



南開大學
Nankai University

南开大学

计算机学院和密码与网络空间安全学院

《数据安全》课程作业

半同态加密应用实践

姓名：陆皓喆

学号：2211044

专业：信息安全

指导教师：刘哲理

2025 年 3 月 11 日

目录

| | |
|---------------------------------------------|-----------|
| 1 实验要求 | 2 |
| 2 github 仓库 | 2 |
| 3 实验原理 | 2 |
| 3.1 同态加密 | 2 |
| 3.2 半同态加密 | 3 |
| 3.3 Paillier 算法 | 3 |
| 4 实验过程 | 4 |
| 4.1 实验环境安装 | 4 |
| 4.1.1 安装 python 环境 | 4 |
| 4.1.2 安装 phe 库 | 4 |
| 4.1.3 验证环境正确性 | 5 |
| 4.2 基于 Python 的 phe 库完成加法和标量乘法的验证 | 5 |
| 4.3 隐私信息获取 | 7 |
| 4.4 扩展实验探索 | 9 |
| 5 实验心得与体会 | 10 |

1 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。

2 github 仓库

本次实验的有关代码和文件，都已经上传至我的个人 github 中。
您可以通过访问[此链接](#)来查阅我的代码文件。

3 实验原理

3.1 同态加密

同态加密 (HE, homomorphic encryption) 是一种加密算法，它可以通过对密文进行运算得到加密结果，解密后与明文运算的结果一致。主要原理如下所示：

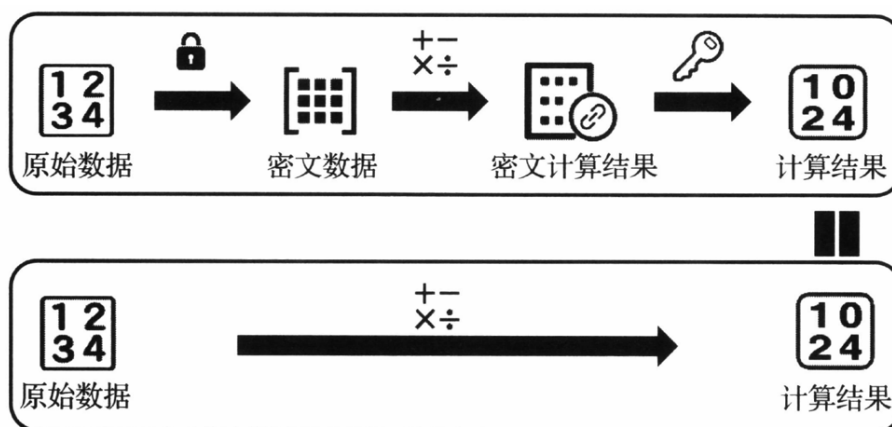


图 3.1: 同态加密原理

同态加密主要基于公钥密码体制构建，它允许将加密后的密文发给任意的第三方进行计算，并且在计算前不需要解密，可以在不需要密钥方参与的情况下，在密文上直接进行计算。

同态加密主要有以下几种类型：

- **加法同态**：如果加密函数 E 满足 $E(m_1) \times E(m_2) = E(m_1 + m_2)$ ，就称该加密方案具有加法同态性。这意味着对两个明文 m_1 和 m_2 分别加密后得到的密文进行乘法运算，其结果解密后与对 m_1 和 m_2 先进行加法运算再加密的结果相同。
- **乘法同态**：若加密函数 E 满足 $E(m_1)^n = E(m_1 \times n)$ ，则称该加密方案具有乘法同态性。即对明文 m_1 的密文进行 n 次幂运算，解密后与 m_1 乘以 n 再加密的结果一致。
- **全同态**：如果一个加密方案既满足加法同态又满足乘法同态，或者能在一定条件下支持任意多项式计算，就称其为全同态加密方案，可在密文上进行任意复杂的计算。

3.2 半同态加密

半同态加密 (Semi-Homomorphic Encryption) 是同态加密的一种特殊形式, 它只支持一种特定类型的同态计算 (通常是加法同态或者乘法同态), 而不是像全同态加密那样支持任意的计算。以下是关于半同态加密实验原理的阐述。

半同态加密的实验原理主要基于特定的数学困难问题来构造加密算法, 使得在密文上能够进行某种运算, 并且运算结果解密后与在明文上进行相应运算的结果一致。

以加法同态的半同态加密为例, 假设存在一个加密函数 E 和一个解密函数 D , 对于两个明文 m_1 和 m_2 :

