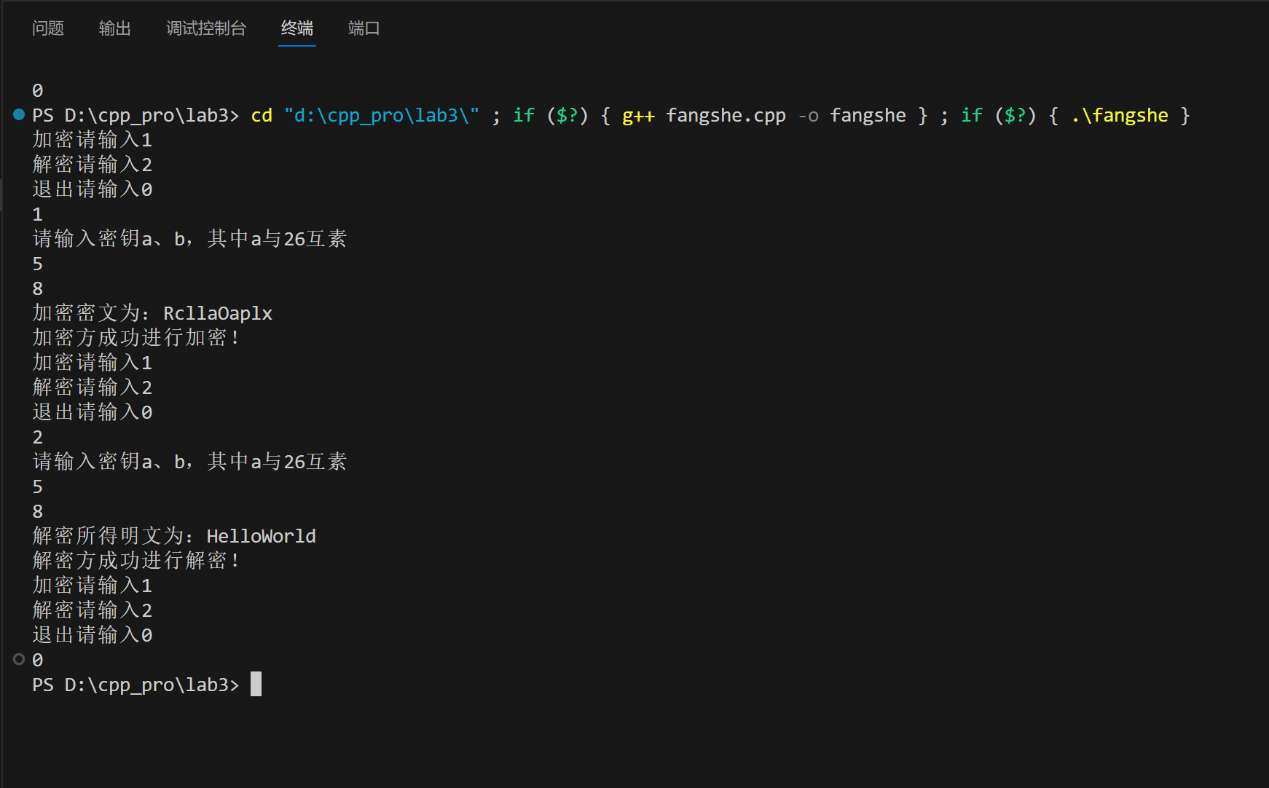
ifisPrime原理即调用gcd函数计算a和b的最大公约数，如果最大公约数为1，则返回1，否则返回0。gcd 即使用辗转相除法（欧几里德算法）计算a和b的最大公约数，直到b为0，返回a的值。

getinverse函数：计算a在模m下的乘法逆元，主要原理是通过遍历1到m-1的整数，找到满足(i \* a) % m = 1的数i，并返回i。如果不存在则返回0。

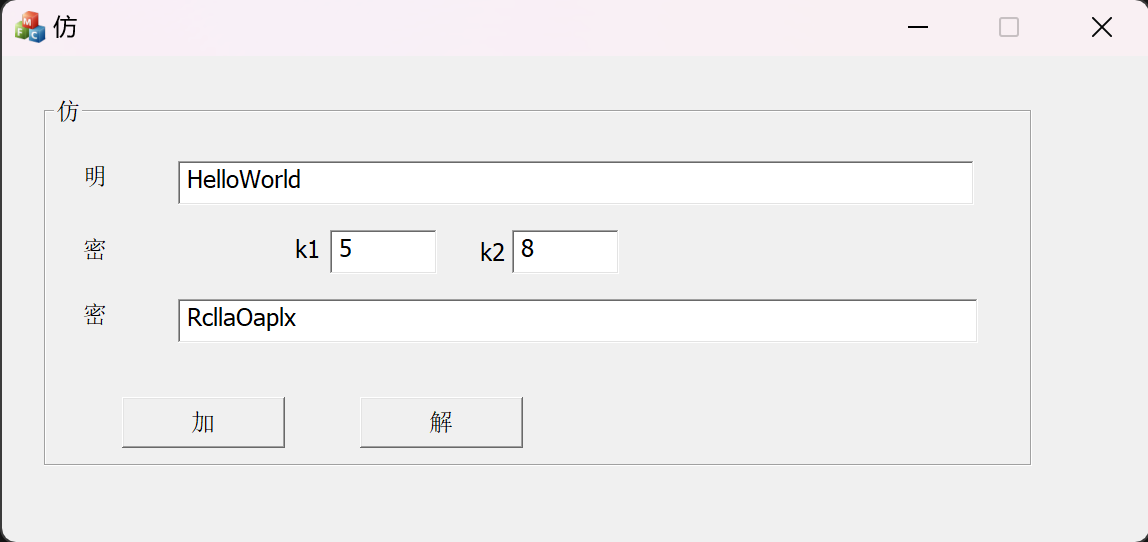
加密函数主要利用e(x)=(ax+b) mod m公式进行加密，它的解密函数思想即d(x)=a^(x-b)\ mod m，利用getinverse函数计算a在模m下的乘法逆元，即可以求得明文。

##### 四、实验数据记录（或仿真及软件设计）

先对明文文件origin.txt（内容为HelloWorld）进行加密，密钥为a=7,b=8，加密后的内容存储到encrypt.txt中。再对密文文件encrypt.txt进行解密，密钥为a=5,b=8，解密后的结果存放到decrypt.txt中



利用给的小工具对加密解密流程进行测试，可以看出加解密流程正确



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1. 思考仿射变换为何要求k1与26互素？

当k1与26不互素时，会导致下面两种情况的发生：

* 1. 存在某个数字a，使得(a \* k1) mod 26 = 0，这意味着在进行仿射变换时，存在一些明文字符被映射到了另一些字符的同一个位置上，从而降低了加密强度。
  2. 密文字符中可能出现一些无法解密的字符，因为它们没有对应的明文字符。具体来说，如果gcd(k1, 26) ≠ 1，则不存在一个数x，使得(x \* k1) mod 26 = 1，也就是说，在进行解密操作时，有些密文字符无法还原成相应的明文字符。

因此，在仿射密码中，要求选择的k1必须与26互素，以保证加密过程的安全性，并且能够保证解密操作的可行性。

1. 思考仿射变换与凯撒密码的关系？

凯撒密码是一种简单的替换密码，它将每个字母向右或向左移动一个固定的数量（位移），从而生成密文。这个位移量可以看作是一个参数，类似于仿射变换中的矩阵。但是，凯撒密码只对字母进行了简单的移位操作，而没有对其进行缩放、旋转或错切等变换。与凯撒密码不同，仿射变换是一种更复杂的线性变换，其中包括平移、旋转、缩放和错切等操作。由于仿射变换的灵活性，它被广泛用于图像处理和计算机视觉领域中的几何变换。

## 实验四

### 对称密码算法-DES密码

##### 一、实验目的

1. 学习理解DES算法的工作原理
2. 设计DES加密程序，利用DES加密算法完成对文件加解密。
3. 连接远程实验系统，下载相关工具进行验证，记录实验过程和结果。
4. 对实验结果进行分析和评价。

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

**基本原理**: DES（Data Encryption Standard）是一种对称加密算法，其基本原理是通过利用相同的密钥对数据进行加密和解密，以确保数据的机密性。

DES算法的加密步骤如下：

1. 密钥生成：首先需要生成一个长度为64位的密钥，这个密钥将被用于加密和解密过程中。
2. 初始置换：将明文数据按照特定的规则进行重新排列，生成新的数据序列。这个步骤旨在增加加密强度和随机性。
3. 分组：将初始置换后的数据分成8个64位的块。
4. 加密轮次：DES算法共有16个加密轮次，每轮都需要进行以下四个步骤：

a) 扩展置换：将64位的数据块扩展成48位。

b) 轮密钥生成：根据当前轮数生成一个48位的轮密钥，用于加密过程中的异或操作。

c) S盒替换：将48位的数据块按照6位一组分成8组，每组使用不同的S盒进行替换，得到新的32位数据块。

d) P盒置换：对32位数据块进行重新排列，得到新的32位数据块。

1. 交换左右半部分：将最后一轮加密后得到的左半部分和右半部分交换。
2. 逆初始置换：将加密后的数据按照特定的规则进行重新排列，生成新的数据序列，作为密文输出

解密步骤与加密步骤相同，只是轮密钥的使用顺序与加密时相反。

**代码解析**:

主函数提供了DES算法加密和解密两种模式，输入1为加密模式，要求用户分别输入明文文件名、秘钥文件名和加密后的密文存储文件名，并调用DES\_Encrypt函数进行加密操作。输入2为解密模式，则要求用户分别输入密文文件名、秘钥文件名和解密后的明文存储文件名，并调用DES\_Decrypt函数进行解密操作。输入0则退出循环。

