



北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

网络空间安全导论

第五章实验报告

基于 Paillier 算法的匿名电子投票流 程实现

目录

| | |
|--|-----------|
| 1 课程实验原理及要求 | 2 |
| 1.1 实验原理 | 2 |
| 1.1.1 Paillier算法简介 | 2 |
| 1.1.2 Paillier算法简化 | 2 |
| 1.2 实验要求 | 3 |
| 1.2.1 实验描述 | 3 |
| 1.2.2 实验分析 | 3 |
| 2 实验环境 | 3 |
| 3 实验步骤 | 3 |
| 3.1 设计程序 | 3 |
| 3.1.1 Paillier程序设计 | 3 |
| 3.1.2 test_Paillier程序设计 | 8 |
| 3.1.3 test_Paillier_plus程序设计 | 9 |
| 3.1.4 Vote_Paillier程序设计 | 9 |
| 3.2 运行程序 | 13 |
| 3.2.1 test_Paillier程序运行展示 | 13 |
| 3.2.2 test_Paillier_plus程序运行展示 | 15 |
| 3.2.3 Vote_Paillier程序运行展示 | 17 |
| 4 总结 | 19 |
| 5 参考文献 | 19 |

1 课程实验原理及要求

1.1 实验原理

1.1.1 Paillier算法简介

密钥生成:

1.随机选择两个大素数 p, q 满足 $\gcd(pq, (p-1)(q-1)) = 1$, 且满足 p, q 长度相等

2.计算 $n = pq$ 以及 $\lambda = \text{lcm}(p-1, q-1)$, 这里 lcm 表示最小公倍数, $|n|$ 为 n 的比特长度

3.随机选择整数 $g \leftarrow Z_{n^2}^*$

4.定义 L 函数: $L(x) = \frac{x-1}{n}$, 计算 $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$

5.公钥 $\{n, g\}$, 私钥 $\{\lambda, \mu\}$

加密:

1.输入明文消息 m , 满足 $0 \leq m < n$

2.选择随机数 r , 满足 $0 \leq r < n$ 且 $r \in Z_n^*$

3.计算密文 $c = g^m r^n \bmod n^2$

解密:

1.输入密文 c , 满足 $c \in Z_{n^2}^*$

2.计算明文消息 $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$

同态加:

1.对于密文 c_1 和 c_2 , 计算 $c = c_1 \cdot c_2 \bmod n^2$

1.1.2 Paillier算法简化

在实验过程中, 发现如果完全随机选择素数以及, g, r 随机, 会造成加密解密时间比较长, 运算复杂度比较高, 在计算较大数时很不便捷, 因此查阅资料, 在实现Paillier算法时, 对参数的生成做出了以下调整:

首先, 在 p, q 等长时, 有

1. $n = p \cdot q$

2. $\lambda = (p-1)(q-1)$

3. $g = n + 1$

4. $\mu = \Phi(n)^{-1} \bmod n$, $\Phi(n) = (p-1)(q-1)$

1.2 实验要求

1.2.1 实验描述

- 1、编写Paillier算法（密钥生成、加密和解密算法）并验证其加法同态性质
- 2、模拟实现基于Paillier 算法的匿名电子投票流程，了解该算法的应用，加深对同态加密算法的认识

1.2.2 实验分析

- 1、使用python语言编写程序实现paillier算法的密钥生成、加密、解密过程，并且设计程序验证其加法同态性
- 2、使用python语言设计、编写匿名投票程序

2 实验环境

Windows11, python3.9, PyCharm 2021.3.3 (Professional Edition)

3 实验步骤

3.1 设计程序

为了完成实验，设计4个python程序，第一个程序实现Paillier算法的封装，第二个程序展示Paillier算法加解密过程，第3个程序验证Paillier算法同态加密的特性，第4个程序模拟基于Paillier算法的投票流程

3.1.1 Paillier程序设计

为了方便其他几个程序的调用，将Paillier算法封装成Paillier类，其中属性有pubKey与priKey分别表示公钥和私钥，其中有12种方法，对外提供四个方法，分别是`gen_Key()`、`encode()`、`decode()`、`plus()`

```
import math
import random
import gmpy2 as gy
from time import *

class Paillier(object):

    def __init__(self, pubKey=None, priKey=None):
        self.pubKey = pubKey
        self.priKey = priKey

    # 密钥生成算法第一步，获得p和q
    def get_pq(self):
        list=[521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599,
              601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,
              673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
              757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829,
              839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919,
              929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009,
              1013, 1019, 1021]
        pq=[]
        while True:
            p=list[random.randint(0,74)]
            q=list[random.randint(0,74)]
            if (math.gcd(p*q, (p-1)*(q-1))!=1):
                pq.append(p)
                pq.append(q)
                break
            else:
                continue
        return p,q
```

