Lab4——公钥密码算法 RSA

学号：2111033
姓名：艾明旭
年级：2021 级
专业：信息安全

一、实验内容说明

1、实验目的

通过实际编程了解公钥密码算法 RSA 的加密和解密过程，加深对公钥密码算法的了解和使用

2、实验要求

1. 对实验步骤 2，写出生成素数的原理，包括随机数的生成原理和素性检测的内容，并给出程序框图
2. 对实验步骤 3，要求分别实现加密和解密两个功能，并分别给出程序框图。

3、实验步骤

1. 为了加深对 RSA 算法的了解，根据已知参数：p=3，q=11，m=2，手工计算公钥和私钥，并对明文 m 进行加密，然后对密文进行解密
2. 编写一个程序，用于生成 512 比特的素数
3. 利用 2 中程序生成的素数，构建一个 n 的长度为 1024 比特的 RSA 算法，利用该算法实现对明文的加密和解密
4. 在附件中还给出了一个可以进行 RSA 加密和解密的对话框程序 RSATool，运行这个程序加密一段文字，了解 RSA 算法原理

二、实验环境

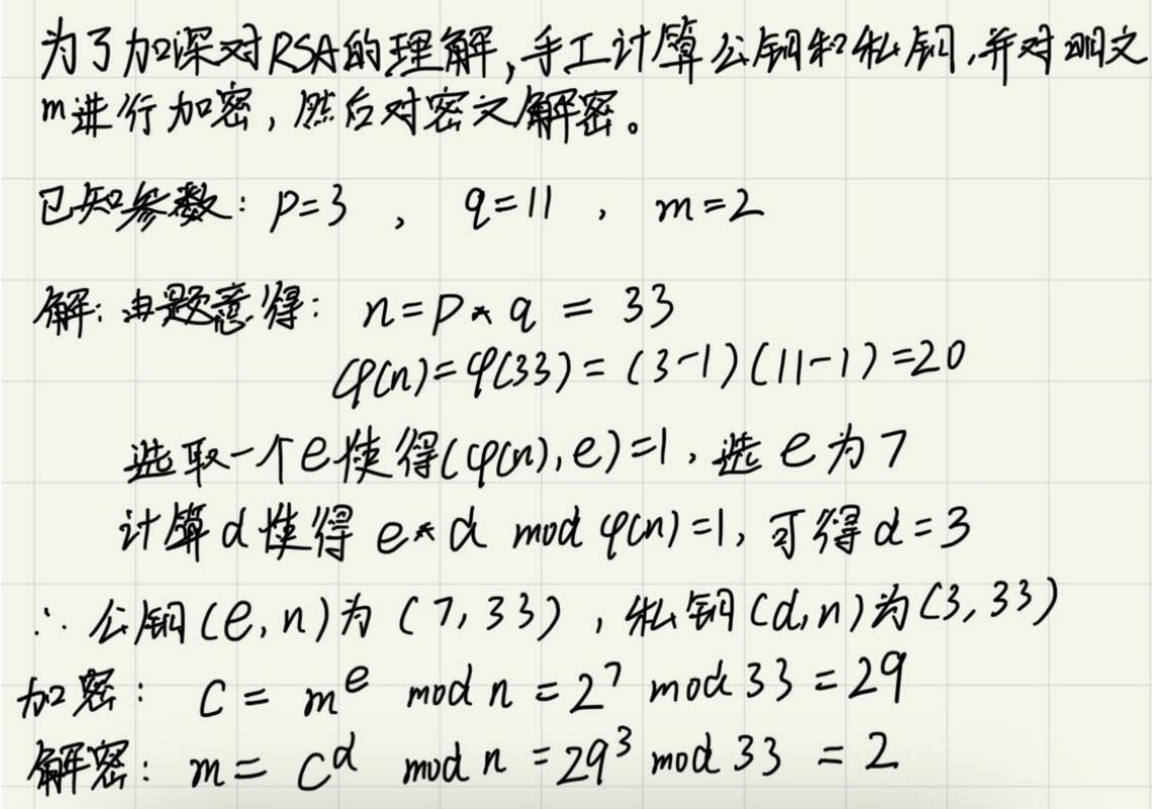
- 操作系统：win11
- 软件系统：visual studio
- 编译工具：vs2022
- 编程语言：C++