加密函数如下

/\*加密文件\*/

int DES\_Encrypt(char \*plainFile, char \*keyStr,char \*cipherFile){

FILE \*plain,\*cipher;

int count;

ElemType plainBlock[8],cipherBlock[8],keyBlock[8];

ElemType bKey[64];

ElemType subKeys[16][48];

if((plain = fopen(plainFile,"rb")) == NULL){

return PLAIN\_FILE\_OPEN\_ERROR;

}

if((cipher = fopen(cipherFile,"wb")) == NULL){

return CIPHER\_FILE\_OPEN\_ERROR;

}

/\*设置密钥\*/

memcpy(keyBlock,keyStr,8);

/\*将密钥转换为二进制流\*/

Char8ToBit64(keyBlock,bKey);

/\*生成子密钥\*/

DES\_MakeSubKeys(bKey,subKeys);

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){

DES\_EncryptBlock(plainBlock,subKeys,cipherBlock);

fwrite(cipherBlock,sizeof(char),8,cipher);

}

}

if(count){

/\*填充\*/

memset(plainBlock + count,'\0',7 - count);

/\*最后一个字符保存包括最后一个字符在内的所填充的字符数量\*/

plainBlock[7] = 8 - count;

DES\_EncryptBlock(plainBlock,subKeys,cipherBlock);

fwrite(cipherBlock,sizeof(char),8,cipher);

}

fclose(plain);

fclose(cipher);

printf("加密成功！\n");

}

这段代码是一个使用DES算法对文件进行加密的函数。主要实现思路如下：打开明文文件和密文文件，如果打开失败则返回相应的错误代码。把密钥转换为二进制流，并生成16个子密钥。每次读取8个字节的明文，如果成功读取，则对其进行加密并写入密文文件中。如果最后一次读取的字节数不够8个字节，则需要进行填充（补全），并在最后一个字节中记录填充的字符数量。关闭文件并输出“加密成功！”提示信息。其中，使用的函数包括：Char8ToBit64：把长度为8的字符串转换为长度为64的二进制流。DES\_MakeSubKeys：根据密钥生成16个子密钥。DES\_EncryptBlock：对长度为8的明文块进行加密，得到长度为8的密文块。

解密函数如下

int DES\_Decrypt(char \*cipherFile, char \*keyStr,char \*plainFile){

FILE \*plain, \*cipher;

int count,times = 0;

long fileLen;

ElemType plainBlock[8],cipherBlock[8],keyBlock[8];

ElemType bKey[64];

ElemType subKeys[16][48];

if((cipher = fopen(cipherFile,"rb")) == NULL){

return CIPHER\_FILE\_OPEN\_ERROR;

}

if((plain = fopen(plainFile,"wb")) == NULL){

return PLAIN\_FILE\_OPEN\_ERROR;

}

/\*设置密钥\*/

memcpy(keyBlock,keyStr,8);

/\*将密钥转换为二进制流\*/

Char8ToBit64(keyBlock,bKey);

/\*生成子密钥\*/

DES\_MakeSubKeys(bKey,subKeys);

/\*取文件长度 \*/

fseek(cipher,0,SEEK\_END);/\*将文件指针置尾\*/

fileLen = ftell(cipher); /\*取文件指针当前位置\*/

rewind(cipher); /\*将文件指针重指向文件头\*/

while(1){

/\*密文的字节数一定是8的整数倍\*/

fread(cipherBlock,sizeof(char),8,cipher);

DES\_DecryptBlock(cipherBlock,subKeys,plainBlock);

times += 8;

if(times < fileLen){

fwrite(plainBlock,sizeof(char),8,plain);

}

else{

break;

}

}

/\*判断末尾是否被填充\*/

if(plainBlock[7] < 8){

for(count = 8 - plainBlock[7]; count < 7; count++){

if(plainBlock[count] != '\0'){

break;

}

}

}

if(count == 7){/\*有填充\*/

fwrite(plainBlock,sizeof(char),8 - plainBlock[7],plain);

}

else{/\*无填充\*/

fwrite(plainBlock,sizeof(char),8,plain);

}

fclose(plain);

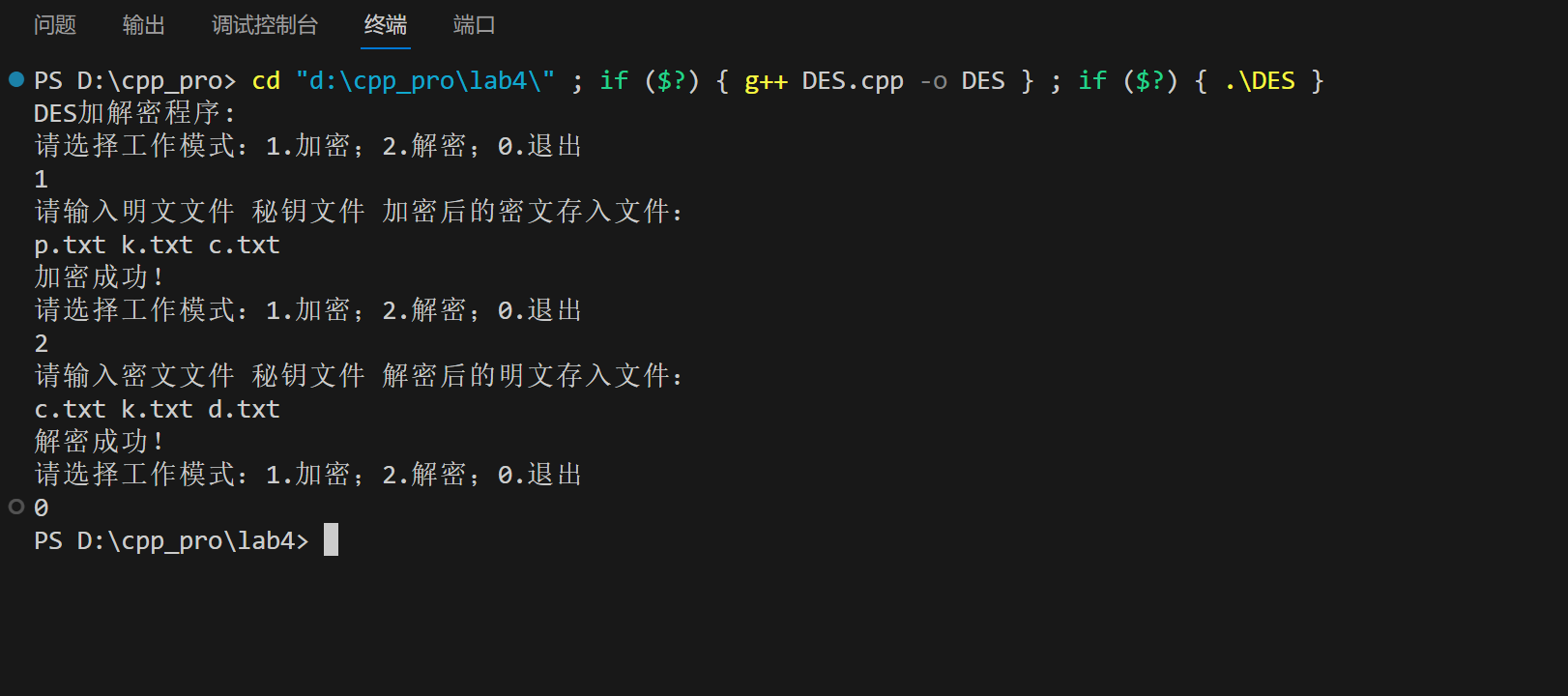
fclose(cipher);

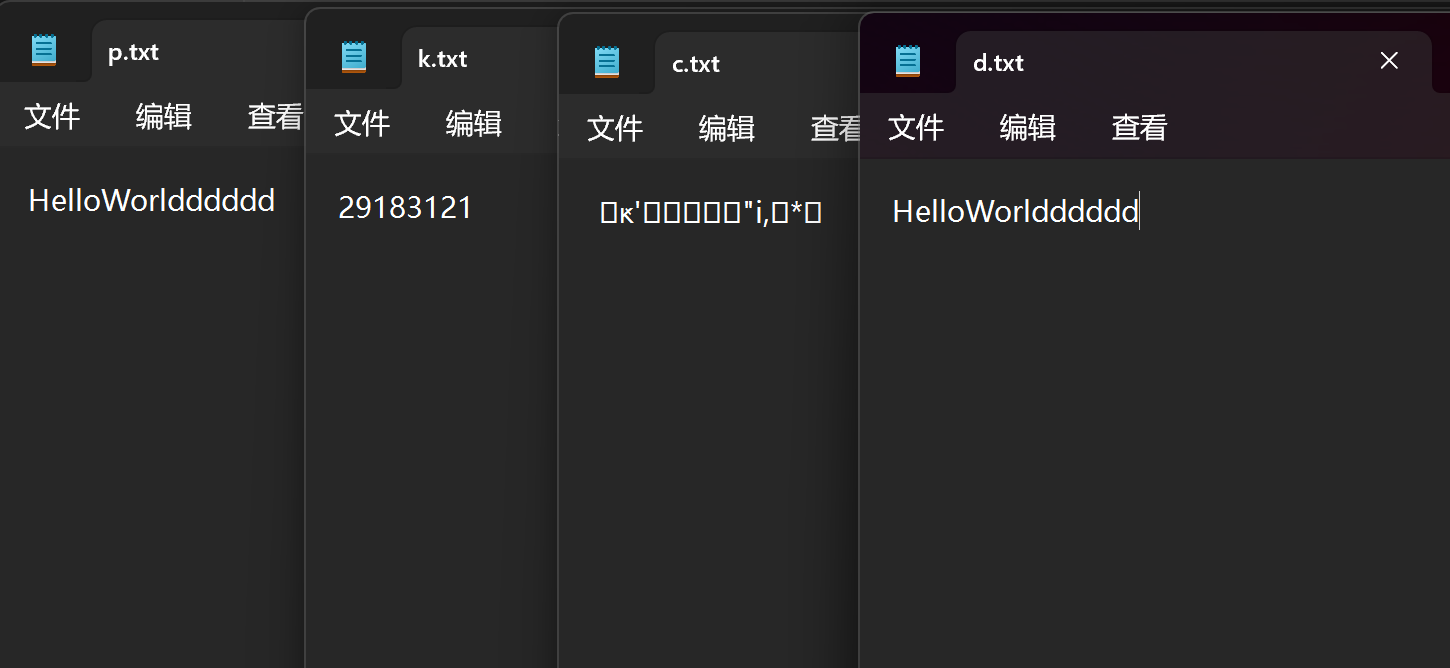
printf("解密成功！\n");

