



南开大学  
Nankai University

南 开 大 学

网 络 空 间 安 全  
学 院

数据安全实验报告

---

半同态加密应用实践

---

2111408 周钰宸

年级：2021 级

专业：信息安全

指导教师：刘哲理

2024 年 3 月 20 日

## 摘要

Paillier 加密算法是 Paillier 等人 1999 年提出的一种基于判定  $n$  阶剩余类难题的典型密码学加密算法，具有加法同态性，是半同态加密方案。本次实验完成了基于 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。

**关键字：**数据安全 同态加密 隐私信息获取 Paillier 算法

## 目录

<b>一、 实验要求</b>	<b>1</b>
<b>二、 实验原理</b>	<b>1</b>
(一) 同态加密 . . . . .	1
(二) 半同态加密 . . . . .	1
(三) Paillier 算法 . . . . .	2
<b>三、 实验准备</b>	<b>2</b>
(一) 安装 phe 库 . . . . .	2
(二) phe 库练习 . . . . .	3
<b>四、 实验过程</b>	<b>5</b>
(一) 隐私信息获取 . . . . .	6
(二) 拓展探索 . . . . .	7
<b>五、 实验总结与思考</b>	<b>10</b>

## 一、实验要求

1. 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。
2. 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。

本次实验基于上述实验要求，完成了 Paillier 算法的隐私信息获取 (PIR)，并额外完成了拓展实验的探索，接下来将详细介绍实验内容。

## 二、实验原理

### (一) 同态加密

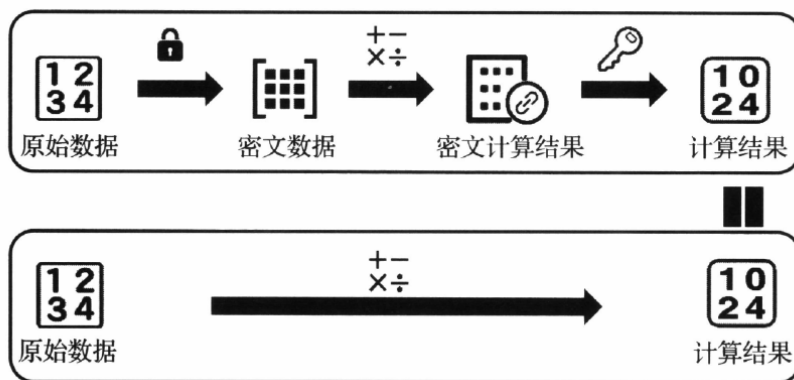


图 1: 同态加密原理

如1所示，**同态加密 (Homomorphic Encryption, HE)** 是一种特殊的加密方法，它允许我们在加密数据上执行计算，例如加法和乘法，而不会泄露原始明文的任何信息。计算的结果仍然是加密的，只有拥有密钥的用户可以解密并获得处理后的明文结果。

**“注意：加法和乘法同态是相对明文而言所执行的操作，而非密文上执行的运算形式。”**

### (二) 半同态加密

半同态加密 (partial homomorphic encryption, 简称 PHE) 仅支持单一类型的密文域同态运算 (加或乘同态)；

一个常见的应用场景是安全计算 (Secure Computation)。在安全计算中，两个或多个参与方希望在不暴露各自私有数据的情况下，进行某些计算并得到计算结果。PHE 可以应用在安全计算中，其中参与方可以将他们的私有数据加密后共享给其他方进行计算，而不用担心数据泄露。



图 2: 半同态加密应用

本次实验重点的 Paillier 算法就属于半同态加密算法。

### (三) Paillier 算法

1. **理论原理：**Paillier 算法是一种半同态加密算法，它基于数论的困难问题，如大数的分解和离散对数问题。其核心是基于 RSA 加密算法的改进，使用了同余运算和模幂运算。**加密过程：**

- 首先，选择两个大质数  $p$  和  $q$  并计算它们的乘积  $n = p \cdot q$ 。
- 接着，计算  $\lambda = \text{LCM}(p-1, q-1)$ ，即  $p-1$  和  $q-1$  的最小公倍数。
- 选择一个整数  $g$ ，确保  $g$  是  $n$  平方剩余模  $n^2$  的生成元。
- 对明文  $m$  进行加密，首先计算  $c = g^m \cdot r^n \bmod n^2$ ，其中  $r$  是一个随机数，且满足  $0 < r < n$  且  $\text{gcd}(r, n) = 1$ 。

