

**《数据安全》课程实验报告**

**实验2：半同态加密应用实践**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2112060

姓 名 孙蕗

1. **实验要求**
2. 基于Pailier算法实现隐私信息获取:从服务器给定的m个消息中获取其中一个，不得向服务器泄露获取了哪一个消息，同时客户端能完成获取消息的解密。
3. 扩展实验:有能力的同学可以在客户端保存对称密钥k，在服务器端存储m个用对称密钥k加密的密文，通过隐私信息获取方法得到指定密文后能解密得到对应的明文。
4. **实验原理**
5. Paillier算法描述

Paillier方案满足加密方案的标准安全定义：语义安全，即在选择明文攻击下的密文的不可区分性。

1. 密钥生成

· 随机选择两个质数p和q,尽可能地保证p和q的长度接近或相等(安全性高);

·计算n=pq和λ=lcm(p-1,q-1),其中Icm表示最小公倍数；

·随机选择,考虑计算性能优化，通常会选择g=n+1;

·计算,其中;

·公钥为(n,g);

·私钥为(λ,μ)

1. 加密算法

对于任意明文消息,任意选择一个随机数,计算得到密文c:

注意：密文c要比明文m长度要长。

1. 解密算法

对于密文,计算得到明文m:

1. 半同态加密虽然还不能同时支持加法和乘法运算，不能支持任意的计算，但是因为其与全同态相比，具有较高性能，因此，仍然具有极为广泛的应用场景，且在现实应用中起到了中重要的作用。一类典型的应用体现在隐私保护的数据聚合上。由于加法同态加密可以在密文上直接执行加和操作，不泄露明文，在到多方协作的统计场景中，可完成安全的统计求和的功能。在加密数据库SQL查询场景，在数据库不可信的情况下，可以通过部署协议和代理来保护请求者的查询隐私。其中，PHE可以用来完成安全数据求和、均值的查询。
2. 隐私信息获取：基于Paillier协议进行设计。对Paillier的标量乘的性质进行扩展，数值“0”的密文与任意数值的标量乘也是0；数值“1”的密文与任意数值的标量乘将是数值本身。利用这个特性，可进行如下设计

|  |
| --- |
| 服务器端：产生数据列表data\_list={m1,m2,...,mn}  客户端：  ·设置要选择的数据位置为pos  ·生成选择向量select\_list={0,...,1...,0},其中，仅有pos的位置为1  ·生成密文向量enc\_list={E(0),...,E(1)...,E(0)}  ·发送密文向量enc\_list给服务器  服务器端：  ·将数据与对应的向量相乘后累加得到密文  c=m1\*enc\_list[1]...+mn\*enc\_list[n]  ·返回密文c给客户端  客户端：解密密文c得到想要的结果 |

1. 3DES

扩展实验使用3DES完成。

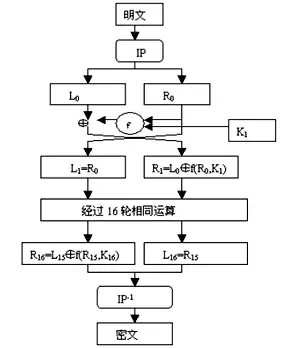
DES使用56位的密钥和64位的明文块进行加密。DES算法的分组大小是64位，因此，如果需要加密的明文长度不足64位，需要进行填充；如果明文长度 超过64位，则需要使用分组模式进行分组加密。

DES算法的分组大小是64位，由于DES算法的密钥长度只有56位，因此DES算法存在着弱点，容易受到暴力破解和差分攻击等攻击手段的威胁。因此，在实际应用中DES已经被更加安全的算法所取代，如AES算法等。