}

DES算法解密的步骤如下：打开密文文件和明文文件，如果打开失败则返回相应的错误代码。把密钥转换为二进制流，并生成16个子密钥。每次读取8个字节的密文，如果成功读取，则对其进行解密并写入明文文件中。判断最后一个字节中记录的填充字符数，如果有填充则去掉填充，否则直接写入剩余的字节。关闭文件并输出“解密成功！”提示信息。其中，使用的函数包括：Char8ToBit64：把长度为8的字符串转换为长度为64的二进制流。DES\_MakeSubKeys：根据密钥生成16个子密钥。DES\_DecryptBlock：对长度为8的密文块进行解密，得到长度为8的明文块。

##### 四、实验数据记录（或仿真及软件设计）





##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1、根据实验原理的讲解，回答下列问题：

​ (1) DES每一个明文分组的长度是多少位？这个明文分组加密后的密文是多少位？

​ DES加密算法的明文分组长度为64位，即8个字节。经过加密后，每一个明文分组将会生成64位（8个字节）的密文。

​ (2) 在DES算法的各种置换中，哪个置换为DES提供了最好的安全性？

​ S盒置换为DES提供了最好的安全性。S盒置换将每个6比特的输入映射到一个4比特的输出，这意味着它通过一个非线性的转换对每个6比特的输入进行加密。由于每个S盒都有自己独特的变换规则，并且DES使用8个不同的S盒，因此它们共同提供了DES的强安全性。

2、DES的S盒在设计时就能够防止某些类型的攻击，当1991年Biham和Shamir发现的差分攻击的方法时，美国国家安全局就已承认某些未公布的S盒设计原则正是为了使得差分密码分析变得不可行。也就是说，差分密码分析在DES最初被研发时就已为IBM的研发者所知，但是这种方法却被保密了将近20年，直到Biham和Shamir又独立地发现了这种攻击。目前，DES加密方法已经被认为是不安全的了，请同学查阅相关资料，列出两种DES的分析方法。

* 差分密码分析 差分密码分析是一种广泛应用于对分组密码进行攻击的技术。它利用了加密算法中相邻明文之间的差异，并基于这些差异对密钥进行推断。在DES中，差分密码分析是一种强大的攻击方法，因为DES的S盒具有良好的差分性质。差分密码分析可以通过构造满足一定条件的明文对来推断子密钥，然后通过将这些子密钥组合来获得完整的密钥。
* 线性密码分析 线性密码分析是另一种常用的对分组密码进行攻击的技术。它利用了加密算法中明文、密文和密钥之间的线性关系，以及某些位上的偏差来推断出密钥。在DES中，线性密码分析可以被用来找到DES的轮函数的一些比较弱的特征，然后利用这些特征来推断出子密钥。但是，与差分密码分析相比，线性密码分析需要更多的数据量和计算资源。

## 实验五

### 对称密码算法-AES密码

##### 一、实验目的

1. 学习理解AES算法的工作原理
2. 设计AES加密程序，利用AES加密算法完成对文件加解密。
3. 连接远程实验系统，下载相关工具进行验证，记录实验过程和结果。
4. 对实验结果进行分析和评价。

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

AES（Advanced Encryption Standard）是一种对称加密算法，它将明文分成固定长度的块，然后通过多轮运算对每个块进行加密。下面是AES加密的流程：

1. 密钥扩展。AES加密使用一个密钥，密钥长度可以为128位、192位或256位。在加密之前，需要对密钥进行扩展，生成一系列子密钥。密钥扩展算法会根据密钥长度和AES的加密轮数，计算出需要的子密钥。
2. 初始轮密钥加。将明文块与第一个子密钥进行异或操作。
3. 多轮加密。AES加密使用多轮加密，每轮都包含四个步骤：字节代换、行移位、列混淆和轮密钥加。这四个步骤在所有轮中重复执行，直到达到指定轮数。
   * 字节代换：将每个字节替换为一个固定的值，这个值由S盒提供。S盒是一个256×8的查找表，它使用输入字节的不同组合来提供输出值。这个步骤实现了非线性变换。
   * 行移位：对每行进行循环移位操作。第一行不变，第二行向左移动一个字节，第三行向左移动两个字节，第四行向左移动三个字节。这个步骤实现了一些位移的效果，增加了变换的复杂性。
   * 列混淆：对每列进行矩阵乘法操作。列混淆使用一个固定的矩阵进行乘法，这个矩阵是通过数学分析得出的。这个步骤实现了另一种非线性变换。
   * 轮密钥加：将每个字节与轮密钥进行异或操作。
4. 最终轮。最后一轮没有列混淆步骤，只有字节代换、行移位和轮密钥加步骤。
5. 密文输出。经过多轮加密之后，得到密文块。如果还有其他的明文块需要加密，重复上述过程即可。

main函数主要是对AES加密算法进行了测试。其流程如下：用户输入一个128位的十六进制格式的密钥以及明文，然后使用sscanf函数将其转换为字节类型的密钥以及明文，并逐一存放进key与message中。然后声明一个w数组，用于存储扩展密钥后的结果。调用KeyExpansion函数，将key作为参数传入该函数，进行密钥扩展操作，并将扩展后的结果按顺序存储进w数组中。调用encrypt函数进行AES加密操作，输出加密后的16进制密文结果。之后进行解密，输入相同的密钥，即可以对message进行AES解密操作，解密后的结果会转换成16进制输出。

下面是AES中的密钥扩展函数，用于从一个128位的密钥中生成一系列轮密钥（round key）。在AES中，加密和解密过程都需要使用这些轮密钥。

//密钥扩展函数 - 对128位密钥进行扩展得到 w[4\*(Nr+1)]

void KeyExpansion(byte key[4 \* Nk], word w[4 \* (Nr + 1)]){

word temp;

int i = 0;

// w[]的前4个就是输入的key

while (i < Nk){

w[i] = Word(key[4 \* i], key[4 \* i + 1], key[4 \* i + 2], key[4 \* i + 3]);

++i;

}

i = Nk;

while (i < 4 \* (Nr + 1)) {

temp = w[i - 1]; // 记录前一个word

if (i % Nk == 0){

word temp2 = RotWord(temp);

w[i] = w[i - Nk] ^ SubWord(temp2) ^ Rcon[i / Nk - 1];

}

else

w[i] = w[i - Nk] ^ temp;

++i;

}

}

该函数接收一个长度为4Nk的字节数组key，其中Nk表示输入的密钥长度；并生成一个长度为4\*(Nr+1)的轮密钥数组w，其中Nr表示AES的轮数（对于128位密钥而言，Nr=10）。该函数首先将输入的密钥分成Nk个32位的字，并将它们存入w数组的前Nk个位置。然后，它进入一个循环，生成剩余的轮密钥。在每一轮中，它会先记录前一个word，然后根据当前轮数和Nk是否整除当前轮数，分别进行RotWord、SubWord和Rcon等操作，最终得到新的轮密钥。其中，RotWord操作将一个word的四个字节循环左移一位，SubWord操作将一个word的四个字节通过S-box进行替换，Rcon操作则是一个常量数组，用于生成某些特定的轮密钥。最后，通过异或操作将各个word组合起来，生成新的轮密钥。

下面是加密函数

void encrypt(byte in[4 \* 4], word w[4 \* (Nr + 1)]){

word key[4];

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

key[i] = w[i];

AddRoundKey(in, key);

for (int round = 1; round<Nr; ++round)

{

SubBytes(in);

ShiftRows(in);

MixColumns(in);

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

key[i] = w[4 \* round + i];

AddRoundKey(in, key);

}

SubBytes(in);

ShiftRows(in);

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

key[i] = w[4 \* Nr + i];

AddRoundKey(in, key);

}

具体而言，加密函数首先通过AddRoundKey操作将输入in与密钥的前4个字单元进行异或运算。然后，在接下来的Nr-1轮中，它会对输入执行一系列操作——SubBytes（使用S盒将输入的每个字节替换为另一个字节）、ShiftRows（将每行字节向左移动特定数量的位置）和MixColumns（将每列字节乘以特定矩阵并进行模多项式加法运算）——并使用当前轮数的密钥key进行AddRoundKey（简单的异或操作，它将输入与当前轮次的密钥进行异或运算）操作。最后，在第Nr轮中，它只执行SubBytes、ShiftRows和AddRoundKey操作，没有MixColumns操作。

下面是解密函数

void decrypt(byte in[4 \* 4], word w[4 \* (Nr + 1)]){

word key[4];

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

key[i] = w[4 \* Nr + i];

AddRoundKey(in, key);

for (int round = Nr - 1; round>0; --round)

{

InvShiftRows(in);

InvSubBytes(in);

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

key[i] = w[4 \* round + i];

AddRoundKey(in, key);

InvMixColumns(in);

}

InvShiftRows(in);

InvSubBytes(in);

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

key[i] = w[i];