- **加密过程:** 分别对明文 m_1 和 m_2 进行加密, 得到密文 $c_1 = E(m_1)$ 和 $c_2 = E(m_2)$ 。这里的加密函数 E 是基于特定的数学原理构造的, 比如在一些基于数论的加密方案中, 可能会利用大整数分解、离散对数等困难问题来生成密文。
- **密文运算:** 在密文上进行特定的运算, 对于加法同态加密, 就是对密文进行乘法运算 (因为其满足 $E(m_1) \times E(m_2) = E(m_1 + m_2)$), 得到 $c = c_1 \times c_2$ 。
- **解密过程:** 将密文运算的结果 c 进行解密, 即 $D(c)$ 。根据加法同态的性质, $D(c) = D(c_1 \times c_2) = D(E(m_1 + m_2)) = m_1 + m_2$, 也就是解密后的结果与在明文上先进行加法运算再加密后解密的结果是相同的。

半同态加密在实际应用中, 例如在隐私保护的数据聚合场景下, 当需要对多个加密的数值进行求和操作时 (如果是加法同态加密), 可以直接在密文上进行计算, 然后将结果解密得到总和, 而无需解密每个单独的数值, 从而保护了数据的隐私性。

3.3 Paillier 算法

Paillier 算法是一种基于数论的加法同态加密算法, 以下是其算法的具体步骤:

1. 密钥生成

- 选取两个大素数 p 和 q , 且 p 和 q 满足一定的条件 (例如 $\gcd(pq, (p-1)(q-1)) = 1$)。
- 计算 $n = pq$ 和 $\lambda = \text{lcm}(p-1, q-1)$, 其中 lcm 表示最小公倍数。
- 选取一个整数 g , 使得 g 满足 $g \in \mathbb{Z}_{n^2}^*$ 且 $L(g^\lambda \bmod n^2)$ 与 n 互质, 其中 $\mathbb{Z}_{n^2}^*$ 是模 n^2 的乘法群, $L(x) = \frac{x-1}{n}$ 。
- 公钥为 (n, g) , 私钥为 λ 。

2. 加密

- 对于明文消息 $m \in \{0, 1, \dots, n-1\}$, 选择一个随机数 $r \in \{0, 1, \dots, n-1\}$, 且 r 与 n 互质。
- 计算密文 $c = g^{mr} \bmod n^2$ 。

3. 解密

- 对于密文 c , 计算 $m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$ 。

4. 加法同态性验证

- 假设存在两个明文 m_1 和 m_2 ，其对应的密文分别为 $c_1 = g^{m_1} r_1^n \bmod n^2$ 和 $c_2 = g^{m_2} r_2^n \bmod n^2$ 。
- 计算 $c = c_1 c_2 \bmod n^2 = (g^{m_1} r_1^n)(g^{m_2} r_2^n) \bmod n^2 = g^{m_1+m_2} (r_1 r_2)^n \bmod n^2$ 。
- 对 c 进行解密得到的结果为 $m_1 + m_2$ ，这就验证了 Paillier 算法的加法同态性，即可以在密文上进行加法运算，解密后得到与明文相加相同的结果。

4 实验过程

4.1 实验环境安装

4.1.1 安装 python 环境

首先，我们需要在 windows 系统中安装 python 环境。由于本人的 windows 环境已经安装了 python，所以我们直接在命令行中输入 python，查看其对应版本即可。

```
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python
Python 3.11.9 | packaged by Anaconda, Inc. | (main, Apr 19 2024, 16:40:41) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

图 4.2: 查看 python 环境

我们发现，本机的 python 版本为 3.11.9，所以我们不需要再进行 python 的安装了。然后，我们输入 `from phe import paillier`，检测是否安装 phe 环境，输出如下所示：

```
>>> from phe import paillier
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'phe'
>>> |
```

图 4.3: 查看 phe 是否安装

我们得到了提示：ModuleNotFoundError: No module named 'phe'，说明我们并没有安装这个库。然后，我们输入 `exit()`，退出 python 环境即可。

```
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python
Python 3.11.9 | packaged by Anaconda, Inc. | (main, Apr 19 2024, 16:40:41) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from phe import paillier
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'phe'
>>> exit()
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes>
```

图 4.4: 退出 python 环境

4.1.2 安装 phe 库

下面我们来完成 phe 库的安装。我们在命令行中输入 `pip install phe` 进行安装即可。

