

**学 生 实 验 报 告**

**（理工类）**



课程名称： 密码学实验 专业班级：22网络空间安全

学生学号： 2218000121 学生姓名： 陈宣儒

所属院部：网络安全学院 指导教师： 黄丹丹

**2023 ——2024 学年 第 1 学期**

金陵科技学院教务处制

**实验报告书写要求**

实验报告原则上要求学生手写，要求书写工整。若因课程特点需打印的，要遵照以下字体、字号、间距等的具体要求。纸张一律采用A4的纸张。

格式要求：全文大标题为黑体，四号，1.5倍行距。

**实验报告书写说明**

实验报告中一至四项内容为必填项，包括实验目的和要求；实验仪器和设备；实验内容与过程；实验结果与分析；填写注意事项

（1）细致观察，及时、准确、如实记录。

（2）准确说明，层次清晰。

（3）尽量采用专用术语来说明事物。

（4）外文、符号、公式要准确，应使用统一规定的名词和符号。

（5）应独立完成实验报告的书写，严禁抄袭、复印，一经发现，以零分论处。

**实验报告批改说明**

实验报告的批改要及时、认真、仔细，一律用红色笔批改。实验报告的批改成绩采用百分制，具体评分标准由各院部自行制定。

**实验成绩评价及分析**

实验成绩表中实验项目名称、学时与实验大纲、实验计划表中信息保持一致；实验课结束后学生对该门实验课就自身能力产出展开评价；任课教师结合该课对应的能力要求、毕业指标给予评语。

**实验报告装订要求**

实验批改完毕后，任课老师将每门课程的每个实验项目的实验报告以自然班为单位、按学号升序排列，装订成册。

实验项目名称：Caesar密码 实验学时： 2

同组学生姓名： 无 实验地点： 4号科技楼1204

实验日期： 2023.9.6 实验成绩：

批改教师： 黄丹丹 批改时间： 2022.9.8

实验1 Caesar密码

一、基于OBE模式的实验目的和要求

1、了解古典密码的几种分类。

2、掌握Caesar密码的加解密原理。

3、利用编写的Caesar密码程序破译简单的Caesar挑战密文。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、密码学是一门古老的学科，起源于在古代军事作战中如何隐密地传递信息的问题。从古希腊时期一直到1949年，都属于古典密码的发展时期。古典密码主要分为两种：代换密码和置换密码。代换密码指的是根据代换表或替换表将明文逐字母换成其他的字母来产生密文；置换密码指的是将明文中的字母重新排列来产生密文。

2、在古罗马时期，执政官凯撒(Caesar)在军事作战中使用了一种密码用于与其将军们通信，后人称其为“Caesar密码”。它是一种移位密码，描述相当简单，设

**明文**均为英文字母，

**密钥*k***（也就是偏移量）是0~25的整数，

**密文**，均为英文字母。

**1）加密**

先将明文中每个对应到0~25的整数，得到；

再根据密钥*k*将作偏移，得到

，

记为，仍然是0~25的整数；

最后将对应回英文字母，得到密文。

**2）解密**

类似于加密，只需要把密钥换成是*26-k*，就能将密文解密成明文。

**3）示例**

明文为*attacknow*，密钥为*18*，加密时将每个字母用相隔18个位置后的字母

替换，得到密文为sllsucfgo；

再取密钥为*26-18=8，*用于解密密文sllsucfgo，得到明文*attacknow。*

提示点：

1）可能会用到char\*, int, CString之间的相互转化

2）‘a’~’z’的ASCII码值分别是97~122，‘A’~’Z’的ASCII码值分别是65~90

3、程序代码：

#include **<**stdio**.**h**>  
void** encryption**(){**    **int** key**;**    printf**("请输入密钥：");**    scanf**("%d",&**key**);**    getchar**();**    **char** word**[**100**];**    **char** passwd**[**100**];**    **for(int** i**=**0**;**i**<**100**;**i**++){**        word**[**i**]=**'\0'**;**        passwd**[**i**]=**'\0'**;**    **}**    **int** num**=**0**;**    **char** a**;**    printf**("请输入明文：");**    **while((**a**=**getchar**())!=**'\n'**){**        word**[**num**]=**a**;***//保存明文*        **if(**word**[**num**]<=**'z'**&&**word**[**num**]>=**'a'**){***//凯撒密码只转换字母*            passwd**[**num**]=**'a'**+(**a**+**key**-**'a'**+**26**)%**26**;**        **}else** **if(**word**[**num**]<=**'Z'**&&**word**[**num**]>=**'A'**){**            passwd**[**num**]=**'A'**+(**a**+**key**-**'A'**+**26**)%**26**;**        **}else{**            passwd**[**num**]=**word**[**num**];**        **}**      
        num**++;**    **}**      
    printf**("加密后的文本：%s**\n**",**passwd**);**    printf**("**\n**");  
}  
  
void** decryption**(){**    **char** word**[**100**];**    **char** passwd**[**100**];**    **for(int** i**=**0**;**i**<**30**;**i**++){**        word**[**i**]=**'\0'**;**        passwd**[**i**]=**'\0'**;**    **}**    **int** num**=**0**;**    printf**("请输入密文：");**    **char** a**;**    getchar**();**    **while((**a**=**getchar**())!=**'\n'**){**        passwd**[**num**]=**a**;**        num**++;**    **}**    num**=**0**;**    **for(int** i**=**1**;**i**<**26**;**i**++){**        **while(**passwd**[**num**]!=**'\0'**){**            **if(**passwd**[**num**]<=**'z'**&&**passwd**[**num**]>=**'a'**){**                word**[**num**]=**'a'**+(**passwd**[**num**]+**i**-**'a'**+**26**)%**26**;**            **}else** **if(**passwd**[**num**]<=**'Z'**&&**passwd**[**num**]>=**'A'**){**                word**[**num**]=**'A'**+(**passwd**[**num**]+**i**-**'A'**+**26**)%**26**;**            **}else{**                word**[**num**]=**passwd**[**num**];**            **}**      
            num**++;**        **}**        printf**("第%d次解密文本：%s**\n**",**i**,**word**);**        num**=**0**;**    **}**    printf**("**\n**");  
}  
int** main**()  
{**    **int** x**;**    printf**("凯撒密码**\n**加密请输入数字1**\n**解密请输入数字2**\n**退出请输入数字3**\n**");**    scanf**("%d",&**x**);**    **while(**x**!=**3**){**        **if(**x**==**1**){**            encryption**();***//加密*        **}else{**            decryption**();***//解密*        **}**        printf**("凯撒密码**\n**加密请输入数字1**\n**解密请输入数字2**\n**退出请输入数字3**\n**");**        scanf**("%d",&**x**);**    **}**      
    **return** 0**;** **}** 四、实验结果与分析

Caesar挑战密文为 ESTD TD XJ DPNCPE XPDDLRP，

解密后所得到的明文为 THIS IS MY SECRET MESSAGE， 密钥为 15 。

测试结果截图与分析如下：

图片包含 文本

描述已自动生成

先调用一次验证程序的正确性

图片包含 日历

描述已自动生成

再使用循环输出每一种情况，最终找到第15次的输出为正确内容。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

凯撒密码的理论很简单，就是对字母进行转换，但是实操去写代码时发现了很多问题。首先就是凯撒密码只对字母进行转换，对于数字、空格等字符都是不需要操作的。其次就是凯撒密码时把字母转换成字母，需要mod26来保证在字母的范围里。做实验时注意看文档，以为凯撒密码规则和课本上的一样，加解密密钥是一样的。但是文档里给的示例明确说了，解密密钥是26-key，导致在课上只做出了11这个密钥，如果按照示例里密钥就是15。修改代码后能够成功实现预期结果了。

实验项目名称：扩展Euclid 算法 实验学时： 2

同组学生姓名： 无 实验地点： 4号科技楼1204

实验日期： 2023.9.13 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.9.16

实验2 扩展Euclid 算法

一、基于OBE模式的实验目的和要求

1、掌握扩展欧几里德算法基本原理；

2、熟悉求解两数的最大公因子；

3、掌握求解模空间中的乘法逆元。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、设计实现算法程序，Euclid 算法求a, b的最大公因子gcd (a, b)。

EUCLID (a, b)

1. X  a; Y  b；

2. if Y= 0 then return X= (a, b)；

3. if Y=1 then return Y= (a, b)；

4. R=X mod Y；

5. X=Y；

6. Y=R；

7. goto 2.

测试示例：当 a=3737, b=2701时，求出相应的gcd (a, b) = 37。

2、设计实现算法程序，Extended Euclid算法求b在mod a 下的乘法逆元，若逆元不存在程序需正确响应。

EXTENDED EUCLID (a, b)

1. (1, 0, a); (0, 1, b);

2. if , then return ；no inverse;

3. if , then return； ;

4.  ;

5. ;

6. ;

7.  ;

8. goto 2.