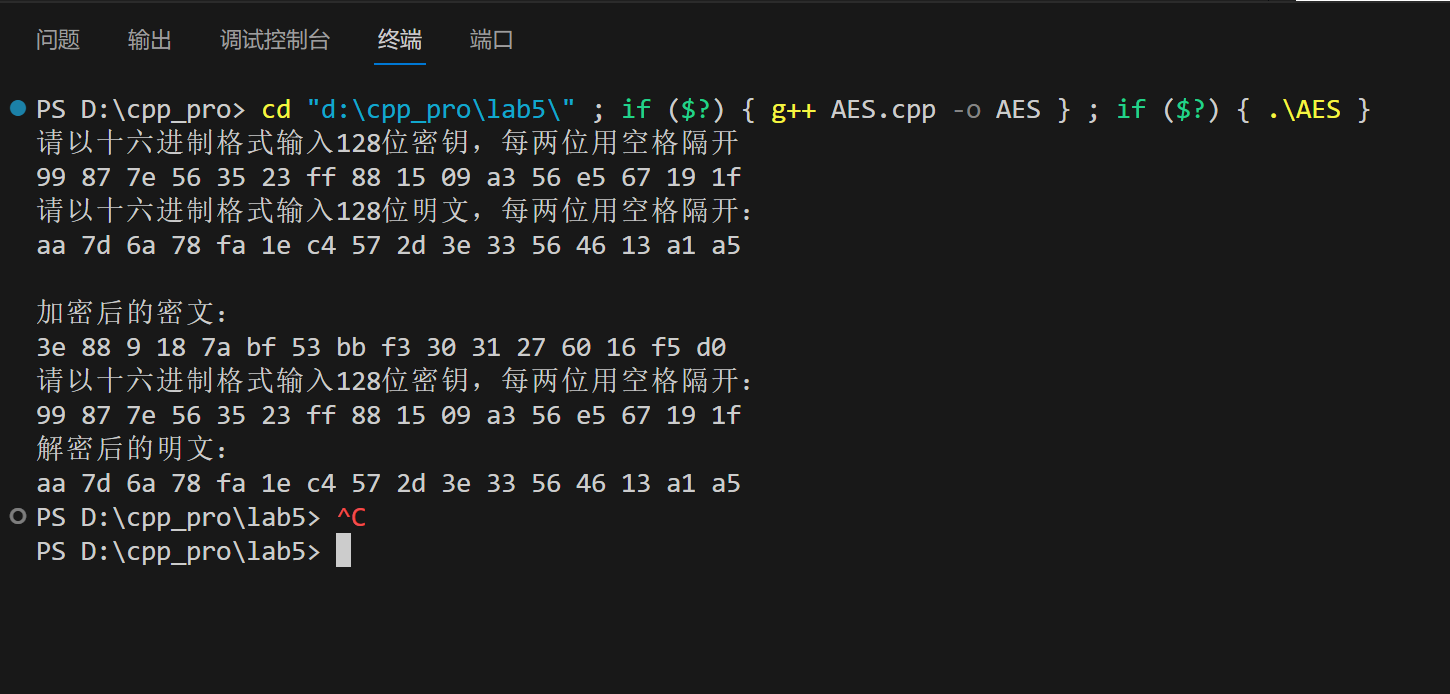
AddRoundKey(in, key);

}

解密函数首先从w中提取最后一轮密钥，并使用AddRoundKey操作将其与输入进行异或运算。随后，在接下来的Nr-1轮中，该函数执行一系列操作：首先对输入执行InvShiftRows操作，然后执行InvSubBytes操作，并使用当前轮数的密钥与输入进行AddRoundKey操作。接着，它执行InvMixColumns操作，以撤销AES加密算法中的MixColumns操作。在第Nr轮中，函数仅执行InvShiftRows、InvSubBytes和AddRoundKey操作，没有InvMixColumns操作。最后，函数使用密钥扩展数组中的第一个字作为最终轮次的密钥，并使用AddRoundKey操作对输入进行最后一次解密。

##### 四、实验数据记录（或仿真及软件设计）

输入密钥：99 87 7e 56 35 23 ff 88 15 09 a3 56 e5 67 19 1f以及明文：aa 7d 6a 78 fa 1e c4 57 2d 3e 33 56 46 13 a1 a5，进行相应的加密，可以得出加密的结果是3e 88 9 18 7a bf 53 bb f3 30 31 27 60 16 f5 d0，输入对应密钥再对加密后的结果进行解密操作得到原始明文aa 7d 6a 78 fa 1e c4 57 2d 3e 33 56 46 13 a1 a5。



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1. 在AES加密的10轮循环中，前9轮与第10轮有什么不同？

前9轮和第10轮使用的密钥是不同的。其次，在前9轮中，AES算法使用的转换操作是SubBytes、ShiftRows、MixColumns和AddRoundKey。这些操作将输入块分割成16个字节，然后通过置换、替换和线性变换来混淆它们。在第10轮中，AES算法只使用SubBytes、ShiftRows和AddRoundKey三种转换，而没有MixColumns操作。MixColumns操作将每个字节与特定矩阵相乘并进行模多项式加法运算，增强了算法的扰动和非线性性质。但是，在最终一轮中，该操作被省略，因为它会使得解密过程更加困难。

1. “AES算法和DES算法一样，都是对比特进行操作从而完成加密的”，你认为这句话对吗？

尽管AES算法和DES算法都是对比特进行操作，但它们的加密方式是不同的。在DES算法中，明文被分成64位块，并使用56位密钥进行加密。算法将每个块分为左右两个32位半块，并对右半块应用一系列置换、替换、移位和异或运算。算法重复16轮以上的步骤来加密每个块，每轮都涉及到了密钥的子集。而在AES算法中，明文被分成128位块，并使用128位密钥进行加密。算法将每个块视为一个4x4矩阵，并执行SubBytes、ShiftRows、MixColumns和AddRoundKey等转换操作，使得每个元素都有机会与其他元素之间相互影响。AES算法重复10轮以上的步骤来加密每个块，每轮都需要使用轮密钥的不同子集。因此，AES和DES算法都是基于对比特进行操作的块密码算法，但它们具体的加密方式和操作过程是不同的。

## 实验六

### 非对称密码算法-RSA密码

##### 一、实验目的

1. 掌握RSA生成公钥私钥的过程；
2. 熟练掌握利用RSA密钥对指定内容的加解密进程。
3. 设计RSA加密工具，利用RSA加密算法对文件进行加密。

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

RSA（Rivest–Shamir–Adleman）是一种非对称加密算法，其加解密过程如下：

1. 密钥生成：随机生成两个大素数p和q，并计算模数n=p\*q、φ(n)=(p-1)\*(q-1)，选择一个整数e（1<e<φ(n)）作为公钥，使得e和φ(n)互质，然后计算d，使得d\*e = 1 (mod φ(n))，即d是e在模φ(n)意义下的乘法逆元。公钥为{e, n}，私钥为{d, n}。
2. 加密：将明文转换成整数M（通常使用编码方案进行转换），然后用公钥{e, n}来计算密文C = M^e (mod n)。
3. 解密：接收到密文C后，使用私钥{d, n}来计算出明文M = C^d (mod n)。

主函数实现了一个基本的RSA加密算法。通过获取输入的两个素数p和q，计算得到模数n。然后输入公钥e，公钥e应该满足1<e<φ(n)，且e与(p - 1)\*(q - 1)互质，所以会对输入的e进行一个是否是素数的判断。再根据欧拉函数求出私钥d。然后将明文m加密为密文c，并输出密文；同时也能将密文解密为明文。在加密和解密过程中，使用了encrypt()函数来实现加解密操作。encrypt()函数实际上是是一个幂取余的过程。

下面是判断整数是否为素数的函数Prime(int m)，其输入参数为一个整数m。这个函数用于判断输入的p,q是否是素数。

int Prime(int m){

int i, k;

if (m<2) return 0;

k = sqrt(m);

for (i = 2; i <= k; i++)

if (m%i == 0)

break;

//素数返回1,否则返回0

if (i>k)

return 1;

else

return 0;

}

这段代码的实现逻辑如下：

* 首先，如果输入的m小于2，直接返回0，因为小于2的数都不是素数。
* 接着，定义变量k等于m的平方根（sqrt(m)）向下取整的值，这是因为一个数如果有因子n，那么必定存在另一个因子m/n。而且n和m/n中必有一个小于或等于sqrt(m)，另一个大于或等于sqrt(m)。因此只需要从2遍历到k就可以找到m的所有因子，从而判断m是否为素数。
* 然后使用for循环，从2开始遍历到k，依次判断m能否被i整除。如果存在可以整除的i，则跳出循环。
* 最后，在循环结束后，检查i是否大于k。若大于k，说明m没有除了1和自身之外的其它因子，即m是素数，返回1；否则，说明m有其他因子，返回0。

下面是判断两数是否互素的函数，主要用于判断公钥e是否与(p - 1)\*(q - 1)互质

int husu(int x, int y){

int t;

while (y){

t = x;

x = y;

y = t%y;

}

//x与y互素时返回0，x与y不互素时返回1

if (x == 1)

return 0;

else

return 1;

