

南開大學

《数据安全》课程实验报告

实验 2：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_____2112060
姓 名_____孙璐

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。

二、实验原理

1. Paillier 算法描述

Paillier 方案满足加密方案的标准安全定义:语义安全,即在选择明文攻击下的密文的不可区分性。

(1) 密钥生成

- 随机选择两个质数 p 和 q , 尽可能地保证 p 和 q 的长度接近或相等(安全性高);
- 计算 $n=pq$ 和 $\lambda=\text{lcm}(p-1, q-1)$, 其中 lcm 表示最小公倍数;
- 随机选择 $g \in Z_n^*$, 考虑计算性能优化, 通常会选择 $g=n+1$;
- 计算 $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$, 其中 $L(x) = \frac{x-1}{n}$;
- 公钥为 (n, g) ;
- 私钥为 (λ, μ)

(2) 加密算法

对于任意明文消息 $m \in Z_n$, 任意选择一个随机数 $r \in Z_n^*$, 计算得到密文

$$c: c = E(m) = g^m r^n \bmod n^2$$

注意: 密文 c 要比明文 m 长度要长。

(3) 解密算法

对于密文 $c \in Z_{n^2}^*$, 计算得到明文 m : $m = D(c) = L(c^\lambda \bmod n^2) * \mu \bmod n$

2. 半同态加密虽然还不能同时支持加法和乘法运算, 不能支持任意的计算, 但是因为其与全同态相比, 具有较高性能, 因此, 仍然具有极为广泛的应用场景,

且在现实应用中起到了中重要的作用。一类典型的应用体现在隐私保护的数据聚合上。由于加法同态加密可以在密文上直接执行加和操作，不泄露明文，在到多方协作的统计场景中，可完成安全的统计求和的功能。在加密数据库 SQL 查询场景，在数据库不可信的情况下，可以通过部署协议和代理来保护请求者的查询隐私。其中，PHE 可以用来完成安全数据求和、均值的查询。

3. 隐私信息获取：基于 Paillier 协议进行设计。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。利用这个特性，可进行如下设计

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 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

4. 3DES

扩展实验使用 3DES 完成。

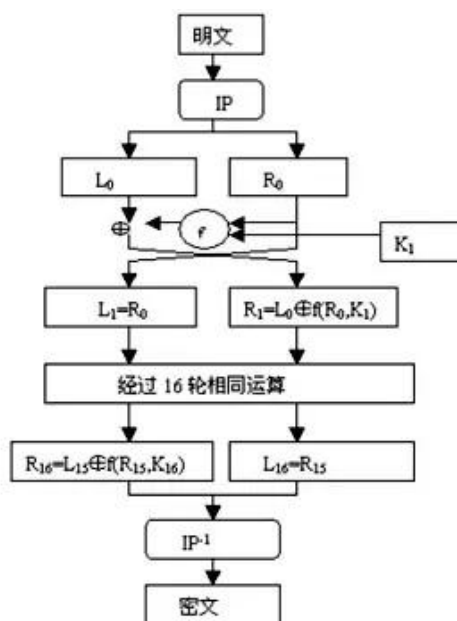
DES 使用 56 位的密钥和 64 位的明文块进行加密。DES 算法的分组大小是 64 位，因此，如果需要加密的明文长度不足 64 位，需要进行填充；如果明文长度超过 64 位，则需要使用分组模式进行分组加密。

DES 算法的分组大小是 64 位，由于 DES 算法的密钥长度只有 56 位，因此 DES 算法存在着弱点，容易受到暴力破解和差分攻击等攻击手段的威胁。因此，在实际应用中 DES 已经被更加安全的算法所取代，如 AES 算法等。

(1) DES 加密

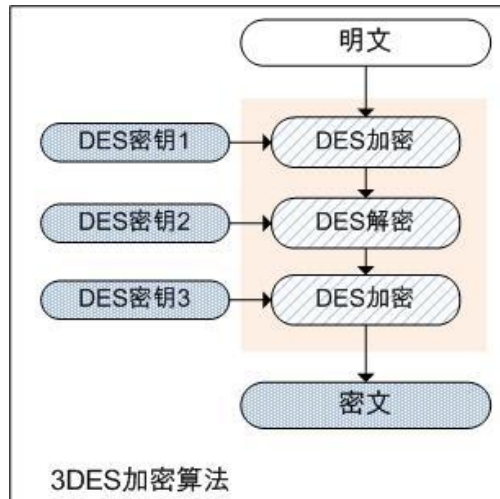
当输入了一条 64 位的数据之后，DES 将通过以下步骤进行加密。

- A. 初始置换（IP 置换）：将输入的 64 位明文块进行置换和重新排列，生成新的 64 位数据块。
- B. 加密轮次：DES 加密算法共有 16 个轮次，每个轮次都包括四个步骤
- 将 64 位数据块分为左右两个 32 位块。
 - 右侧 32 位块作为输入，经过扩展、异或、置换等操作生成一个 48 位的数据块。这个 48 位的数据块被称为“轮密钥”，它是根据加密算法的主密钥生成的子密钥。
 - 将左侧 32 位块和轮密钥进行异或运算，结果作为新的右侧 32 位块。
 - 将右侧 32 位块与原来的左侧 32 位块进行连接，生成一个新的 64 位数据块，作为下一轮的输入。
- C. 末置换（FP 置换）：在最后一个轮次完成后，将经过加密的数据块进行置换和重新排列，得到加密后的 64 位密文。