这部分代码是Paillier算法中密钥生成的第一步：获得 p, q

定义了 $get_{pq}()$ 方法， $list = [...]$ 中存放的是提前用脚本生成好的素数

利用 $while$ 结构来检测生成的参数，当满足 $gcd(pq, (p-1)(q-1)) = 1$ 时，退出循环，返回 p, q 的值

```

#密钥生成算法第二步，获得N和栏目大(x)
def calculate_N(self, p, q):
    N = p * q
    x = (p - 1) * (q - 1)
    return N, x

#密钥生成算法第三步，获得g, y
def get_random(self, N, X):
    u = 0
    while u == 0:
        g = N + 1
        u = gy.invert(X, N)
    return g, u

#构造L(x)
def L_x(self, N, x):
    z = (x - 1) // N
    return z

#加密过程
def enc_m(self, publicKey, message):
    N = publicKey[0]
    g = publicKey[1]
    r = random.randint(1, N)
    c = (g ** message % N ** 2) * (r ** N % N ** 2) % N ** 2
    return c

#解密过程
def dec_c(self, N, privateKey, crypto):
    if crypto > N ** 2:

```

这部分代码主要包含 $calculate_N()$ 、 $get_random()$ 、 $L_x()$ 、 $enc_m()$

其中 x 表示算法中的 λ ， u 表示算法中的 μ

对于构造的 L_x 函数，在编写程序调试的过程中发现，采用“/”有时会计算出错，后来上网查找资料发现python使用“/”计算结果为浮点数，导致有bug，于是改为“//”，其计算结果仍然是整数

在加密函数 $enc_m()$ 中，编写程序后发现加密计算时间很长，于是对计算密文的方式进行了化简，在一定程度上减小了算法的时间复杂度，原理为幂模运算的性质，即 $(a * b) \bmod p = (a \bmod p * b \bmod p) \bmod p$ ，化简后得到计算密文的公式为 $c = (g^m \bmod n^2 * r^n \bmod n^2) \bmod n^2$

```

#解密过程
def dec_c(self, N, priverKey, crypto):
    if crypto > N**2:
        return 0
    else:
        x=priverKey[0]
        u=priverKey[1]
        z=self.L_x(N,crypto**x % N**2)
        m=z*u % N
        return m

#同态加法验证
def com(self, message1, message2, publicKey, N):
    N = publicKey[0]
    g = publicKey[1]
    r = random.randint(1, N)
    c1 = (g ** message1 % N ** 2) * (r ** N % N ** 2) % N ** 2
    print("明文1加密后的密文为: ", c1)
    c2 = (g ** message2 % N ** 2) * (r ** N % N ** 2) % N ** 2
    print("明文2加密后的密文为: ", c2)
    c=c1*c2 % N ** 2
    print("密文为相乘后得到: ", c)
    return c

def gen_Key(self):
    p,q=self.get_pq()
    print('p = ',p)
    print('q = ',q)

    N,x=self.calculate_N(p,q)

```

这部分代码主要包含`dec_c()`、`com()`

`com()`主要功能是实现两个明文加密后，密文相乘，返回相乘后的结果，用于验证Paillier算法的同态加性性质

```
86 def gen_Key(self):
87     p,q=self.get_pq()
88     print('p = ',p)
89     print('q = ',q)
90
91     N,x=self.calculate_N(p,q)
92     print("N = ",N)
93     print("x = ",x)
94     g,u=self.get_random(N,x)
95     print("g = ",g)
96     print("u = ",u)
97
98     self.pubKey=[N,g]
99     self.priKey=[x,u]
100    print("publicKey = ",self.pubKey)
101    print("priKey = ",self.priKey)
102
103    #加密
104    def encode(self,message):
105        return self.enc_m(self.pubKey,message)
106
107    #解密
108    def decode(self,message):
109        return self.dec_c(self.pubKey[0],self.priKey,message)
110
111    #相加
112    def plus(self,message1,message2):
113        return self.com(message1,message2,self.pubKey,self.pubKey[0])
114
```

这部分代码主要包含`gen_Key()`、`encode()`、`decode()`、`plus()`

`gen_Key()`实现密钥生成，`encode()`实现加密，`decode()`实现解密，`plus()`实现对两个明文分别加密后计算密文的乘积

3.1.2 test_Paillier程序设计

```
1 from Paillier import Paillier
2 from time import *
3
4 if __name__=="__main__":
5     print("-----")
6     pa = Paillier()
7     print("加解密参数如下:")
8     pa.gen_Key()
9     print("-----")
10    message=eval(input("请输入需要加密的明文: "))
11    Enc_begin_time=time()
12    print("加密前明文: ",message)
13    c=pa.encode(message)
14    Enc_end_time=time()
15    Enc_time=Enc_end_time-Enc_begin_time
16    print("加密后密文: ",c)
17    print("加密运行时间: ",Enc_time)
18    print("-----")
19    Dec_begin_time=time()
20    print("需要解密的密文: ",c)
21    m=pa.decode(c)
22    Dec_end_time=time()
23    Dec_time=Dec_end_time-Dec_begin_time
24    print("解密后的明文: ",m)
25    print("解密运行时间: ",Dec_time)
26
```

