

# 《数据安全》实验报告

姓名：穆禹宸 学号：2012026 班级：信息安全法学、双学位班

## 实验名称

半同态加密应用实践

## 实验要求

### 基础实验

基于Paillier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

### 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。

## 实验过程

### 环境配置

实验使用python语言来完成。由于python环境我已经装好了，因此直接进行后续工作。

实验需要安装 `phe` 库，以使用 `Paillier` 算法。

直接使用如下代码进行安装：

```
1 pip install phe
```

然后验证安装是否成功，在命令行之中进入python环境，然后输入如下命令：

```
1 from phe import paillier
```

这时我们会看到以下验证结果：

```
C:\ 命令提示符 - python
Microsoft Windows [版本 10.0.22000.1455]
(c) Microsoft Corporation。保留所有权利。

C:\Users\20120>python
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> from phe import paillier
>>> _
```

然后再查看一下 `phe` 库的版本，使用如下命令：

```
1 import phe
2 print(phe.__version__)
```

然后得到如下结果：

```
C:\Users\20120>python
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> import phe
>>> print(phe.__version__)
1.5.0
>>>
```

这里显示我们的phe库是1.5.0版本，这已经是现在最新的版本了。需要注意的是，1.4版本与1.5版本的最大不同在于对字节操作的不同，也正是因为此原因，我在后续的扩展实验中没有采取复杂的加密算法，后面我会进一步介绍。

## 基础实验

基于 Python 的 phe 库完成加法和标量乘法的验证。

这里都是一些基础操作，代码如下所示：

```
1 from phe import paillier # 开源库
2 import time # 做性能测试
3
4 ##### 设置参数
5 print("默认私钥大小:", paillier.DEFAULT_KEYSIZE)
6 #生成公私钥
```

```

7 public_key, private_key = paillier.generate_paillier_keypair()
8 # 测试需要加密的数据
9 message_list = [3.1415926,100,-4.6e-12]
10
11 ##### 加密操作
12 time_start_enc = time.time()
13 encrypted_message_list = [public_key.encrypt(m) for m in message_list]
14 time_end_enc = time.time()
15 print("加密耗时s: ",time_end_enc-time_start_enc)
16 print("加密数据 (3.1415926) :",encrypted_message_list[0].ciphertext())
17
18 ##### 解密操作
19 time_start_dec = time.time()
20 decrypted_message_list = [private_key.decrypt(c) for c in encrypted_message_list]
21 time_end_dec = time.time()
22 print("解密耗时s: ",time_end_dec-time_start_dec)
23 print("原始数据 (3.1415926) :",decrypted_message_list[0])
24
25 ##### 测试加法和乘法同态
26 a,b,c = encrypted_message_list # a,b,c分别为对应密文
27 a_sum = a + 5 # 密文加明文, 已经重载了+运算符
28 a_sub = a - 3 # 密文加明文的相反数, 已经重载了-运算符
29 b_mul = b * 6 # 密文乘明文, 数乘
30 c_div = c / -10.0 # 密文乘明文的倒数
31
32 print("a+5 密文:",a.ciphertext()) # 密文纯文本形式
33 print("a+5=",private_key.decrypt(a_sum))
34 print("a-3=",private_key.decrypt(a_sub))
35 print("b*6=",private_key.decrypt(b_mul))
36 print("c/-10.0=",private_key.decrypt(c_div))
37
38 ##密文加密文
39 print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a+b))
40 #报错, 不支持a*b, 即两个密文直接相乘
41 #print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))

```

运行之后我们得到如下的输出结果:

```

1 默认私钥大小: 3072
2 加密耗时s: 0.6568384170532227
3 加密数据 (3.1415926) : 90227437529683995199187350458922722761245812002295170901684
4 解密耗时s: 0.1875007152557373
5 原始数据 (3.1415926) : 3.1415926
6 a+5 密文: 90227437529683995199187350458922722761245812002295170901684908892717394
7 a+5= 8.1415926

```

```
8 a-3 0.141592600000000007
9 b*6= 600
10 c/-10.0= 4.6e-13
11 True
```

这都和我们在理论课上所讲的内容对应。

## 基于 Python 的 phe 库完成隐私信息获取的功能

服务器端拥有多个数值，要求客户端能基于 Paillier 实现从服务器读取一个指定的数值并正确解密，但服务器不知道所读取的是哪一个

首先，我们要基于 Paillier 协议进行设计。

对 Paillier 的标量乘的性质进行扩展，我们知道：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。

