

南开大学

数据安全课程实验报告

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：2113997

姓名：齐明杰

班级：信安2班

1 实验目的

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。

2 实验原理

2.1 同态加密

同态加密 (Homomorphic Encryption, HE) 是将数据加密后，对加密数据进行运算处理，之后对数据进行解密，解密结果等同于数据未进行加密，并进行同样的运算处理。同态加密的概念最初在1978年，由Ron Rivest, Leonard Adleman和Michael L. Dertouzos共同提出，旨在解决在不接触数据的前提下，对数据进行加工处理的问题。

目前，同态加密支持的运算主要为加法运算和乘法运算。按照其支持的运算程度，同态加密分为**半同态加密** (Partially Homomorphic Encryption, PHE) 和**全同态加密** (Fully Homomorphic Encryption, FHE)。半同态加密在数据加密后只支持加法运算或乘法运算中的一种，根据其支持的运算的不同，又称为加法同态加密或乘法同态加密。半同态加密由于机制相对简单，相对于全同态加密技术，拥有着更好的性能。全同态加密对加密后的数据支持任意次数的加法和乘法运算。

2.2 Paillier算法

Paillier算法是一种公钥密码体系，由法国学者Pascal Paillier在1999年提出。该算法的最大特点是其具有加法同态性质，即能够在加密数据的状态下进行数学加法操作，并在解密后得到与明文直接相加相同的结果。这一特性使得Paillier算法在隐私保护、安全计算等领域非常有用。下面我将详细介绍Paillier算法的基本原理以及它在实现隐私信息获取方面的应用。

Paillier算法基本原理:

Paillier加密体系包括密钥生成、加密和解密三个主要步骤:

1. 密钥生成

- 选择两个大素数 p 和 q ，计算 $n = pq$ 和 $\lambda = lcm(p - 1, q - 1)$ (lcm 表示最小公倍数)
- 选择一个随机数 g ，其中 g 在模 n^2 的乘法群中。
- 计算 $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ ，这个 $L(x) = \frac{x-1}{n}$
- 公钥为 (n, g) ，私钥为 (λ, μ) 。

2. 加密

- 对于一个明文消息，选择一个随机数，其中 $0 < m < n$ 且 $\gcd(r, n) = 1$ 。
- 密文 c 计算为 $c = g^m \cdot r^n \bmod n^2$ 。

3. 解密

- 给定一个密文 c ，利用私钥 (λ, μ) 计算明文 $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$ 。

Paillier算法的一个关键特性是其**加法同态性**，具体表现为：

- 如果有两个明文 m_1 和 m_2 ，它们的加密密文分别为 c_1 和 c_2 ，那么 $m_1 + m_2$ 的密文就可以通过 $c_1 \cdot c_2 \bmod n^2$ 计算得到。
- 这意味着我们可以在不解密密文的情况下，直接对加密后的数据进行加法运算。

其另一个特性是**标量乘同态性**，具体表现为：

- 给定一个明文 m 和一个标量 k ，以及 m 的加密形式 $c = g^m \cdot r^n \bmod n^2$ ，我们可以在不直接解密 c 的情况下，计算出 km 的加密形式。
- 这可以通过将密文 c 乘以 g^k 的 n 次方模 n^2 实现，即 $c' = c^k \bmod n^2$ 。

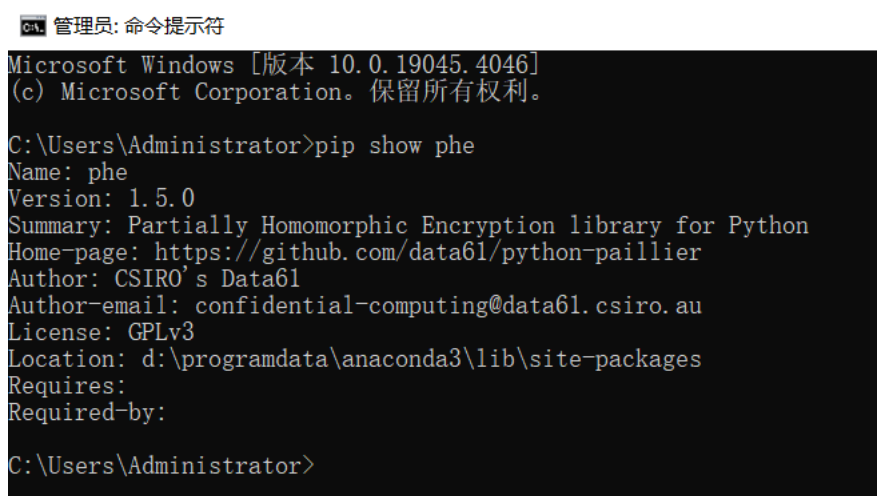
3 实验过程

3.1 环境配置

- 使用 `phe` 库实现 *Paillier* 加密算法。该库提供了 *Paillier* 公钥密码体系的实现，包括密钥生成、加密和解密等功能。
- *Python* 编程语言进行实验代码的编写。