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、手工计算

根据实验步骤 1 的参数，可以手工计算 RSA 算法如下所示：



2、流程分析

(1)大素数生成

在本次实验中最重要的就是重载大数，因为在 c++里，int 类型所能容纳的大小并不足以支持我们的加密需求，我们的数的大小范围已经远远超过其能表示的数的范围，所以得为这些大的数字重新定义各种运算，在重载之后，程序实现起来将会非常的方便，我将分为以下几步进行操作：

1、重载大数类

我们设计如下类 `bigint`，其含有一个大小为 1025 的 bool 数组，以及代表正负的 bool 型变量 flag

2、构造函数

- 无参构造函数：全置为 0

- 参数为 String 的构造函数对每个字符进行判断，每个字符对应 0000~1111 中的一个，进行遍历赋值
- 参数为 int 的构造函数，采用除二取余，进行赋值

3、重载运算符

我们对+、-、*、/、%、>、<、==进行重载：

- 例如+，我们首先判断符号，对于不同情况进行处理（如 `a + b`,b 是负数，则将 b 符号位取反，`return a - b`。后续符号处理方法类似，不再进行说明），对于两个都为正的情况，我们用一个 `temp` 存储进位信息，然后循环对每一位进行加法，同时存储进位信息用于下一位加法。
- -减法与+加法类似
- * 用一个 `bigint` 类存储每次相乘的结果 `result`，例如 `a * b`，循环检测 b 的每一位是否为 1，同时对 a 进行左移 1 位，若 b 的该位为 1，则将 a 加到 `result` 中，最后返回结果
- / 与乘法类似，`a / b`，循环对 b 左移 1 位，在循环时，若 `a >= b`，就让 `a = a - b`,并将 `result` 的那一位置 1，当 a < b 时循环停止
- % 计算 `a - (a / b) * b` 即可
- 比较运算符号 我们只需要相减然后判断是否全为 0 即可。