基于这个特性，我们可以如下巧妙的设计：

**服务器端：** 产生数据列表  $\text{data\_list} = \{m_1, m_2, \dots, m_n\}$

**客户端：**

- 设置要选择的数据位置为  $\text{pos}$
- 生成选择向量  $\text{select\_list} = \{0, \dots, 1, \dots, 0\}$ ，其中，仅有  $\text{pos}$  的位置为 1
- 生成密文向量  $\text{enc\_list} = \{E(0), \dots, E(1), \dots, E(0)\}$
- 发送密文向量  $\text{enc\_list}$  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  $c = m_1 * \text{enc\_list}[1] + \dots + m_n * \text{enc\_list}[n]$
- 返回密文  $c$  给客户端

**客户端：** 解密密文  $c$  得到想要的结果

以上为教材中所写内容，但不够详细，下面我将更加细致的讲解一下这个协议的目的和功能以及是如何实现的。

协议目的

这个协议的设计是为了保护客户端的隐私，即客户端不希望服务器端知道它要读取的具体数值。因此，协议中的加密操作是针对服务器端进行的，即客户端将要读取的位置加密后发送给服务器端，服务器端进行加密求和后再将结果返回给客户端，客户端再使用私钥进行解密得到要读取的数值。

这样做的好处是，客户端的隐私得到了保护，即使服务器端知道了加密后的位置，也无法得知具体要读取的数值。同时，由于加密操作是在客户端进行的，服务器端只能得到加密后的结果，无法得知具体的数值，保证了数据的隐私性和安全性。

因此，这个协议的加密操作是对服务器端保密的，目的是保护客户端的隐私。

## 整体流程

1. 服务器端保存了一些数值，客户端生成了公私钥，并随机选择了一个要读取的位置。
2. 客户端生成了一个密文选择向量，其中只有要读取的位置对应的元素为True，其他元素为False。
3. 服务器端对每个数值和密文选择向量进行乘法运算，并将结果相加得到一个密文。
4. 客户端使用私钥对密文进行解密，得到了要读取的数值。

## 具体实现

首先导入了开源库phe中的Paillier加密算法和Python自带的random库。然后，设置了一些参数，包括服务器端保存的数值列表、数值列表的长度、客户端生成的公私钥以及客户端随机选择的要读取的位置。最后，输出了要读取的位置。

```
1 from phe import paillier # 开源库
2 import random # 选择随机数
3
4 ##### 设置参数
5 # 服务器端保存的数值
6 message_list = [100,200,300,400,500,600,700,800,900,1000]
7 length = len(message_list)
8 # 客户端生成公私钥
9 public_key, private_key = paillier.generate_paillier_keypair()
10 # 客户端随机选择一个要读的位置
11 pos = random.randint(0,length-1)
12 print("要读起的数值位置为: ",pos)
```

然后生成了一个密文选择向量，其中只有要读取的位置对应的元素为True，其他元素为False。这里就对应了我们前面所说的数值“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。具体地，代码使用了一个for循环，遍历了数值列表中的每个元素。对于每个元素，如果它的下标等于要读取的位置，则将对应的密文选择向量元素设为True，否则设为False。然后，使用公钥对每个密文选择向量元素进行加密，并将加密结果存储在enc\_list列表中。

```
1 ##### 客户端生成密文选择向量
2 select_list=[]
3 enc_list=[]
4 for i in range(length):
5     select_list.append( i == pos )
6     enc_list.append( public_key.encrypt(select_list[i]) )
```

然后在服务器端进行了加密求和的运算。这段加密求和过程比较简单，对于每个元素，将其与对应的密文选择向量元素相乘，并将结果累加到变量c中。最后，输出加密求和的结果。

```
1 ##### 服务器端进行运算
2 c=0
3 for i in range(length):
4     c = c + server_list[i] * enc_list[i]
5 print("产生密文: ",c.ciphertext())
```

最后，在客户端进行解密操作，得到了要读取的数值。

```
1 ##### 客户端进行解密
2 m=private_key.decrypt(c)
3 print("得到数值: ",m)
```