这个程序用于测试Paillier算法的密钥生成、加密解密过程，通过命令台输入参数，引入Paillier.py中的Paillier类实现Paillier加解密过程，其中引入了time库，来查看加解密的计算时间

3.1.3 test_Paillier_plus程序设计

```

1  from Paillier import Paillier
2  import time
3
4  if __name__ == "__main__":
5      print("-----")
6      print("加解密参数如下：")
7      pa=Paillier()
8      pa.gen_Key()
9      print("-----")
10     m1=eval(input("请输入需要加密的明文1："))
11     m2=eval(input("请输入需要加密的明文2："))
12     print("-----")
13     c=pa.plus(m1,m2)
14     m=pa.decode(c)
15     print("解密后明文为：",m)

```

这个程序用于验证Paillier算法的同态加法，引入Paillier.py中的Paillier类，通过命令行输入变量，来查看密文相乘后，解密出来的明文是否等于原文之和

3.1.4 Vote_Paillier程序设计

程序流程图如下：



编写程序模拟投票流程，从程序流程图可以发现主要有3方参与，在程序中，考虑用3个不同的函数来模拟投票方、计算方、公布方，在主函数中对其依次调用，实现模拟投票的流程

```

1  from Paillier import Paillier
2
3
4  #投票工作：每个elector对每个candidate进行投票，然后将每个elector的投票结果进行加密，返回chiper_lists
5  def vote(can,ele):
6      chiper_lists=[]
7      for i in range(1,ele+1):
8          chiper_list = []
9
10         print("-----Elector{},Please vote-----".format(i))
11         for j in range(1,can+1):
12             chiper_list.append(pa.encode(eval(input("Candidate{}:".format(j)))))
13         print("Elector{},your result is : ".format(i),chiper_list)
14         chiper_lists.append(chiper_list)
15
16     return chiper_lists
17
18 #计数方主要的工作：获得vote的结果，然后运算其乘积，返回一个候选人得分密文列表
19 def count(chiper_lists,can,ele,N):
20     chiper_list=[]
21     for i in range(can):
22         chiper_list.append(1)
23     print('-----')
24     print("Please wait for a moment, the cpu is computing.")
25     for i in chiper_lists:
26         count=0
27         for j in i:
28             chiper_list[count]=(chiper_list[count] % N**2) * (j % N ** 2) % N**2
29             count=count+1
30
31     return chiper_list

```

这部分的代码主要包括`vote()`、`count()`，分别用来模拟投票方与计算方，引入编写的Paillier.py中的Paillier类

其中，主函数向`vote()`中传递候选人数`can`以及投票人数`ele`，随后在`for`循环中依次进行投票，将每个人的结果保存在列表里(`chiper_lists`)，该列表的每个元素是不同投票人的投票结果（经过加密后的），不同投票人的结果也是一个列表(`chiper_list`)

`count()`函数，接受从主函数传递过来的所有投票结果列表(`chiper_lists`)以及其他必要参数，对投票结果进行计算聚合，最后返回密文状态下的候选人的最终票数结果列表(`chiper_list`)

```
32
33     #公布者：拿到count的密文列表对其解密，输出候选人得分
34     def theory(chiper_list):
35         result=[]
36         for i in chiper_list:
37             result.append(pa.decode(i))
38
39         return result
40
41     #对结果进行排序
42     def sort(result):
43         index=[]
44         for i in range(0,len(result)):
45             index.append(i)
46
47         #冒泡排序
48         for i in range(0,len(result)):
49             for j in range(i,len(result)):
50                 if result[j] > result[i]:
51                     t=index[i]
52                     index[i]=index[j]
53                     index[j]=t
54                 else:
55                     continue
56
57         return index
58
```

