一、 实践目的

实现 RSA 的密钥生成、数据加密、数字签名。

二、 实践内容

密钥生成包括生成两个大素数 p,q，计算 n=p×q 和 Φ(n)=(p-1)(q-1)，然后选择与 Φ(n)互素且小于 Φ(n)的整数 e，计算 $d=e^{-1} \bmod \Phi(n)$，最后得到公钥{e,n}和私钥{d,n}。数据加密是指用公钥{e,n}对指定的明文进行加密。数字签名是指用私钥{d,n}对指定的明文进行加密。

三、 实践环境

硬件环境：Windows10 系统

软件环境： Visual Studio Code

四、 实践过程与步骤

1. 大数的基本运算实现

由于 C 语言在处理大数字时没有特别好用的工具，不论是用 int 类型还是 long long 类型进行运算时都很容易造成溢出导致程序崩溃，于是我的想法是将所有数字都用字符串表示，并且实现关于数字字符串的基本运算。

这里我参考了：C 语言实现 大数运算 加减乘除模运算 超详细 石小参的博客-CSDN 博客_c 语言大数模运算，在实际运用中，发现运算数有负数时会产生 bug，于是我在该基础上补足了包含负数的运算规则，成功实现了大数字字符串的加法、减法、乘法、除法、取模这五种运算。以下时部分代码

（1） 加法运算

```c
char *add(char *str1, char *str2)
{
    if(str1[0] != '-' && str2[0] != '-'){ //两正数相加
        //  确定较大，较小的字符串，并获取它们的长度
        int n = maxer(str1, str2);
        char *max_str = n >= 0 ? str1 : str2;
        char *min_str = n >= 0 ? str2 : str1;
        int len_max = strlen(max_str);
        int len_min = strlen(min_str);

        //  创建 较大字符串 长度大1 的空间 存储结果
        char *str = malloc(MAX_SIZE + 1);
        // str[len_max + 1] = '\0';
        memset(str,'\0',MAX_SIZE+1);

        //  进行加法运算
        int plus = 0;        // plus 为 进位
        int i = len_max - 1; // 标记 较大字符串 最低位字符的位置
        int j = len_min - 1; // 标记 较小字符串 最低位字符的位置
        for (; i >= 0; i--)
        {
            if (j >= 0)
            {
                str[i + 1] = max_str[i] + min_str[j] - '0' + plus; // 字符相加
                j--;                                               // 位置左移 1
                plus = str[i + 1] > '9' ? 1 : 0;                   // 判断是否进位
                if (plus == 1)
                    str[i + 1] -= 10; // 进位 则减 10
            }
            else // 较小字符串的字符 全部加完
            {
                str[i + 1] = max_str[i] + plus;
                plus = str[i + 1] > '9' ? 1 : 0;
                if (plus == 1)
                    str[i + 1] -= 10;
            }
        }

        //  根据 最高位 是否 进位，确定 返回指针
        if (plus == 0)
            return str + 1; // 返回 str 右移1位的位置
        else
        {
            str[0] = '1'; // str首位 赋 '1'
            return str;   // 返回 str
        }

    } else if(str1[0] == '-' && str2[0] == '-') {   //两负数相加，相当于两个数的正数部分相加再取负
        char *tmp = add(str1+1,str2+1);
        char *res = (char*)malloc(strlen(tmp)+2);
        memset(res,'\0',strlen(tmp)+2);
        res[0] = '-';
        for(int i=1,j=0; j<strlen(tmp); i++,j++){
            res[i] = tmp[j];
        }
        return res;
    } else if(str1[0] != '-' && str2[0] == '-') {   //一正一负相加，相当于第一个数减去第二个数的正数部分
        return sub(str1,str2+1);
    } else {
        return sub(str2,str1+1);                    //一负一正相加，相当于第二个数减去第一个数的正数部分
    }
}

/**...
char *sub(char *str1,char *str2) {...
```