```
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> pip install phe
Collecting phe
  Obtaining dependency information for phe from https://files.pythonhosted.org/packages/53/7c/1c514f3e030ff69ee2a184fca3f1514c1d32653ca00869d884b4f981e564/phe-1.5.0-py2.py3-none-any.whl.metadata
    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)
Installing collected packages: phe
Successfully installed phe-1.5.0
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> |
```

图 4.5: 安装 phe

我们发现，成功完成了 phe 库的安装！

4.1.3 验证环境正确性

然后，我们需要验证环境的正确性。我们再次进入 python 环境，输入 `from phe import paillier`，发现不存在报错，说明我们的 phe 库安装成功！

```
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python
Python 3.11.9 | packaged by Anaconda, Inc. | (main, Apr 19 2024, 16:40:41) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from phe import paillier
>>>
```

图 4.6: phe 库安装成功

4.2 基于 Python 的 phe 库完成加法和标量乘法的验证

安装完 phe 库之后，我们对其进行简单的验证。我们使用提供的代码进行测试，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))

```

得到的结果如下所示:

```

PS E:\学学\本科\大三下\数据安全\Lab\Lab02\codes> python test_phe.py
默认密钥大小: 3072
加密耗时: 0.07844257354736328
加密数据 (3.1415926) : 1255726367953980925963598474976442982821018221024876046170864340101813812790871416447515489711535044718263770122601629779988816958024739288168288583832258
47684505861002105039145642345129344181992214347571506885912054574332486301921049967385128460260369225385767703475920924673225542689566926447269527213195129660019754096559876260
303865797914339480385846635584848923182643460821105944565210487345012680087725275086733337356757161002574434064222609544088414507512014929702100341702523997818905979618079644004
04670660167715313322262242381828376759763571675753417311825272219550187159435455332045455421047666133124563057031235415109683228750772685318799197555687788182788899436145803319
182882759276105289378733070670373494444583549588636077667834187586888340241118776223900829943199292936571970808570361753553739857739740537384143597388274202683372927716599216
344880176348141671446549411629132251235308972524101645076327570156403066795395789836835461637970639813876632515180426159333982475340973146275906655625248469063303243568300043
233329866781384549045504828005177823585797455741613348015900641048161124042845751384310511488239857889047988826876548097569968359276440622735283727692774935071706458829880634380
49034192330324015931747378525355391493219825316573271370183837880013268574451698021754100345595602091177886366998822790472816079376862223620919915120763827889799865055243928308
962176635430781274460878735056638127568662709673408363296048656455816462413545638749123686917355897979567719039628898247017039455737913575258354618663297822047249878596486758695
442779370236159972888724754119747837358599716683328977236081262302135236109447172813973056780216426477332805496133638597060387792435696242944267502942383783838874862970682838
3219298049240585014764669514354719770413065054628413878421644344503087140514341148126366160737439994204
解密耗时: 0.01573967933654785
原始数据 (3.1415926) : 3.1415926
a+5 密文: 12557263679539809259635984749764429828210182210248760461708643401018138127908714164475154897115350447182637701226016297799888169580247392881682885838322584768450586100
21050391456423451293441819922143475715068859120545743324863019210499673851284602603692253857677034759209246732255426895669264472695272131951296600197540965598762603038657079143
394803854686355848489231826434608211059445652104873450126800877252750867333373567571610025744340642226095440884145075120149297021003417025239978189059796180796440040467060410771
51313222622423818283767597635716757534173118252722195501871594354553320454554210476661331245630570312354151096832287507726853187991975556877881827888994361458033191828827592761
052893787330706703734944445035495886360776678341875868883402411187762239008299431992929365719708085703617535537398577397405373841435973882742026833729277165992163448801763481
416714465494116291322512353089725241016450763275701564030667953957898368354616379706398138766325151804261593339824753409731462759066556252484690633032435683000432333298667813
845490455048280051778235857974557416133480159006410481611240428457513843105114882398578890479888268765480975699683592764406227352837276927749350717064588298806343804903419233032
40159317473785253553914932198253165732713701838378800132685744516980217541003455956020911778863669988227904728160793768622236209199151207638278897998650552439283089621766354307
812744608787350566381275686627096734083632960486564558164624135456387491236869173558979795677190396288982470170394557379135752583546186632978220472498785964867586954427793702361
599728887247541197478373585997166833289772360812623021352361094471728139730567802164264773328054961336385970603877924356962429442675029423837838388748629706828383219298049240
585014764669514354719770413065054628413878421644344503087140514341148126366160737439994204
a+5= 8.1415926
a-3 = 0.14159260000000007
b*6= 600
c/-10.0= -4.6e-13
True
PS E:\学学\本科\大三下\数据安全\Lab\Lab02\codes>