完整代码如下所示：

```
1 from phe import paillier # 开源库
2 import random # 选择随机数
3
4 ##### 设置参数
5 # 服务器端保存的数值
6 message_list = [100,200,300,400,500,600,700,800,900,1000]
7 length = len(message_list)
8 # 客户端生成公私钥
9 public_key, private_key = paillier.generate_paillier_keypair()
10 # 客户端随机选择一个要读的位置
11 pos = random.randint(0,length-1)
12 print("要读起的数值位置为: ",pos)
13
14 ##### 客户端生成密文选择向量
15 select_list=[]
16 enc_list=[]
17 for i in range(length):
18     select_list.append( i == pos )
19     enc_list.append( public_key.encrypt(select_list[i]) )
20
21 # for element in select_list:
22 #     print(element)
23 # for element in enc_list:
24 #     print(private_key.decrypt(element))
25
```



```

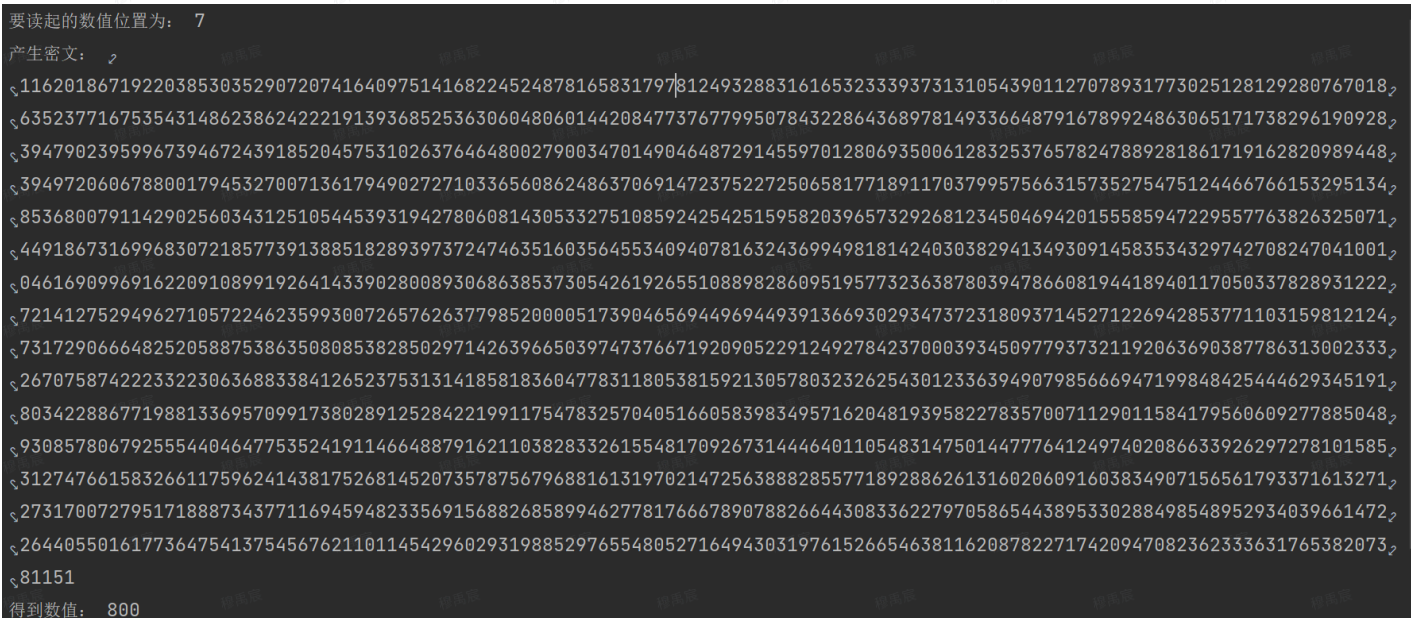
26 ##### 服务器端进行运算
27 c=0
28 for i in range(length):
29     c = c + message_list[i] * enc_list[i]
30 print("产生密文: ",c.ciphertext())
31
32 ##### 客户端进行解密
33 m=private_key.decrypt(c)
34 print("得到数值: ",m)