1. DES加密

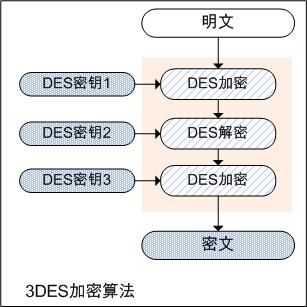
当输入了一条64位的数据之后，DES将通过以下步骤进行加密。

1. 初始置换（IP置换）：将输入的64位明文块进行置换和重新排列，生成新的64位数据块。
2. 加密轮次：DES加密算法共有16个轮次，每个轮次都包括四个步骤
3. 将64位数据块分为左右两个32位块。
4. 右侧32位块作为输入，经过扩展、异或、置换等操作生成一个48位的数据块。这个48位的数据块被称为“轮密钥”，它是根据加密算法的主密钥生成的子密钥。
5. 将左侧32位块和轮密钥进行异或运算，结果作为新的右侧32位块。
6. 将右侧32位块与原来的左侧32位块进行连接，生成一个新的64位数据块，作为下一轮的输入。
7. 末置换（FP置换）：在最后一个轮次完成后，将经过加密的数据块进行置换和重新排列，得到加密后的64位密文。



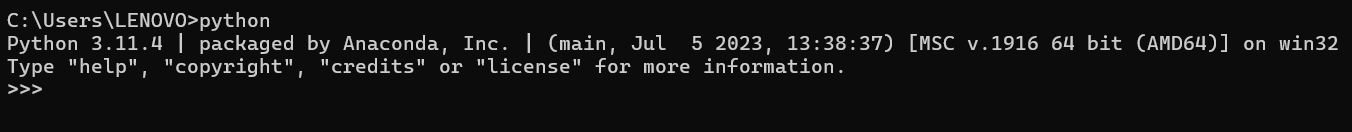
1. 3DES

3DES就是使用DES加密3次，使用3个密钥进行加解密。3DES使用了三个密钥，将DES算法的加密过程重复三次，从而大大增强了安全性。当三重密钥均相同时，前两步相互抵消，相当于仅实现了一次加密，因此可实现对普通DES加密算法的兼容。3DES的密钥长度为168位，远高于DES算法的56位密钥长度，但仍有可能被暴力破解。



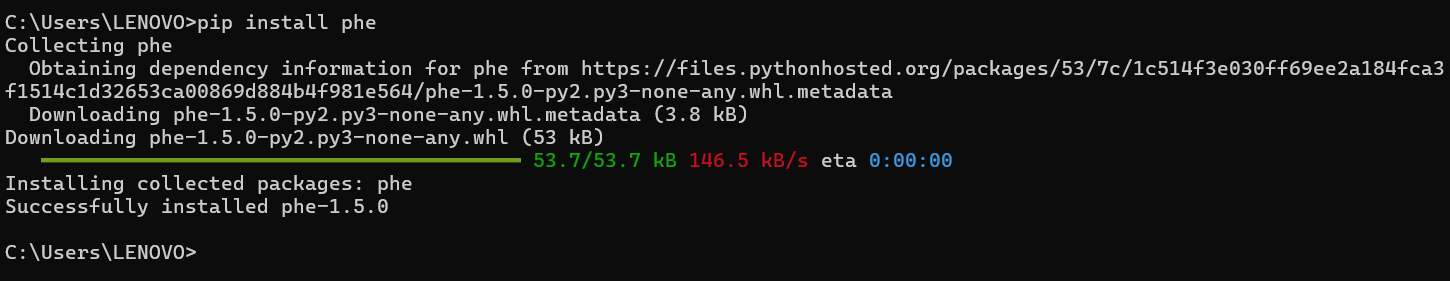
1. **实验过程**
2. **环境配置**
3. 在Windows下安装python环境，注意安装过程要勾选Add Python.exe to PATH将python程序路径加到系统的环境变量中。

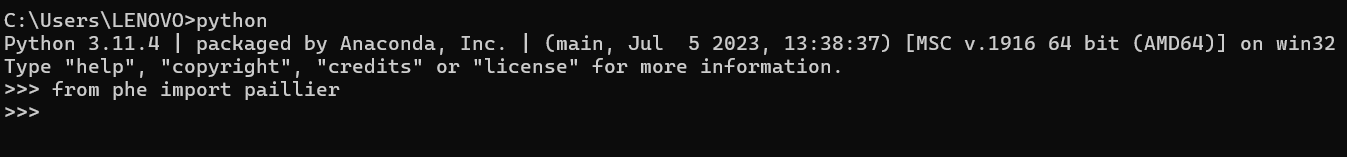
安装完毕打开控制台，输入python指令显示如下表明已经安装成功并进入python运行环境。