安装库phe:

```
1 pip install phe
2 pip show phe
```



管理员: 命令提示符

```
Microsoft Windows [版本 10.0.19045.4046]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Administrator>pip show phe
Name: phe
Version: 1.5.0
Summary: Partially Homomorphic Encryption library for Python
Home-page: https://github.com/data61/python-paillier
Author: CSIRO's Data61
Author-email: confidential-computing@data61.csiro.au
License: GPLv3
Location: d:\programdata\anaconda3\lib\site-packages
Requires:
Required-by:

C:\Users\Administrator>
```

版本为 `Version1.5.0`。

下面是一个集成了 `phe` 库几个基本用法（密钥生成、数据加密、数据解密、以及利用同态性质进行加法运算）的 *Python* 代码示例：

```
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
    t(a*b))

```

运行结果:

```

• (base) PS D:\study\大三\数据安全\实验\2> cd d:/study/大三/数据安全/实验/2
• (base) PS D:\study\大三\数据安全\实验\2> & D:/ProgramData/Anaconda3/python.exe d:/study/大三/数据安全/实验/2/use.py
默认私钥大小: 3072
加密耗时s: 0.6424469947814941
加密数据 (3.1415926) : 361788185804293110030821078633591247324725126363940529894942841773749527220708904161750031806404441
27628290415162494358225951183290045319865298362813237839036756809046929786170183812823010499765099773164598905396093461224
18070007641650684727429260552654021256184169951842448896603432257327084472943523317801698595931743342329513647517314381933
29483849731549803986402215418485149273191203463403981855620782619979762158369998129300967884207507289716461126203291263810
12476981607009513691253795779173002631414106516084066239294700613812730505880623599514925547996981747959675413983686654734
81600940463573472993822640112431428458756727825275502144398989440151550897768491232246154868001805487672122824563046473294
39820391135718556388934015298137480417029464858247704321356024902906742135313339759064382871767120092438652433411426716417
83563125283482202940603226183290198169553409945723311493149484662048268983958896787440217812439418538101522816298395248270
92764965492533994383292512319021692196612845706073361307188917091640855431188472539231989155951761185495237985070854319826
15852404070515024886786430041170624531776979933584733387358393747491371592965731299263584979114884755876460602250668893919
95361984078131066344753415170797155133063595518719889513002926797852160368039502013739926061366913654306112410984875542259
84624961157662759869484048447033418777956598909032091651114687952484490824982956569479316612712608124173300627925438320856
94400011757831852121548914347274098012529479541209582048823687221381216562336728626515901711304800349239376597216537359973
解密耗时s: 0.18174242973327637
原始数据 (3.1415926) : 3.1415926
a+5 密文: 3617881858042931100308210786335912473247251263639405298949428417737495272207089041617500318064044413495962772581
24943582259511832900453198652983628132378390367568090469297861701838128230104997650997731645989053960934612242398728754607
06847274292605526540212561841699518424488966034322573270844729435233178016985959317433423295136475173143819332369687574651
98039864022154184851492731912034634039818556207826199797621583699981293009678842075072897164611262032912638105719274483192
951369125379577917300263141410651608406623929470061381273050588062359951492554799698174795967541398368665473487142066085959
34729938226401124314284587567278252755021443989894401515508977684912322461548680018054876721228245630464732949361962338739

```

```
399438329251231902169219661284570607336130718891709164085
502488678643004117062453177697993358473338735839374749137
106634475341517079715513306359551871988951300292679785216
275986948404844703341877795659890903209165111468795248449
185212154891434727409801252947954120958204882368722138121
a+5= 8.1415926
a-3 0.14159260000000007
b*6= 600
c/-10.0= 4.6e-13
True
(base) PS D:\study\大三\数据安全\实验\2>
```

3.2 基于Paillier算法实现隐私信息获取

在基于Paillier算法实现隐私信息获取的场景中，客户端与服务器之间的数据交互过程包含了密钥生成、加密选择向量的生成、加密信息的交换、以及最终信息的解密过程。

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。利用这个特性，下面详细解析整个过程：

1. 密钥生成

- **客户端操作：**使用Paillier算法生成公私钥对。公钥将用于加密信息，而私钥用于解密。这一步骤是整个隐私信息获取过程的基础。

2. 加密选择向量的生成

- **客户端操作：**客户端确定它希望从服务器获取的消息的索引。然后，客户端生成一个选择向量，向量的长度与服务器消息的数量相等。在这个向量中，只有对应客户端想获取的消息索引位置的元素为1，其余均为0。客户端使用公钥将这个向量加密，得到加密的选择向量。