```

运行上述代码，得到如下结果：

- 1 要读起的数值位置为： 7
- 2 产生密文： 1162018671922038530352907207416409751416822452487816583179781249328831
- 3 得到数值： 800

截图如下所示，图为pycharm结果：



```

要读起的数值位置为： 7
产生密文： 1162018671922038530352907207416409751416822452487816583179781249328831
116201867192203853035290720741640975141682245248781658317978124932883161653233393731310543901127078931773025128129280767018,
635237716753543148623862422219139368525363060480601442084773767799507843228643689781493366487916789924863065171738296190928,
394790239599673946724391852045753102637646480027900347014904648729145597012806935006128325376578247889281861719162820989448,
394972060678800179453270071361794902727103365608624863706914723752272506581771891170379957566315735275475124466766153295134,
853680079114290256034312510544539319427806081430533275108592425425159582039657329268123450469420155585947229557763826325071,
449186731699683072185773913885182893973724746351603564553409407816324369949818142403038294134930914583534329742708247041001,
046169099691622091089919264143390280089306863853730542619265510889828609519577323638780394786608194418940117050337828931222,
72141275294962710572246235993007265762637798520000517390465694496449391366930293473723180937145271226942853771103159812124,
731729066648252058875386350808538285029714263966503974737667192090522912492784237000393450977937321192063690387786313002333,
267075874222332230636883384126523753131418581836047783118053815921305780323262543012336394907985666947199848425444629345191,
803422886771988133695709917380289125284221991175478325704051660583983495716204819395822783570071129011584179560609277885048,
930857806792555440464775352419114664887916211038283326155481709267314446401105483147501447776412497402086633926297278101585,
312747661583266117596241438175268145207357875679688161319702147256388828557718928862613160206091603834907156561793371613271,
273170072795171888734377116945948233569156882685899462778176667890788266443083362279705865443895330288498548952934039661472,
264405501617736475413754567621101145429602931988529765548052716494303197615266546381162087822717420947082362333631765382073,
81151
得到数值： 800

```

## 扩展实验

要求为：“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。”

首先，我选择了异或加密算法。下面先介绍异或加密算法。

### 异或加密算法

异或加密是一种**对称加密算法**。此算法的原理为：利用了计算机中的异或计算，异或计算(符号记为‘^’)的原理是，相同为 0，不同为 1。

如下所示：

```
1 0 ^ 0 = 0
2 1 ^ 1 = 0
3 1 ^ 0 = 1
4 0 ^ 1 = 1
```

本加密算法的核心在于由于  $1 \wedge 0$  或  $0 \wedge 1$  的结果都为 1，因此不能直接由结果 1 推出原来的明文到底是 0 还是 1，达到保护明文的目的。

那么，对同一个明文 B，使用密钥 A 对其进行两次异或运算，可得到明文 B，我们正是利用了这一特性，进行实现加密解密。则完整的加解密算法如下：

加密算法：

1. 明文与密钥按照异或运算规则进行异或运算，产生密文
2. 将得到的密文进行base64 编码，最后返回

解密算法：

1. 将密文进行 base64 解码，得到原始的密文
2. 将密文按照相同的异或运算规则与密文进行异或，得到明文

当然，为了保证长度相同，我们还应该进行取余和zero padding等处理方式。

## 整体流程

1. 服务器端保存了一些数值，使用异或加密算法对其进行加密，得到加密后的数值列表。
2. 客户端生成了公私钥，并随机选择了一个要读取的位置。
3. 客户端生成了一个密文选择向量，其中只有要读取的位置对应的元素为True，其他元素为False。
4. 服务器端对每个加密后的数值和密文选择向量进行乘法运算，并将结果相加得到一个密文。
5. 客户端使用私钥对密文进行解密，得到了用密钥k加密后的数值。
6. 客户端使用异或解密算法对加密后的数值进行解密，得到了原始数值。