这部分代码主要包括`theory()`、`sort()`，用来模拟公布方以及对结果的排序

`theory()`函数将传递过来的密文状态下的投票结果`chiper_list` 进行解密，最后返回明文状态下的投票结果`result`

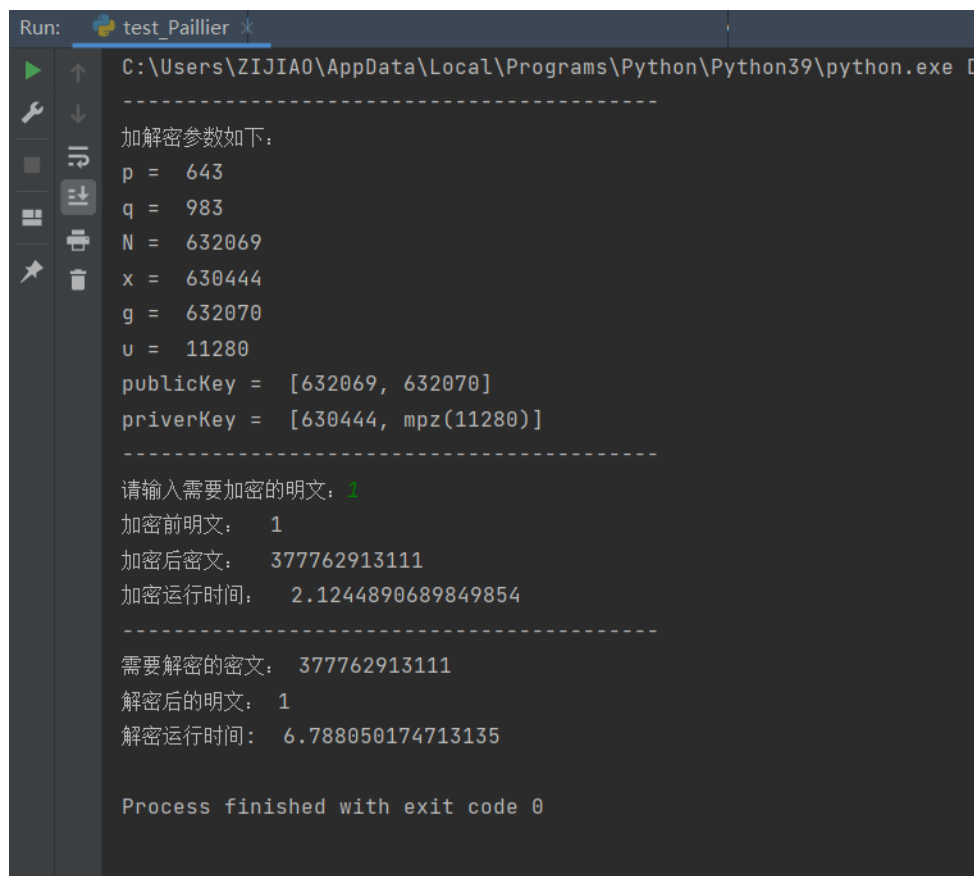
`sort()`函数对明文结果从大到小进行排序，返回候选人排名的索引`Index`

```
59 ▶ if __name__ == "__main__":
60     candidate=eval(input("Please input the number of candidates:"))
61     electors=eval(input("Please input the number of electors:"))
62
63     #System init
64     pa=Paillier()
65     pa.gen_Key()
66     publicKey=pa.pubKey
67     chiper_lists=vote(candidate,electors)
68
69     chiper_list=count(chiper_lists,candidate,electors,publicKey[0])
70
71     result=theory(chiper_list)
72
73     print("-----result-----")
74     for i in range(len(result)):
75         print("Candidate{}:".format(i+1),result[i])
76
77     print("-----sorted-----")
78
79     index=[]
80     index=sort(result)
81
82     for i in index:
83         print("Candidate{}:".format(i+1),result[i])
84     print(index)
```

这部分代码是主函数的代码部分，首先获得候选人以及选举人的数量，然后初始化，生成密钥，接着依次调用`vote()`、`count()`、`theory()`、`sort()`，并且对结果进行展示