2. **解密过程：**对密文  $c$  进行解密，计算  $L(c^\lambda \bmod n^2) \cdot \mu \bmod n$ ，其中  $L(x) = \frac{x-1}{n}$ ，而  $\mu$  是  $\lambda$  的逆元模  $n$ 。

3. **应用：**Paillier 算法在安全多方计算、隐私保护数据挖掘和电子投票等领域具有重要应用。它提供了同态加密的功能，使得在加密状态下对数据进行计算成为可能，同时保护了数据的隐私性。

## 三、实验准备

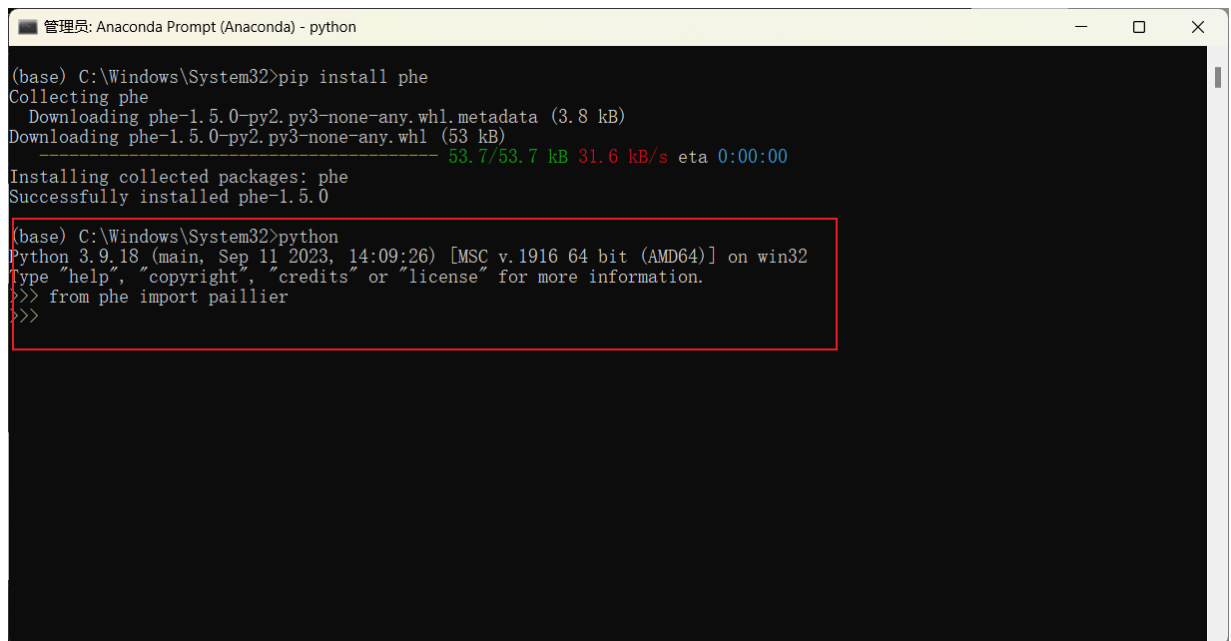
### (一) 安装 phe 库

本次实验中我将以我的 Windows 操作系统中的 Python 环境进行实验。首先为了能够正确地调用 Paillier 算法，需要先安装其对应的库。打开 Anaconda 后，安装对应的 Phe 库，通过如下命令：

```
1 (base) C:\Windows\System32>pip install phe
```

安装成功后，通过以下命令验证安装成功，成功后不再提示错误信息：

```
1 >>> from phe import paillier
```



```
(base) C:\Windows\System32>pip install phe
Collecting phe
  Downloading phe-1.5.0-py2.py3-none-any.whl.metadata (3.8 kB)
Downloading phe-1.5.0-py2.py3-none-any.whl (53 kB)
----- 53.7/53.7 kB 31.6 kB/s eta 0:00:00
Installing collected packages: phe
Successfully installed phe-1.5.0

(base) C:\Windows\System32>python
Python 3.9.18 (main, Sep 11 2023, 14:09:26) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from phe import paillier
>>>
```