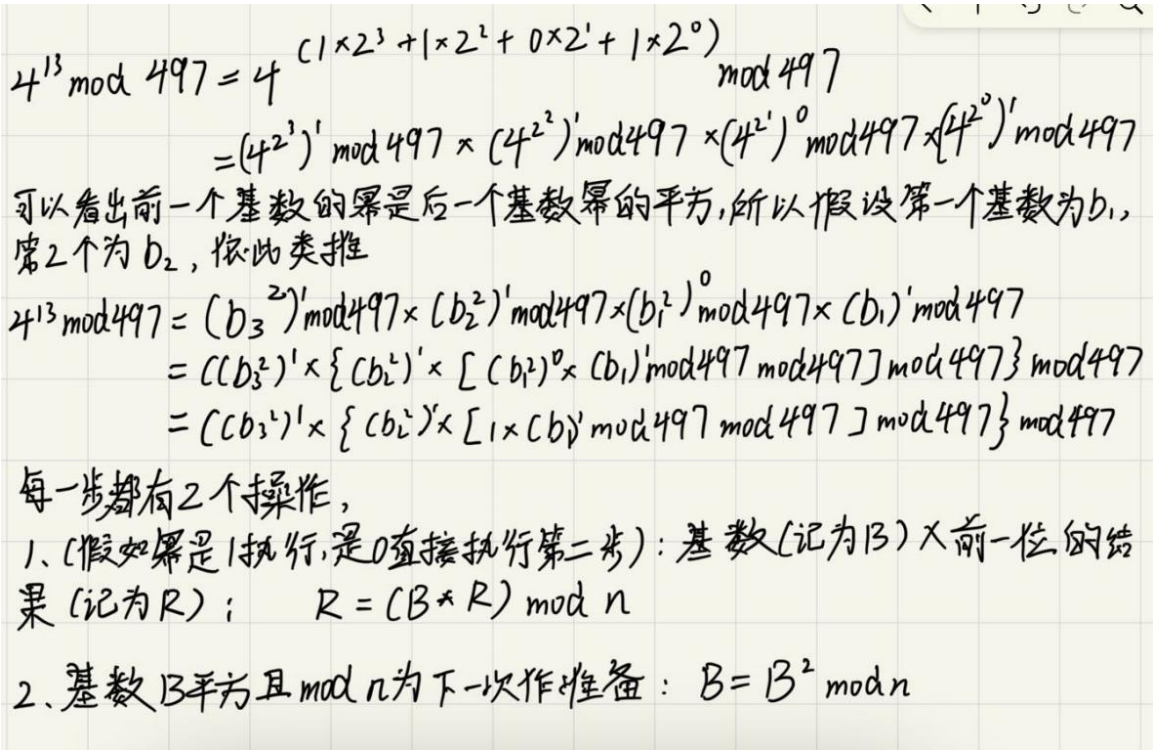
关键函数如下：

- 随机函数 `random(int n)`生成一个位数为 n 的数，将最后一位置 1，其他位随机生成
- 模乘函数 `modmul(a，b，c)`循环 每次循环先计算乘法结果然后取模再进行下一轮
- 模幂函数如下所示：

```
bigint expmod(bigint e, bigint n)
{
    bigint c("1");
    int i = zyl_max - 1;
    while (!e[i]) {
        i--;
    }
    int j;
    for (j = i; j >= 0; j--) {
        c = modMul(c, c, n);
        if (e[j]) {
            c = modMul(c, (*this), n);
```

```
            }
        }
        return c;
    }
```

对于此函数的原理大致如下图所示：



- 求逆元函数 `bigint inv(bigint x)`,我们采用扩展欧几里得算法求即可
- 输出函数 `print()`每四位截取进行输出，对四位进行判断 `0000~1111`

4、素数生成器

将重载大数类这个硬骨头啃下后，程序接下来的步骤就变得非常简单了（`2000` 以内的 `303` 个素数已经提前写到一个结构体中去了）

- 首先是随机数生成器：

```
void random(int n)
{
    bigint temp;
    srand((int)time(0));
    for (int i = 0; i < n; i++)
    {
        int x = rand() % 2;
        if (x == 1)
            num[i] = true;
        else
```

```
                num[i] = false;
        }
        for (int i = zyl_max - 1; i >= n; i--)
        {
            num[i] = false;
        }
        num[n - 1] = true;
        num[0] = true;
    }
```

- 然后是判断是否是素数的函数：

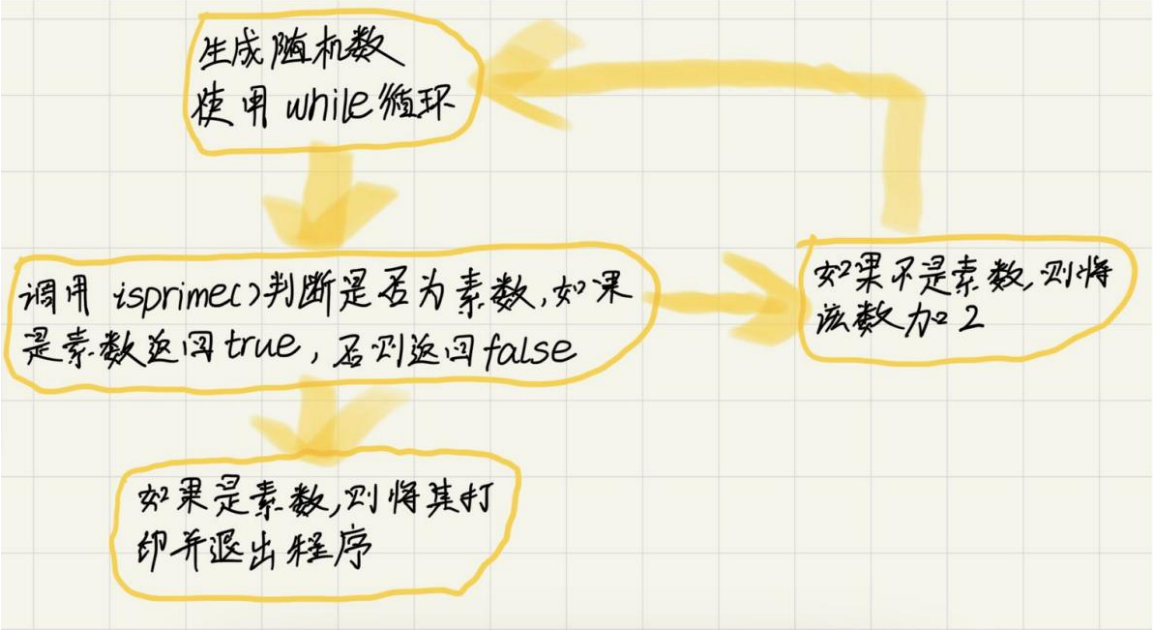
```
bool isPrime(bigint a)
{
    const bigint ZERO(0);
    for (int i = 0; i < 303; i++)
    {
        bigint p(pri[i]);
        bigint d(1);
        bigint c = (a % p);
        if (c == ZERO)
            return false;
    }
    return true;
}
```