3.2 运行程序

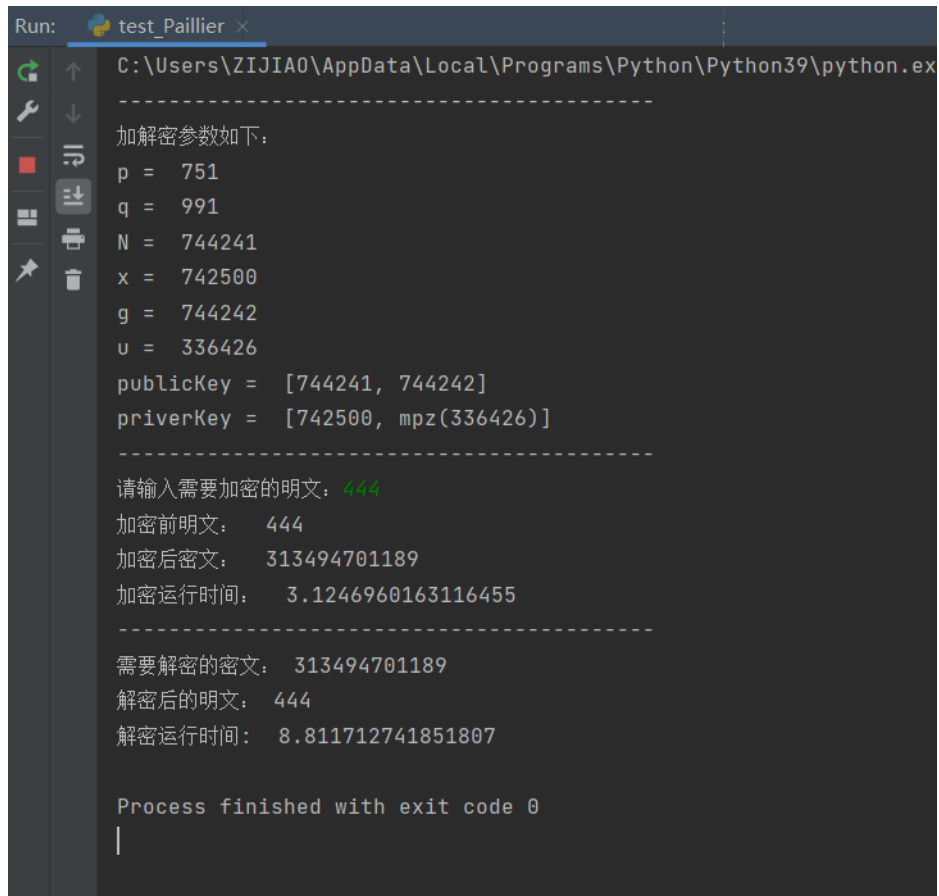
3.2.1 test_Paillier程序运行展示



The screenshot shows a terminal window titled "Run: test_Paillier". The command executed is "C:\Users\ZIJIA0\AppData\Local\Programs\Python\Python39\python.exe D". The output of the program is as follows:

```
-----  
加解密参数如下:  
p = 643  
q = 983  
N = 632069  
x = 630444  
g = 632070  
u = 11280  
publicKey = [632069, 632070]  
priverKey = [630444, mpz(11280)]  
-----  
请输入需要加密的明文: 1  
加密前明文: 1  
加密后密文: 377762913111  
加密运行时间: 2.1244890689849854  
-----  
需要解密的密文: 377762913111  
解密后的明文: 1  
解密运行时间: 6.788050174713135  
  
Process finished with exit code 0
```

输入明文为1时程序运行结果展示

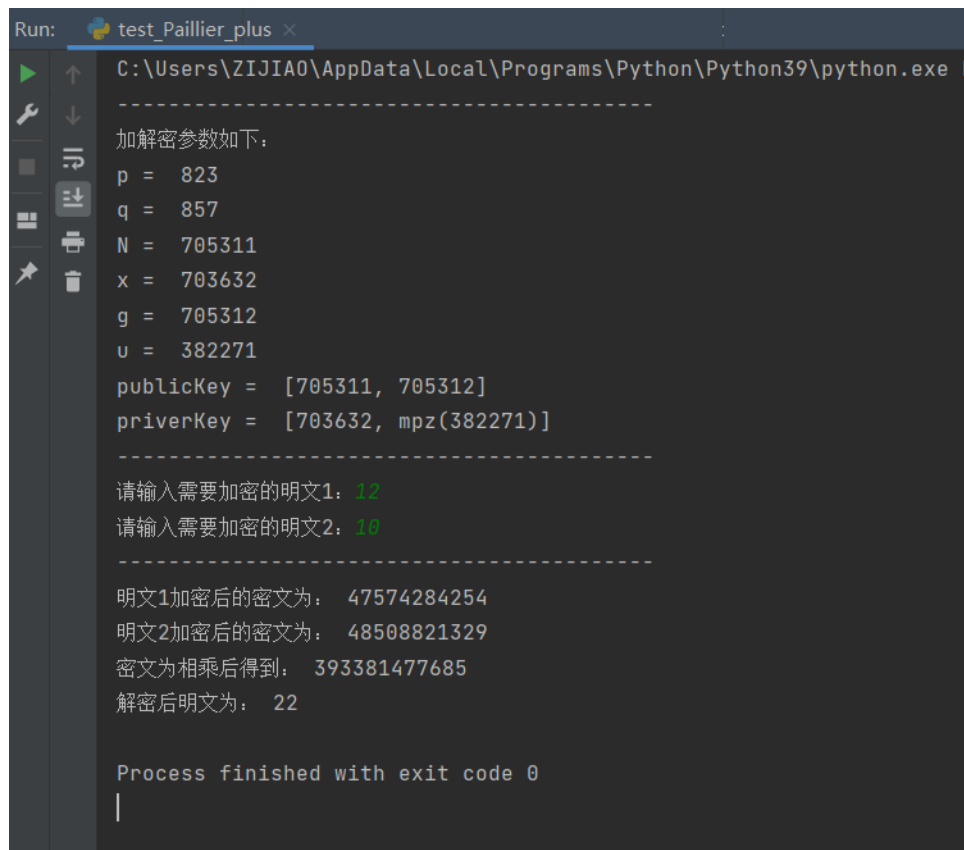


```
Run: test_Paillier x
C:\Users\ZIJIA0\AppData\Local\Programs\Python\Python39\python.exe
-----
加解密参数如下:
p = 751
q = 991
N = 744241
x = 742500
g = 744242
u = 336426
publicKey = [744241, 744242]
priverKey = [742500, mpz(336426)]
-----
请输入需要加密的明文: 444
加密前明文: 444
加密后密文: 313494701189
加密运行时间: 3.1246960163116455
-----
需要解密的密文: 313494701189
解密后的明文: 444
解密运行时间: 8.811712741851807

Process finished with exit code 0
|
```