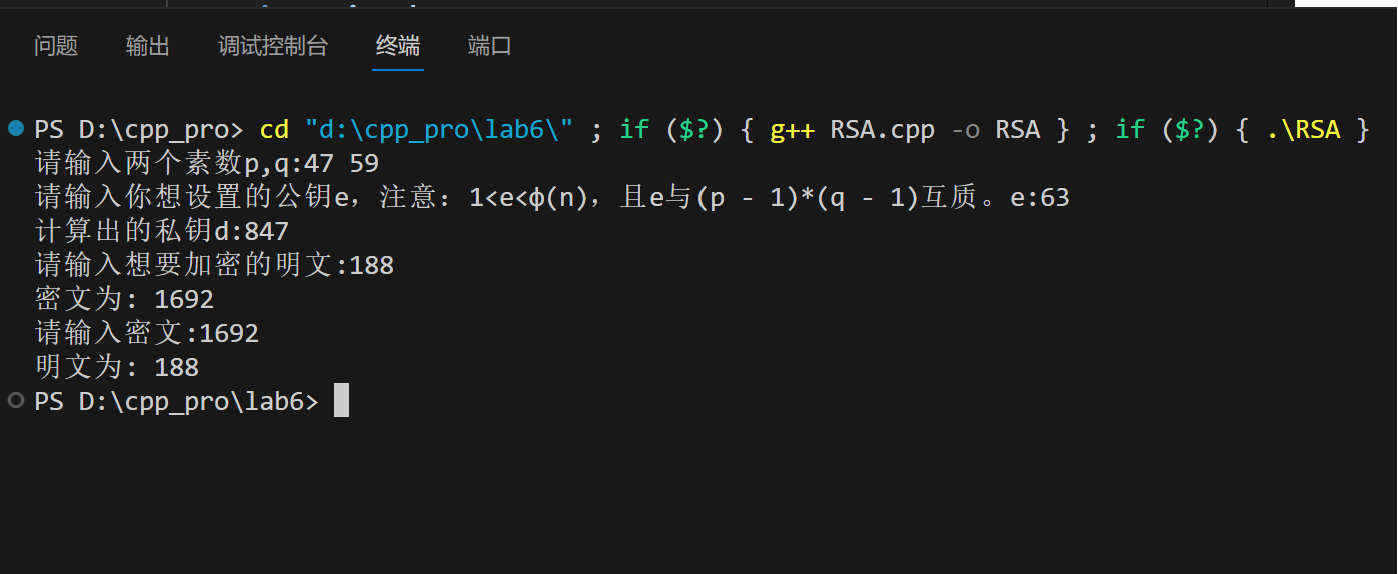
}

这个函数的具体实现流程如下：

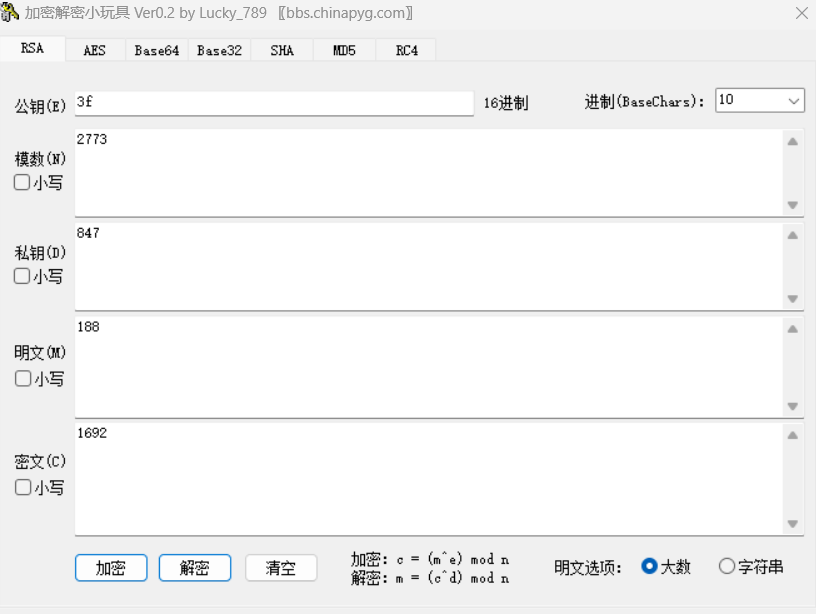
* 首先，使用while循环进行辗转相除法（又称欧几里得算法），求出x和y的最大公约数，保存在变量t中，直到y等于0，此时x就是x和y的最大公约数。
* 然后，如果x等于1，说明x和y互质，因为只有当两个数的最大公约数为1时，它们才是互质的，返回0。
* 最后，如果x不等于1，说明x和y不互质，返回1。

##### 四、实验数据记录（或仿真及软件设计）

输入两个素数p=47，q=59，则n=p\*q，再输入设置的公钥e=63，满足了1<e<φ(n)，且e与(p - 1)\*(q - 1)互质。计算出的私钥d为847，进行加密后密文c=m^e (mod n)为1692，密文解密后的结果m = c^d (mod n)为188.



利用给出的小工具去判断加密解密流程是否正确，注意这里公钥63的格式应该输入成十六进制的3f，右侧的进制选择10，明文选项选择大数。通过使用工具验证，可知RSA加解密流程正确。



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1、简述RSA的公钥生成算法

RSA公钥生成算法的步骤如下：

1. 随机选择两个不相等的质数p和q
2. 计算n = p \* q
3. 计算欧拉函数φ(n) = (p-1) \* (q-1)
4. 随机选择一个整数e，满足1 < e < φ(n)，且e与φ(n)互质（即它们没有除了1以外的公共因子）
5. 计算e关于模φ(n)的乘法逆元d，即d ≡ e⁻¹ (mod φ(n))
6. 公钥为(n, e)，私钥为(n, d)

其中，公钥由(n, e)组成，私钥由(n, d)组成。在实际使用中，p和q应该足够大并且随机选择，以确保 RSA 的安全性

2、“无法证明RSA算法是安全的”，你认为这句话对吗？

在密码学中，任何算法都无法被证明是绝对安全的，因为即使看似很安全的算法仍然可能被攻击者破解或者利用漏洞攻击。RSA算法也不例外。RSA算法的安全性与所选用的素数的长度有直接关系。一般来说，选用更长的素数会使攻击者通过暴力破解的难度增大，从而提高了RSA算法的安全性。总之，虽然没有一种加密算法可以被完全证明是绝对安全的，但是，如果正确使用并采取适当的安全措施，RSA算法可以提供足够的保护来满足大部分实际安全需求。

## 实验七

### Hash算法-MD5密码

##### 一、实验目的

1. 学习MD5算法生成文件摘要的过程；
2. 设计实现MD5文件校验工具，完成MD5算法对文件加密。
3. 了解MD5算法特点和应用场景

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

##### 三、实验基本原理及步骤（或方案设计及理论计算）

**MD5算法生成文件摘要的过程**: MD5（Message-Digest Algorithm 5）是一种常用的哈希算法，可以将任意长度的消息摘要成一个固定长度的数字指纹。生成文件摘要的过程如下：

1. 初始化：MD5算法会先初始化一些常量和变量，包括4个32位寄存器A、B、C、D，以及一个64字节的缓冲区。
2. 填充消息：将要计算摘要的消息进行填充，使其长度满足512位（64字节）的倍数，填充规则为在消息末尾添加一个1，然后补0直到消息长度满足条件。
3. 处理消息块：将填充后的消息按照512位（64字节）分组，每次处理一组，共进行n个循环，其中n等于消息长度除以512的整数部分。
4. 更新寄存器：对每个消息块进行操作，从而更新寄存器A、B、C、D的值。
5. 生成摘要：当所有消息块都被处理后，将A、B、C、D四个寄存器中的内容按照大端序（高位字节在前，低位字节在后）连接起来，即可得到128位的摘要。
6. 输出摘要：最后返回128位的MD5摘要作为输出结果。

主函数如下

int main(){

printf("MD5加密程序：\n");

while(1)

{

printf("请输入加密文件名称（输入exit则退出程序）\n");

gets(filename); //用get函数,避免scanf以空格分割数据,

if (filename[0]==34) filename[strlen(filename)-1]=0,strcpy(filename,filename+1); //支持文件拖曳,但会多出双引号,这里是处理多余的双引号

if (!strcmp(filename,"exit")) exit(0); //输入exit退出

if (!(fp=fopen(filename,"rb")))

{

printf("Can not open this file!\n"); //以二进制打开文件

continue;

}

fseek(fp, 0, SEEK\_END); //文件指针转到文件末尾

if((len=ftell(fp))==-1)

{

printf("Sorry! Can not calculate files which larger than 2 GB!\n"); //ftell函数返回long,最大为2GB,超出返回-1

fclose(fp);

continue;

}

rewind(fp); //文件指针复位到文件头

A=0x67452301,B=0xefcdab89,C=0x98badcfe,D=0x10325476; //初始化链接变量

flen[1]=len/0x20000000; //flen单位是bit

flen[0]=(len%0x20000000)\*8;

memset(x,0,64); //初始化x数组为0

fread(&x,4,16,fp); //以4字节为一组,读取16组数据

for(i=0; i<len/64; i++) //循环运算直至文件结束

{

md5();

memset(x,0,64);

fread(&x,4,16,fp);

}

((char\*)x)[len%64]=128; //文件结束补1,补0操作,128二进制即10000000

if(len%64>55) md5(),memset(x,0,64);

memcpy(x+14,flen,8); //文件末尾加入原文件的bit长度

md5();

fclose(fp);

printf("请输入输出文件名称:\n");

gets(filename);

fp=fopen(filename,"w+");

fprintf(fp,"%08x%08x%08x%08x\n",PP(A),PP(B),PP(C),PP(D)); //高低位逆反输出

fclose(fp);

printf("加密完成，请查看！\n");

printf("\n");