```

图 4.7: 测试 phe

我们发现, 成功完成了加密与解密, 并且最后输出 true, 说明所有的结果都是正确的。

为了验证 Paillier 算法确实不能实现乘法运算, 我们将最后一行的注释去除, 进行测试, 得到如下报错信息:

```
Traceback (most recent call last):
  File "E:\学学\本科\大三下\数据安全\Lab\Lab02\codes\test_phe.py", line 41, in <module>
    print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))
    ~^~
  File "D:\anaconda3\Lib\site-packages\phe\paillier.py", line 508, in __mul__
    raise NotImplementedError('Good luck with that...')
NotImplementedError: Good luck with that...
```

图 4.8: phe 乘法报错

所以，我们成功验证了该算法只支持同态加法，不支持同态乘法的运算。

4.3 隐私信息获取

接下来，我们开始正式的实验。

首先，我们需要基于 Paillier 协议来进行设计。我们对于 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。

根据这个特点，我们可以设计出相应的方案：

- **服务器端：**产生数据列表 $data_list = \{m_1, m_2, \dots, m_n\}$
- **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
- **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
- **客户端：**解密密文 c 得到想要的结果

所以，我们按照课本上的代码，进行尝试。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)

```

我们进行运行，得到结果，如下所示：

```

test_phe.py | get_private_information.py x
get_private_information.py > ...
1 from phe import paillier # 开源库 Import "phe" could not be resolved
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
9 PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\get_private_information.py"
要读起的数值位置为: 1
产生密文: 1462791717038242231496584632095611425497430974070020004444350525763366216908352591985939862248338405965767711678069482166857016669166920697399371521895491146850186233
0282273705060708021584615712154742402608170623266204373230937700222173183818509146108412700908597306055135013760698112388473481409338919758817467631590761101903127767498939566022
69327711277462655174453132592395707459388609204639225510317132301733062792284632881905076553154026830566914841216854185834581282223401691228770175626190033503632013222961469433
356998543478913099900849859778582298197228645379361162658826166716754959866435345862782186995164560368780960752199369159834857777182396690204026742512988964422768232047803315
9976598190422375225583769465483419754089950611057715076380216650332844672550967172878074563484609460715525932002063593729821755879503977303376658973108531280962272477660177582829
625454686628782583558364277232316848548607070766728327659929059953885835500794036625293206386145407806876311413708755071049471693451395709532298607668198389935201208204878534420
661702478173837528552790729167628517911045380105794753818197782945266440310638607170120427517571207183364508740853372699407506955421347478835245764184317691151881498984209867215
206388888387034647558816273815300394707547182634792481442515495545791088870113812249383820830889636503137500256366718169437091468874654753429513859296144359980078936964275108
4722487175213798912398994244028172146288477923567663246208277481679671897968136546725134233366929241508804979373527607974268685774699982619882216446148841014009612760582568411537
0308751726848319578258597259893626613388819193112305737341005023726017491682678559671042535029820108026671631016374394520566257281320298798887832759098019367970644671287915686
0191953427197464278115411669966358177745292305422079362365063827573192043824321516012880155
得到数值: 200
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes>

```

图 4.9: 隐私信息获取

在上图中，我们可以发现，成功地完成了隐私的获取，达到了我们的实验要求。

4.4 扩展实验探索

在扩展实验中，我们需要完成的是通过对称加密密钥来安全地实现隐私信息的获取。对称加密有很多种，比如 AES、DES、移位密码等等，我们在此处就选取 AES 来作为我们的对称加密体制。

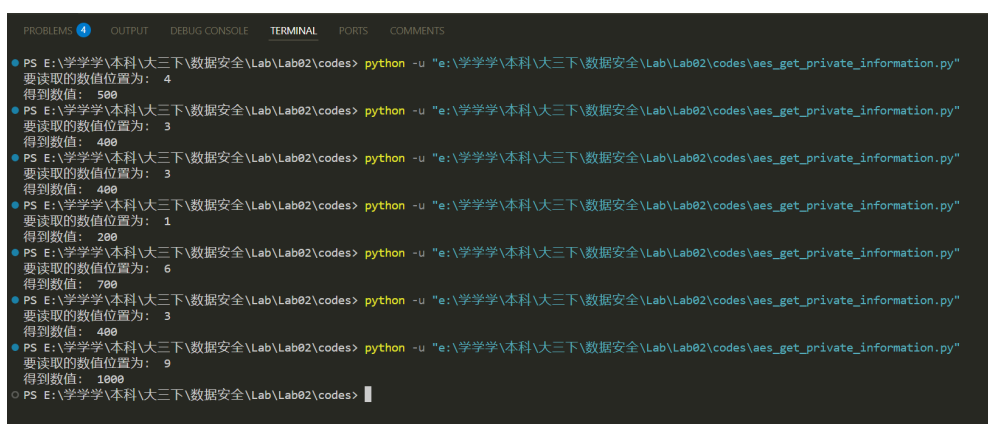
我们直接在源代码上进行改动即可。改动后的代码如下所示：