3. 加密信息的交换

- **客户端到服务器：**客户端将加密的选择向量发送给服务器。
- **服务器操作：**服务器收到加密的选择向量后，使用它与存储的消息进行运算。具体来说，服务器对每一个消息进行加权，使用选择向量中的加密0或加密1与每条消息相乘，然后将所有的结果相加得到一个加密的总和。这个加密的总和包含了客户端想要的那条消息的加密形式，而由于Paillier加密的同态性质，这一切操作都在加密状态下完成，服务器无法知道客户端选择的是哪条消息。

4. 最终信息的解密

- **服务器到客户端：**服务器将加密的总和发送回客户端。
- **客户端操作：**客户端收到加密的总和后，使用私钥对其进行解密，得到其所需要的消息。

流程如下图所示：

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 pos
- 生成选择向量 $\text{select_list}=\{0, \dots, 1, \dots, 0\}$ ，其中，仅有 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 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

对应代码如下：

```
1 from phe import paillier
2 import random
3
4 # 假设的服务器端消息
5 server_messages = [10, 20, 30, 40, 50] # 服务器保存的消息列表
6 length = len(server_messages)
7
8 # 生成Paillier密钥对
9 public_key, private_key = paillier.generate_paillier_keypair()
10
11 # 客户端知道它想要的消息索引
12 index = random.randint(0, len(server_messages) - 1)
13 print(f"客户端想要的消息索引是：{index}")
14
15 # 客户端生成加密的选择向量
16 select_vector = [public_key.encrypt(int(i == index)) for i in
17 range(length)]
18 print(f"客户端生成的加密选择向量是：{select_vector}")
19
20 # 服务器计算加密的总和
21 encrypted_sum = sum(server_messages[i] * select_vector[i] for i in
22 range(length))
23 print(f"服务器计算的加密总和是：{encrypted_sum}")
24
25 # 客户端解密以获取感兴趣的消息
26 desired_message = private_key.decrypt(encrypted_sum)
27 print(f"客户端获取的消息是：{desired_message}")
```

运行结果：


```

• (base) PS D:\study\大三\数据安全\实验\2> & D:/ProgramData/Anaconda3/python.exe d:/study/大三/数据安全/实验/2/1.py
客户端想要的消息索引是：2
客户端生成的加密选择向量是：[<phe.paillier.EncryptedNumber object at 0x00000193B94824C0>, <phe.paillier.EncryptedNumber
2460>, <phe.paillier.EncryptedNumber object at 0x00000193B9482550>, <phe.paillier.EncryptedNumber object at 0x00000193B9
ncryptedNumber object at 0x00000193B94825B0>]
服务器计算的加密总和是：<phe.paillier.EncryptedNumber object at 0x00000193B94826A0>
客户端获取的消息是：30
○ (base) PS D:\study\大三\数据安全\实验\2>

```

3.3 拓展实验

我们可以通过结合Paillier同态加密和简单的异或(XOR)对称加密技术，安全地实现隐私信息的获取。在客户端保存对称密钥 k ，而服务器端存储使用对称密钥 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。

1. **对称密钥共享**：实验开始前，客户端和服务端通过一个安全的通道共享对称密钥 k 。这个密钥在后续的异或加密和解密过程中将被用到。
2. **服务器端消息加密**：服务器端拥有一个消息列表，使用对称密钥 k 对每条消息进行异或加密。加密操作是通过将每条消息与密钥 k 进行异或运算来完成的。这些加密后的消息将被存储在服务器上，等待客户端的查询。
3. **Paillier密钥对生成**：客户端使用Paillier算法生成公钥和私钥。公钥将用于加密选择向量，而私钥将用于解密服务器返回的加密总和。
4. **生成加密的选择向量**：客户端决定它想获取的消息的索引，并生成一个选择向量，其中只有所选消息的位置为1，其余为0。然后，客户端使用Paillier公钥加密这个选择向量。
5. **服务器计算加密总和**：服务器接收到加密的选择向量后，对其存储的加密消息进行处理，计算出加密的总和。这个过程利用了Paillier同态加密的特性，使得服务器无需知道客户端所需的具体消息内容。
6. **客户端获取并解密消息**：客户端接收到服务器返回的加密总和后，使用Paillier私钥进行解密，获取到加密的消息。随后，客户端再次使用对称密钥 k 对这条加密的消息进行异或操作，解密得到原始的明文消息。