主要逻辑是将 2000 以内的 303 个素数进行取余运算，若出现余数为 0 的情况，则不是素数

- 接下来就是生成一个大素数，其代码如下：

```
bigint getPrime()
{
    bigint temp;
    bigint TWO("2");
    temp.random(512);
    while (true) {
        if (isPrime(temp)) {
            return temp;
        }
        temp = temp + TWO;
    }
}
```

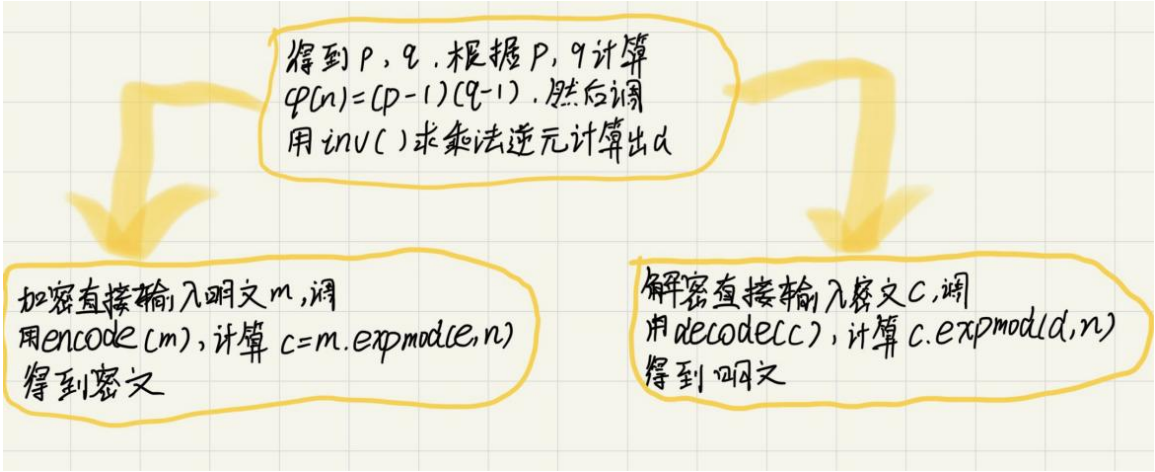
- 生成素数的程序流程图如下所示：



(2)RSA 流程

1. 由于我们已经在大数类里提前预备好了各种函数，所以我们直接调用即可
2. 首先我们获取 p、q 两个数，然后计算（p-1）*（q-1），d 等数据
3. 根据用户输入来决定是加密还是解密
4. 根据选择使用 expmod()的参数得到结果
5. 由于我们产生的素数只是试除了 2000 以内的素数，可能生成的并不是素数，所以我们这里 p、q 采用 RSA 工具生成的素数以验证程序正确性
6. E 就使用工具里默认的 10001

- 其流程图如下所示：



3、代码实现

在程序流程中已经列出了部分函数，这里对没有列出的部分重要函数做以补充：

大素数类中的各种函数

- 首先有三种构造函数，一种是默认的，一种是数字的，输入字符的较难，需要逐位进行比较，如下所示：

```
bigint(string wxn)
{
    int i, j = 0;
    int temp;
    for (i = 0; i < zyl_max; i++) {
        num[i] = 0;
    }
    for (i = (wxn.length() - 1); i >= 0; i--) {
        if (wxn[i] >= 'a' && wxn[i] <= 'f') {
            temp = wxn[i] - 'a' + 10;
        }
        else {
            if (wxn[i] >= 'A' && wxn[i] <= 'F') {
                temp = wxn[i] - 'A' + 10;
            }
            else {
                temp = wxn[i] - '0';
            }
        }
        if (temp / 8) {
            num[4 * j + 3] = 1;
        }
        if ((temp % 8) / 4) {
            num[4 * j + 2] = 1;
        }
        if ((temp % 4) / 2) {
            num[4 * j + 1] = 1;
        }
        if (temp % 2) {
            num[4 * j] = 1;
        }
        j++;
    }
    flag = 0;
}
```