图 3: phe 安装成功

图3可以看到 **phe 库安装成功，完成了实验准备的第一步**。接下来为了让我能够对同态加密和 phe 库能够更加熟练地应用，首先将进行一个练习。

## (二) phe 库练习

接下来为了后来能够更好地理解 phe 库原理并运用，**我将首先以基于 phe 库完成加法和标量乘法的验证**。

```
1  #!/usr/bin/env python
2  # -*- encoding: utf-8 -*-
3  """
4  @Project : Temp
5  @File    : Paillier.py
6  @IDE     : PyCharm
7  @Author  : ErwinZhou
8  @Date    : 2024/3/18 22:59
9  """
10 from phe import paillier # 开源库
11 import time # 做性能测试
12
13 # 设置参数
14 print("默认私钥大小: ", paillier.DEFAULT_KEYSIZE)
15 # 生成公私钥
16 public_key, private_key = paillier.generate_paillier_keypair()
17 # 测试需要加密的数据
18 message_list = [3.1415926, 100, -4.6e-12]
19 # 加密操作
```

```

20 time_start_enc = time.time()
21 encrypted_message_list = [public_key.encrypt(m) for m in message_list]
22 time_end_enc = time.time()
23 print("加密耗时s: ", time_end_enc - time_start_enc)
24 print("加密数据 (3.1415926) :", encrypted_message_list[0].ciphertext())
25 # 解密操作
26 time_start_dec = time.time()
27 decrypted_message_list = [private_key.decrypt(c) for c in
    encrypted_message_list]
28 time_end_dec = time.time()
29 print("解密耗时s: ", time_end_dec - time_start_dec)
30 print("原始数据 (3.1415926) :", decrypted_message_list[0])
31 # 测试加法和乘法同态
32 a, b, c = encrypted_message_list # a,b,c分别为对应密文
33 a_sum = a + 5 # 密文加明文, 已经重载了+运算符
34 a_sub = a - 3 # 密文加明文的相反数, 已经重载了-运算符
35 b_mul = b * 6 # 密文乘明文, 数乘
36 c_div = c / -10.0 # 密文乘明文的倒数
37 print("a+5 密文:", a.ciphertext()) # 密文纯文本形式
38 print("a+5=", private_key.decrypt(a_sum))
39 print("a-3", private_key.decrypt(a_sub))
40 print("b*6=", private_key.decrypt(b_mul))
41 print("c/-10.0=", private_key.decrypt(c_div))
42 # 密文加密文
43 print((private_key.decrypt(a) + private_key.decrypt(b)) ==
    private_key.decrypt(a + b))
44 # 报错, 不支持a*b, 即两个密文直接相乘
45 #
    print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))

```

### 代码思路如下

1. 导入所需模块: 首先, 代码导入了 `phe.paillier` 模块, 该模块提供了 Paillier 加密算法的实现, 以及 `time` 模块, 用于性能测试。
2. 设置参数: 接着, 代码使用 `paillier.DEFAULT_KEYSIZE` 来设置默认私钥的大小。这个参数决定了生成的密钥的长度, 从而影响加密的强度。
3. 生成公私钥: 然后, 通过调用 `paillier.generate_paillier_keypair()` 函数生成了公钥和私钥对。这对密钥将用于加密和解密操作。
4. 加密和解密操作: 在这一步, 代码将定义的数据列表中的每个元素使用生成的公钥进行加密, 并使用私钥进行解密。这一过程展示了 Paillier 加密算法的基本用法。
5. 测试同态性质: 最后, 代码对加密后的数据进行同态加法和乘法操作, 并使用私钥解密结果以验证同态性质。这个步骤展示了 Paillier 加密算法在同态加密方面的应用。

结果如下：

```

Paillier
加密耗时: 0.524977224395752
加密数据 (3.1415926) :
296284399780291117755598789411121171352074980601347436479346974534785550396621705028494159991774442654537115639830371056869097284595644031965228314490857041457024667872086023
6764057885273781637858991365423182938576936982126676294794063005362122955362703185781349113526432752819294410655117805311891604120280846290537635562825348065117491341479221
4559675406963066534350515215030356019249164403299405704877883319813380072986239524079773546537593268398824220417977057129853667647108289329467451557668668742649980312038184
229219622203376225744878266295745106221669544992789365117417689348193650861998248264372782339650541709598335661989214648213171078272800426579965808365719697523362607463336201
629343480770725380856973869149383418592264869633784335834916619611638782604196864461974839319271718670103772461180759089925394825370361399869784121972854592412701444962920723
363856056577275001436152445104841854055025230877302610033856962025112669531649014287797278697665401239465604802237359670440221439609860080406090381515013648866543310994906
93087954016961119407338584500502859714050768309464053357730034161331619402705354838551763273074576886057221736008596388392804108727404648395855127351651891779072529884148622
27166676161981779936788309789790894324964482984316711850927799557854987363347476867971920559112081517018517744057337744950974246342259543747145762528013886407964801096130854
739457843266479199208954606738043433318680385340803057133798758971279263765602749969259069687485414678988788065195593374532933596069860619838224808045305757561163750608799800
34472567151237805557792275007657240176503985820898026197868398999162600742353346519998583798762831773236093433235474924275420325758315512219068848185842741190623139209976969
9782692404587475040902454836740343178582826099893610287892923857977961348834062661420667085795559853784070201