（2） 减法运算

```c
char *sub(char *str1,char *str2) {
    if(str1[0]!='-'&&str2[0]!='-') {    //两正数相减
        //  确定结果 正负 或 0
        int is_neg = maxer(str1, str2);
        if (is_neg == 0)
            return "0"; // 若为 0, 直接返回 "0" 结束
        char *max_str = is_neg > 0 ? str1 : str2;
        char *min_str = is_neg > 0 ? str2 : str1;
        int len_max = strlen(max_str);
        int len_min = strlen(min_str);

        char *str = malloc(MAX_SIZE + 1);
        // str[len_max + 1] = '\0';
        memset(str,'\0',MAX_SIZE+1);

        //  进行减法运算
        int plus = 0; // plus 为退位
        int i = len_max - 1;
        int j = len_min - 1;
        for (; i >= 0; i--)
        {
            if (j >= 0)
            {
                str[i + 1] = max_str[i] - min_str[j] + '0' + plus; // 字符相减
                j--;
                plus = str[i + 1] < '0' ? -1 : 0; // 是否 退位
                if (plus == -1)
                    str[i + 1] += 10; // 退位 则加 10
            }
            else
            {
                str[i + 1] = max_str[i] + plus;
                plus = str[i + 1] < '0' ? -1 : 0;
                if (plus == -1)
                    str[i + 1] += 10;
            }
        }

        //  确定最后一个 前位0 的位置, 默认 str首位 为'0'
        int n_zero = 0;
        for (int i = 1; i <= len_max; i++)
        {
            if (str[i] != '0')
                break;
            n_zero++;
        }

        //  根据 正负 和 最后一个前位0的位置 确定 返回指针
        if (is_neg < 0)
        {
            str[n_zero] = '-';   // 将最后一个前置0的 赋为 '-'
            return str + n_zero; // 返回'-'的位置
        }
        else
            return str + n_zero + 1; // 返回最后一个前置0 右移1位的位置
    } else if(str1[0]=='-'&&str2[0]=='-') {      //两负数相减，相当于第二个数的正数部分减去第一个数的正数部分
        return sub(str2+1,str1+1);
    } else if(str1[0]!='-'&&str2[0]=='-'){      //一正一负相减，相当于第一个数加上第二个数的正数部分
        return add(str1,str2+1);
    } else {                                    //一负一正相减，相当于两个负数相加
        char *_str2 = sub("0",str2);
        return add(str1,_str2);
    }
}
```

由于篇幅原因仅在这里展示加法运算与减法运算的代码，详细代码可在源码文件中查看。

2. 密钥生成

在实现了大数的基本运算后，就可以开始进行密钥的生成了。

生成密钥首先要生成大素数。我的想法是在 10^10 以上范围内进行素数检测，取 1000 个素数到数组中，在从数组中随机选取两个素数作为生成的两个大素数。

以下是生成大素数数组的函数，最开始的数是一个奇数，然后判断其是否是素数，如果不是，这个奇数+8，然后继续判断，如果是素数，就存入数组中，最后返回一个存储了 1000 个大素数的数组。

```
long long *bigPrime(){
    long long *res = (long long*)malloc(sizeof(long long)*1000);
    srand(time(0));
    int x = rand()%100;
    int y = rand()%100+100;
    long long rangeFrom = least+x*n_1w;
    long long rangeTo = least+y*n_1w;
    if(rangeFrom%2==0) rangeFrom++; //确保最开始的数是个奇数
    int j=2,count = 0;
    for(long long i = rangeFrom; i<=rangeTo; i += 8 ){
        for(j=2; j*j<=i; j++){
            if(i%j == 0) break;
        }
        if(j*j>i){
            res[count++] = i;
        }
        if(count==1000) break;
    }
    return res;
}
```

获取大素数数组后，我们从中随机选取两个素数作为 p,q,并且将它们转化为 10 进制字符串，方便我们接下来的运算

```
long long *num = bigPrime();     //得到一个有1000个随机大素数的数组
srand(time(0));                  //重置随机数种子
//获取两个随机数
int x = rand()%500;
int y = rand()%500+500;
//以上述两个随机数为下标在大素数数组中选出p,q,并且将它们转化为十进制字符串
char *str1 = Dec2Str(num[x]);
char *str2 = Dec2Str(num[y]);
```

p，q 相乘得到 n

```
//p,q相乘得到n
char *str3 = mul(str1,str2);
```

计算 Φ(n)=(p-1)*(q-1)

```
//计算(p-1)*(q-1)
char *str1_1 = sub(str1,"1");    //p-1
char *str2_1 = sub(str2,"1");    //q-1
char *Fn = mul(str1_1,str2_1);   //Fn = (p-1)*(q-1)
```