测试示例：

当n = 3337 = 47\*71，*ϕ*(*n*) = 46\*70 **=** 3220，e = 79时，求出*ϕ*(*n*) = 1019。

3、程序代码：

#include<stdio.h>void Euclid(){    int a,b;    printf("请输入a，b：");    scanf("%d %d",&a,&b);    int X,Y,R;    X=a,Y=b;    while(1){        if(Y==0){            printf("gcd( %d , %d ) = %d",a,b,X);            return;        }else if(Y==1){            printf("gcd( %d , %d ) = %d",a,b,Y);            return;        }        R=X%Y;        X=Y;        Y=R;        }}void ExtendedEuclid(){    int a,b;    printf("请输入a，b：");    scanf("%d %d",&a,&b);    int x1,x2,x3,y1,y2,y3;    int t1,t2,t3;    x1=1,x2=0,x3=a,y1=0,y2=1,y3=b;    while(1){        if(y3==0){            printf("%d mod %d 没有逆元",a,b);            return;        }else if(y3==1){            printf("%d mod %d 的逆元是 %d",b,a,y2);            return;        }        int q=x3/y3;        t1=x1-q\*y1,t2=x2-q\*y2,t3=x3-q\*y3;        x1=y1,x2=y2,x3=y3;        y1=t1,y2=t2,y3=t3;    }}int main(){    printf("请选择程序:\n1.求最大公因数\n2.求逆元\n");    int flag;    scanf("%d",&flag);    if(flag==1){        Euclid();    }else{        ExtendedEuclid();    }    return 0;}

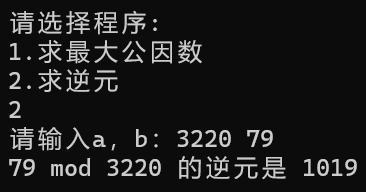
四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

为了让程序的交互性更好，使用if分支让用户选择需要的功能。先是实现了求最大公因数的功能。



再次调用程序，选择求逆元，输入数据，输出正确内容。

五、基于OBE模式的学生自我评价与体会

本次实验主要学习了欧几里得算法，并在了解掌握之后通过C语言编程实现算法。先是辗转相除法求最大公因数。原理很简单，就是几个数换位取余直至为1或者为0。然后是欧几里得拓展算法求逆元。总体思路也是大差不差，是三个数换位取余，只不过会出现一种没有逆元的情况。通过本次实验我对欧几里得算法有了深刻的理解，并能进行运用。

实验项目名称： SM4密码算法 实验学时： 4

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.9.20 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.9.30

实验3 SM4密码算法

一、基于OBE模式的实验目的和要求

1、掌握SM4密码算法的基本原理

2、学会使用C/C++实现SM4密码算法

二、实验仪器和设备

Visual C/C++

三、实验过程

SM4密码是我国国家密码管理局公布的国内第一个商用密码标准，于2006年2月公布，是中国无线局域网安全标准推荐使用的分组密码算法。

**1、加密算法**

SM4算法的分组长度为128比特，密钥分组长度也为128比特。加密算法与密钥扩展算法都采用32轮迭代的Feistel结构，以字节（8位）和字（32位）为单位进行数据处理。

设输入的明文为四个字（128比特长），输入的轮密钥为，共32个字。输出的密文为四个字（128比特长）。加密算法可描述如下：

（1） 



（2） 

其中，*F*是轮函数，*T*是合成变换，*rki*是第*i*+1轮轮密钥，*R*是反序变换。

在加密算法之后还需要一个反序变换R，目的是使加密与解密流程保持一致，解密只需将加密密钥逆序使用。

合成变换*T*由非线性变换和线性变换*L*复合而成，设输入为字*X*，则先对*X* 进行非线性变换，再进行线性*L*变换。记为



非线性变换是由4个S盒并置构成，该S盒可以通过查表实现，见表1。设输入为（4个字节），输出为（4个字节），则



线性变换*L*是以字为处理单位的线性变换，其输入输出都是32位的字，设*L*的输入为字*B*，输出为字*C*，则



其中，为循环左移。

表1 SM4密码算法的S盒

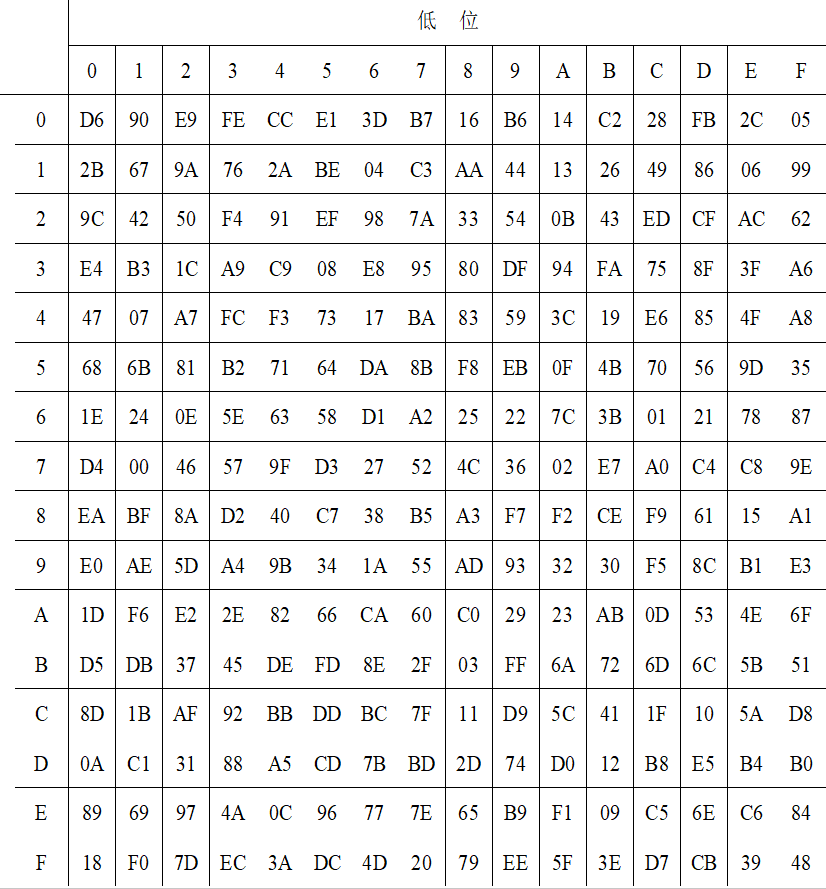




图2 SM4加密算法流程图

**2、解密算法**

SM4解密算法与加密算法相同，只是轮密钥的使用顺序相反，解密轮密钥是加密轮密钥的逆序。

算法的输入为密文和轮密钥，输出为明文。

**3、密钥扩展算法**

设输入的加密密钥为 ，输出轮密钥为，密钥扩展算法可描述如下，其中为中间数据：



变换**与加密算法轮函数中的*T*基本相同，只将其中的线性变化*L*修改为以下的**：



系统参数：（4个字）

*FK* =（*FK0, FK1, FK2, FK3*）=（a3b1bac6, 56aa3350, 677d9197, b27022dc）

固定参数：（32个字）

，

。

**4、编写设计SM4密码算法**

（1）输入128比特明文和密钥，利用SM4密码对明文加密并输出密文。

（2）输入SM4密码加密的128比特密文和密钥，对密文进行解密。

（3）要求有对应的程序调试记录和验证记录。

**提示点：****1、**首先需要定义SM4密码算法中的基本函数，循环左移、非线性变换（S盒）、加密中的线性变换*L*、密钥扩展算法中的线性变换**。部分函数参考代码如下：

（1）Rotl(x, y)定义为将32位的x循环左移y位

**#define Rotl(x, y) ((x << y) | (x >> (32 - y)))**

（2）ByteSub(A)为非线性变换，A是1个字，取A的4个字节分别进行S盒变换

**#define ByteSub(A) (Sbox[(A) >> 24 & 0xFF] << 24 | Sbox[(A) >> 16 & 0xFF] << 16 | Sbox[(A) >> 8 & 0xFF] << 8 | Sbox[(A) & 0xFF])**

（3）加密中的线性变换*L*

**#define L1(B) ((B) ^ Rotl(B, 2) ^ Rotl(B, 10) ^ Rotl(B, 18) ^ Rotl(B, 24))**

（4）密钥扩展算法中的线性变换**

**#define L2(B) ((B) ^ Rotl(B, 13) ^ Rotl(B, 23))**

**2、**所使用参数的初始定义：

（1）定义S盒

const unsigned char Sbox[256] = {

0xd6,0x90,0xe9,0xfe,0xcc,0xe1,0x3d,0xb7,0x16,0xb6,0x14,0xc2,0x28,0xfb,0x2c,0x05,

0x2b,0x67,0x9a,0x76,0x2a,0xbe,0x04,0xc3,0xaa,0x44,0x13,0x26,0x49,0x86,0x06,0x99,

0x9c,0x42,0x50,0xf4,0x91,0xef,0x98,0x7a,0x33,0x54,0x0b,0x43,0xed,0xcf,0xac,0x62,

0xe4,0xb3,0x1c,0xa9,0xc9,0x08,0xe8,0x95,0x80,0xdf,0x94,0xfa,0x75,0x8f,0x3f,0xa6,

0x47,0x07,0xa7,0xfc,0xf3,0x73,0x17,0xba,0x83,0x59,0x3c,0x19,0xe6,0x85,0x4f,0xa8,

0x68,0x6b,0x81,0xb2,0x71,0x64,0xda,0x8b,0xf8,0xeb,0x0f,0x4b,0x70,0x56,0x9d,0x35,

0x1e,0x24,0x0e,0x5e,0x63,0x58,0xd1,0xa2,0x25,0x22,0x7c,0x3b,0x01,0x21,0x78,0x87,

0xd4,0x00,0x46,0x57,0x9f,0xd3,0x27,0x52,0x4c,0x36,0x02,0xe7,0xa0,0xc4,0xc8,0x9e,

0xea,0xbf,0x8a,0xd2,0x40,0xc7,0x38,0xb5,0xa3,0xf7,0xf2,0xce,0xf9,0x61,0x15,0xa1,

0xe0,0xae,0x5d,0xa4,0x9b,0x34,0x1a,0x55,0xad,0x93,0x32,0x30,0xf5,0x8c,0xb1,0xe3,

0x1d,0xf6,0xe2,0x2e,0x82,0x66,0xca,0x60,0xc0,0x29,0x23,0xab,0x0d,0x53,0x4e,0x6f,

0xd5,0xdb,0x37,0x45,0xde,0xfd,0x8e,0x2f,0x03,0xff,0x6a,0x72,0x6d,0x6c,0x5b,0x51,

0x8d,0x1b,0xaf,0x92,0xbb,0xdd,0xbc,0x7f,0x11,0xd9,0x5c,0x41,0x1f,0x10,0x5a,0xd8,

0x0a,0xc1,0x31,0x88,0xa5,0xcd,0x7b,0xbd,0x2d,0x74,0xd0,0x12,0xb8,0xe5,0xb4,0xb0,

0x89,0x69,0x97,0x4a,0x0c,0x96,0x77,0x7e,0x65,0xb9,0xf1,0x09,0xc5,0x6e,0xc6,0x84,

0x18,0xf0,0x7d,0xec,0x3a,0xdc,0x4d,0x20,0x79,0xee,0x5f,0x3e,0xd7,0xcb,0x39,0x48

};