## 具体代码

首先，编写异或加密函数，代码如下所示：

```
1 # 采取异或加密解密，只是为了简便起见。但是，需要注意的是，异或加密就是一种对称密钥算法。
2 def xor_encrypt_decrypt(data, key):
3     # 将整型转换为字节数组
4     data_bytes = data.to_bytes(((data.bit_length() + 7) // 8), byteorder='big')
5     key_bytes = key.to_bytes(((key.bit_length() + 7) // 8), byteorder='big')
```



```

6     # 将字节数组转换为可变的bytearray类型
7     data_bytearray = bytearray(data_bytes)
8     key_bytearray = bytearray(key_bytes)
9     # 遍历字节数组，对每个字节进行异或操作
10    for i in range(len(data_bytearray)):
11        data_bytearray[i] ^= key_bytearray[i % len(key_bytearray)]
12    # 将加密/解密后的字节数组转换为整型
13    return int.from_bytes(data_bytearray, byteorder='big')

```

注意，这里我为了简化操作，并没有对原字符串类型进行操作，而是直接采取了int型进行byte的转换。

然后我自定义了一个key，如下所示：

```

1  # key也是为了简化操作，设置为int型123456
2  key=123456

```

然后加密，得到要保存的加密后的数据：

```

1  ##### 设置参数
2  # 服务器端保存的数值
3  message_list = [100,200,300,400,500,600,700,800,900,1000]
4  # 服务器要保存的加密后数值
5  server_list=[]
6  for i in message_list:
7      encrypted_data = xor_encrypt_decrypt(i, key)
8      server_list.append(encrypted_data)
9      # print(encrypted_data)
10 # 加密后的数组
11 print("密钥k加密后的数组:", server_list)

```

然后在服务端，我们使用加密后的数据进行计算：

```

1  ##### 服务器端进行运算
2  c=0
3  for i in range(length):
4      c = c + server_list[i] * enc_list[i]
5  print("产生密文: ",c.ciphertext())

```

注意，这里使用的是server\_list，即加密后的数据！

最后，进行解密：

```
1 ##### 客户端进行解密
2 m=private_key.decrypt(c)
3 print("得到用密钥k加密后的数值:",m)
4 #然后再进行异或解密
5 decrypted_data = xor_encrypt_decrypt(m, key)
6 print("得到原始数值: ",decrypted_data)
```

完整代码如下所示：

```
1 from phe import paillier # 开源库
2 import random # 选择随机数
3 # 采取异或加密解密，只是为了简便起见。但是，需要注意的是，异或加密就是一种对称密钥算法。
4 def xor_encrypt_decrypt(data, key):
5     # 将整型转换为字节数组
6     data_bytes = data.to_bytes((data.bit_length() + 7) // 8, byteorder='big')
7     key_bytes = key.to_bytes((key.bit_length() + 7) // 8, byteorder='big')
8     # 将字节数组转换为可变的bytearray类型
9     data_bytearray = bytearray(data_bytes)
10    key_bytearray = bytearray(key_bytes)
11    # 遍历字节数组，对每个字节进行异或操作
12    for i in range(len(data_bytearray)):
13        data_bytearray[i] ^= key_bytearray[i % len(key_bytearray)]
14    # 将加密/解密后的字节数组转换为整型
15    return int.from_bytes(data_bytearray, byteorder='big')
16 # key也是为了简化操作，设置为int型123456
17 key=123456
18 ##### 设置参数
19 # 服务器端保存的数值
20 message_list = [100,200,300,400,500,600,700,800,900,1000]
21 # 服务器要保存的加密后数值
22 server_list=[]
23 for i in message_list:
24     encrypted_data = xor_encrypt_decrypt(i, key)
25     server_list.append(encrypted_data)
26     # print(encrypted_data)
27 # 加密后的数组
28 print("密钥k加密后的数组:", server_list)
29 length = len(server_list)
30 # 客户端生成公私钥
31 public_key, private_key = paillier.generate_paillier_keypair()
32 # 客户端随机选择一个要读的位置
```

```

33 pos = random.randint(0,length-1)
34 print("要读起的数值位置为:",pos)
35
36 ##### 客户端生成密文选择向量
37 select_list=[]
38 enc_list=[]
39 for i in range(length):
40     select_list.append( i == pos )
41     enc_list.append( public_key.encrypt(select_list[i]) )
42
43 ##### 服务器端进行运算
44 c=0
45 for i in range(length):
46     c = c + server_list[i] * enc_list[i]
47 print("产生密文:",c.ciphertext())
48
49 ##### 客户端进行解密
50 m=private_key.decrypt(c)
51 print("得到用密钥k加密后的数值:",m)
52 #然后再进行异或解密
53 decrypted_data = xor_encrypt_decrypt(m, key)
54 print("得到原始数值:",decrypted_data)