```

图 4: 验证结果 1

```

解密耗时: 0.1515820026397705
原始数据 (3.1415926) : 3.1415926
a+5 密文:
296284399780291117755598789411121171352074980601347436479346974534785550396621705028494159991774442654537115639830371056869097284595644031965228314490857041457024667872086023
6764057885273781637858991365423182938576936982126676294794063005362122955362703185781349113526432752819294410655117805311891604120280846290537635562825348065117491341479221
4559675406963066534350515215030356019249164403299405704877883319813380072986239524079773546537593268398824220417977057129853667647108289329467451557668668742669980312038184
229219622203376225744878266295745106221669544992789365117417689348193650861998248264372782339650541709598335661989214648213171078272800426579965808365719697523362607463336201
629343480770725380856973869149383418592264869633784335834916619611638782604196864461974839319271718670103772461180759089925394825370361399869784121972854592412701444962920723
363856056577275001436152445104841854055025230877302610033856962025112669531649014287797278697665401239465604802237359670440221439609860080406090381515013648866543310994906
93087954016961119407338584500502859714050768309464053357730034161331619402705354838551763273074576886057221736008596388392804108727404648395855127351651891779072529884148622
27166676161981779936788309789790894324964482984316711850927799557854987363347476867971920559112081517018517744057337744950974246342259543747145762528013886407964801096130854
739457843266479199208954606738043433318680385340803057133798758971279263765602749969259069687485414678988788065195593374532933596069860619838224808045305757561163750608799800
34472567151237805557792275007657240176503985820898026197868398999162600742353346519998583798762831773236093433235474924275420325758315512219068848185842741190623139209976969
9782692404587475040902454836740343178582826099893610287892923857977961348834062661420667085795559853784070201
a+5= 8.1415926
a-3 0.141592600000000007
b*6= 600
c/-10.0= 4.6e-13
True

```

图 5: 验证结果 2

图4和图5清楚地展示了加密和解密的过程，耗时，最后的 **True** 也代表了加密解密成功，验证成功。

同时值得注意的是，如果取消最后那里，理论上会产生报错，报错是因为 **Paillier 加密算法并不支持直接对两个密文进行乘法操作**。Paillier 加密算法只支持同态加法，而不支持同态乘法。因此，当试图执行 `private_key.decrypt(a * b)` 这样的操作时，会导致报错。

```

Traceback (most recent call last):
  File "D:\Codes\Others\Temp\Paillier.py", line 45, in <module>
    print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))
  File "D:\Anaconda\lib\site-packages\phe\paillier.py", line 508, in __mul__
    raise NotImplementedError('Good luck with that...')
NotImplementedError: Good luck with that...

```

图 6: 验证报错

图6确实出现了报错，这进一步验证了 Paillier 的加法同态性。

## 四、实验过程

现在正式开始实验，

## (一) 隐私信息获取

这里我们首先要基于 **Pailler 协议的一些性质来进行设计**：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。**由此我们给出设计：**

1. **服务器端**：产生数据列表  $data\_list = m_1, m_2, \dots, m_n$
2. **客户端**：
  - 设置要选择的数据位置为  $pos$
  - 生成选择向量  $select\_list = 0, \dots, 1, \dots, 0$ ，其中，仅有  $pos$  的位置为 1
  - 生成密文向量  $enc\_list = E(0), \dots, E(1), \dots, E(0)$
  - 发送密文向量  $enc\_list$  给服务器
3. **服务器端**：
  - 将数据与对应的向量相乘后累加得到密文  $c = m_1 * enc\_list[1] + \dots + m_n * enc\_list[n]$
  - 返回密文  $c$  给客户端
4. **客户端**：解密密文  $c$  得到想要的结果。