（2）定义固定参数（32个字）

const unsigned int CK[32] = {

0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,

0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,

0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,

0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,

0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,

0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,

0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,

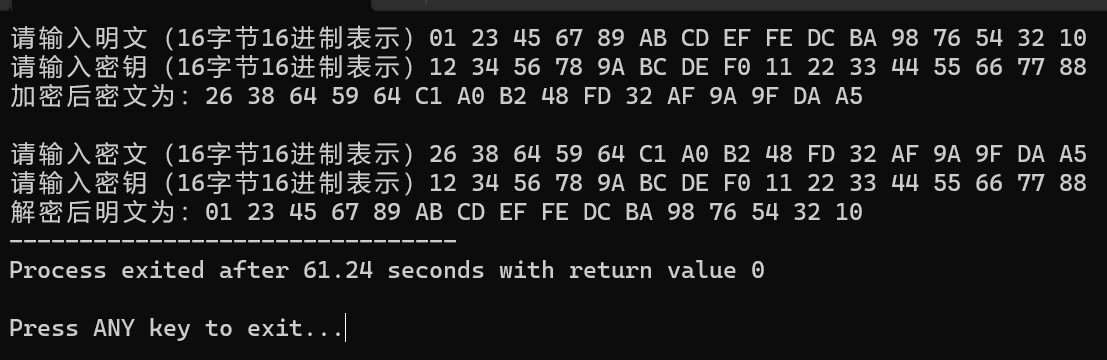
0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279 };

**5、程序代码：**

#include <stdio.h>#include <stdlib.h>#define Rotl(x, y) ((x << y) | (x >> (32 - y)))  
#define ByteSub(A) (Sbox[(A) >> 24 & 0xFF] << 24 | Sbox[(A) >> 16 & 0xFF] << 16 | Sbox[(A) >> 8 & 0xFF] << 8 | Sbox[(A) & 0xFF])  
#define L1(B) ((B) ^ Rotl(B, 2) ^ Rotl(B, 10) ^ Rotl(B, 18) ^ Rotl(B, 24))  
#define L2(B) ((B) ^ Rotl(B, 13) ^ Rotl(B, 23))  
  
const unsigned char Sbox[256] = {    0xd6,0x90,0xe9,0xfe,0xcc,0xe1,0x3d,0xb7,0x16,0xb6,0x14,0xc2,0x28,0xfb,0x2c,0x05,    0x2b,0x67,0x9a,0x76,0x2a,0xbe,0x04,0xc3,0xaa,0x44,0x13,0x26,0x49,0x86,0x06,0x99,    0x9c,0x42,0x50,0xf4,0x91,0xef,0x98,0x7a,0x33,0x54,0x0b,0x43,0xed,0xcf,0xac,0x62,    0xe4,0xb3,0x1c,0xa9,0xc9,0x08,0xe8,0x95,0x80,0xdf,0x94,0xfa,0x75,0x8f,0x3f,0xa6,    0x47,0x07,0xa7,0xfc,0xf3,0x73,0x17,0xba,0x83,0x59,0x3c,0x19,0xe6,0x85,0x4f,0xa8,    0x68,0x6b,0x81,0xb2,0x71,0x64,0xda,0x8b,0xf8,0xeb,0x0f,0x4b,0x70,0x56,0x9d,0x35,    0x1e,0x24,0x0e,0x5e,0x63,0x58,0xd1,0xa2,0x25,0x22,0x7c,0x3b,0x01,0x21,0x78,0x87,    0xd4,0x00,0x46,0x57,0x9f,0xd3,0x27,0x52,0x4c,0x36,0x02,0xe7,0xa0,0xc4,0xc8,0x9e,    0xea,0xbf,0x8a,0xd2,0x40,0xc7,0x38,0xb5,0xa3,0xf7,0xf2,0xce,0xf9,0x61,0x15,0xa1,    0xe0,0xae,0x5d,0xa4,0x9b,0x34,0x1a,0x55,0xad,0x93,0x32,0x30,0xf5,0x8c,0xb1,0xe3,    0x1d,0xf6,0xe2,0x2e,0x82,0x66,0xca,0x60,0xc0,0x29,0x23,0xab,0x0d,0x53,0x4e,0x6f,    0xd5,0xdb,0x37,0x45,0xde,0xfd,0x8e,0x2f,0x03,0xff,0x6a,0x72,0x6d,0x6c,0x5b,0x51,    0x8d,0x1b,0xaf,0x92,0xbb,0xdd,0xbc,0x7f,0x11,0xd9,0x5c,0x41,0x1f,0x10,0x5a,0xd8,    0x0a,0xc1,0x31,0x88,0xa5,0xcd,0x7b,0xbd,0x2d,0x74,0xd0,0x12,0xb8,0xe5,0xb4,0xb0,    0x89,0x69,0x97,0x4a,0x0c,0x96,0x77,0x7e,0x65,0xb9,0xf1,0x09,0xc5,0x6e,0xc6,0x84,    0x18,0xf0,0x7d,0xec,0x3a,0xdc,0x4d,0x20,0x79,0xee,0x5f,0x3e,0xd7,0xcb,0x39,0x48  
  
};const unsigned int CK[32] = {    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279  
};void KeyExpansion(unsigned int\* rk, unsigned char\* key){    unsigned int K[36];    unsigned int MK[4];    unsigned int tmp;    int i;      
    for (i = 0; i < 4; ++i) {        MK[i] = (key[4 \* i] << 24) | (key[4 \* i + 1] << 16) | (key[4 \* i + 2] << 8) | key[4 \* i + 3];    }      
    K[0] = MK[0] ^ 0xA3B1BAC6;    K[1] = MK[1] ^ 0x56AA3350;    K[2] = MK[2] ^ 0x677D9197;    K[3] = MK[3] ^ 0xB27022DC;      
    for (i = 0; i < 32; ++i) {        tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];        tmp = ByteSub(tmp);        K[i + 4] = K[i] ^ L2(tmp);        rk[i] = K[i + 4];    }}unsigned int F(unsigned int X0){    unsigned int t, y;      
    t = X0 ^ Rotl(X0, 2) ^ Rotl(X0, 10) ^ Rotl(X0, 18) ^ Rotl(X0, 24);    y = ByteSub(t);    y ^= Rotl(t, 13) ^ Rotl(t, 23);      
    return y;}void Encrypt(unsigned char\* input, unsigned char\* output, unsigned int\* rk){    unsigned int X[36];    unsigned int i, t, tmp;      
    for (i = 0; i < 4; ++i) {        X[i] = (input[4 \* i] << 24) | (input[4 \* i + 1] << 16) | (input[4 \* i + 2] << 8) | input[4 \* i + 3];    }      
    for (i = 0; i < 32; ++i) {        t = X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i];        tmp = F(t);        X[i + 4] = X[i] ^ tmp;    }      
    for (i = 0; i < 4; ++i) {        output[4 \* i] = (X[35 - i] >> 24) & 0xFF;        output[4 \* i + 1] = (X[35 - i] >> 16) & 0xFF;        output[4 \* i + 2] = (X[35 - i] >> 8) & 0xFF;        output[4 \* i + 3] = X[35 - i] & 0xFF;    }}void Decrypt(unsigned char\* input, unsigned char\* output, unsigned int\* rk){    unsigned int X[36];    unsigned int i, t, tmp;      
    for (i = 0; i < 4; ++i) {        X[i] = (input[4 \* i] << 24) | (input[4 \* i + 1] << 16) | (input[4 \* i + 2] << 8) | input[4 \* i + 3];    }      
    for (i = 0; i < 32; ++i) {        t = X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[31 - i];        tmp = F(t);        X[i + 4] = X[i] ^ tmp;    }      
    for (i = 0; i < 4; ++i) {        output[4 \* i] = (X[35 - i] >> 24) & 0xFF;        output[4 \* i + 1] = (X[35 - i] >> 16) & 0xFF;        output[4 \* i + 2] = (X[35 - i] >> 8) & 0xFF;        output[4 \* i + 3] = X[35 - i] & 0xFF;    }}int main(){    unsigned int rk[32];    printf("请输入明文（16字节16进制表示）");    unsigned char plaintext[16];    for(int i=0;i<16;i++){        scanf("%hhx", &plaintext[i]);    }    unsigned char ciphertext[16];    unsigned char decryptedtext[16];    printf("请输入密钥（16字节16进制表示）");    unsigned char key[16];    for(int i=0;i<16;i++){        scanf("%hhx", &key[i]);    }    KeyExpansion(rk, key);  
    Encrypt(plaintext, ciphertext, rk);  
    printf("加密后密文为：");    for (int i = 0; i < 16; ++i) {        printf("%02X ", ciphertext[i]);    }    printf("\n");    unsigned char ciphertext2[16];    printf("\n请输入密文（16字节16进制表示）");    for (int i = 0; i < 16; ++i) {        scanf("%hhx", &ciphertext2[i]);    }      
    printf("请输入密钥（16字节16进制表示）");    unsigned char key2[16];    for(int i=0;i<16;i++){        scanf("%hhx", &key2[i]);    }      
    unsigned int rk2[32];    KeyExpansion(rk2, key2);    Decrypt(ciphertext, decryptedtext, rk2);      
    printf("解密后明文为：");    for (int i = 0; i < 16; ++i) {        printf("%02X ", decryptedtext[i]);    }      
    return 0;}

