



南開大學  
Nankai University

南 開 大 學

网络空间安全学院

数据安全实验报告

---

## 实验 2：半同态加密应用实践

---

姓名：付政烨

学号：2113203

年级：2021 级

专业：信息安全、法学

2024 年 4 月 1 日

# 目录

一、 实验要求	1
二、 实验内容	1
(一) 环境配置	1
(二) 基础实验	1
1. 即基于 Python 的 phe 库完成加法和标量乘法的验证	1
(三) 基于 Python 的 phe 库完成隐私信息获取的功能	3
(四) 扩展实验	5
1. 基于高级加密标准 (AES)	5
2. 基于新式流加密算法 (ChaCha20Poly1305)	6
三、 实验心得与体会	8

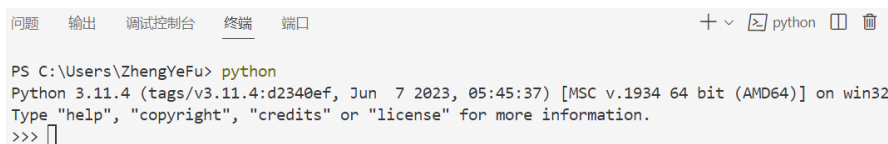
## 一、实验要求

- **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。

## 二、实验内容

### (一) 环境配置

#### 1. 安装 python 环境



```
PS C:\Users\ZhengYeFu> python
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 1

- #### 2. 安装 phe 库：
- 输入命令：pip install phe 完成 phe 库的安装。（Pip 是 python 的一个安装库的工具，可执行文件在 python 安装目录下可以找到。）



```
PS C:\Users\ZhengYeFu> 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)
    53.7/53.7 kB 462.7 kB/s eta 0:00:00
Installing collected packages: phe
Successfully installed phe-1.5.0

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

图 2

- #### 3. 验证环境正确性：
- 再次进入 python 环境，输入 python 代码：from phe import paillier。发现不出现错误信息，说明环境安装成功。



```
PS C:\Users\ZhengYeFu> python
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from phe import paillier
>>>
```

图 3

### (二) 基础实验

#### 1. 即基于 Python 的 phe 库完成加法和标量乘法的验证

测试课本给出一个集成的演示代码（第二章-实验 2.1）如下：

```

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

```

代码利用 Paillier 同态加密库演示了加密、解密和同态操作的基本流程。首先, 它显示了库默认的私钥大小并生成一对公私钥。接下来, 它使用公钥加密一组特定数据 (3.1415926, 100, -4.6e-12), 并记录加密所需时间, 同时展示了加密后的第一个数据的密文。然后, 代码使用私钥解密这些数据, 并计算解密所需的时间, 以及数据解密结果。在同态操作部分, 代码通过重载的运算符实现了加密数据的加法、减法、乘法和除法运算, 并验证了加法运算的同态性质。

```

PS C:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2> & "c:\Users\ZhengYeFu\AppData\Local\Programs\Python\Python311\python.exe" "c:\Users\ZhengYeFu\.vscode\extensions\ms-python.debugpy-2024.2.0-win32-x64\bundle
d\libs\debugpy_launcher\...\.debugpy_launcher" "51377" "--" "c:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2\test1.py"
默认私钥大小: 3072
加密耗时: 0.828845739364624
加密数据 (3,1415926): 323948746238884952075127178698127626651712740463980856745051149958655226995847360154717363714970308220267230797235148600618639799772293949019873076388316641985615304778912142007169663210449
1870170145236523270642249153308857539759852598667155345632524841631893565465062115271183481783058876802791532440702268617806244805271650126790286968099011313596230858510218718884664134921446298174512167540
2173315555984033611621855758639457122407098786937758801820483686597898776965388348892402987318185912344095808376033085128798035491827568418060835444191849315051371933953264680589804675774339547293445718919454242
7885461873090705975118892182514847171642121404650575978009985183802044715587687297474629010000013957170401957653435999843343615394414157049973519837148204586472710988373053635461523549996494397044518317188156684
205365737820371118829296150695375878019762642767899521660278915997039081803727170786695786638639065617285863098483219916032781287807779949756091457746700602731672029294071473722229162744655135518371533643513855
709929551396224304177821067963986738058265101972886882234316895202002509512078164565207142545477000506041547171733132084569419532342032450446845461705931395186941517784033504046090910845006236762079840740606123
027891590703980186372771707866957866386390656173858636984832199160327812878077799497560914577467006027316720292940714737222291627444655135518371533643513855780929551396224304177821067963986738058265101972886882234
31669520200250951207816456526714254547700050604154717173313208456941953234203245044684546170593139518694151778403350404609091084500623676207984074060612312219626349682047199230628911060188739795985139080339493
605895047715283437134344038210001169500947032578244932345652399854248793401177738640064098949235434814322975578096045179271039648312016301459546082461441506468546515925796091401029358368016777195234306292824938467
6524508540969549751957809993356800487653576961118239242064367620660901056769499181101269912338761008629019434504468454912613965216256828909646845676049706566111377801782688108975252047109796994279996835870785449
908231083917459245579560266884402404390938362763148475043257223053040563243583672862463269799238942
a=5 0.1415926
a-3 0.14159260000000007
b^6= 600
c/-10.0= 4.6e-13
True