选择 e，这里我让程序默认选择了常用的素数 65537

```
char *e = "10001";               //e:选用常用的65537, H(e)=10001
```

计算 $d = e^{-1} \bmod \Phi(n)$,这里我才用了扩展欧几里得算法，参考了：[辗转相除法求模逆（C 语言）_qq2672909406 的博客-CSDN 博客_c 语言求模逆元]

```
//计算d = e^(-1) mod Fn
char *e_d = Hex2Dec(e);
printf("e:%s\n",e_d);
char *d;
char *a = Fn;
char *b = e_d;
char *q1,*r;
char *u = "0", *v = "1",*t;
while(strcmp(b,"0")){
    q1 = divi(a,b);
    char *tmp = mul(b,q1);
    r = sub(a,tmp);
    a = b;
    b = r;
    t = v;
    char *tmp1 = mul(q1,v);
    v = sub(u,tmp1);
    u = t;
}
//最后如果 u<0 , d = u + Fn, 否则 d = u
if(u[0]=='-') d = add(u,Fn);
else d = u;
```

最后将得到的数据写入到文件中。

3. 加密与签名

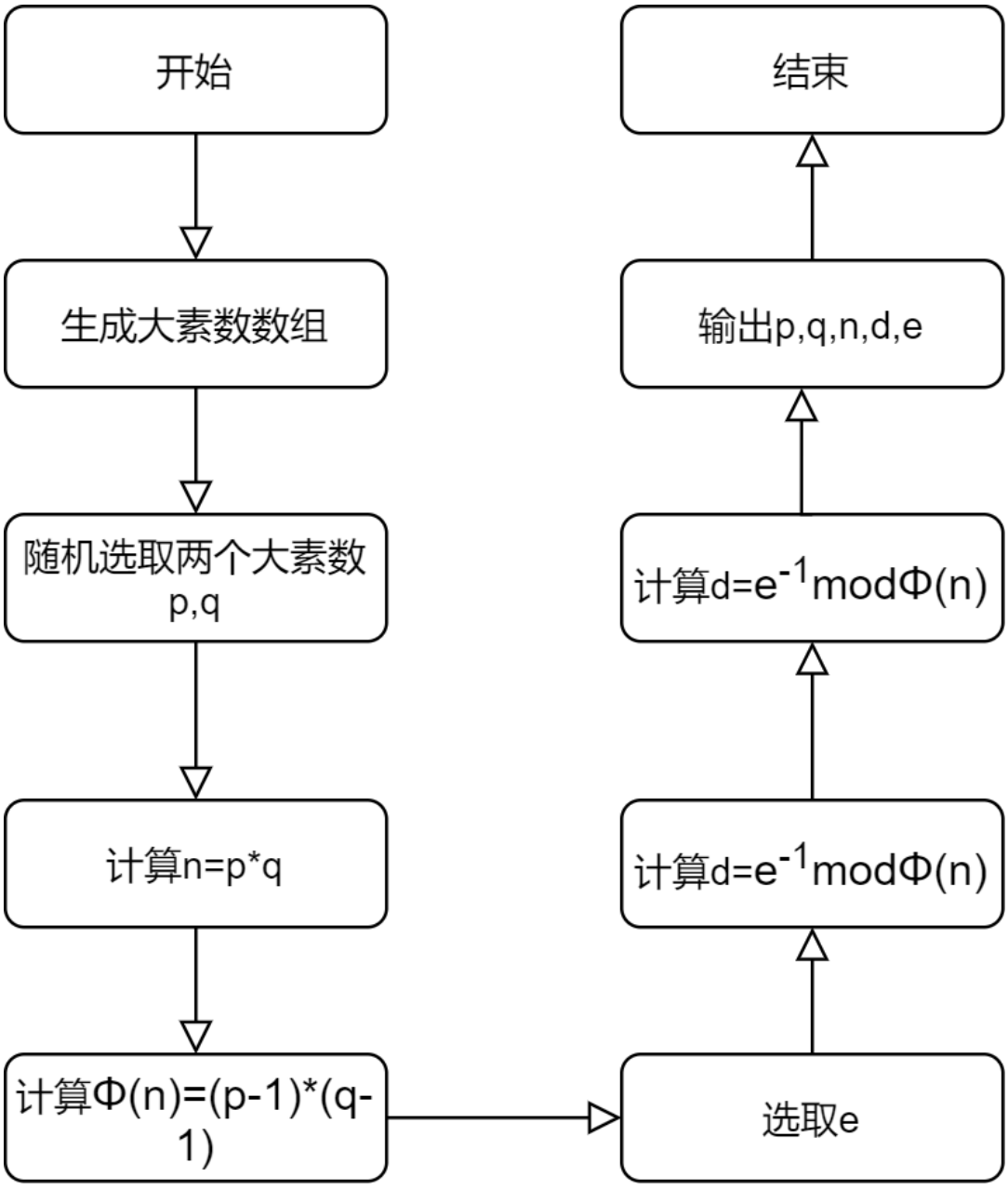
加密，解密，签名的算法其实都是一样的，只是选择的数据不同而已。

加密，解密，签名的算法都可以写成 a = b^k mod n。由于指数可能很大，在进行运算时如果循环轮数太多的话程序运行时间将非常长，于是这里我采用的是乘法-乘方算法，参考了课件，代码如下：

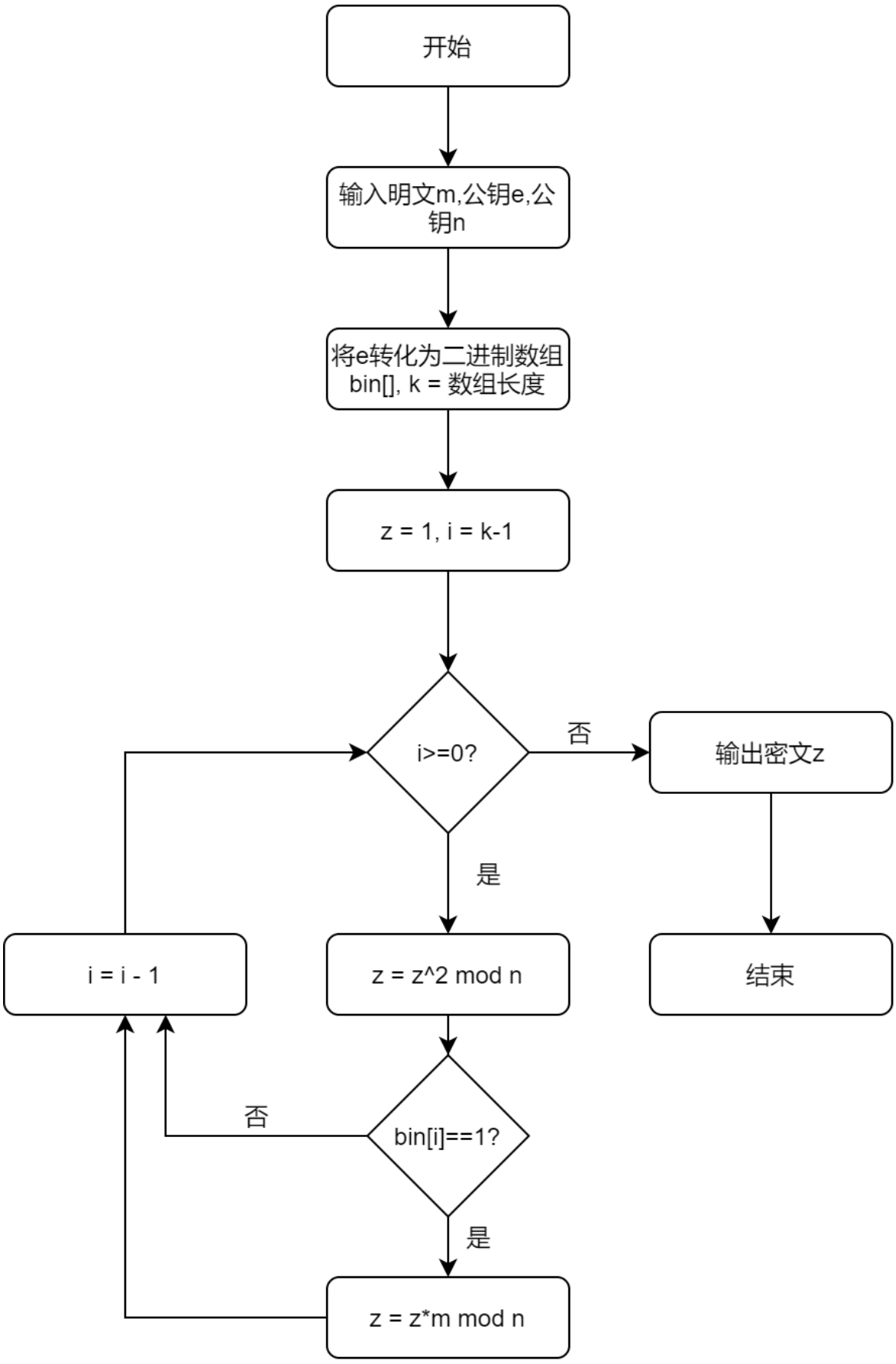
```
char *EncryptOrDecrypt(char *n,char *mOrc,char *eOrd){
    int bin[MAX_SIZE];  //将e或d转为二进制数组
    int k = Hex2Bin(bin,eOrd);
    char *tmp1 = mOrc;
    char *z = "1";
    for(int i=k-1; i>=0; i--) {
        z = mul(z,z);
        z = mol(z,n);
        if(bin[i]==1){
            z = mul(z,mOrc);
            z = mol(z,n);
        }
    }
    return z;
}
```