(2) 3DES

3DES 就是使用 DES 加密 3 次，使用 3 个密钥进行加解密。3DES 使用了三个密钥，将 DES 算法的加密过程重复三次，从而大大增强了安全性。当三重密钥均相同时，前两步相互抵消，相当于仅实现了一次加密，因此可实现对普通 DES 加密算法的兼容。3DES 的密钥长度为 168 位，远高于 DES 算法的 56 位密钥长度，但仍有可能被暴力破解。



三、 实验过程

(一) 环境配置

1. 在 Windows 下安装 python 环境,注意安装过程要勾选 Add Python.exe to PATH 将 python 程序路径加到系统的环境变量中。

安装完毕打开控制台,输入 python 指令显示如下表明已经安装成功并进入 python 运行环境。

```
C:\Users\LENOVO>python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. 安装 phe 库

输入 pip install phe 完成 phe 库的安装。安装完成后,进入 python 环境,输入 from phe import paillier, 如果不出现错误信息,说明环境安装成功

```
C:\Users\LENOVO>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 146.5 kB/s eta 0:00:00
Installing collected packages: phe
Successfully installed phe-1.5.0
C:\Users\LENOVO>
```

```
C:\Users\LENOVO>python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from phe import paillier
>>>
```

(二) 基于 Python 的 phe 库完成加法和标量乘法的验证

```
from phe import paillier #开源库
import time #做性能测试

#####设置参数
print("默认私钥大小: ",paillier.DEFAULT_KEYSIZE)#生成公私钥
# 默认私钥大小为 3072

# 生成公私钥
public_key,private_key =paillier.generate_paillier_keypair()

#测试需要加密的数据（设置 3 个要加密的数据）
message_list =[3.1415926,100,-4.6e-12]

#####加密操作
time_start_enc=time.time()
# 内嵌 for 循环
# 使用 public_key.encrypt 完成加密
encrypted_message_list=[public_key.encrypt(m) for m in message_list]
time_end_enc=time.time()
print("加密耗时 s:",time_end_enc-time_start_enc)
print("加密数据 (3,1415926):",encrypted_message_list[0].ciphertext())

#####解密操作
time_start_dec=time.time()
# 内嵌 for 循环
decrypted_message_list=[private_key.decrypt(c) for c in
encrypted_message_list]
time_end_dec=time.time()
# 使用 private_key.decrypt 完成解密
```

```
print("解密耗时 s:",time_end_dec-time_start_dec)
print("原始数据(3.1415926):",decrypted_message_list[0])

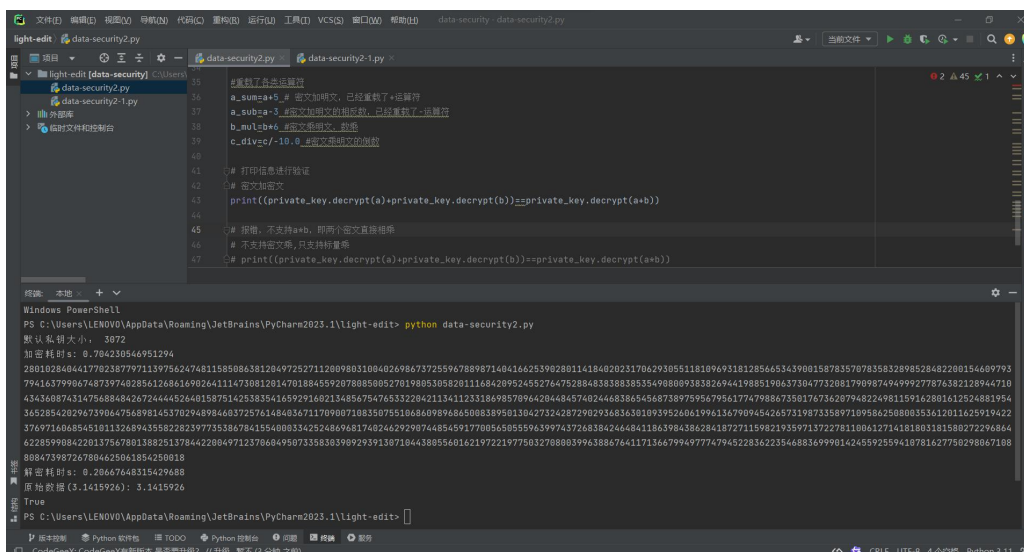
#####测试加法和乘法同态
a,b,c=encrypted_message_list #a,b,c 分别为对应密文