基于上述思路，这里参考书中代码：

```
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)

```

```

要读取的数值位置为: 6
产生密文:
15538402924142202517473716939096388696505914868619193930865525190517619554845464118113345786005875540183960359166698199079294101646927461931041598271171768757932670988812322166
482142478934395662373952439375254082565524580248647781395046236582290228195941500368038377184076487338451834015426945828110886880685951650665894132418617656742787251253325321
38794499477577409919246180612293379314768401099138174772069491514429244022667659930911487507787792369672265611662450522359781912746188109396366646653353670602085152511290781
520366152921810891997268173151115340480130621827748459031388564680226250510798120566764677450957642995235555805615027209918502704827654912272962661369139904472407185528254127
40761212080499521074901030169946712384955650942381069695498216360673935786125795637022145265924197040579513774001292608908016510632272130683196881210323816080684549289496943
427815295671700339378009203141521272444853325855637436484994983522380648836198539635428525901453898126714777459993599749699360704857456573376945801726824185868799786084294554
2071903229703940053340556298690094243855774231620863681076541033335400512799942187011843245644979507913818836407401644141043520259188073680107445941107176751259980187177043673
281425095159687778473778166056287943500321365427778829379546526951428985512151578122477760434911161917368432624541571104820423447913259148053631963314246798462257952931788691
356734244983271706541769861090848357237584931965781693547559452922687647009298825514005092720716347708412612621182451726116387748250733188162613924653306318747585563829373362
949795818704500270273729840231746804830461146950327456335235450720523392216862305057114756108571215148252635175339389093635400518231958403156284154899081235983804125406961780
76492860399236184445924733666087960986964352889793686992366788760645850538678297623896555809407896065385499199
得到数值: 700
进程已结束,退出代码为 0

```

图 7: 隐私信息获取结果

图7可以看到成功地完成了隐私获取,达到了实验要求。

## (二) 拓展探索

在拓展探索这个阶段,我选择借助 Crypto.Cipher 库里的 AES 包的 AES 算法进行实现。

具体而言:客户端使用对称密钥  $k$  对消息列表进行加密,并将加密后的数据发送给服务器端;服务器端使用 Paillier 密文进行同态计算,得到选择的密文的加权和,并将结果发送给客户端;客户端使用 Paillier 私钥解密服务器端计算得到的 Paillier 密文,然后再使用对称密钥  $k$  进行解密,最终得到了选择的密文对应的明文数据。

代码如下:

```

1 #!/usr/bin/env python
2 # -*- encoding: utf-8 -*-
3 """
4 @Project : Temp
5 @File : Paillier.py
6 @IDE : PyCharm
7 @Author : ErwinZhou
8 @Date : 2024/3/18 22:59
9 """
10 from Crypto.Cipher import AES
11 import random

```

```
12 from phe import generate_paillier_keypair
13 def encrypt_message_list(message_list, password):
14
15     aes = AES.new(password, AES.MODE_ECB)
16     encrypt_list = []
17     for message in message_list:
18         en_text = aes.encrypt(message.to_bytes(length=16, byteorder='big',
19             signed=False))
20         encrypt_list.append(en_text)
21     return encrypt_list
22
23 def generate_select_list(length, pos):
24
25     select_list = [i == pos for i in range(length)]
26     return select_list
27
28 def encrypt_select_list(select_list, public_key):
29
30     enc_list = [public_key.encrypt(select) for select in select_list]
31     return enc_list
32
33 def server_computation(encrypt_message_list, enc_list):
34
35     c = 0
36     for i in range(len(encrypt_message_list)):
37         trans = int().from_bytes(encrypt_message_list[i], byteorder='big',
38             signed=False)
39         c += trans * enc_list[i]
40     return c
41
42 def decrypt_and_decode(ciphertext, private_key, aes):
43
44     m = private_key.decrypt(ciphertext)
45     m = m % (2**128) # 确保m的大小在128位以内
46     m = aes.decrypt(m.to_bytes(length=32, byteorder='big', signed=True))
47     m = int().from_bytes(m, byteorder='big', signed=True)
48     return m
49
50 def main():
51     password = b'1548164742548705'
52     message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
53     length = len(message_list)
54     encrypted_messages = encrypt_message_list(message_list, password)
```

```
54 pos = random.randint(0, length - 1)
55 print("要读起的数值位置为: ", pos)
56
57 select_list = generate_select_list(length, pos)
58 public_key, private_key = generate_paillier_keypair()
59 enc_list = encrypt_select_list(select_list, public_key)
60
61 c = server_computation(encrypted_messages, enc_list)
62 print("产生密文: ", c.ciphertext())
63
64 m = decrypt_and_decode(c, private_key, AES.new(password, AES.MODE_ECB))
65 print('未经过AES解密的密文为: ', m)
66 print("经过AES解密后得到数值: ", m)
67 if __name__ == "__main__":
68     main()
```