```

然后，我们得到了如下结果：

- 1 密钥k加密后的数组： [101, 201, 206, 114, 22, 954, 862, 706, 614, 522]
- 2 要读起的数值位置为： 6
- 3 产生密文： 1043766326359742644271684013147952022319223863069615448713729321916580
- 4 得到用密钥k加密后的数值： 862
- 5 得到原始数值： 700

如下是pycharm的运行截图：

```
密钥k加密后的数组: [101, 201, 206, 114, 22, 954, 862, 706, 614, 522]
```

```
要读起的数值位置为: 6
```

```
产生密文:
```

```
104376632635974264427168401314795202231922386306961544871372932191658007635880671897086809676675908519948123776193181541888,
289644115793042822993767439878965839992050523295506361685719668775058630954809760233793605169529925177941669836633018596925,
787747340532073963950277523146617250160865083918828847694855521251967101278629351921488712951946845295261997237907571276099,
93600674850592122195237550790488530826754260648252682633927876082900058387051584834458114173882917961503358973105103392093,
817209381002943598186302612186922089875723870477713129232695494747242100315823655649174801505888286536795098881061894172966,
714483358132234396220935269287587323080493115486870704321288291827181737168492967734035366567426117025948067655213548575405,
258498882242733492328723321878484313238275793316807769258524134316000575685409889063987308500456514691868738549407300253254,
486649697475432017752413140724592332607744122341211525483557644803093212563354749292101059752188169384660361991632692841738,
516586636253049485867212506214127771920366592006823075838849927770575198572102548855093509454550640473159688496052166018040,
224575795568510488898220860319077254011472049367918549414974117319377747444732278899335404816644112357908783134694877604545,
156826535876932649409930101667974444537872593514038030650063134360489925844886887940920503473770909135442381470362498348353,
590445812244371980749670200119943791766463392627908509994753082930991958565445614605765940577193372817077202128285328546244,
082069790347333776448912609785240111226357669099848661440352051255118117463868200533519836524324155297520889122580617116545,
859848129278582787572787121096252425350642659325866755195722579038519059863336037299375636375188759455560350727439480439737,
827750425695092700515121066399984464840901368889923417022801179320151776573921286474075576402279204900319378475998962151818,
87163
```

```
486849797475432017752413140724592332607744122341211525483557644803093212563354749292101059752188169384660361991632692841738
516586636253049485867212506214127771920366592006823075838849927770575198572102548855093509454550640473159688496052166018040
224575795568510488898220860319077254011472049367918549414974117319377747444732278899335404816644112357908783134694877604545
156826535876932649409930101667974444537872593514038030650063134360489925844886887940920503473770909135442381470362498348353
590445812244371980749670200119943791766463392627908509994753082930991958565445614605765940577193372817077202128285328546244
082069790347333776448912609785240111226357669099848661440352051255118117463868200533519836524324155297520889122580617116545
859848129278582787572787121096252425350642659325866755195722579038519059863336037299375636375188759455560350727439480439737
827750425695092700515121066399984464840901368889923417022801179320151776573921286474075576402279204900319378475998962151818
87163
```

```
得到用密钥k加密后的数值: 862
```

```
得到原始数值: 700
```

可以看见得到了我们预期的结果，证明本次扩展实验成功！

## 心得体会

本次实验，我先通过已经给出的参考代码，熟悉了phe库的一些基本操作，也明白了半同态加密的一些基本原理。事实上，我认为，本次实验最重要的是看懂基础实验的代码，尤其是其协议目的，弄清楚对服务器端加密的具体含义，是为了保护用户端的隐私，这样才能明白整个算法和后续的扩展实验的意义。

对于扩展实验，我在实验过程中遇到不少困难，比如我一开始想采用AES等加密方法进行加密，但是一方面那些方法并不满足对称密钥的条件，另一方面由于phe库会将数据改成byte型，不利于我后续使用int型进行操作，频繁的数据类型转换也给我带来了不少困难。最终，我选择简化了操作，采取异或加密的方式，一方面其符合对称加密的要求，另一方面也降低了实验难度。

总体上说，本次实验难度不大，我的收获很大！