代码如下：

```

1  from phe import paillier
2  import random
3
4  # 假设的服务器端消息和对称密钥k
5  server_messages = [10, 20, 30, 40, 50]
6  k = random.randint(1, 100) # 对称密钥k，假设已安全共享给客户端和服务端
7  length = len(server_messages)
8
9  # 使用对称密钥k对服务器消息进行异或加密
10 encrypted_server_messages = [msg ^ k for msg in server_messages]
11
12 # 生成Paillier密钥对
13 public_key, private_key = paillier.generate_paillier_keypair()
14
15 # 客户端知道它想要的消息索引

```

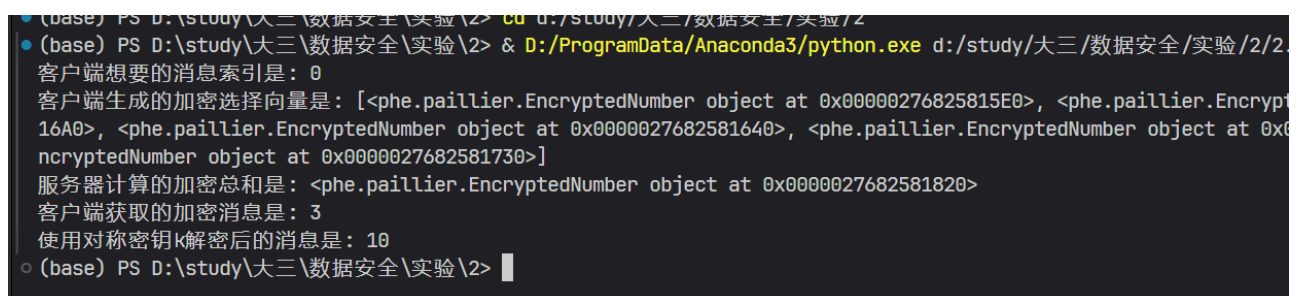


```

16 index = random.randint(0, len(encrypted_server_messages) - 1)
17 print(f"客户端想要的消息索引是: {index}")
18
19 # 客户端生成加密的选择向量
20 select_vector = [public_key.encrypt(int(i == index)) for i in
range(length)]
21 print(f"客户端生成的加密选择向量是: {select_vector}")
22
23 # 服务器计算加密的总和
24 encrypted_sum = sum(encrypted_server_messages[i] * select_vector[i] for i
in range(length))
25 print(f"服务器计算的加密总和是: {encrypted_sum}")
26
27 # 客户端解密以获取感兴趣的加密消息
28 desired_encrypted_message = private_key.decrypt(encrypted_sum)
29 print(f"客户端获取的加密消息是: {desired_encrypted_message}")
30
31 # 使用对称密钥k对加密的消息进行解密
32 desired_message = desired_encrypted_message ^ k
33 print(f"使用对称密钥k解密后的消息是: {desired_message}")

```

运行结果如下:



```

(base) PS D:\study\大三\数据安全\实验\2> cd D:/study/大三/数据安全/实验/2/
(base) PS D:\study\大三\数据安全\实验\2> & D:/ProgramData/Anaconda3/python.exe d:/study/大三/数据安全/实验/2/2/
客户端想要的消息索引是: 0
客户端生成的加密选择向量是: [<phe.paillier.EncryptedNumber object at 0x00000276825815E0>, <phe.paillier.Encrypt
16A0>, <phe.paillier.EncryptedNumber object at 0x0000027682581640>, <phe.paillier.EncryptedNumber object at 0x0
ncryptedNumber object at 0x0000027682581730>]
服务器计算的加密总和是: <phe.paillier.EncryptedNumber object at 0x0000027682581820>
客户端获取的加密消息是: 3
使用对称密钥k解密后的消息是: 10
(base) PS D:\study\大三\数据安全\实验\2>

```

4 实验心得

完成这个基于Paillier算法实现隐私信息获取的实验, 以及其扩展实验, 让我深刻体会到了同态加密技术在数据隐私保护方面的强大应用潜力。通过这次实验, 我不仅学习了Paillier算法的基本原理和操作, 还实践了如何在保护隐私的前提下进行安全的数据交换。

Paillier算法的加法同态特性为我们在不暴露具体内容的情况下, 对加密数据进行操作提供了可能。这种技术可以广泛应用于云计算、多方安全计算等领域, 特别是在需要数据共享但又又要保护数据隐私的场景中极具价值。

此外, 通过结合简单的异或对称加密技术, 实验的扩展部分展示了如何安全地处理加密数据, 同时保证数据传输过程的安全性和数据本身的隐私性。这不仅加深了我对同态加密技术的理解, 也让我对数据加密和隐私保护有了更全面的认识。