```

图 4

### (三) 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  $data\_list = \{m_1, m_2, \dots, m_n\}$

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

进而，开发具体代码如下：

```

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

```

```
15 for i in range(length):
16     select_list.append( i == pos )
17     enc_list.append( public_key.encrypt( select_list [i]) )
18     ##### 服务器端进行运算
19 c=0
20 for i in range(length):
21     c = c + message_list[i] * enc_list [i]
22     print("产生密文: ",c.ciphertext())
23     ##### 客户端进行解密
24 m=private_key.decrypt(c)
25     print("得到数值: ",m)
```

本段落引用自教科书内容，指出原文描述并未深入详尽。以下部分旨在通过更为精细的阐述，对该协议的安全性进行分析：

### (1) 服务器端的数据隐私性

服务器端的数据隐私性基于 Paillier 加密协议的同态加密特性。同态加密允许对密文进行特定的算术运算，而不需要将其解密。在本设计中，这个特性被用于**执行密文上的标量乘法运算**。

1. 当服务器接收到加密的选择向量（enc\_list）时，它对每个数据项进行标量乘法运算。因为选择向量中除了客户端想要的数据位置是加密的 1 之外，其他都是加密的 0，根据 Paillier 的特性，乘以加密的 0 得到的还是加密的 0，乘以加密的 1 得到的是原数值的加密形式。
2. 这意味着在加和操作完成后，服务器发送回客户端的密文实际上只包含客户端所请求的那个数据项的加密形式，其他数据项由于乘以了加密的 0，对最终结果没有贡献。
3. 在整个过程中，服务器只处理密文，无法识别哪些数据被请求，保证了数据处理过程的隐私性。

### (2) 客户端的数据访问控制

客户端的数据访问控制是**通过密文选择向量的精确构造实现的，确保客户端仅能访问到特定位置的数据**。

1. 客户端构造选择向量时，在想要访问的数据位置放置 1，其他位置放置 0。这个向量经过 Paillier 公钥加密后，确保了只有特定位置被“选中”。
2. 由于 Paillier 加密的同态性质，只有加密的 1 与数据项相乘才会保留该数据项的值，加密的 0 与任何数据项相乘都会得到加密的 0，这意味着只有客户端“选中”的数据会被正确地返回。
3. 客户端通过解密返回的密文，即可获取到所需的数据，而无需获取或解密服务器上的其他数据。这不仅保证了客户端的访问控制权，也避免了不必要的数据泄露。

### (3) 安全性保障的原理

- **加密隔离**：所有敏感操作都在加密形态下进行，即使是服务器也不能看到明文数据，保证了信息的隐私性和安全性。
- **最小化数据暴露**：通过精确控制加密向量，确保仅有必要的数据被处理和暴露，遵循数据处理的最小必要原则。



图 5

#### (四) 扩展实验

本次实验是基于课本给出的第二章-实验 2.2 代码的基础上完成的吗，旨在深入探索不同加密技术在客户端-服务器模型中的应用。本实验通过构建一个客户端与服务器间通信的简易示例，依次采用高级加密标准（AES）与新兴流加密算法 ChaCha20Poly1305，展示了隐私信息安全获取的过程。此实践有助于深入掌握对称加密机制的原理，特别是上述两种加密技术的工作机制及其在现代加密通讯中的实际应用。同时，本实验进一步加强了对于客户端-服务器架构中，数据安全性及隐私保护重要性的理解。