四、实验结果与分析

测试结果截图与分析如下：



先输入16字节（128位）明文，再输入16字节密钥，经过sm4算法后加密成16字节的密文，再把加密出来的密文输入进去，输入相同的密钥，经过解密后解出原文。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验，我们尝试用C语言实现SM4算法，SM4是国标商用密码，实现难度比前几次实验要难得多，但是好在老师给了不少提示。在实验过程中，我按照给定的步骤进行了加密算法、解密算法和密钥扩展算法的编写设计，并进行了相应的调试和验证。通过这个实验，我对SM4密码算法有了更深入的理解。我了解到SM4算法是一种具有高安全性和高效率的分组密码算法。我熟悉了SM4算法的加密和解密过程，以及密钥扩展算法的实现方法。

实验项目名称： AES密码算法 实验学时： 8

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.11.20 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.11.29

实验4 AES密码算法

一、基于OBE模式的实验目的和要求

1、了解分组密码的起源与涵义。

2、掌握AES密码的加解密原理。

3、编程设计AES密码算法。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、2000年10月2日美国商务部部长Norman Y. Mineta宣布经过三年来世界著名密码专家之间的竞争,比利时密码专家 Joan Daemen 和 Vincent Riimen 提出的“ Rijndael 数据加密算法”最终获胜。2001年11月26日，NIST正式公布高级加密标准AES，并于2002年5月26日正式生效，成为美国的官方政府标准。AES被设计成三个密钥长度128/192/256比特用于加密长度为128比特的分组，相应的轮数为10/12/14。AES算法是SPN结构的代表。

2、算法细节

**明文**是128bit序列，划分为16个字节，并把字节数据块按a00, a10, a20, a30, a01, a11, a21, a31, a02, a12, a22, a32, a03, a13, a23, a33 顺序映射为状态字节矩阵。

**密钥长度为128比特**的AES加密流程如下：

（1）密钥编排：由密钥扩展算法将128比特的初始密钥扩展为11个128比特的轮密钥K0, K1…K10, 每一个轮密钥同样表示为字节矩阵。

（2）密钥白化（初始轮）：将明文状态矩阵与第一个轮密钥K0 异或加运算。

（3）执行9轮完全相同的轮变换，轮函数分为四步（中间轮）：字节代换、行移位、列混合、轮密钥加。

（4）执行最后一轮轮变换（最后轮）：字节代换、行移位、轮密钥加。

（5）步骤（4）输出的字节矩阵按照顺序排列即为密文。

详见图1：

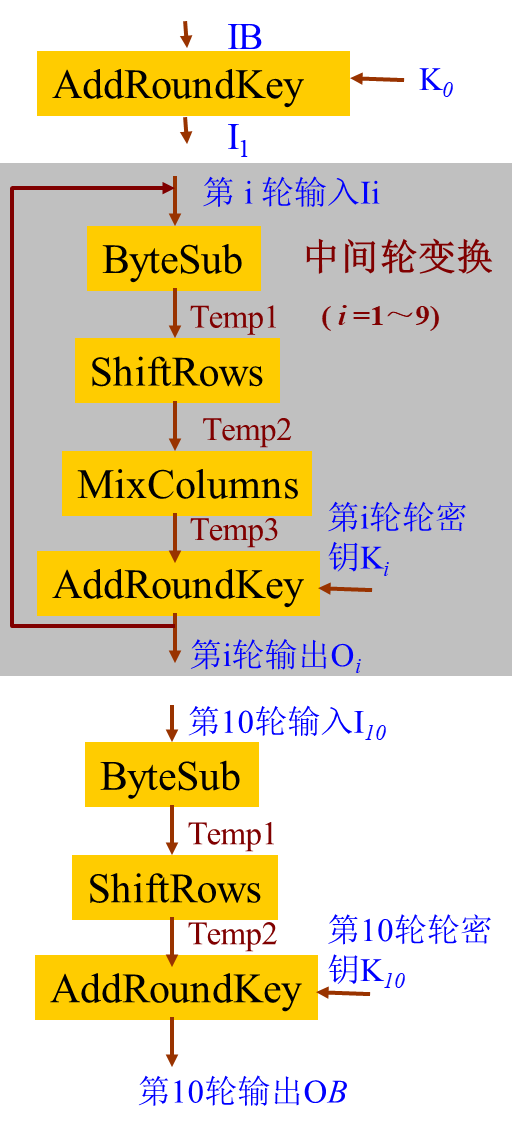


图1 AES加密

解密算法与加密算法的计算网络都相同，只是将各计算部件换为对应的逆部件，解密算法的轮密钥依次为：K10, InvMixColumns(K9), …, InvMixColumns(K1), K0。列混合变换中乘法使用的4字节向量为 ( 03 01 01 02)，逆列混合变换中乘法使用的4字节向量为 ( 0B 0D 09 0E)。

密钥扩展算法将128比特的初始密钥扩展为44个字W0, W1…W43, 每一个字为32比特，算法流程如图2：

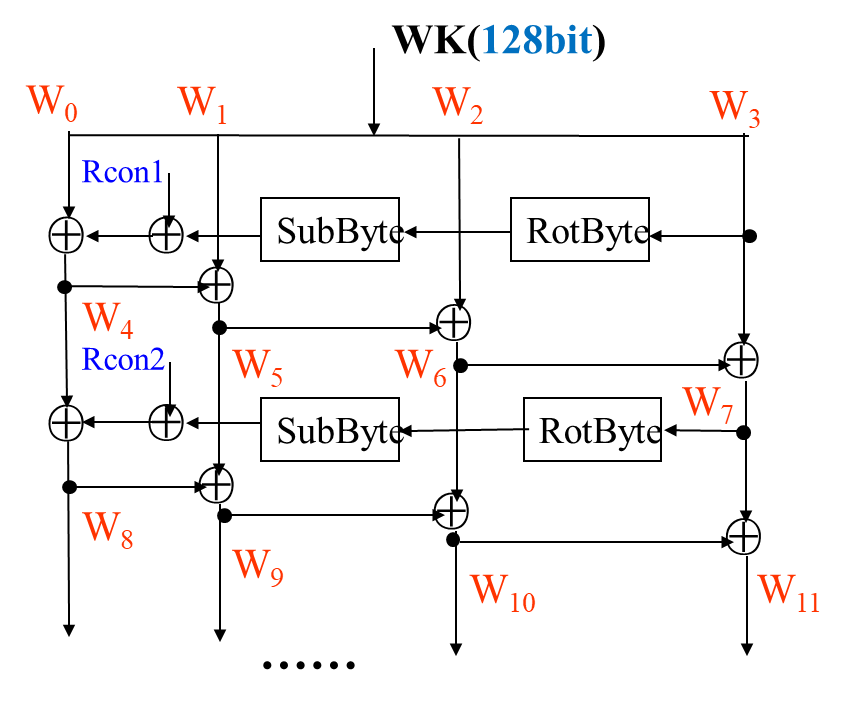


图2 AES密钥编排

3、编写设计AES密码算法

（1）输入128比特明文和密钥，利用AES密码对明文加密并输出密文。

（2）输入AES密码加密的128比特密文和密钥，对密文进行解密。

（3）要求有对应的程序调试记录和验证记录。

提示点：

1. 使用模块化编程，将算法实现的功能写成子函数后供上一层函数调用。
2. 编写函数时多输出中间变量查看结果。
3. 首先需要定义AES密码算法中的基本函数，例如字节代换、行移位、列混合、轮密钥加、密钥扩展算法、有限域上的乘法运算（字节乘法）等。部分函数参考代码如下：

void AddRoundKey( unsigned char \*a , unsigned char \*Key ) { // 轮密钥加

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

a[i] ^= Key[i] ;

}

void SubBytes( unsigned char \*input ) { // S盒字节代换

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

input[i] = S[input[i]] ;

}

void InvSubBytes( unsigned char \*input ) { // 逆S盒字节代换

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

input[i] = IS[input[i]] ;

}

void ShiftRows( unsigned char \*a ) { // 行移位—矩阵按列展开

unsigned char b[16] ;

b[ 0] = a[ 0] ; b[ 4] = a[ 4] ; b[ 8] = a[ 8] ; b[12] = a[12] ;

b[ 1] = a[ 5] ; b[ 5] = a[ 9] ; b[ 9] = a[13] ; b[13] = a[ 1] ;

b[ 2] = a[10] ; b[ 6] = a[14] ; b[10] = a[ 2] ; b[14] = a[ 6] ;

b[ 3] = a[15] ; b[ 7] = a[ 3] ; b[11] = a[ 7] ; b[15] = a[11] ;

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

a[i] = b[i] ;