运行结果见：六、实践结果与分析

五、 程序设计方案



密钥生成流程图


开始
输入明文m,公钥e,公钥n
将e转化为二进制数组bin[], k = 数组长度
z = 1, i = k-1
i>=0?
否
输出密文z
是
i = i - 1
z = z^2 mod n
结束
否
bin[i]==1?
是
z = z*m mod n

加密流程图

利用到的数据结构：

int[]，long long[]，char[]，指针，动态内存分配(malloc)，宏

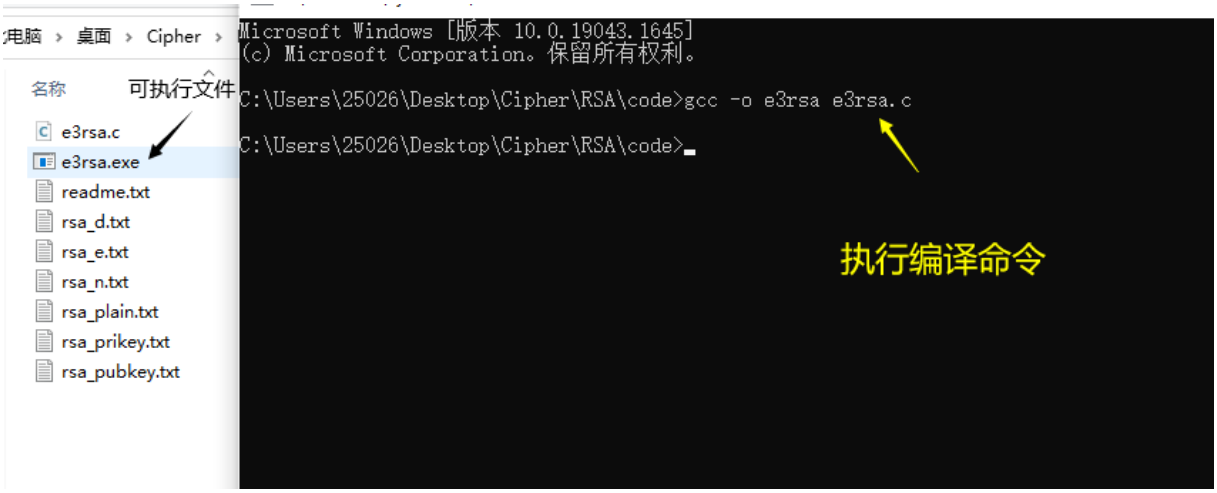