输入明文为444时程序运行结果展示

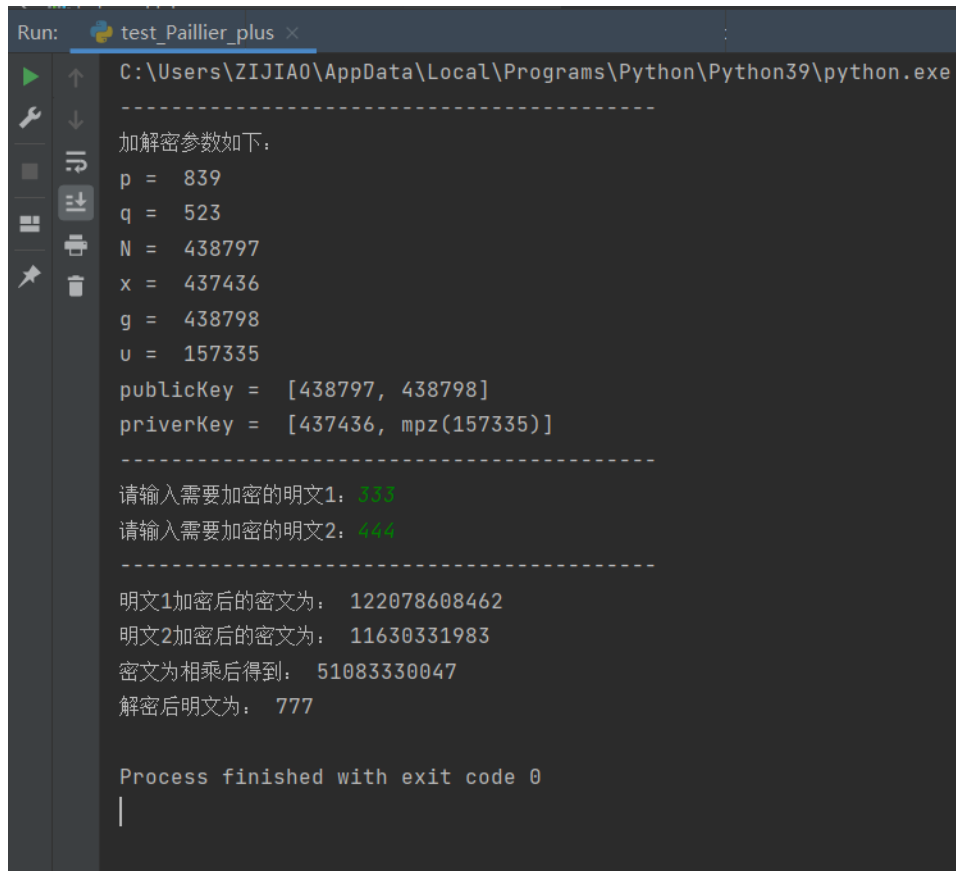
3.2.2 test_Paillier_plus程序运行展示



```
Run: test_Paillier_plus x
C:\Users\ZIJIAO\AppData\Local\Programs\Python\Python39\python.exe D
-----
加解密参数如下:
p = 823
q = 857
N = 705311
x = 703632
g = 705312
u = 382271
publicKey = [705311, 705312]
priverKey = [703632, mpz(382271)]
-----
请输入需要加密的明文1: 12
请输入需要加密的明文2: 10
-----
明文1加密后的密文为: 47574284254
明文2加密后的密文为: 48508821329
密文为相乘后得到: 393381477685
解密后明文为: 22

Process finished with exit code 0
|
```