}

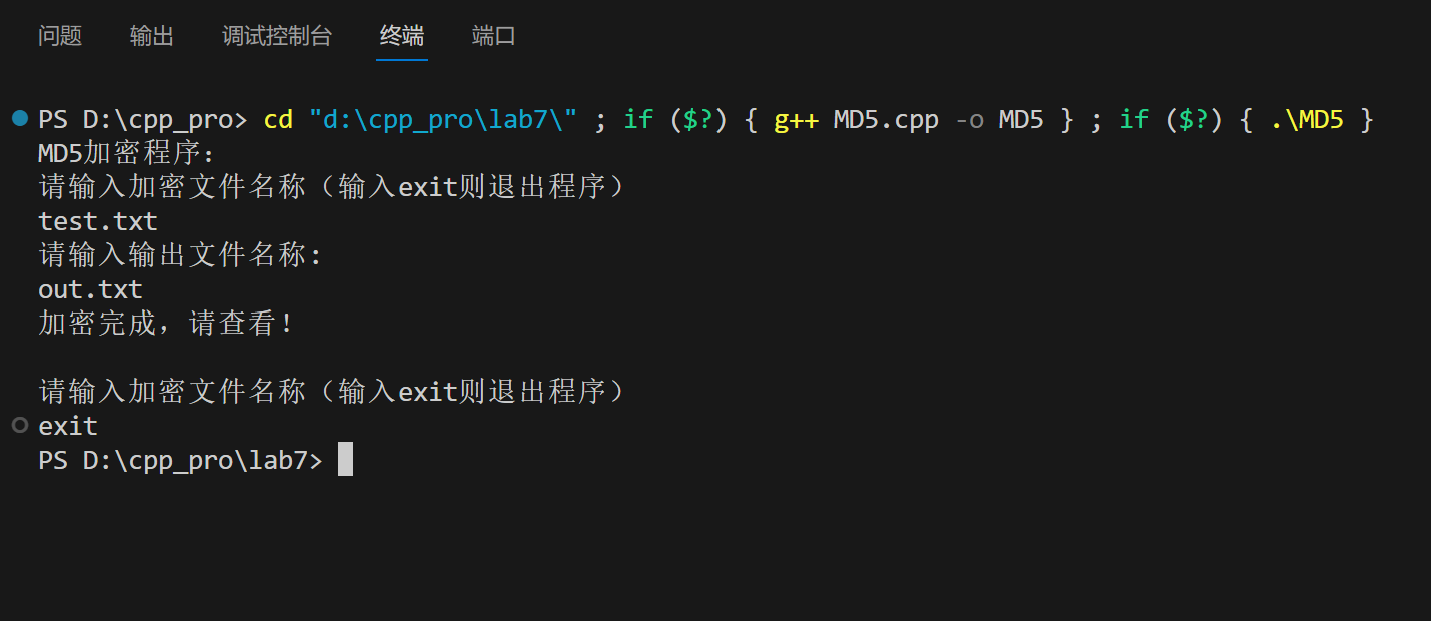
return 0;

}

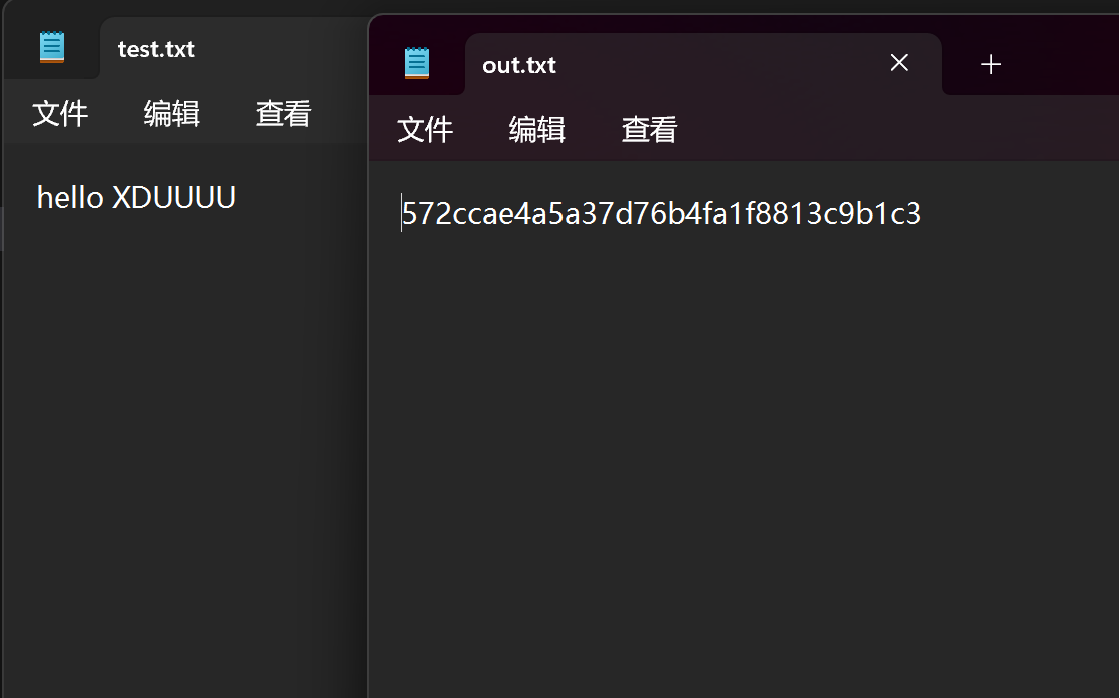
这段代码是一个MD5加密程序的主函数，使用printf()输出一条提示信息，让用户输入要加密的文件名称。使用gets()函数获取用户输入的文件名称，并存储到filename变量中。使用fopen()函数以二进制模式打开用户输入的文件。然后使用fseek()函数将文件指针移动到文件末尾，使用ftell()函数获取文件长度，单位为字节。、如果文件长度超过了2GB，则无法计算文件摘要，程序会输出错误信息并继续等待用户输入。下面需要初始化链接变量：使用四个32位寄存器A、B、C、D分别初始化为0x67452301、0xefcdab89、0x98badcfe、0x10325476，作为链接变量。再分块处理文件数据：将文件按照64字节（512位）分组，每次处理一组，共进行len/64次循环。在循环中，调用md5()函数来更新链接变量的值。文件结束补位：在文件末尾添加一个1和若干个0，使得文件长度为512的倍数。如果文件末尾不足64字节，则补位操作需要跨越两个分组进行。最后将四个32位链接变量的值高低位逆序输出，并将输出结果写入到用户指定的输出文件中。

##### 四、实验数据记录（或仿真及软件设计）

下面是MD5加密程序



文件如图所示



##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1.MD5生成摘要的长度是多少位？

MD5生成的摘要长度是128位。

## 实验八

### 密码应用-文件安全传输

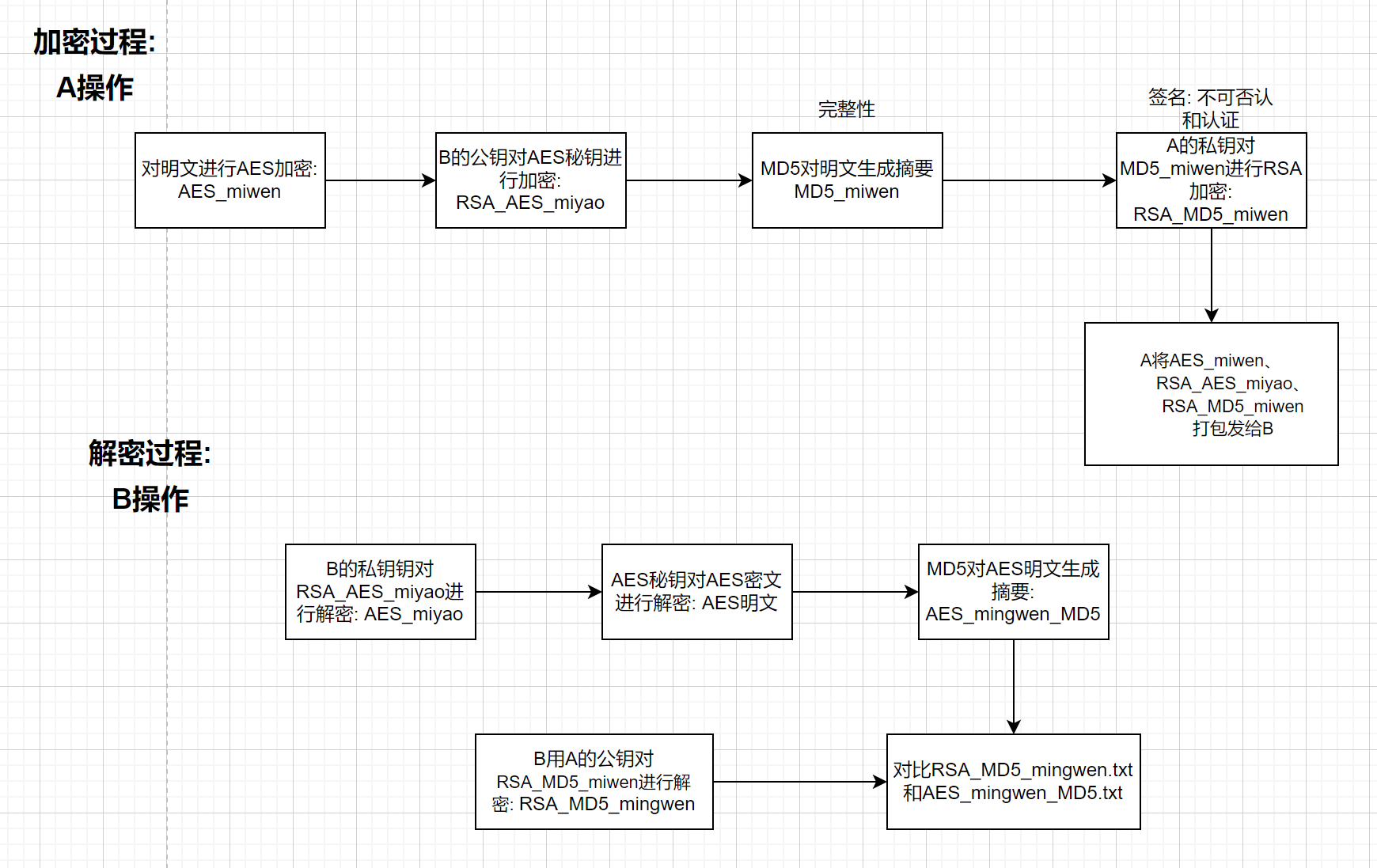
##### 一、实验目的

1. 掌握安全通信中常用的加密算法
2. 掌握数字签名过程
3. 掌握文件安全传输的基本步骤

##### 二、实验所用仪器（或实验环境）

​ VSCode+MinGW

三、实验基本原理及步骤（或方案设计及理论计算）



参数选取:

RSA A：选择p=61，q=53，e=17 得到公钥（e，n）=（17，3233） 私钥（d，n）=（2573，3233）

RSA B:选择p=47，q=59，e=63 得到公钥（e，n）=（63，2773） 私钥（d，n）=（847，2773）

明文: levit

AES秘钥: 12345678KJNMLSDG

**加密过程(A):**

AES密文: 640246746 37579803911 321220508 1223758774 116

用B的公钥对AES秘钥进行RSA加密 RSA\_AES\_miyao : ۯ Ϳ ʭ ؆ ĳ β ճ ਝ ѽ ॥ ¤ ઁ উ ׮ ३ ષ

对明文计算MD5摘要,MD5\_miwen: ea152a3ff6469c391ee8c79a01191d80

A的私钥对MD5\_miwen加密, RSA\_MD5\_miwen: ા ௚ ం ௟ ા ۰ χ χ స ص స ਘ ō ۰ ਘ ௚ ұ ō ਘ ા ਐ ௚ ௚ ਘ ௚ Ɵ ұ ਐ ষ ࢍ

最后把加密生成的文件AES\_miwen.txt、RSA\_AES\_miyao.txt、RSA\_MD5\_miwen.txt 打包发给终端B

**解密过程(B)**

终端B的私钥对RSA\_AES\_miyao.txt 进行RSA解密, 得到:AES\_Key

对AES\_miwen解密，得到明文AES\_mingwen.txt

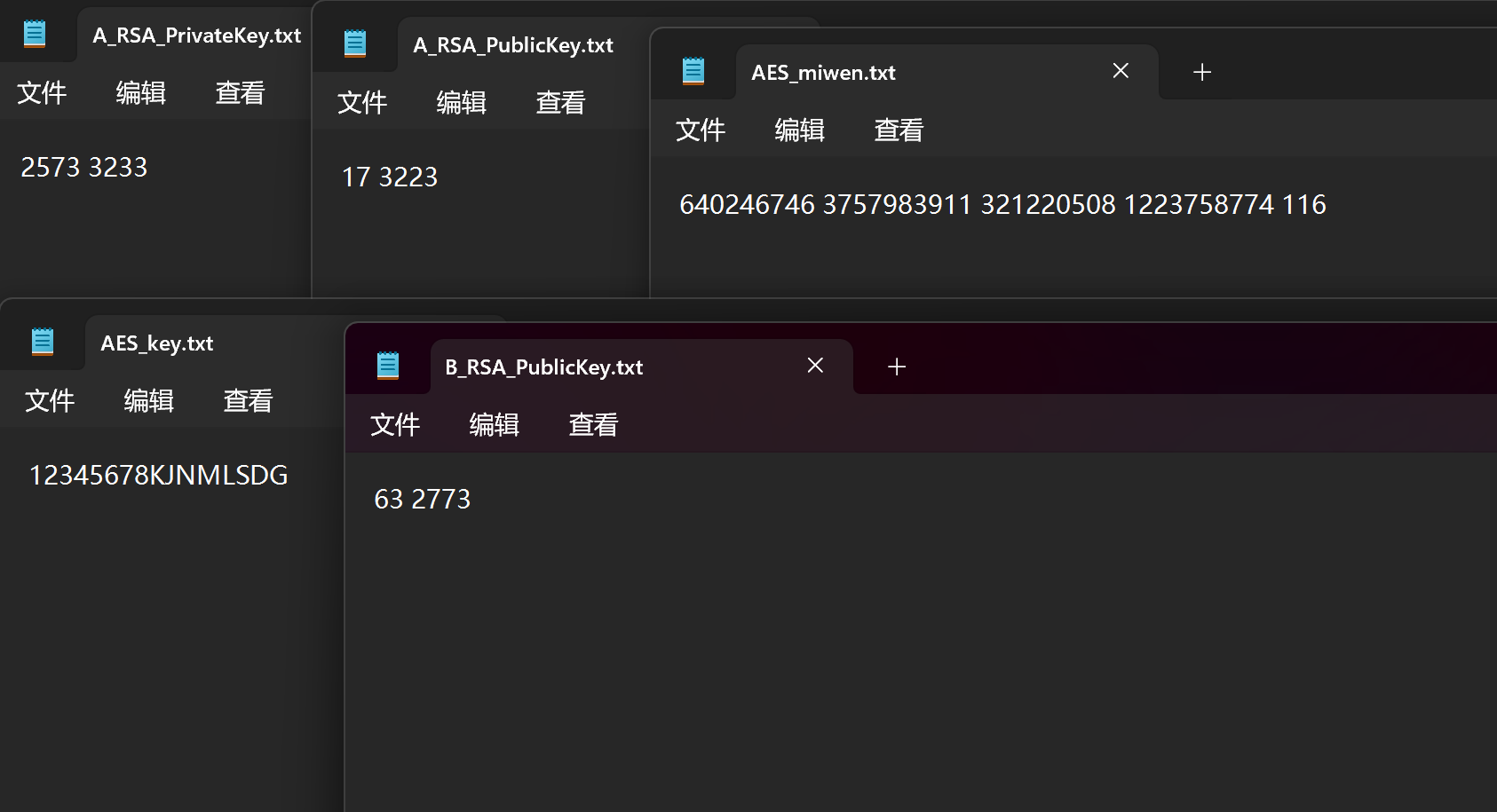
用MD5算法对所得明文AES\_mingwen.txt进行加密生成解密后的明文摘要。得到： AES\_mingwen\_MD5.txt

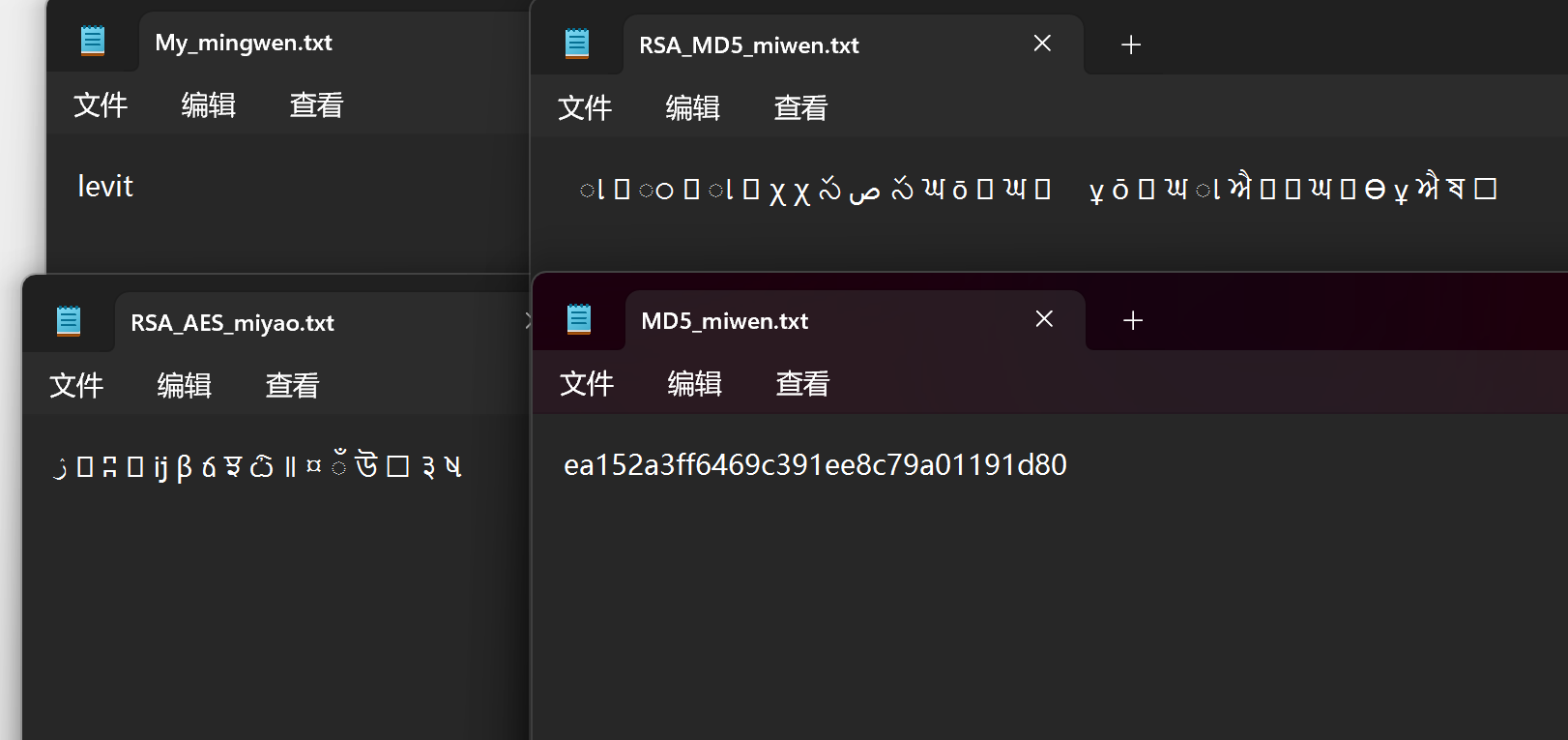
RSA算法用终端A 的公钥对RSA\_MD5\_miwen.txt 进行解密，得到: RSA\_MD5\_mingwen.txt