IO 接口：

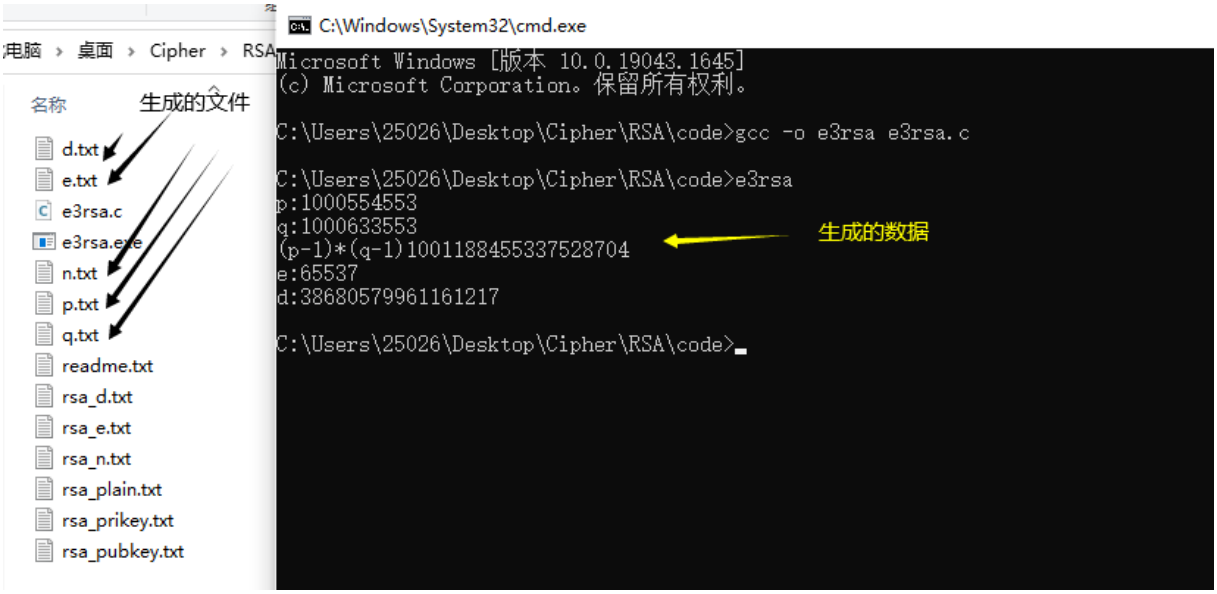
C 自带的 FILE 类型

六、 实践结果与分析

1. 密钥生成：

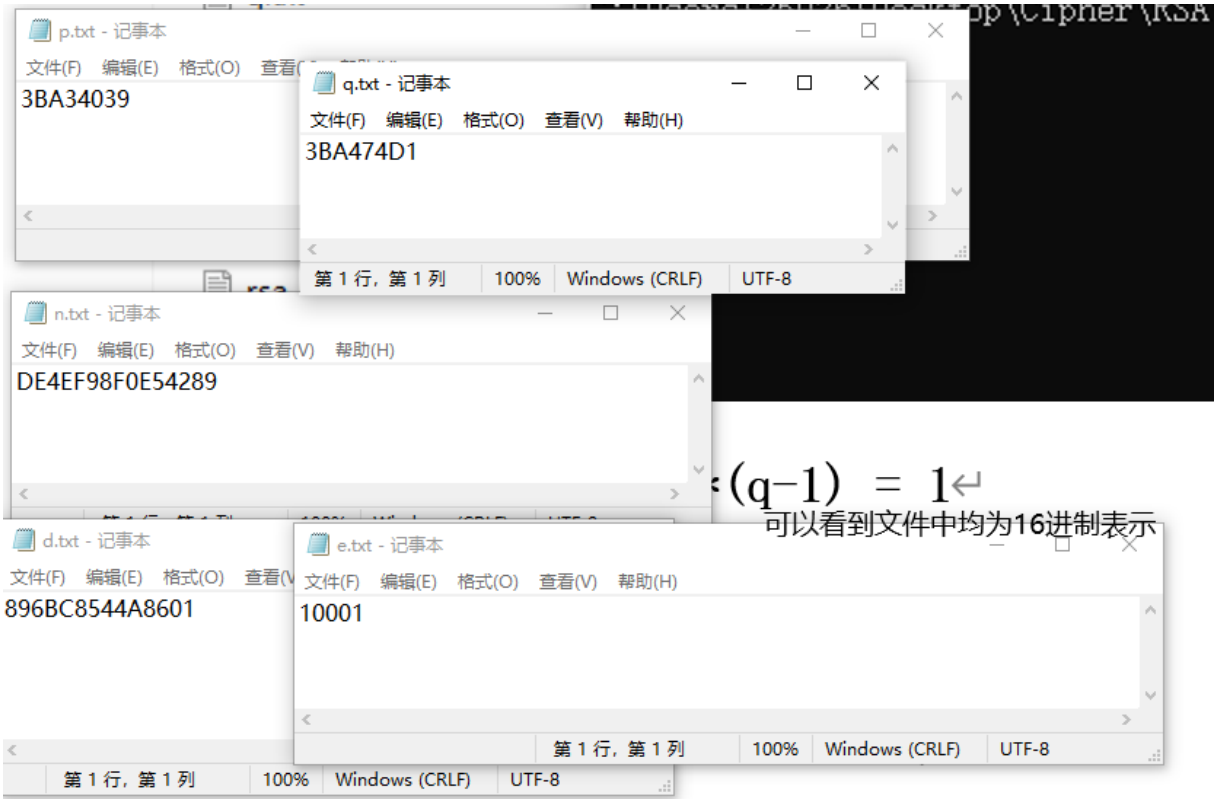
在 code 文件夹下执行命令：gcc -o e3rsa e3rsa.c 生成可执行程序 e3rsa.exe，直接执行命令 e3rsa，默认生成 p,q,n,e,d 并且写入到文件 p.txt，q.txt，n.txt，e.txt，d.txt 中。以下是运行截图：