}

void InvShiftRows( unsigned char \*a ) { // 逆行移位变换

unsigned char b[16] ;

b[ 0] = a[ 0] ; b[ 4] = a[ 4] ; b[ 8] = a[ 8] ; b[12] = a[12] ;

b[ 1] = a[13] ; b[ 5] = a[ 1] ; b[ 9] = a[ 5] ; b[13] = a[ 9] ;

b[ 2] = a[10] ; b[ 6] = a[14] ; b[10] = a[ 2] ; b[14] = a[ 6] ;

b[ 3] = a[ 7] ; b[ 7] = a[11] ; b[11] = a[15] ; b[15] = a[ 3] ;

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

a[i] = b[i] ;

}

void MixColumns( unsigned char \*a ) { // 列混合

unsigned char b[16] ;

b[ 0] = Mul(0x02,a[0]) ^ Mul(0x03,a[1]) ^ a[2] ^ a[3];

b[ 1] = Mul(0x02,a[1]) ^ Mul(0x03,a[2]) ^ a[3] ^ a[0];

b[ 2] = Mul(0x02,a[2]) ^ Mul(0x03,a[3]) ^ a[0] ^ a[1];

b[ 3] = Mul(0x02,a[3]) ^ Mul(0x03,a[0]) ^ a[1] ^ a[2];

b[ 4] = Mul(0x02,a[4]) ^ Mul(0x03,a[5]) ^ a[6] ^ a[7];

b[ 5] = Mul(0x02,a[5]) ^ Mul(0x03,a[6]) ^ a[7] ^ a[4];

b[ 6] = Mul(0x02,a[6]) ^ Mul(0x03,a[7]) ^ a[4] ^ a[5];

b[ 7] = Mul(0x02,a[7]) ^ Mul(0x03,a[4]) ^ a[5] ^ a[6];

b[ 8] = Mul(0x02,a[8]) ^ Mul(0x03,a[9]) ^ a[10] ^ a[11];

b[ 9] = Mul(0x02,a[9]) ^ Mul(0x03,a[10]) ^ a[11] ^ a[8];

b[10] = Mul(0x02,a[10]) ^ Mul(0x03,a[11]) ^ a[8] ^ a[9];

b[11] = Mul(0x02,a[11]) ^ Mul(0x03,a[8]) ^ a[9] ^ a[10];

b[12] = Mul(0x02,a[12]) ^ Mul(0x03,a[13]) ^ a[14] ^ a[15];

b[13] = Mul(0x02,a[13]) ^ Mul(0x03,a[14]) ^ a[15] ^ a[12];

b[14] = Mul(0x02,a[14]) ^ Mul(0x03,a[15]) ^ a[12] ^ a[13];

b[15] = Mul(0x02,a[15]) ^ Mul(0x03,a[12]) ^ a[13] ^ a[14];

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

a[i] = b[i] ;

}

1. 所使用参数的初始定义

unsigned char S[256] = { // S盒--按行展开为一元数组0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76,

0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0,

0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15, 0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75, 0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84, 0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF, 0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8, 0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2, 0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73, 0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB, 0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79, 0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08, 0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A, 0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E, 0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF, 0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16};

unsigned char IS[256] = { // 逆S盒

0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,

0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,

0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,

0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,

0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,

0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,

0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,

0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,

0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,

0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,

0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,

0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,

0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,

0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,

0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,

0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d} ;

4、程序代码：

#include <stdio.h># include <stdlib.h># include <string.h>unsigned char S[256] = {    0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76,    0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0,      
    0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15,      
    0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,      
    0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84,      
    0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,      
    0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8,      
    0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2,      
    0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73,      
    0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,      
    0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,      
    0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08,      
    0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A,      
    0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E,      
    0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF,     
    0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16  
};unsigned char IS[256] = {    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D  
};void SubBytes( unsigned char \*input )   {      
    for( int i = 0 ; i < 16 ; i ++ )        input[i] = S[input[i]] ;}void InvSubBytes(unsigned char \*input) {    for (int i = 0; i < 16; i++) {        input[i] = IS[input[i]];    }}void ShiftRows( unsigned char \*a )     {    unsigned char b[16] ;    b[ 0] = a[ 0] ;    b[ 4] = a[ 4] ;    b[ 8] = a[ 8] ;    b[12] = a[12] ;    b[ 1] = a[ 5] ;    b[ 5] = a[ 9] ;    b[ 9] = a[13] ;    b[13] = a[ 1] ;    b[ 2] = a[10] ;    b[ 6] = a[14] ;    b[10] = a[ 2] ;    b[14] = a[ 6] ;    b[ 3] = a[15] ;    b[ 7] = a[ 3] ;    b[11] = a[ 7] ;    b[15] = a[11] ;    for( int i = 0 ; i < 16 ; i ++ )        a[i] = b[i] ;}void InvShiftRows(unsigned char \*a) {    unsigned char b[16];    b[ 0] = a[ 0];   b[ 4] = a[ 4];   b[ 8] = a[ 8];   b[12] = a[12];    b[ 1] = a[13];   b[ 5] = a[ 1];   b[ 9] = a[ 5];   b[13] = a[ 9];    b[ 2] = a[10];   b[ 6] = a[14];   b[10] = a[ 2];   b[14] = a[ 6];    b[ 3] = a[ 7];   b[ 7] = a[11];   b[11] = a[15];   b[15] = a[ 3];    for (int i = 0; i < 16; i++) {        a[i] = b[i];    }}unsigned char Mul(unsigned char a, unsigned char b) {    unsigned char result = 0;    while (b > 0) {        if (b & 1) {            result ^= a;        }        if (a & 0x80) {            a = (a << 1) ^ 0x1b;        } else {            a <<= 1;        }        b >>= 1;    }    return result;}static void MixColumns(unsigned char \*a){    unsigned char b[16] ;    b[ 0] = Mul(0x02,a[0]) ^ Mul(0x03,a[1]) ^ a[2] ^ a[3];    b[ 1] = Mul(0x02,a[1]) ^ Mul(0x03,a[2]) ^ a[3] ^ a[0];    b[ 2] = Mul(0x02,a[2]) ^ Mul(0x03,a[3]) ^ a[0] ^ a[1];    b[ 3] = Mul(0x02,a[3]) ^ Mul(0x03,a[0]) ^ a[1] ^ a[2];    b[ 4] = Mul(0x02,a[4]) ^ Mul(0x03,a[5]) ^ a[6] ^ a[7];    b[ 5] = Mul(0x02,a[5]) ^ Mul(0x03,a[6]) ^ a[7] ^ a[4];    b[ 6] = Mul(0x02,a[6]) ^ Mul(0x03,a[7]) ^ a[4] ^ a[5];    b[ 7] = Mul(0x02,a[7]) ^ Mul(0x03,a[4]) ^ a[5] ^ a[6];    b[ 8] = Mul(0x02,a[8]) ^ Mul(0x03,a[9]) ^ a[10] ^ a[11];    b[ 9] = Mul(0x02,a[9]) ^ Mul(0x03,a[10]) ^ a[11] ^ a[8];    b[10] = Mul(0x02,a[10]) ^ Mul(0x03,a[11]) ^ a[8] ^ a[9];    b[11] = Mul(0x02,a[11]) ^ Mul(0x03,a[8]) ^ a[9] ^ a[10];    b[12] = Mul(0x02,a[12]) ^ Mul(0x03,a[13]) ^ a[14] ^ a[15];    b[13] = Mul(0x02,a[13]) ^ Mul(0x03,a[14]) ^ a[15] ^ a[12];    b[14] = Mul(0x02,a[14]) ^ Mul(0x03,a[15]) ^ a[12] ^ a[13];    b[15] = Mul(0x02,a[15]) ^ Mul(0x03,a[12]) ^ a[13] ^ a[14];    for( int i = 0 ; i < 16 ; i ++ )        a[i] = b[i] ;      
}static void InvMixColumns(unsigned char \*a){    unsigned char b[16];    b[0] = Mul(0x0E, a[0]) ^ Mul(0x0B, a[1]) ^ Mul(0x0D, a[2]) ^ Mul(0x09, a[3]);    b[1] = Mul(0x09, a[0]) ^ Mul(0x0E, a[1]) ^ Mul(0x0B, a[2]) ^ Mul(0x0D, a[3]);    b[2] = Mul(0x0D, a[0]) ^ Mul(0x09, a[1]) ^ Mul(0x0E, a[2]) ^ Mul(0x0B, a[3]);    b[3] = Mul(0x0B, a[0]) ^ Mul(0x0D, a[1]) ^ Mul(0x09, a[2]) ^ Mul(0x0E, a[3]);    b[4] = Mul(0x0E, a[4]) ^ Mul(0x0B, a[5]) ^ Mul(0x0D, a[6]) ^ Mul(0x09, a[7]);    b[5] = Mul(0x09, a[4]) ^ Mul(0x0E, a[5]) ^ Mul(0x0B, a[6]) ^ Mul(0x0D, a[7]);    b[6] = Mul(0x0D, a[4]) ^ Mul(0x09, a[5]) ^ Mul(0x0E, a[6]) ^ Mul(0x0B, a[7]);    b[7] = Mul(0x0B, a[4]) ^ Mul(0x0D, a[5]) ^ Mul(0x09, a[6]) ^ Mul(0x0E, a[7]);    b[8] = Mul(0x0E, a[8]) ^ Mul(0x0B, a[9]) ^ Mul(0x0D, a[10]) ^ Mul(0x09, a[11]);    b[9] = Mul(0x09, a[8]) ^ Mul(0x0E, a[9]) ^ Mul(0x0B, a[10]) ^ Mul(0x0D, a[11]);    b[10] = Mul(0x0D, a[8]) ^ Mul(0x09, a[9]) ^ Mul(0x0E, a[10]) ^ Mul(0x0B, a[11]);    b[11] = Mul(0x0B, a[8]) ^ Mul(0x0D, a[9]) ^ Mul(0x09, a[10]) ^ Mul(0x0E, a[11]);    b[12] = Mul(0x0E, a[12]) ^ Mul(0x0B, a[13]) ^ Mul(0x0D, a[14]) ^ Mul(0x09, a[15]);    b[13] = Mul(0x09, a[12]) ^ Mul(0x0E, a[13]) ^ Mul(0x0B, a[14]) ^ Mul(0x0D, a[15]);    b[14] = Mul(0x0D, a[12]) ^ Mul(0x09, a[13]) ^ Mul(0x0E, a[14]) ^ Mul(0x0B, a[15]);    b[15] = Mul(0x0B, a[12]) ^ Mul(0x0D, a[13]) ^ Mul(0x09, a[14]) ^ Mul(0x0E, a[15]);    for (int i = 0; i < 16; i++)        a[i] = b[i];}void AddRoundKey( unsigned char \*a , unsigned char \*Key )  {  //  轮密钥加      for( int i = 0 ; i < 16 ; i ++ )        a[i] ^= Key[i] ;}void AesEncrypt(unsigned char \*blk, unsigned char \*key, int Nr){    int round;    AddRoundKey(blk,key);    for(round=1;round<=(Nr-1);round++)      
    {        SubBytes(blk);          
        ShiftRows(blk);          
        MixColumns(blk);      
        AddRoundKey(blk,key);    }    SubBytes(blk);    ShiftRows(blk);    AddRoundKey(blk,key);}void ScheduleKey(unsigned char \*inkey,unsigned char \*outkey, int Nk, int Nr){    unsigned char temp[4],t;    int m,n;      
    for(n=0;n<(4\*Nk);n++){        outkey[n]=inkey[n];    }    n=Nk;      
    unsigned char RC[10];    RC[0]=1;    for(int i=1;i<10;i++){        RC[i]=Mul(0x02,RC[i-1]);    }      
    while(n<(4\*(Nr+1)))   
    {        for(m=0;m<4;m++)   
            temp[m]=outkey[(4\*(n-1))+m];      
        if(n%Nk==0)        {            t=temp[0];temp[0]=temp[1];temp[1]=temp[2];temp[2]=temp[3];temp[3]=t;            for(m=0;m<4;m++){                temp[m]=S[temp[m]];            }            temp[0]^=RC[(n/Nk)-1];        }        for(m=0;m<4;m++){            outkey[(4\*n)+m]=outkey[(4\*(n-Nk))+m]^temp[m];        }        ++n;    }}void AesDecrypt(unsigned char \*blk, unsigned char \*key, int Nr)//解密方法一 替换步骤顺序和加密不一样{    AddRoundKey(blk, key);    for (int round = Nr - 1; round > 0; round--) {        InvShiftRows(blk);        InvSubBytes(blk);        AddRoundKey(blk, key);        InvMixColumns(blk);    }    InvShiftRows(blk);    InvSubBytes(blk);    AddRoundKey(blk, key);}int main(void){    unsigned char pt[16], key[17];    unsigned char skey[15 \* 16];    int i= 0;    printf("请输入16字节明文：");    scanf("%s", pt);//    getchar();    printf("明文为（128比特）：");    for (i = 0; i < 16; i++) {        printf("%02x", pt[i]);    }    printf("\n");    printf("请输入16字节密钥：");    scanf("%s", key);//    getchar();    ScheduleKey(key, skey, 4, 10);    AesEncrypt(pt, skey, 10);    printf("加密后密文为：");    for (i = 0; i < 16; i++) {        printf("%02x", pt[i]);    }    printf("\n");    AesDecrypt(pt, skey, 10);    printf("解密后明文为：");    for (i = 0; i < 16; i++) {        printf("%c", pt[i]);    }      
    printf("\n解密后明文为（128比特）：");    for (i = 0; i < 16; i++) {        printf("%02x", pt[i]);    }  
    return 0;}