- 接下来是运算符的重载，在这里只列出加法，其余与之类似：

```
friend bigint operator+(bigint a, bigint b)
{
    bigint result;
    int i;
    bool temp = 0;
    if (a.flag == b.flag) {
        for (i = 0; i < zyl_max; i++) {
            if (a[i] == 0 && b[i] == 0 && temp == 0) {
                result.make(i, 0);
```

```
                temp = 0;
            }
            else {
                if ((a[i] == 1 && b[i] == 0 && temp == 0) ||
                    (a[i] == 0 && b[i] == 1 && temp == 0) ||
                    (a[i] == 0 && b[i] == 0 && temp == 1)) {
                    temp = 0;
                    result.make(i, 1);
                }
                else {
                    if ((a[i] == 1 && b[i] == 1 && temp == 0) ||
                        (a[i] == 0 && b[i] == 1 && temp == 1) ||
                        (a[i] == 1 && b[i] == 0 && temp == 1)) {
                        temp = 1;
                        result.make(i, 0);
                    }
                    else {
                        if (a[i] == 1 && b[i] == 1 && temp == 1) {
                            temp = 1;
                            result.make(i, 1);
                        }
                    }
                }
            }
        }
        result.flag = a.flag;
    }
    if (a.flag == 0 && b.flag == 1) {
        b.Num_Not();
        return a - b;
    }
    if (a.flag == 1 && b.flag == 0) {
        a.Num_Not();
        return b - a;
    }
    if (temp) {
        cout << "Overflow" << endl;
    }
    return result;
}
```

- 然后是刚刚提到的利用欧几里得算法的求逆元函数和打印函数

```
bigint inv(bigint x)
{
    bigint ZERO("0"), ONE("1");
    bigint x1 = ONE, x2 = ZERO, x3 = x;
    bigint y1 = ZERO, y2 = ONE, y3 = (*this);
    bigint t1, t2, t3;
    if (y3 == ONE) {
        return ONE;
```

```
    }
    bigint q;
    bigint g;
    do {
        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;
    } while (!(y3 == ONE));
    g = y2;
    if (!(g > ZERO))
        g = x + g;
    return g;
}

    // 16 进制的打印
void print()
{
    if (this->flag == 1) {
        cout << '-';
    }
    char result[zyl_max];
    int i;
    for (i = zyl_max - 1; i >= 0; i--) {
        if ((*this)[i] == 1) {
            break;
        }
    }
    i++;
    int k;
    int length = 0;
    switch (i % 4) {
    case 1:
        length = i + 3;
        break;
    case 2:
        length = i + 2;
        break;
    case 3:
        length = i + 1;
        break;
    case 0:
        length = i;
        break;
```

```
        }
        for (k = 0; k < length; k = k + 4) {
            if ((*this)[k] == 0 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 0) {
                result[k / 4] = '0';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 0) {
                result[k / 4] = '1';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 0) {
                result[k / 4] = '2';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 0) {
                result[k / 4] = '3';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 0) {
                result[k / 4] = '4';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 0) {
                result[k / 4] = '5';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 0) {
                result[k / 4] = '6';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 0) {
                result[k / 4] = '7';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 1) {
                result[k / 4] = '8';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 1) {
                result[k / 4] = '9';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 1) {
                result[k / 4] = 'A';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 0 && (*this)[k + 3] == 1) {
                result[k / 4] = 'B';
            }
```

```
            if ((*this)[k] == 0 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 1) {
                result[k / 4] = 'C';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 0 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 1) {
                result[k / 4] = 'D';
            }
            if ((*this)[k] == 0 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 1) {
                result[k / 4] = 'E';
            }
            if ((*this)[k] == 1 && (*this)[k + 1] == 1 && (*this)[k + 2]
== 1 && (*this)[k + 3] == 1) {
                result[k / 4] = 'F';
            }
        }
        if (i == 0) {
            cout << '0' << endl;
        }
        else {
            for (i = (k / 4) - 1; i >= 0; i--) {
                cout << result[i];
            }
            cout << endl;
        }
    }
```

判断素数和生成素数刚刚已经列出过了，这里就不再赘述

接下来是 RSA 方面的各个函数

- 首先是其构造函数，计算公钥和私钥

```
RSA(bigint a, bigint b)
{
    bigint one("1");
    p = a;
    q = b;
    n = p * q;
    f = (p - one) * (q - one);
    bigint curr("10001");
    // 默认选取的公钥e为10001，和老师给的工具中相同
    e = curr;
    // d为私钥
    d = e.inv(f);
}
```