经验证：d*e mod (p-1)*(q-1) = 1

并且所有文件中的数据均为 16 进制表示：



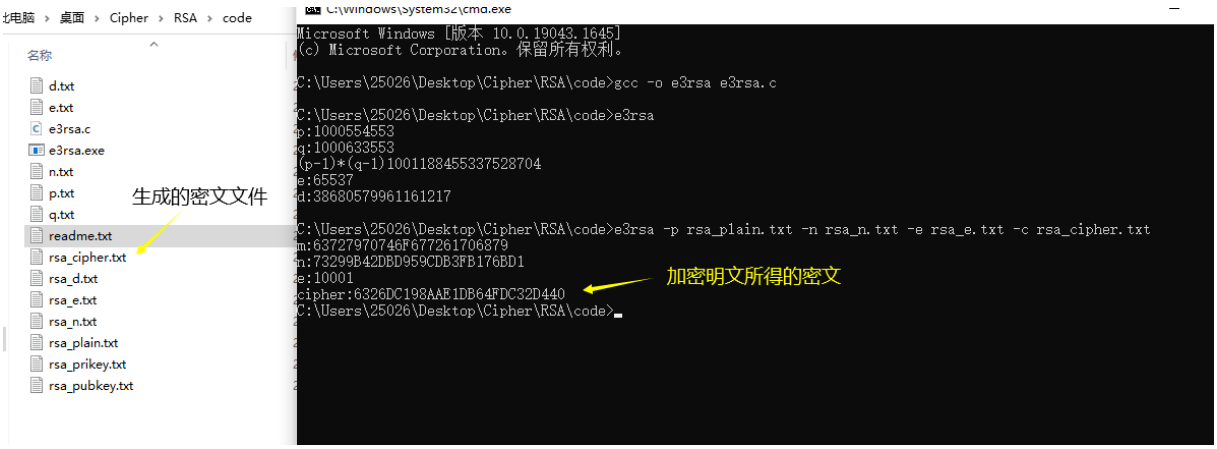
经验证：16 进制的数据与 10 进制的数据一一对应，无误

2. **加密与签名：**

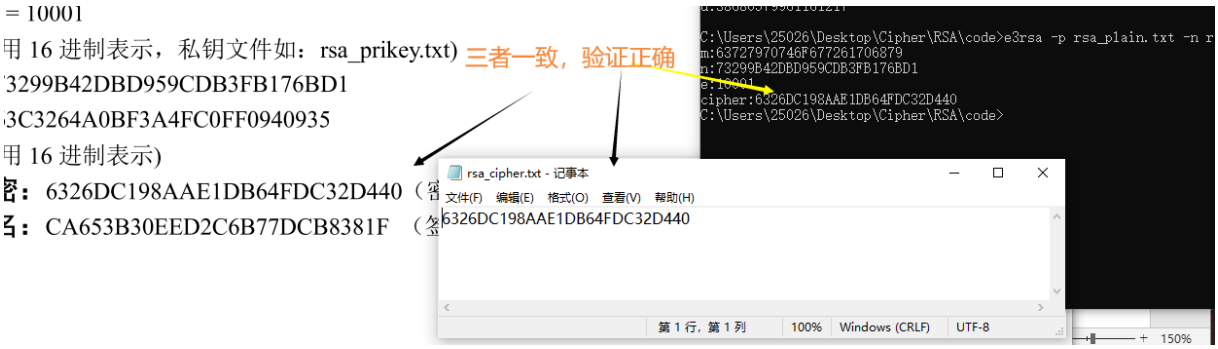
加密：

文件夹中的 rsa_n.txt,rsa_e.txt,rsa_d.txt 为 pdf 中的数据，用于验证算法正确性。

执行命令：e3rsa -p rsa_plain.txt -n rsa_n.txt -e rsa_e.txt -c rsa_cipher.txt