四、实验结果与分析

测试结果截图与分析如下：

图形用户界面, 应用程序

描述已自动生成、

先调用Mul函数运行测试用例，来验证Mul函数正确性。结果与PPT所提到的一致，证明Mul函数正确。

文本

描述已自动生成

用字符串接收明文，将明文输出16进制bit串便于后续比较。再用字符串的形式接收密钥。用16进制输出被处理后的密文比特串。再运行解密程序。同时输出解密后明文的字符串形式和16进制比特串形式，比较处理前后明文，发现一致。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验，通过C语言编程实现AES加密功能。通过本次实验我对AES算法有了更深刻的认识。在实验过程中，需要设计模块和函数来实现AES算法的各个步骤，如密钥扩展、字节替代、行移位和列混淆等，这要求对AES算法有充分的了解，通过本次实验我发现了自己对AES掌握上还有所欠缺。总的来说，不仅练习了我的编程能力，还让我加深了对AES算法的理解，此外还让我知道自己的不足之处。

实验项目名称： 快速指数和M-R算法 实验学时： 4

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.11.29 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.12.06

实验5 快速指数和M-R算法

一、基于OBE模式的实验目的和要求

1、掌握快速指数算法，并会运用其实现模幂运算；

2、掌握Miller-Rabin算法，并会运用其检测整数的素性。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、设计实现算法程序，利用快速指数算法求*am*(mod *n*)。

FastExponent (*a*, *m*, *n*)



1）d = 1;

2）for *i* = k－1 downto 0 do

{

d← d2mod *n* ；

if b*i* =1 then d ← (d×*a* )mod *n*

}

return d.

测试示例：当 *a* = 160, m = 77, n = 221时，求出*am*(mod *n*)。

2、设计实现算法程序Miller-Rabin素性检测算法。

WITNESS (*a*，*n*)

1）将*n*-1表示为二进制形式 ；

2）d = 1;

for *i* = k downto 0 do

{

x ← d;

d ← d2mod *n* ；

if d = 1 and (x ≠ 1) and (x ≠ n-1) then return *FALSE*;

if b*i* = 1 then d ← (d×*a* )mod *n*；

}

if d ≠ 1 then return *FALSE*;

return *TRUE*.

测试示例：1）当*n* = 3337时，随机选择参数*a* （5个左右），使得算法输出为FALSE，因此输出*n*不是素数；2）当 *n* = 2333时，随机选择参数*a*（至少10个），使得算法输出都是TRUE，因此说明*n* 有很大概率是素数，否则是强伪素数。例如，当10个参数测试通过时*n*是素数的概率至少为  > 99.999%。

3、程序代码：

#include <stdio.h>#include <stdlib.h>int FastExponent(int a, int m, int n) {    int d = 1;    int binaryM[16] = {0};    int i = 0;  
    while (m > 0) {        binaryM[i] = m % 2;        m = m / 2;        i++;    }      
    for (int j = i - 1; j >= 0; j--) {        d = (d \* d) % n;        if (binaryM[j] == 1) {            d = (d \* a) % n;        }    }    return d;}int WITNESS(int a,int n) {    if (n <= 1)   
        return 0;   
      
    if (n == 2 || n == 3)        return 1;      
    if (n % 2 == 0)        return 0;   
      
    int d = n-1;    int k = 0;      
    while (d % 2 == 0) {        d = d / 2;        k++;    }      
    int x, witness;    for (int i = 0; i < a; i++) {        witness = (rand() % (n - 3)) + 2;   
        x = FastExponent(witness, d, n);        if (x == 1 || x == n - 1) {            continue;    
        }          
        for (int j = 0; j < k - 1; j++) {            x = FastExponent(x, 2, n);              
            if (x == n - 1) {                break;    
            }              
            if (x == 1) {                return 0;   
            }        }          
        if (x != n - 1) {            return 0;        }    }      
    return 1;}int main() {    int n;    printf("输入需要验证的数字：");    scanf("%d",&n);    if (WITNESS (10, n)) {        printf("%d 有很大可能是素数\n", n);    }else{        printf("%d 不是素数\n", n);    }      
    return 0;}

四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

文本

描述已自动生成

首先调用快速指数算法计算数字较大的次幂运算，这种算法的好处是每次结果都会小于n，输出结果如图。

文本

描述已自动生成

文本

描述已自动生成

在调用M-R算法，M-R算法里需要用到快速指数算法。判断次数选择10，这精确率已经达到99.999%，这几乎就是100%，但是数学上需要严谨，所以输出“很大可能”，但是对于密码学来说这个精度已经够用了。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

通过本次实验我学习到了快速指数算法和M-R算法，快速指数算法能在一定程度上解决次幂运算结果太大不方便继续模运算，快速指数算法就是将次幂转化成二进制形式，根据二进制位数按照平方或乘法运算规则进行计算。通过这种算法可以将每次运算结果都控制在小于模数的范围内，可以大大减小计算量。M-R算法是借助快速指数算法进行大素数判断。虽然不是100%，但是对于判断次数为10来说已经很精确了。总之，通过本次实验，我不止学到了新的算法用于处理较大数据，还锻炼了C语言编程能力，让我受益匪浅。

实验项目名称： RSA密码算法 实验学时： 6

同组学生姓名： 无 实验地点：科技楼4-1204

实验日期： 2023.12.6 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.12.15

实验6 RSA密码算法

一、基于OBE模式的实验目的和要求

1、了解公钥密码的起源与涵义。

2、掌握RSA密码的加解密原理。

3、实现RSA密码并输出结果。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、RSA密码由R. Rivest、A. Shamir和L. Adleman于1978年提出，是最著名的公钥密码，能够抵抗到目前为止已知的绝大多数密码攻击，已被ISO推荐为公钥[数据加密标准](http://baike.baidu.com/view/1519129.htm" \t "http://baike.baidu.com/_blank)。RSA密码的安全性基于这样一个事实：将两个大素数p,q相乘十分容易，但对其乘积N=pq作因子分解却极其困难。在本实验中，假设p,q均为len比特素数，具体描述如下：