1. 安装phe库

输入pip install phe完成phe库的安装。安装完成后，进入python环境，输入from phe import paillier，如果不出现错误信息，说明环境安装成功

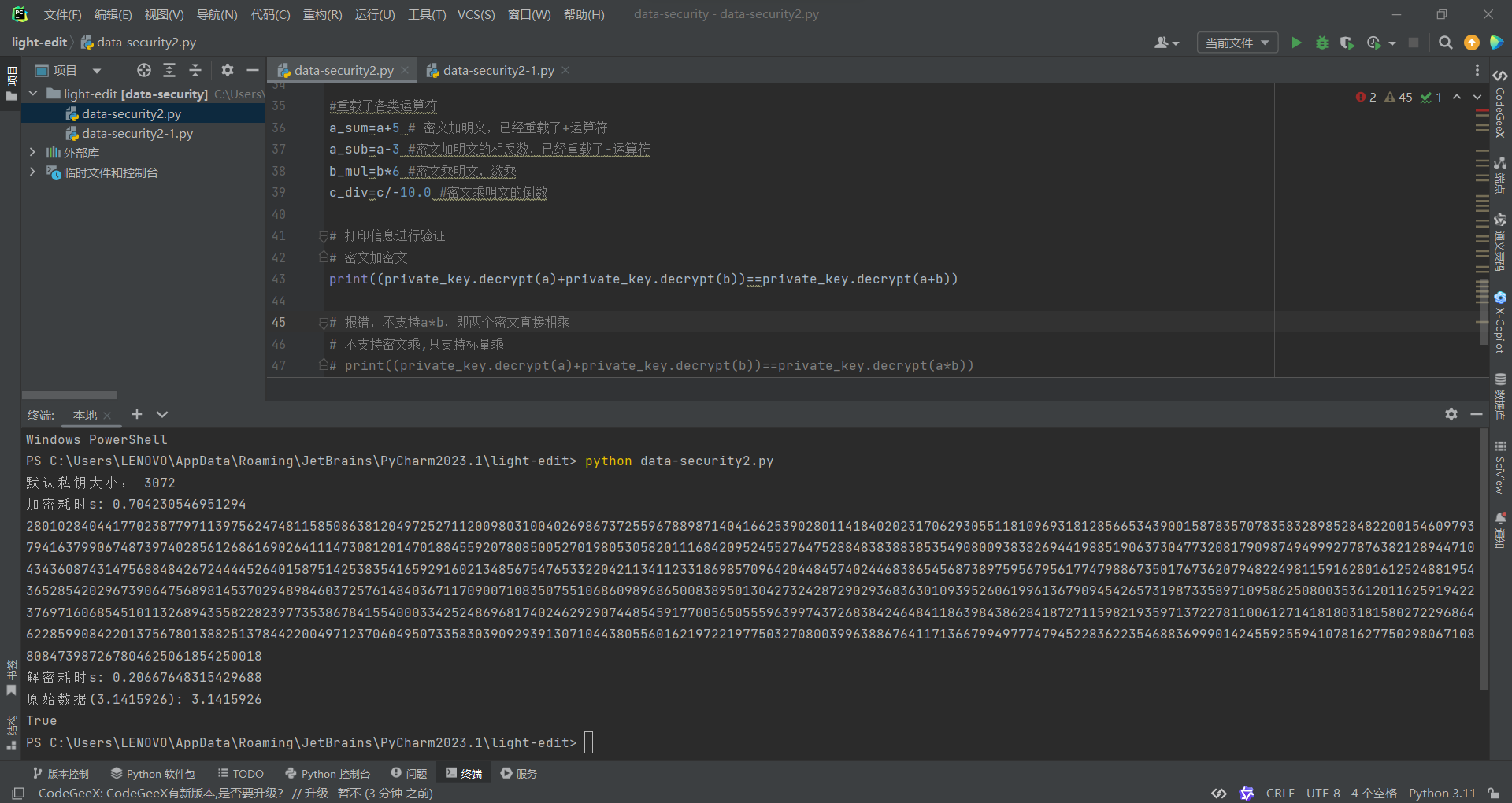




1. **基于Python的phe库完成加法和标量乘法的验证**

|  |
| --- |
| from phe import paillier #开源库  import time #做性能测试  #####################设置参数  print("默认私钥大小：",paillier.DEFAULT\_KEYSIZE)#生成公私钥  # 默认私钥大小为3072  # 生成公私钥  public\_key,private\_key =paillier.generate\_paillier\_keypair()  #测试需要加密的数据（设置3个要加密的数据）  message\_list =[3.1415926,100,-4.6e-12]  #####################加密操作  time\_start\_enc=time.time()  # 内嵌for循环  # 使用public\_key.encrypt完成加密  encrypted\_message\_list=[public\_key.encrypt(m) for m in message\_list]  time\_end\_enc=time.time()  print("加密耗时s:",time\_end\_enc-time\_start\_enc)  print("加密数据(3,1415926):",encrypted\_message\_list[0].ciphertext())  ####################解密操作  time\_start\_dec=time.time()  # 内嵌for循环  decrypted\_message\_list=[private\_key.decrypt(c) for c in encrypted\_message\_list]  time\_end\_dec=time.time()  # 使用private\_key.decrypt完成解密  print("解密耗时s:",time\_end\_dec-time\_start\_dec)  print("原始数据(3.1415926):",decrypted\_message\_list[0])  ####################测试加法和乘法同态  a,b,c=encrypted\_message\_list #a,b,c分别为对应密文  #重载了各类运算符  a\_sum=a+5 # 密文加明文，已经重载了+运算符  a\_sub=a-3 #密文加明文的相反数，已经重载了-运算符  b\_mul=b\*6 #密文乘明文，数乘  c\_div=c/-10.0 #密文乘明文的倒数  # 打印信息进行验证  # 密文加密文  print((private\_key.decrypt(a)+private\_key.decrypt(b))==private\_key.decrpt(a+b))  # 报错，不支持a\*b，即两个密文直接相乘  # 不支持密文乘,只支持标量乘  #print((private\_key.deerypt(a)+private\_key.decrypt(b))==private\_key.decrypt(a\*b)) |