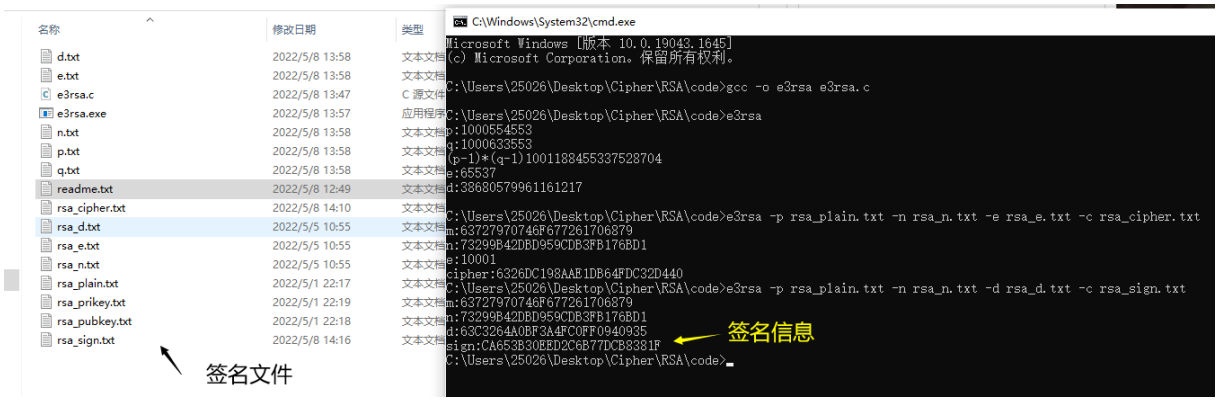


验证正确性：

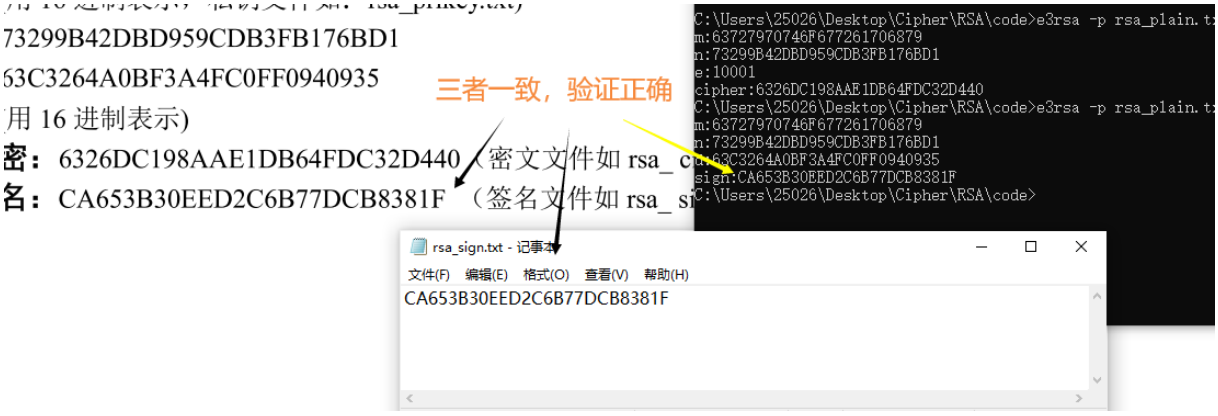


签名：

执行命令：e3rsa -p rsa_plain.txt -n rsa_n.txt -d rsa_d.txt -c rsa_sign.txt



验证正确性：



以上可以看到密钥生成，加密签名都没有问题。但是在大素数随机生成这里我没有做的很好，其实这里的随机只是伪随机，而且大素数的范围始终在

1000000001~1001990001，虽然大素数是大于 10^10 的，但总体范围不大。