##### 1. 基于高级加密标准（AES）

AES（高级加密标准）是一种广泛使用的对称加密算法，用于保护电子数据的安全。AES 由美国国家标准与技术研究院（NIST）在 2001 年正式采纳为标准，它是一种块加密算法，意味着它以固定大小的数据块进行加密（通常是 128 位）。AES 支持多种密钥长度（128 位、192 位和 256 位），加密强度随密钥长度的增加而提高。由于其高效性、安全性和易用性，AES 广泛应用于各种软件和硬件中，以保护传输和存储的数据不被未经授权访问。

```

1 import random
2 from cryptography.fernet import Fernet
3
4 # 设置参数
5 message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
6 length = len(message_list)
7 pos = random.randint(0, length - 1)
8 print("要读取的数值位置为:", pos)
9 # 生成密钥
10 key = Fernet.generate_key()
11 cipher = Fernet(key)
12 # 客户端生成选择向量
13 select_list = [1 if i == pos else 0 for i in range(length)]
14 # 服务器端加密消息列表
15 encrypted_list = [cipher.encrypt(str(num).encode()) for num in message_list]
16 # 客户端请求指定位置的密文
17 encrypted_selected = encrypted_list[pos]
18 # 客户端进行解密
19 decrypted_selected = cipher.decrypt(encrypted_selected)
20 selected_number = int(decrypted_selected.decode())
21

```



```
22 print("得到数值:", selected_number)
```

代码演示了如何使用基于高级加密标准 (AES) 的 Fernet 模块安全地在客户端和服务端之间传输和访问指定的数据。在这个模型中, 服务器端首先将一系列预定义的数值进行加密并存储。客户端通过生成一个密钥和对应的加密器来安全地选择并请求服务器上的特定数据。通过对所选数据的位置进行加密请求, 客户端能够从服务器接收到加密的数据, 然后使用相同的密钥进行解密, 以获取原始数值。整个过程强调了数据在存储和传输过程中的安全性, 同时确保了客户端可以有效且安全地访问服务器端存储的数据, 而不泄露选择的具体位置信息。

```
PS C:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2> c::; cd
\\.vscode\extensions\ms-python.debugpy-2024.2.0-win32-x64\bundled
要读取的数值位置为: 1
得到数值: 200
PS C:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2> █
```

图 6

## 2. 基于新式流加密算法 (ChaCha20Poly1305)

ChaCha20Poly1305 是由 Daniel J. Bernstein 设计的一种组合加密标准, 融合了高性能的流加密算法 ChaCha20 与高安全性的认证标签算法 Poly1305。这种加密方案不仅保障了数据的机密性和完整性, 而且还能确保数据传输的真实性, 适合在各种环境下, 尤其是性能受限的环境中高效运作。作为一种新式的加密算法, ChaCha20Poly1305 因其卓越的性能和安全性, 被 Google 采用并推广使用, 广泛应用于安全通信协议中, 如 TLS 和 QUIC, 以加强网络数据的保护。

使用 ChaCha20 进行加密替代 Fernet 的情况下, 我们需要做一些调整, 因为 ChaCha20 与 Fernet 在使用上有所不同。**ChaCha20 是一个流密码, 它需要一个随机的 nonce (一次性数字), 用于加密过程, 而且它不像 Fernet 那样自动处理数据的加密和解密。**这意味着我们需要手动管理 nonce, 并确保在加密和解密时使用相同的 nonce。下面是使用 ChaCha20 代替 Fernet 的代码示例。(注意: 由于 Python 的 cryptography 库支持 ChaCha20, 但使用方式略有不同)

```
1 import os
2 import random
3 from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305
4
5 # 设置参数
6 message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
7 length = len(message_list)
8 pos = random.randint(0, length - 1)
9 print("要读取的数值位置为:", pos)
10
11 # 生成密钥
12 key = ChaCha20Poly1305.generate_key()
13 cipher = ChaCha20Poly1305(key)
14
15 # 客户端生成选择向量 (本示例中未直接使用, 但提供了如何操作的示例)
16 select_list = [1 if i == pos else 0 for i in range(length)]
17
18 # 服务器端加密消息列表
```



```
19 # 对每个消息使用独立的nonce
20 nonce_list = [os.urandom(12) for _ in message_list] # 生成一个随机nonce列表
21 encrypted_list = [cipher.encrypt(nonce_list[i], str(num).encode(), None)
22                     for i, num in enumerate(message_list)]
23
24 # 客户端请求指定位置的密文及其nonce
25 encrypted_selected, nonce_selected = encrypted_list[pos], nonce_list[pos]
26
27 # 客户端进行解密
28 decrypted_selected = cipher.decrypt(nonce_selected, encrypted_selected, None)
29 selected_number = int(decrypted_selected.decode())
30
31 print("得到数值:", selected_number)
```