运行后截图如下：



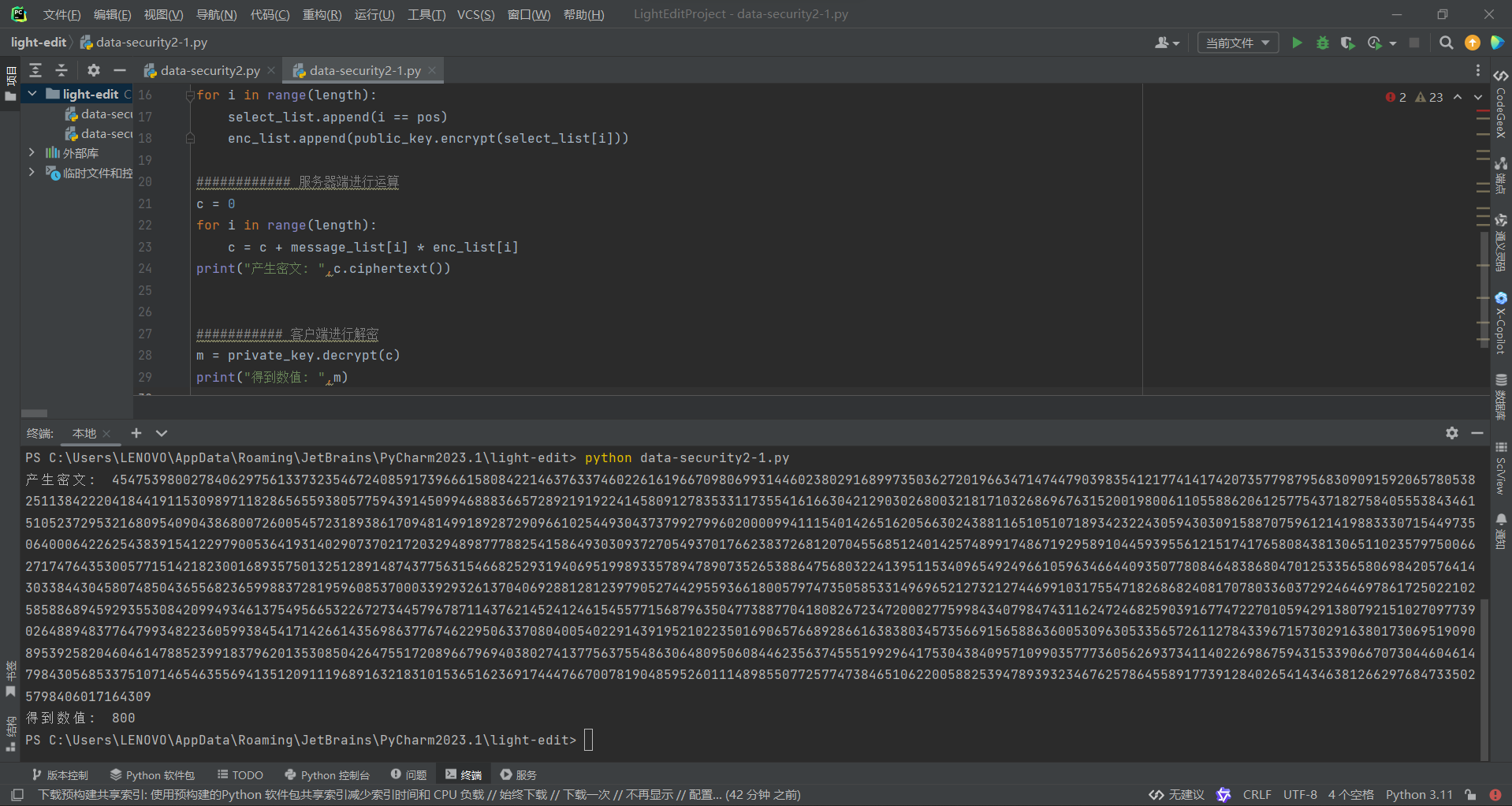
首先定义了需要加密的数据，并自动生成公钥与私钥，默认私钥大小为3072。在完成数据测试后，首先通过公钥对数据3.1415926进行加密，并显示加密耗时时间与加密数据内容。完成加密操作后根据私钥对密文进行解密。

解密耗时相对加密耗时更短，与Paillier的性质相符，即在加密过程中需要做两次指数运算，而解密只需要一次。

1. **利用Python phe库完成隐私信息获取实验**

|  |
| --- |
| from phe import paillier # 开源库  import random # 选择随机数  ################### 设置参数  # 服务器端保存的数值  message\_list =[100,200,300,400,500,600,700,800,900,1000]  length =len(message\_list)  # 客户端生成公私钥，选择读取的位置  public\_key,private\_key = paillier.generate\_paillier\_keypair()  pos = random.randint(0,length-1)  ##########客户端生成密文选择向量  select\_list=[]  enc\_list=[]  for i in range(length):  select\_list.append(i == pos)  enc\_list.append(public\_key.encrypt(select\_list[i]))  ############ 服务器端进行运算  c = 0  for i in range(length):  c = c + message\_list[i] \* enc\_list[i]  print("产生密文: ",c.ciphertext())  ########### 客户端进行解密  m = private\_key.decrypt(c)  print("得到数值: ",m) |