- 接下来是加解密

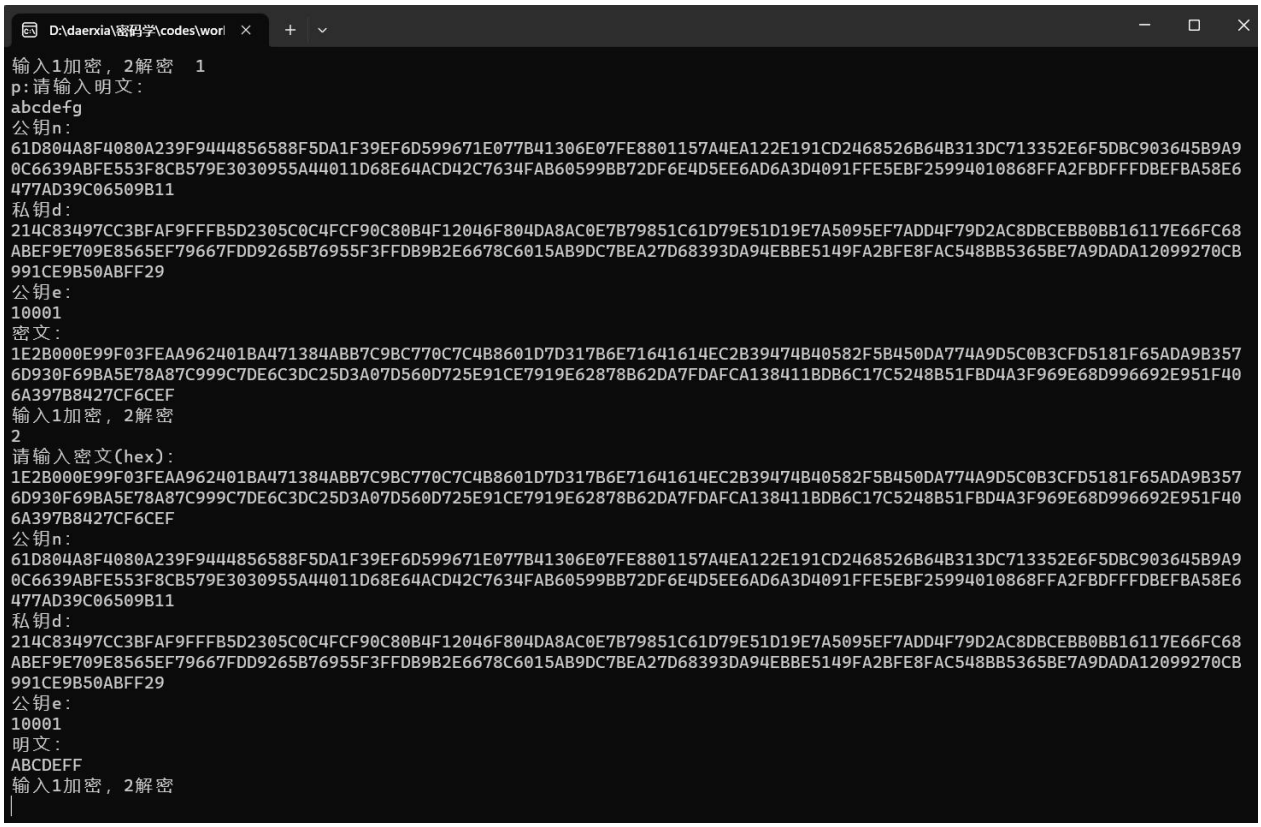
```
void encode(bigint m1)
{
```

```
        m = m1;
        c = m.expmod(e, n);
    }
    void decode(bigint c1)
    {
        c = c1;
        m = c.expmod(d, n);
    }
```

4、结果展示

- 首先是素数生成的结果：

- 然后是 RSA 加解密的结果：



四、总结与展望

1、总结

本次密码学实验使用了 RSA 公钥密码算法进行加密和解密。通过实验，我们了解了 RSA 公钥密码算法的原理和实现过程，以及了解了 RSA 算法的优缺点。实验中，我们通过生成公钥和私钥、加密和解密数据的操作，深入理解了 RSA 算法的实现方法。

通过本次实验，我更加深入地认识了密码学这一重要的领域。密码学在信息安全中起着重要的作用，提高了我对网络安全问题的认识。同时，我也更加了解了 RSA 算法，了解了其在数据加密和解密中的作用。

2、展望

展望未来，我将继续学习和探究密码学相关知识，对现代密码技术进行深入研究，并探索实际应用。希望能够在信息安全领域做出一份贡献。