**系统参数**

大素数p、q：len比特素数（即 且p、q为素数）

乘积N：N = p\*q，约为2\*len比特

N的欧拉函数值：

**公钥**

(N, e)：其中e满足1<e<且gcd (e, ) = 1

**私钥**

d：其中d满足1<d<且，即

**明文：**选取正整数m，满足1<m<N-1

**加密**

密文 

**解密**

明文 

1. 编写程序，实现RSA密码并输出结果。

编程的关键是实现整数运算，包括模加、模乘、模幂等等运算。

3、可能需要的函数

#include<time.h>

#include<iostream>

//复制

void Copy(int &a, int b);

//打印输出

void Print(int a);

//生成随机数

int Rand(int n); //生成1到n之间的随机数

int Rand(int bytes); //生成bit数是8\*Bytes的随机数

//基本运算

int AddMod(int a, int b, int n); // 模加: 计算 a + b mod n

int SubMod(int a, int b, int n); // 模减: 计算 a - b mod n(要求a>b)

int MulMod(int a, int b, int n); // 模乘: 计算 a \* b mod n

int Inv(int a, int n); // 求逆元: 输入a, n, 返回 a^(-1)mod n

int PowMod(int a, int b, int n); // 模幂：计算 a ^ b mod n

//MillerRabin素性检测

bool MillerRabinKnl(int &n); //单次素性判定

bool MillerRabin(int &n, int loop); //素性检测，通过返回1，否则返回0

//生成bit数是8\*Bytes的随机素数

int GenPrime(int bytes);

//生成私钥（求逆元）

int Inverse(int e, int N, int d); //计算

//加密

int Encrypt(int m, int e, int N); //计算c = m^e mod N

//解密

int Decrypt(int c, int d, int N); //计算m = c^d mod N

4、编写主函数，主要步骤有

生成并输出素数p、q及其乘积N=pq；

输出公钥e，计算私钥d；

随机生成并输出明文；

加密并输出密文；

解密并输出解密后的明文。

提示点：

1. 实验中涉及到的数都是非负数，所以减法运算需要先判断大小。
2. 本次实验要求实现p, q约为16bit(2字节)素数的情形，测试可选加密DES算法密钥，例如1334577 99BBCDF F1（8个字节），也可自选。

参考：例如取 p = 11113， q = 33113，N = 367984769，log2N = 29，明文可按照28比特一组进行加密。

5、程序代码：

#include **<**stdio**.**h**>**#include **<**stdlib**.**h**>**#include **<**time**.**h**>  
  
long** **long** **int** gcd**(long** **long** **int** a**,long** **long** **int** b**)** **{**    **while** **(**b **!=** 0**)** **{**        **long** **long** **int** t **=** b**;**        b **=** a **%** b**;**        a **=** t**;**    **}**    **return** a**;  
}  
  
long** **long** **int** extended\_gcd**(long** **long** **int** a**,long** **long** **int** b**,long** **long** **int** **\***x**,long** **long** **int** **\***y**)** **{**    **if** **(**b **==** 0**)** **{**        **\***x **=** 1**;**        **\***y **=** 0**;**        **return** a**;**    **}**    **long** **long** **int** x1**,** y1**;**    **long** **long** **int** gcd **=** extended\_gcd**(**b**,** a **%** b**,** **&**x1**,** **&**y1**);**    **\***x **=** y1**;**    **\***y **=** x1 **-** **(**a **/** b**)** **\*** y1**;**    **return** gcd**;  
}  
  
long** **long** **int** inverse**(long** **long** **int** a**,long** **long** **int** n**)** **{**    **long** **long** **int** x**,** y**;**    **long** **long** **int** gcd **=** extended\_gcd**(**a**,** n**,** **&**x**,** **&**y**);**    **if** **(**gcd **!=** 1**)** **{**        **return** **-**1**;** *// 没有逆元*    **}** **else** **{**        **return** **(**x **%** n **+** n**)** **%** n**;**    **}  
}  
  
long** **long** **int** mod\_pow**(long** **long** **int** a**,long** **long** **int** b**,long** **long** **int** n**)** **{**    **long** **long** **int** r **=** 1**;**    **while** **(**b **>** 0**)** **{**        **if** **(**b **&** 1**)** **{**            r **=** **((long** **long)**r **\*** a**)** **%** n**;**        **}**        a **=** **((long** **long)**a **\*** a**)** **%** n**;**        b **>>=** 1**;**    **}**    **return** r**;  
}  
  
long** **long** **int** is\_prime**(long** **long** **int** n**)** **{**    **if** **(**n **<=** 1**)** **{**        **return** 0**;**    **}**    **if** **(**n **<=** 3**)** **{**        **return** 1**;**    **}**    **if** **(**n **%** 2 **==** 0 **||** n **%** 3 **==** 0**)** **{**        **return** 0**;**    **}**    **for** **(int** i **=** 5**;** i **\*** i **<=** n**;** i **+=** 6**)** **{**        **if** **(**n **%** i **==** 0 **||** n **%** **(**i **+** 2**)** **==** 0**)** **{**            **return** 0**;**        **}**    **}**    **return** 1**;  
}  
  
long** **long** **int** gen\_prime**(int** bits**)** **{**    **long** **long** **int** p**,** count **=** 0**;**    **do** **{**        p **=** rand**()** **%** **(**1 **<<** bits**)** **+** **(**1 **<<** **(**bits **-** 1**));**        count**++;**    **}** **while** **(!**is\_prime**(**p**)** **&&** count **<** 1000**);**    **return** p**;  
}  
  
void** gen\_key\_pair**(int** bits**,long** **long** **int** **\***e**,long** **long** **int** **\***d**,long** **long** **int** **\***N**)** **{**    srand**(**time**(**NULL**));**    printf**("生成p，q...**\n**");**    *// 生成两个大素数p和q*    **long** **long** **int** p **=** gen\_prime**(**bits **/** 2**);**    **long** **long** **int** q **=** gen\_prime**(**bits **/** 2**);**    printf**("p=%lld,q=%lld**\n**",**p**,**q**);**    *//欧拉函数值*    **long** **long** **int** phiN **=** **(**p **-** 1**)** **\*** **(**q **-** 1**);**      
    *// 选择公钥e，满足1 < e < phi(N)且e与phi(N)互质*    **\***e **=** 65537**;**    printf**("计算公钥（N，e）...**\n**");**    *// 计算模数N*    **\***N **=** p**\***q**;**      
    *// 计算私钥d*    **\***d **=** inverse**(\***e**,** phiN**);  
}  
  
long** **long** **int** encrypt**(long** **long** **int** m**,long** **long** **int** e**,long** **long** **int** N**)** **{**    **return** mod\_pow**(**m**,** e**,** N**);  
}  
  
long** **long** **int** decrypt**(long** **long** **int** c**,long** **long** **int** d**,long** **long** **int** N**)** **{**    **return** mod\_pow**(**c**,** d**,** N**);  
}  
  
int** main**()** **{**    **int** bits **=** 32**;**    **long** **long** **int** e**,** d**,** N**;**      
    gen\_key\_pair**(**bits**,** **&**e**,** **&**d**,** **&**N**);**    *//printf("%d\n",count);*    printf**("公钥:（N，e）=（%lld, %lld）**\n**",** N**,** e**);**      
    printf**("计算私钥d...**\n**");**    printf**("私钥: d = %lld**\n**",** d**);**      
    **long** **long** **int** m **=** rand**()** **%** N**;**    **long** **long** **int** c **=** encrypt**(**m**,** e**,** N**);**    **long** **long** **int** decrypted **=** decrypt**(**c**,** d**,** N**);**    printf**("明文m = %lld, 密文c = %lld**\n**解密后的明文(16进制) = %llx**\n**解密后的明文(十进制) = %lld**\n**",** m**,** c**,** decrypted**,** decrypted**);**          
    **return** 0**;  
}**四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

文本

描述已自动生成

先随机生成素数p，q，再利用素数p，q计算公钥私钥，然后再生成明文并对明文进行加密，解密并且输出结果。

并且对代码进行修改，使之能按照16进制输出数据。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验是通过C语言实现rsa算法，rsa算法需要一对公私钥来实现数据的加解密，生成密钥时需要生成大素数，并且素数需要足够大才能满足安全性要求。在这里有一个小坑，就是C语言的int范围不能满足运算过程中的需求，会出现两个正整数相乘的结果出现负数，这就是结果超出int范围导致的，甚至连long int的范围都不够用，只有long long能满足需求。这次实验需要用到前面好几次实验的代码，但是不能直接拿来用，需要进行修改，这样更能增强我的综合能力。

实验项目名称： 椭圆曲线密码算法 实验学时： 6

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.12.13 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.12.22

实验7 椭圆曲线密码算法

一、基于OBE模式的实验目的和要求

1、熟悉ECC加解密算法的构造和运行过程。

2、使用C/C++语言编写实现ECC算法程序，加深对ECC点加法的理解。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、ECC密码体制是IEEE公钥密码标准P1363确定的公钥密码算法之一。有限域*Fp*上的椭圆曲线是指满足方程

*y*2 ≡ (*x*3+*ax*+*b*) mod *p*

的所有点(*x*, *y*)再加上一个无穷远点*O*构成的集合，其中*a*, *b*, *x*和 *y* 均在有限域*Fp*上取值，*p*是素数。对于椭圆曲线上的两点*P*, *Q*, 如果*P*=(*x, y*)，*Q*=(*x, -y*)，那么*S* = *P*+*Q* = *O*，即点(*x*, -*y*)是*P*的加法逆元，表示为 *-P*。