这里对上述代码架构进行简单解释:

#### 1. 生成对称密钥:

- 客户端生成了一个对称密钥  $k$ , 用于加密和解密数据。

#### 2. 对消息列表进行加密:

- 使用 AES 算法对给定的消息列表进行加密, 得到加密后的消息列表。
- 加密过程使用了客户端生成的对称密钥  $k$ 。

#### 3. 生成选择列表:

- 生成一个选择列表, 用于指示要读取的位置。

#### 4. 使用 Paillier 加密选择列表:

- 使用 Paillier 公钥对生成的选择列表进行加密, 得到加密后的选择列表。
- 这个加密过程使用了 Paillier 加密算法, 以确保选择列表的安全传输。

#### 5. 在服务器端进行计算:

- 服务器端接收到加密后的消息列表和加密后的选择列表, 然后对这些数据进行计算。
- 计算过程使用了 Paillier 密文进行同态计算, 以保护数据的隐私性。
- 最终得到一个 Paillier 密文  $c$ , 其中包含了选择的密文的加权和。

#### 6. 解密并解码密文:

- 客户端接收到服务器端计算得到的 Paillier 密文  $c$ , 然后使用 Paillier 私钥对其进行解密。
- 解密后得到的结果再经过 AES 解密, 最终得到了选择的密文对应的明文数据。

实验结果如下：

```
要读取的数值位置为： 9
产生密文：
231808439802139781478562630510452225125581601552717912855363065758866071743386676469554058420803646464916015146959390141032868085305923723819306291927462166849628387767865614
464739298415684405428327463711226464103415894789897804816512277828878508331504859953616547699187639991122082411023085802299519801084172482270356767890765267833405622430725113
398308447608952584825421695496951188888860721132082904231714279493610378471374910913333747763975347647296289068899623050008369486933951551770033622440244747977459469924756690
54790829519995240654214254618185612278150323864328967476760787753198099839761573668192727767820236756142246267375590875859770099788593358131362210361719245468859624796189903
478068283864779859194893857158824197611429139180609417531123295144571762998642078727330181752391648541553184370692590015113408868344328698978468028491745363684135345727753149
330326917064272442484271177063317241234229714377212613102667445010422783122855502171561379820988159608904065932394587439884746202689641791669951443668308135643318583530721625
1912403565747030080811580694614478288282379777565476394875663784908721389451645353578164530908122874990898205443401020535602003367611874872381475925368344899492748552355627915
55749176160653903359462818150080343812125836104513755174387351712386985965814951706815163848942168952163153938783687962343383803679972043025328621618636621806447343624373509
743616809308099080242126776616498491135859540551062794696726901722100734390110468561012609415388250124774441465188154857475827311728198999715329844188546201761187805973206303
058495217918874518404661070493898013358416952925372165320461614613361078035729941925784177982851190428861059704890489653622269178269162757194258597522563389685135941348240079
97268902719480722091258182973853499161749855526746718655815849898819015771424113316481779822286341574230704117
未经过AES解密的密文为： 35919644681305063213418507017736773273634040497988410830506558055227758478312
经过AES解密后得到数值： 35919644681305063213418507017736773273634040497988410830506558055227758478312
```

图 8: 拓展探索结果

图8由此就实现了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的要求。**到此实验全部结束，非常成功！**

## 五、实验总结与思考

本次实验我通过将课堂上老师讲授的同态加密知识在课后进行了复习。跟着参考资料和老师讲解视频的思路，对参考代码进行了研读。

由此我依次完成了基本和拓展探索的实验要求。不仅让我更加深刻理解到半同态加密和 Paillier 算法的原理，更是最后结合 AES 让加密场景更加真实。

总的来说，本次实验我收获颇丰，希望在后面的实验中继续努力，将数据安全领域的知识熟记于心，并在实验中反复巩固，不断探索。

## 参考文献

NIKU