输入的两个明文分别为12、10时程序运行结果展示



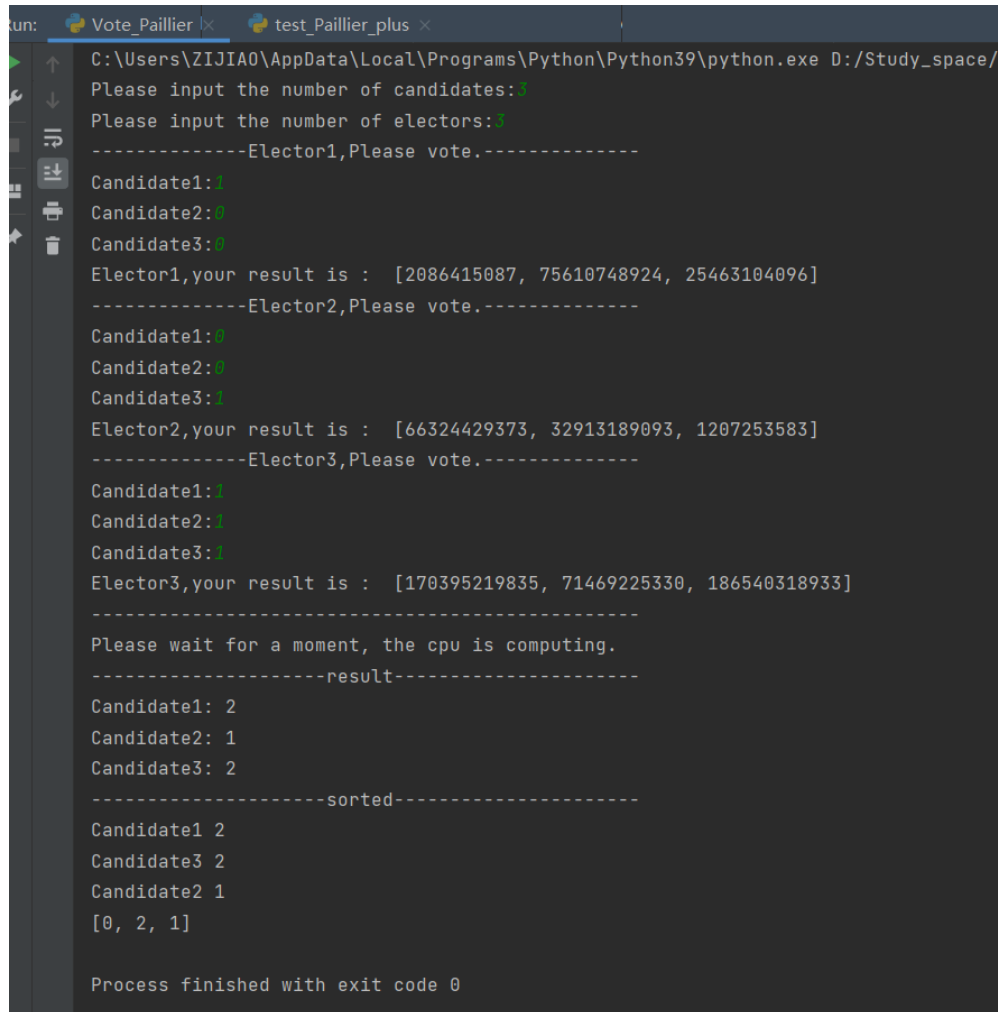
The screenshot shows a terminal window titled 'test_Paillier_plus'. The command prompt is 'C:\Users\ZIJIA0\AppData\Local\Programs\Python\Python39\python.exe'. The output displays the decryption parameters for a Paillier cryptosystem, followed by the encryption of two plaintexts (333 and 444) and the decryption of their product. The process ends with 'Process finished with exit code 0'.

```
Run: test_Paillier_plus ×
C:\Users\ZIJIA0\AppData\Local\Programs\Python\Python39\python.exe
-----
加解密参数如下:
p = 839
q = 523
N = 438797
x = 437436
g = 438798
u = 157335
publicKey = [438797, 438798]
priverKey = [437436, mpz(157335)]
-----
请输入需要加密的明文1: 333
请输入需要加密的明文2: 444
-----
明文1加密后的密文为: 122078608462
明文2加密后的密文为: 11630331983
密文为相乘后得到: 51083330047
解密后明文为: 777

Process finished with exit code 0
|
```