```
1  from Crypto.Cipher import AES
2  from Crypto.Util.Padding import pad, unpad
3  import random
4  import os
5
6  # 生成对称密钥  $k$ 
7  key = os.urandom(16) # AES 密钥长度为 16 字节 (128 位)
8
9  # 服务器端保存的明文消息列表
10 message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
11 length = len(message_list)
12
13 # 服务器端使用对称密钥  $k$  对消息进行加密
14 ciphertext_list = []
15 for message in message_list:
16     # 将消息转换为字节类型
17     message_bytes = str(message).encode('utf-8')
18     # 创建 AES 加密器
19     cipher = AES.new(key, AES.MODE_ECB)
20     # 填充消息
21     padded_message = pad(message_bytes, AES.block_size)
22     # 加密消息
23     ciphertext = cipher.encrypt(padded_message)
24     ciphertext_list.append(ciphertext)
25
26 # 客户端随机选择一个要读的位置
27 pos = random.randint(0, length - 1)
28 print(" 要读取的数值位置为: ", pos)
29
30 # 服务器将指定位置的密文发送给客户端
31 selected_ciphertext = ciphertext_list[pos]
32
33 # 客户端使用对称密钥  $k$  对密文进行解密
34 decipher = AES.new(key, AES.MODE_ECB)
35 # 解密密文
36 decrypted_bytes = decipher.decrypt(selected_ciphertext)
37 # 去除填充
```

```
38 unpadded_bytes = unpad(decrypted_bytes, AES.block_size)
39 # 将字节类型转换为整数
40 decrypted_message = int(unpadded_bytes.decode('utf-8'))
41
42 print(" 得到数值: ", decrypted_message)
```

我们利用 Crypto 等库实现了基于 AES 对称加密算法的信息交互系统。首先,通过 os.urandom 来生成 128 位密钥,对服务器端的整数消息列表依次转换为字节、采用 ECB 模式加密并填充后存入密文列表。客户端随机选定位置,服务器将对应密文发送给客户端,客户端使用相同密钥和模式进行解密,去除填充后将字节转换为整数,最终输出得到的明文数值,整个过程实现了对指定密文的加密存储与解密获取。

运行该代码,得到的结果如下所示:



```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 4
得到数值: 500
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 3
得到数值: 400
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 3
得到数值: 400
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 1
得到数值: 200
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 6
得到数值: 700
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 3
得到数值: 400
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> python -u "e:\学学学\本科\大三下\数据安全\Lab\Lab02\codes\aes_get_private_information.py"
要读取的数值位置为: 9
得到数值: 1000
PS E:\学学学\本科\大三下\数据安全\Lab\Lab02\codes> 
```

图 4.10: AES 加密后的结果

说明我们的 AES 加密实现了对应的功能,验证完成!

5 实验心得与体会

本次实验,我依次完成了实验的基础部分以及扩展部分,并对同态加密的理论知识更加了解了。最后在探索部分中,我使用 AES 加密体制来进行实现,使我对 AES 加密的使用更加熟练了。

总的来说,本次实验,我收获颇丰。希望在后续实验当中,我能够学习到更多的数据安全和隐私保护方面的知识。