比较RSA\_MD5\_mingwen.txt和AES\_mingwen\_MD5.txt有什么不同

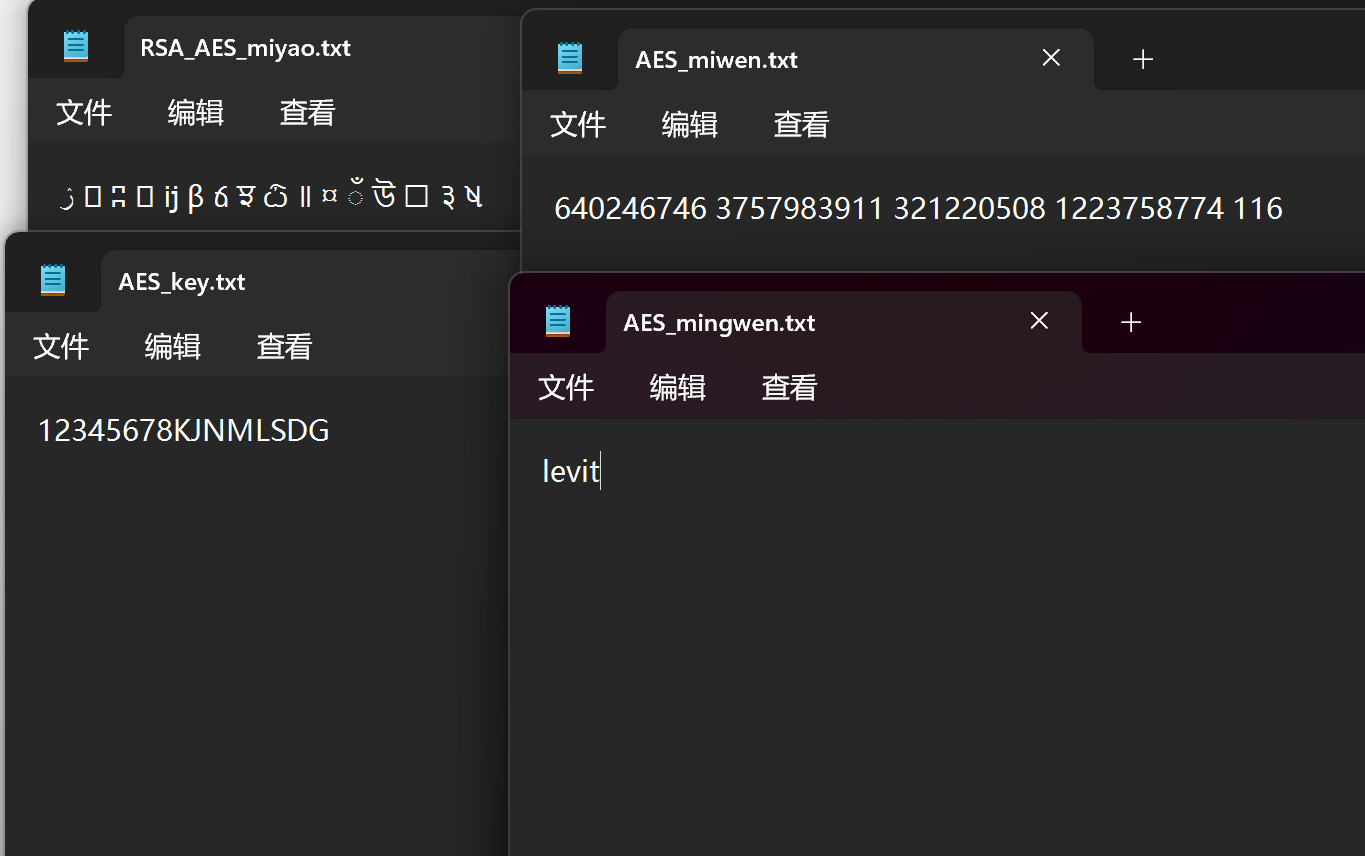
四、实验数据记录（或仿真及软件设计）

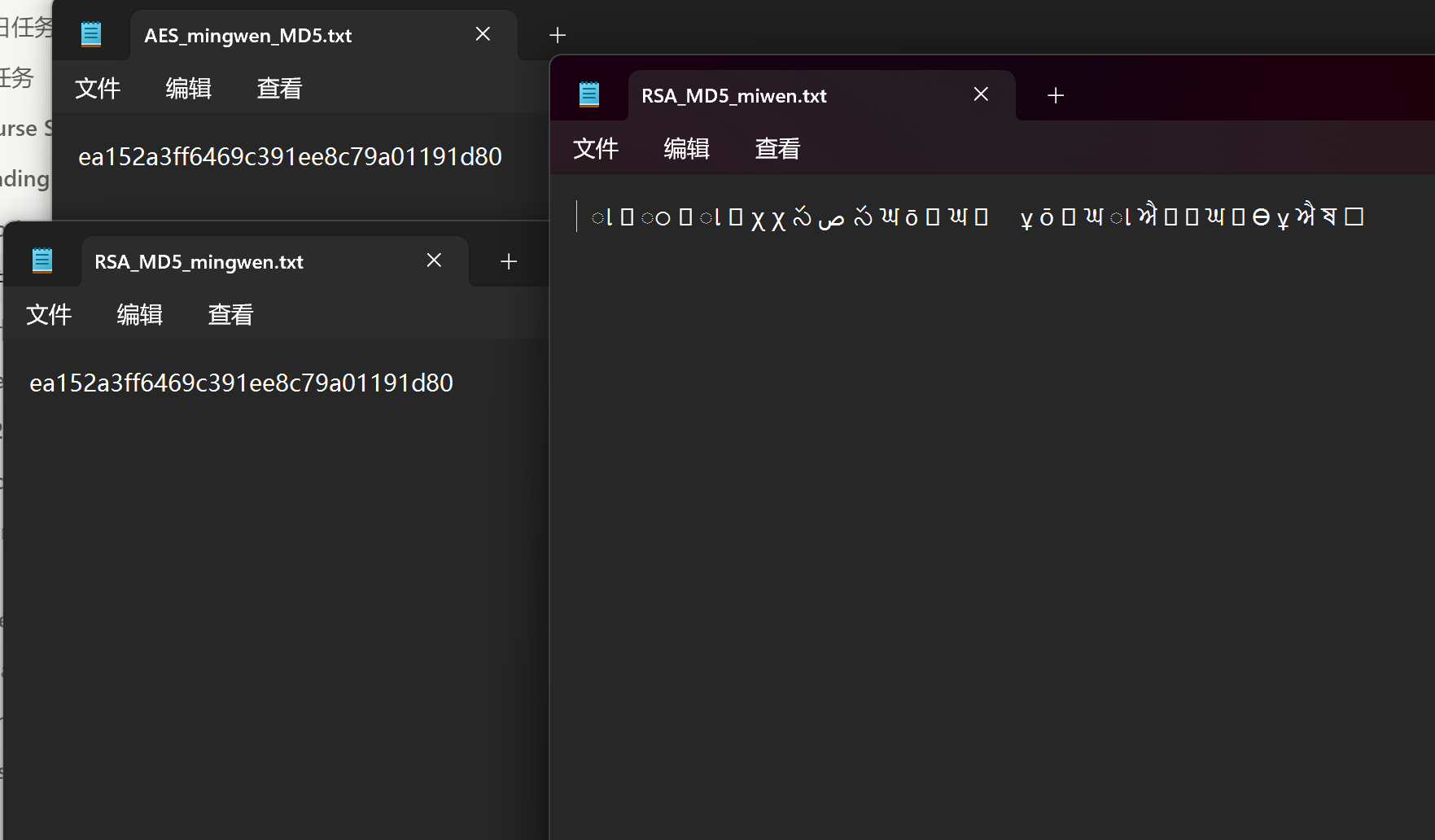
**A:**





**B:**

****

****

##### 五、实验结果分析及回答问题（或测试环境及测试结果）

1. 你所设计的实验步骤，哪些步骤实现了信息传输的保密性、完整性和不可否认性。

AES对明文进行加密以及用RSA加密AES秘钥实现秘钥交换实现了信息传输的保密性

MD5对明文生成摘要实现了信息传输的完整性

用A的RSA私钥对MD5\_miwen进行签名实现了信息传输的不可否认性

2. 在实验设计中为什么不直接应用非对称加密算法直接对明文进行加密传输，而是使用对称加密算法完成对明文的加密工作。

主要原因在于**速度和效率**:

AES：AES是一种对称加密算法，使用相同的密钥进行加密和解密。对称加密算法通常比非对称加密算法要快得多。AES在硬件和软件中的实现都非常高效，能够快速处理大量数据，因此非常适合需要高速度和高吞吐量的应用场景。AES支持128位、192位和256位密钥长度。即使是最低的128位密钥长度也提供了非常高的安全性。

RSA：RSA是一种非对称加密算法，使用公钥进行加密，私钥进行解密。非对称加密算法通常计算复杂度较高，处理速度较慢，尤其是在加密和解密较长的数据时。RSA的安全性依赖于密钥长度。为了达到与AES相当的安全性，RSA需要非常长的密钥（通常是2048位或更长），这进一步增加了计算复杂性和处理时间。

采用RSA+AES的混合加密兼顾了加密效率与安全性: AES用于实际数据的加解密，提供了快速的加密效率，适合大数据量的加密需求。RSA用于安全传输AES密钥，确保密钥在传输过程中不被截获和破解，提供了高安全性的密钥交换机制。

# 二、实验收获

通过此次实验，加深了我对古典加密算法、对称加密算法和非对称加密算法的理解，尤其是自己动手编程之后，对其中的有了更深入的理解。而文件传输的实验，则将理论与实践结合，通过一系列加解密及签名等操作，完成了文件的安全传输。实验中也遇到了一系列问题，通过网上查阅资料、向老师同学请教，最终解决了相关问题。总的来说，收获很多。