运行后截图如下



首先设置了服务器端保存的数值列表message\_list,列表长度length，客户端生成公钥私钥并随机选择了一个要读取的位置。然后客户端生成一个密文选择向量select\_list={0,...,1...,0},其中，仅有pos的位置为1。

客户端遍历数值列表message\_list中的每一个元素，如果与pos相等，则对应的密文选择向量为1否则为0。然后使用公钥对密文选择向量进行加密，存储在密文向量列表enc\_list中。

服务器端进行加密求和的运算，将数据与对应的向量相乘后累加得到密文

c=m1\*enc\_list[1]...+mn\*enc\_list[n]，返回密文c给客户端。

客户端解密密文c得到想要的结果。

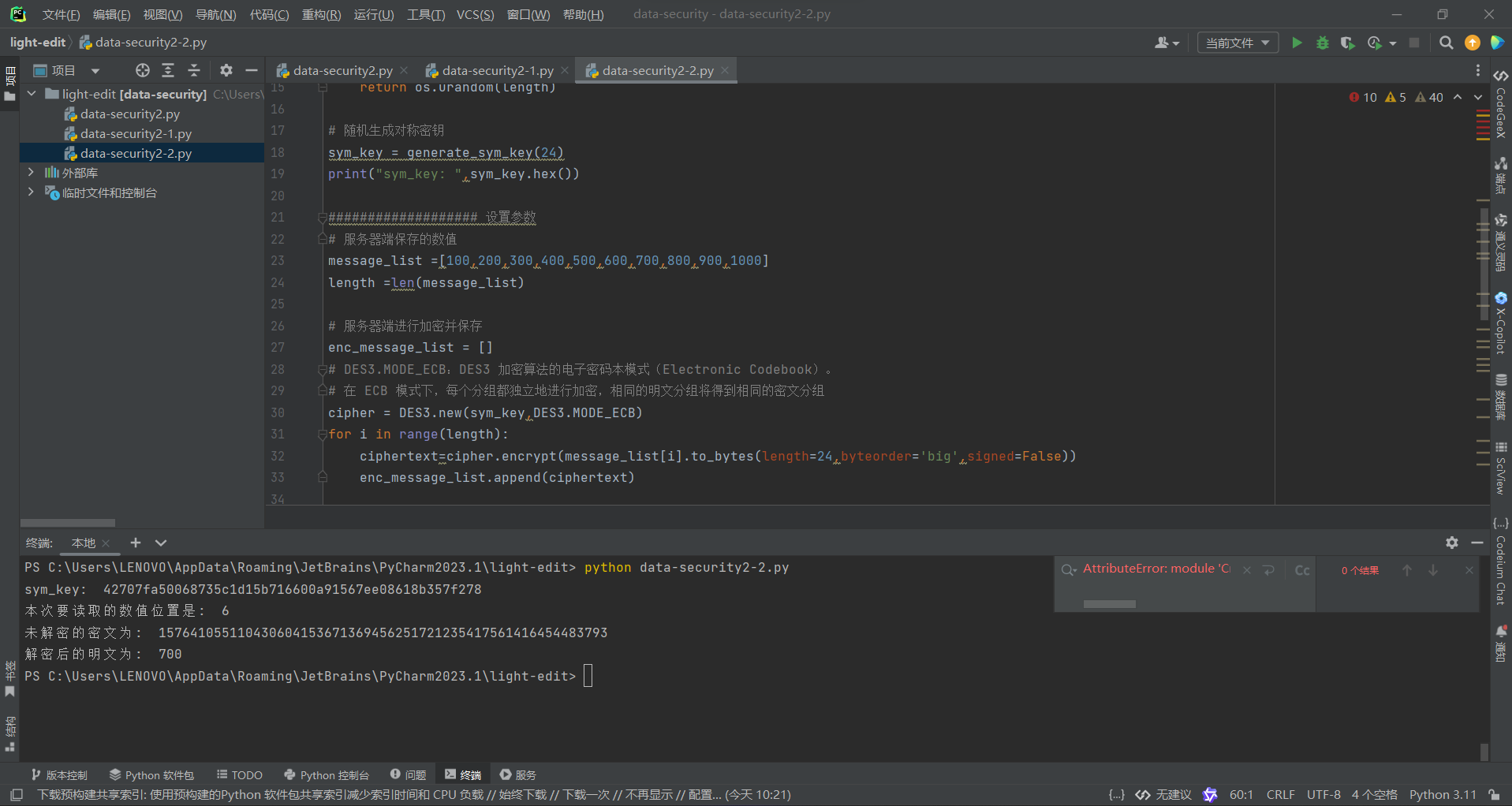
1. **扩展实验**

|  |
| --- |
| from phe import paillier #开源库  import time #做性能测试  from Crypto.Cipher import DES3,AES  from Crypto.Random import get\_random\_bytes  from Crypto.Util.Padding import pad, unpad  import hashlib  import random  import os  # 在客户端保存对称密钥k，  # 在服务器端存储m个用对称密钥k加密的密文，  # 通过隐私信息获取方法得到指定密文后能解密得到对应的明文。  def generate\_sym\_key(length):  return os.urandom(length)  # 随机生成对称密钥  sym\_key = generate\_sym\_key(24)  print("sym\_key: ",sym\_key.hex())  ################### 设置参数  # 服务器端保存的数值  message\_list =[100,200,300,400,500,600,700,800,900,1000]  length =len(message\_list)  # 服务器端进行加密并保存  enc\_message\_list = []  # DES3.MODE\_ECB：DES3 加密算法的电子密码本模式（Electronic Codebook）。  # 在 ECB 模式下，每个分组都独立地进行加密，相同的明文分组将得到相同的密文分组  cipher = DES3.new(sym\_key,DES3.MODE\_ECB)  for i in range(length):  ciphertext=cipher.encrypt(message\_list[i].to\_bytes(length=24,byteorder='big',signed=False))  enc\_message\_list.append(ciphertext)  # 客户端生成公私钥，选择读取的位置  public\_key,private\_key = paillier.generate\_paillier\_keypair()  # 随机选择一个位置读取  pos = random.randint(0,length-1)  print("本次要读取的数值位置是: ",pos)  ##########客户端生成密文选择向量  select\_list=[]  enc\_list=[]  for i in range(length):  select\_list.append(i == pos)  enc\_list.append(public\_key.encrypt(select\_list[i]))  ############ 服务器端进行运算  c = 0  for i in range(length):  trans\_message\_list=int().from\_bytes(enc\_message\_list[i],byteorder='big',signed=False)  c = c + trans\_message\_list \* enc\_list[i]  # print("产生密文: ",c.ciphertext())  ########### 客户端进行解密  m = private\_key.decrypt(c)  print("未解密的密文为: ",m)  m = cipher.decrypt(m.to\_bytes(24, 'big', signed=True))  print("解密后的明文为: ",int().from\_bytes(m,byteorder='big',signed=True)) |

具体思路：

|  |
| --- |
| 客户端  · 设置对称密钥k  服务器端：  · 产生数据列表data\_list={m1,m2,...,mn}  · 存储用对称密钥k加密的密文  客户端：  · 生成公钥私钥  · 设置要选择的数据位置为pos  · 生成选择向量select\_list={0,...,1...,0},其中，仅有pos的位置为1  · 生成密文向量enc\_list={E(0),...,E(1)...,E(0)}  · 发送密文向量enc\_list给服务器  服务器端：  · 把byte转为int进行计算  · 将数据与对应的向量相乘后累加得到密文  c=trans\_message\_list[1]\*enc\_list[1]+...+[n]\*enc\_list[n]  ·返回密文c给客户端  客户端：通过DES，获取的byte解密得到int |

运行结果如下：



选取位置6，得到密文及解密后数值700

1. **实验心得体会**

本次实验，首先在给出的参考代码中熟悉phe的用法，将课上学习的理论与课下实验的实际操作相结合，并进一步掌握半同态加密的基本原理。拓展部分通过查找相关资料，学习对称加密算法的相关知识，从中深化对密码学的认识。