输入的两个明文分别为444，333时程序运行结果展示

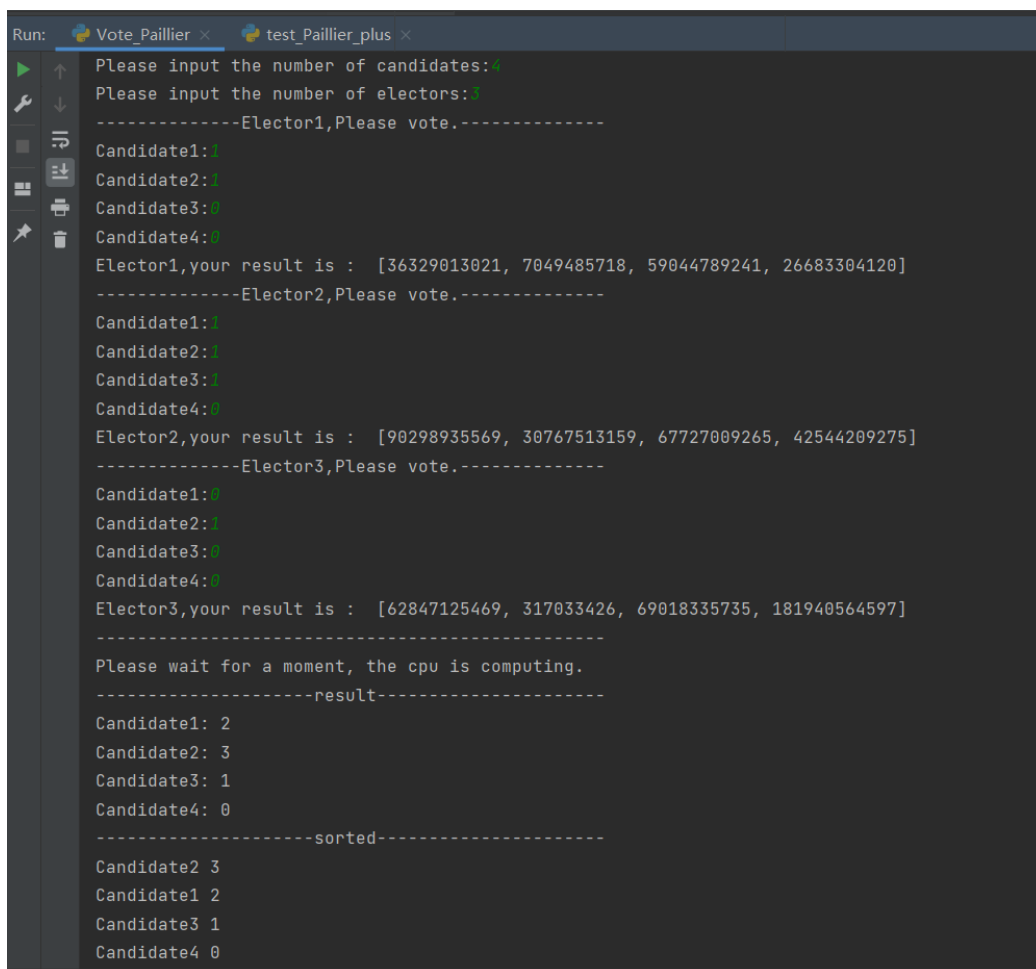
3.2.3 Vote_Paillier程序运行展示



```
run: Vote_Paillier x test_Paillier_plus x
C:\Users\ZIJIA0\AppData\Local\Programs\Python\Python39\python.exe D:/Study_space/
Please input the number of candidates:3
Please input the number of electors:3
-----Elector1,Please vote.-----
Candidate1:1
Candidate2:0
Candidate3:0
Elector1,your result is : [2086415087, 75610748924, 25463104096]
-----Elector2,Please vote.-----
Candidate1:0
Candidate2:0
Candidate3:1
Elector2,your result is : [66324429373, 32913189093, 1207253583]
-----Elector3,Please vote.-----
Candidate1:1
Candidate2:1
Candidate3:1
Elector3,your result is : [170395219835, 71469225330, 186540318933]
-----
Please wait for a moment, the cpu is computing.
-----result-----
Candidate1: 2
Candidate2: 1
Candidate3: 2
-----sorted-----
Candidate1 2
Candidate3 2
Candidate2 1
[0, 2, 1]

Process finished with exit code 0
```

输入3个候选人，3个选举者的运行情况



```
Run: Vote_Paillier x test_Paillier_plus x
Please input the number of candidates:4
Please input the number of electors:3
-----Elector1,Please vote.-----
Candidate1:1
Candidate2:1
Candidate3:0
Candidate4:0
Elector1,your result is : [36329013021, 7049485718, 59044789241, 26683304120]
-----Elector2,Please vote.-----
Candidate1:1
Candidate2:1
Candidate3:1
Candidate4:0
Elector2,your result is : [90298935569, 30767513159, 67727009265, 42544209275]
-----Elector3,Please vote.-----
Candidate1:0
Candidate2:1
Candidate3:0
Candidate4:0
Elector3,your result is : [62847125469, 317033426, 69018335735, 181940564597]
-----
Please wait for a moment, the cpu is computing.
-----result-----
Candidate1: 2
Candidate2: 3
Candidate3: 1
Candidate4: 0
-----sorted-----
Candidate2 3
Candidate1 2
Candidate3 1
Candidate4 0
```

输入4个候选人，3个选举者的运行情况

4 总结

这学期我选到了张川老师的《数据安全与治理》课程，其中同样学到了Paillier算法，而且期末考试还考了Paillier算法的题，两门课程内容的交汇，让我感觉很幸运。将所学的知识转换成代码与程序的过程中遇到了很多问题，首先就是数幂运算导致数很大，自己编写的程序加解密速度很慢，而且会有bug，但是所有的代码都是自己一点一点实现的，有非常高的成就感，同时也对Paillier算法有了更加深入的了解，感叹密码学家的才华与能力，通过这次实验我也意识到了自己的不足，继续努力，还有很多内容要学，还有很多东西不懂。

我自己编写的代码中存在一个问题，就是当数非常大的时候，程序会运行错误，可能是在Paillier算法封装的方法中没有大量使用库函数，导致运算效率很低，运算速度比较慢，解决方法是使用python的gmpy2库中的函数来替换自己编写的简单求模、求幂的运算，或者使用一些优化效果更明显的算法。如Paillier-DJN优化方案，或者中国剩余定理。

5 参考文献

[1] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]//International conference on the theory and applications of cryptographic techniques. Springer, Berlin, Heidelberg, 1999: 223-238.

[2]<https://developer.aliyun.com/article/793131>

[3]部分算法原理内容来源于CSDN、知乎、百度百科等平台。