上述代码展示了如何使用 ChaCha20Poly1305，一种支持认证加密的算法，来安全地加密和解密特定的数据项。首先，它定义了一个数据列表 `message_list`，其中包含了一系列的数值，并随机选择一个位置 `pos` 来模拟客户端想要安全访问的数据项的位置。为了确保加密过程的安全性，代码生成了一个随机密钥，并为列表中的每个元素生成了一个唯一的随机 `nonce`（一次性数字），这是 ChaCha20Poly1305 算法加密和解密过程的要求。对于每个数据项，使用相应的 `nonce` 和生成的密钥进行加密，得到加密后的数据列表 `encrypted_list`。然后，模拟客户端通过选择加密列表中对对应位置的加密数据和 `nonce`，使用相同的密钥和 `nonce` 进行解密操作，以获取原始数据值。以下是对代码的一些关键点说明：

- **Nonce 管理**：在加密通信中，Nonce（Number used ONCE，即一次性数字）是一个只被使用一次的随机或伪随机数值。它的主要作用是增加加密方案中的随机性和安全性，防止重放攻击（即攻击者重复发送之前拦截的加密消息）和确保即使同一消息被多次加密，每次产生的密文也是不同的。在上述代码中，通过 `os.urandom(12)` 生成了一个 12 字节的随机 Nonce。这个函数产生的是足够随机的数据，适合用于加密操作中。12 字节长度是因为 ChaCha20Poly1305 算法要求的 Nonce 大小正好是 12 字节。在加密每个消息时，为每个消息生成一个新的 Nonce 并与之一起加密。在解密时，必须使用相同的 Nonce 进行解密。这意味着，Nonce 需要与密文一起安全地存储或传输，以便接收方能正确解密消息。
- **加密和解密**：加密时，需要提供一个一次性使用的 Nonce、明文数据以及可选的附加数据（用于验证但不加密），以生成密文和确保数据的完整性。解密时，则需使用相同的 Nonce 和附加数据（如果初次加密时使用了）来正确解密密文并验证数据是否在传输过程中被篡改，从而保障了加密通信的安全性和完整性。
- **安全性和性能**：ChaCha20Poly1305 提供了很高的安全性，包括保密性和认证性。Poly1305 部分为每个加密的消息生成一个认证标签（MAC），这个标签在解密时会被验证，以确保数据在传输过程中未被篡改。

```
PS C:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2> c:; cd
.\vscode\extensions\ms-python.debugpy-2024.2.0-win32-x64\bundled
要读取的数值位置为: 5
得到数值: 600
PS C:\Users\ZhengYeFu\Desktop\大三下\数据安全\实验\lab2> █
```

图 7

### 三、 实验心得与体会

在本次实验中,我深入探索了使用 Python 中的 `phe` 库以及加密技术如 AES 和 ChaCha20Poly1305 在客户端-服务器模型中的应用。此实验不仅加深了我对对称加密机制原理的理解,特别是两种加密技术的工作原理及其在现代加密通信中的应用,还增进了我对客户端-服务器架构中数据安全性和隐私保护重要性的认识。

通过实践,我学习到了 Paillier 加密系统的同态特性,它允许在加密数据上直接进行算术操作,如加法和标量乘法。这一特性在实现数据的隐私保护方面极具价值,因为它允许对数据进行加工而不暴露原始数据。例如,在加法和标量乘法验证中,即使数据被加密,我们仍然能够执行计算,并且验证计算的正确性,这对于需要保密的数据分析和处理非常有用。

进一步地,实验通过 AES 和 ChaCha20Poly1305 两种加密方法展示了如何安全地在客户端和服务器之间传输数据。这两种方法虽然在实现上有所不同,但都有效地确保了数据传输的安全性。AES 作为一种块加密标准,其安全性和效率已被广泛认可。而 ChaCha20Poly1305 作为一种较新的流加密算法,它通过提供认证加密确保了数据的机密性、完整性和真实性。在实验中,我了解到管理 `nonce` 值在使用 ChaCha20Poly1305 时至关重要,因为它确保了加密的随机性和安全性。

此外,实验也让我认识到,在设计安全通信协议时,需要仔细考虑数据加密、密钥管理、`nonce` 管理等多个方面,以确保数据传输的安全和效率。通过这次实验,我不仅提升了编程技能,更重要的是,增强了我对加密技术在保护数据安全和隐私方面的应用理解。