如果*P*, *Q*不互为逆元，*P*=(*x*1, *y*1)，*Q*=(*x*2, *y*2)， *S*=*P*+*Q*= (*x*3, *y*3)由以下公式确定：



其中，



倍点运算定义为重复加法，例如4*P* = *P*+*P*+*P*+*P*。

椭圆曲线密码算法的安全性基于椭圆曲线离散对数问题：设*P*为椭圆曲线上的点，*Q* = *xP*（倍点），*x*为正整数，已知*P*和*Q*很难确定出*x*的值。在本实验中，假设p为len比特素数，具体描述如下：

**系统参数**

素数p：len比特素数（即 且p为素数）

椭圆曲线*Ep(a, b)*，及阶为素数*n*的点*P*。

**密钥的生成**

用户随机选取整数*d*，1<*d*<*n*，计算*Q* = *dP*。则公钥为点*Q*，私钥为整数*d*。

**加密（明文无嵌入，随机化加密）**

对明文



随机选取整数*k* (*1<k<n*)，计算倍点*kQ* = (*x, y*), 计算



得到密文*c=*(*C0, c1, c2*) 。

**解密**

对于密文*c=*(*C0, c1, c2*)，先计算*dC0* = (*x, y*)，再计算



得到解密后的明文为 *m=*(*m1, m2*)。

2、编写程序，实现ECC密码并输出结果，编程的关键是实现点加法运算。

3、编写主函数，主要步骤有

生成并输出素数p和椭圆曲线*Ep(a, b)*，及阶为素数*n*的点*P*；（此步可选择完成）

选取私钥*d*；

输出公钥*Q*；

随机生成并输出明文；（或输入明文）

加密并输出密文；

解密并输出解密后的明文。

参考测试用例：例如，系统参数选为素数p = 7，椭圆曲线为*E*：y2 = x3 + 3，椭圆曲线加法群为*E7*(0, 3)（循环群），生成元取其中点*P* = (1, 2)，*P*的阶为13。

4、程序代码：

#include **<**stdio**.**h**>**#include **<**stdlib**.**h**>**   
#include **<**time**.**h**>  
typedef** **struct** **{**    **int** x**;**    **int** y**;  
}** Point**;  
  
int** mod\_inverse**(int** a**,** **int** m**)** **{**    **for** **(int** i **=** 1**;** i **<** m**;** i**++)** **{**        **if** **((**a **\*** i**)** **%** m **==** 1**)** **{**            **return** i**;**        **}**    **}**    **return** 0**;  
}  
  
int** get\_gcd**(int** x**,** **int** y**)** **{**    **if** **(**y **==** 0**)** **{**        **return** x**;**    **}** **else** **{**        **return** get\_gcd**(**y**,** x **%** y**);**    **}  
}**Point point\_addition**(int** a**,** **int** b**,** **int** p**,** Point Q**,** Point P**)** **{**    Point R**;**    **int** x1 **=** Q**.**x**,** y1 **=** Q**.**y**,** x2 **=** P**.**x**,** y2 **=** P**.**y**;**    **int** member**,** denominator**,** gcd\_value**,** inverse\_value**,** k**;**      
    **if** **(**x1 **==** 0 **&&** y1 **==** 0**)** **{**        **return** P**;**    **}**    **if** **(**x2 **==** 0 **&&** y2 **==** 0**)** **{**        **return** Q**;**    **}**      
      
    **if** **(**x1 **==** x2 **&&** y1 **==** y2**)** **{**        member **=** 3 **\*** **(**x1 **\*** x1**)** **+** a**;**        denominator **=** 2 **\*** y1**;**    **}** **else** **{**        member **=** y2 **-** y1**;**        denominator **=** x2 **-** x1**;**          
        **if** **(**member **\*** denominator **<** 0**)** **{**            member **=** abs**(**member**);**            denominator **=** abs**(**denominator**);**            member**=**denominator**\***p**-**member**;**        **}**    **}**      
    gcd\_value **=** get\_gcd**(**member**,** denominator**);**    member **=** member **/** gcd\_value**;**    denominator **=** denominator **/** gcd\_value**;**      
    inverse\_value **=** mod\_inverse**(**denominator**,** p**);**    k **=** **(**member **\*** inverse\_value**)** **%** p**;**      
    R**.**x **=** **(**k **\*** k **-** x1 **-** x2 **+** p**)** **%** p**;**    R**.**y **=** **(**k **\*** **(**x1 **-** R**.**x**)** **-** y1 **+** p**)** **%** p**;**      
    **return** R**;  
}**Point scalar\_multiply**(int** d**,** **int** a**,** **int** b**,** **int** p**,** Point P**)** **{**    Point Q **=** **{**0**,** 0**};**    Point temp **=** P**;**      
    **while** **(**d **!=** 1**)** **{**        Q **=** point\_addition**(**a**,** b**,** p**,** temp**,** P**);**      
        temp **=** Q**;**        d **-=** 1**;**    **}**      
    **if(**Q**.**x**<**0**){**        Q**.**x**=(**Q**.**x**+**p**)%**p**;**    **}**    **if(**Q**.**y**<**0**){**        Q**.**y**=(**Q**.**y**+**p**)%**p**;**    **}**      
    **return** Q**;  
}  
  
int** main**()** **{**    *// 系统参数*    **int** p **=** 7**;**    **int** a **=** 0**;**    **int** b **=** 3**;**    Point G **=** **{**1**,** 2**};**    **int** n **=** 13**;**      
    *// 生成私钥 d*    **int** private\_key\_d **=** 4**;**      
    *// 生成公钥 Q*    Point public\_key\_Q **=** scalar\_multiply**(**private\_key\_d**,** a**,** b**,** p**,** G**);**    *//printf("%d %d",public\_key\_Q.x,public\_key\_Q.y);*    *// 明文*    Point plaintext **=** **{**3**,** 4**};**      
    *// 加密*    srand**(**time**(**0**));**    **int** k **=** rand**()%**7**+**1**;**    Point C0 **=** scalar\_multiply**(**k**,** a**,** b**,** p**,** G**);**    *//printf("C0=(%d %d)\n",C0.x,C0.y);*    Point KP**=**scalar\_multiply**(**k**,**a**,**b**,**p**,**public\_key\_Q**);**    *//printf("KP=%d %d\n",KP.x,KP.y);*      
    Point ciphertext**;**    ciphertext**.**x **=** **(**KP**.**x**\***plaintext**.**x**)%**p**;**    ciphertext**.**y **=** **(**plaintext**.**y **\*** KP**.**y**)** **%** p**;**    *//printf("%d %d\n",ciphertext.x,ciphertext.y);*      
    *// 解密*    Point dC0 **=** scalar\_multiply**(**private\_key\_d**,** a**,** b**,** p**,** C0**);**    *//Point dC0 = {1,5};*    *//printf("dC0=(%d %d)\n",dC0.x,dC0.y);*    Point decrypted\_text**;**    *//decrypted\_text.x = ciphertext.x;*      
    *//printf("%d %d \n",dC0.x,mod\_inverse(dC0.x,p));*    decrypted\_text**.**x **=** **(**ciphertext**.**x **\*** mod\_inverse**(**dC0**.**x**,**p**))** **%** p**;**    **if** **(**decrypted\_text**.**x **<** 0**)** **{**        decrypted\_text**.**x **+=** p**;**    **}**      
    decrypted\_text**.**y **=** **(**ciphertext**.**y**\***mod\_inverse**(**dC0**.**y**,**p**))** **%** p**;**    **if** **(**decrypted\_text**.**y **<** 0**)** **{**        decrypted\_text**.**y **+=** p**;**    **}**      
    *// 输出结果*    printf**("素数域 (p): %d**\n**",** p**);**    printf**("曲线参数 (a, b): %d, %d**\n**",** a**,** b**);**    printf**("基点 (P): (%d, %d)**\n**",** G**.**x**,** G**.**y**);**    printf**("基点阶数 (n): %d**\n**",** n**);**    printf**("私钥 (d): %d**\n**",** private\_key\_d**);**    printf**("公钥 (Q): (%d, %d)**\n**",** public\_key\_Q**.**x**,** public\_key\_Q**.**y**);**    printf**("明文: (%d, %d)**\n**",** plaintext**.**x**,** plaintext**.**y**);**    printf**("随机数：%d**\n**",**k**);**    printf**("密文: ( (%d,%d), %d, %d )**\n**",**C0**.**x**,**C0**.**y**,** ciphertext**.**x**,** ciphertext**.**y**);**    printf**("解密后的文本: (%d, %d)**\n**",** decrypted\_text**.**x**,** decrypted\_text**.**y**);**      
    **return** 0**;  
}**

四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

文本

中度可信度描述已自动生成

设定好椭圆的基本参数a，b，p，基点，然后计算出私钥，公钥，最后进行加密解密操作。

图片包含 文本

描述已自动生成

经过老师的提示，将随机数k也输出出来

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验是用C语言实现ecc算法，算法中最有特点的就是用随机数k进行随机加密，所以在回显中也要输出一下随机数k。既然是随机数，就需要随机生成，可以利用rand（）函数实现伪随机。本次实验还是有着一定难度的。我就在实验过程中遇到了程序中的倍点运算和点加运算的结果不一致，最后发现是λ的运算过程中可能会出现负数的情况，进行修改之后程序终于能正常输出内容了。本次实验不止锻炼了我的C语言能力，还让我对ecc机密算法有了更深刻的认识。