#重载了各类运算符
a_sum=a+5 # 密文加明文, 已经重载了+运算符
a_sub=a-3 #密文加明文的相反数, 已经重载了-运算符
b_mul=b*6 #密文乘明文, 数乘
c_div=c/-10.0 #密文乘明文的倒数

# 打印信息进行验证
# 密文加密文
print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a+b))

# 报错, 不支持 a*b, 即两个密文直接相乘
# 不支持密文乘, 只支持标量乘
#print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))
```

运行后截图如下:



首先定义了需要加密的数据，并自动生成公钥与私钥，默认私钥大小为 3072。在完成数据测试后，首先通过公钥对数据 3.1415926 进行加密，并显示加密耗时间与加密数据内容。完成加密操作后根据私钥对密文进行解密。

解密耗时相对加密耗时更短，与 Paillier 的性质相符，即在加密过程中需要做两次指数运算，而解密只需要一次。

（三）利用 Python phe 库完成隐私信息获取实验

```
from phe import paillier # 开源库

import random # 选择随机数

##### 设置参数

# 服务器端保存的数值
message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]

length = len(message_list)

# 客户端生成公私钥，选择读取的位置
public_key, private_key = paillier.generate_paillier_keypair()

pos = random.randint(0, length-1)

#####客户端生成密文选择向量
select_list=[]
```



```

enc_list=[]
for i in range(length):
    select_list.append(i == pos)
    enc_list.append(public_key.encrypt(select_list[i]))

##### 服务器端进行运算

c = 0
for i in range(length):
    c = c + message_list[i] * enc_list[i]
print("产生密文: ",c.ciphertext())

##### 客户端进行解密

m = private_key.decrypt(c)
print("得到数值: ",m)

```

运行后截图如下

```

PS C:\Users\LENOVO\AppData\Roaming\JetBrains\PyCharm2023.1\light-edit> python data-security2-1.py
产生密文:  4547539802784062975613373235467240859173966615888422146376337460226161966789806993146602380291689973503627201966347147447983983541217741417428735779879568398991592065780538
25113842228184419115309897118286565938857759439145099468883657289219192241458091278353311735941664384212983026800321817103268696763152801980861105588620612577543718275840553843461
510523729532168095409043868072680854572318938417094814991892872989661025449304373799279960208089941154014265162056630243881165105107189342322430594363891588707596121419883330715449735
0640806422625438391541229799085364193148290737021720329489877788254158649303093727054937017662383726812070455685124014257489917486719295891844593956121517417658884381306511023577958066
271747643558057715142182300168935750132512891487437756315466825293194069519989335789478907352653886475680322413951153409654924966105963466448935077808464836804701253356580698420576414
30138443045807485043656823659883728195940853780033929326137040692881281239790527442955936618005797473505853314969652123212744669910317554718268682408170780336037292464697861725022102
58588689459293530820994934613754956653226727344579678711437621452412461545577156879635067738877041808267234720802775998434079847631162472468259039167747227010594291380792151027097739
0264898483776479934822360599384541714266143569863776746229586377888408540229143919521022350169657668928661638380345735669156588634080530963053356572611278433967157302916380173869519090
8953925820460461478852399183796201353085842647551720896679694038027413775637554863064809506084623563745519929641753043840957109903577736056269373411402269867594315339066707304604614
798430568533751071465463556941351209111968916321831015365162369174447667087819048595260114898507725747384651062200588253947893932346762578645589177391284026541434638126429784733502
5798406017164309
得到数值:  808
PS C:\Users\LENOVO\AppData\Roaming\JetBrains\PyCharm2023.1\light-edit>

```

首先设置了服务器端保存的数值列表 message_list, 列表长度 length, 客户端生成公钥私钥并随机选择了一个要读取的位置。然后客户端生成一个密文选择向量 select_list={0,...,1...,0}, 其中, 仅有 pos 的位置为 1。

客户端遍历数值列表 message_list 中的每一个元素, 如果与 pos 相等, 则

对应的密文选择向量为 1 否则为 0。然后使用公钥对密文选择向量进行加密，存储在密文向量列表 `enc_list` 中。

服务器端进行加密求和的运算，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。

（四） 扩展实验

```
from phe import paillier #开源库
import time #做性能测试
from Crypto.Cipher import DES3, AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import hashlib
import random
import os

# 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。

def generate_sym_key(length):
    return os.urandom(length)

# 随机生成对称密钥
sym_key = generate_sym_key(24)
print("sym_key: ", sym_key.hex())

##### 设置参数
# 服务器端保存的数值
```

```

message_list =[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
length =len(message_list)

# 服务器端进行加密并保存
enc_message_list = []
# DES3.MODE_ECB: DES3 加密算法的电子密码本模式（Electronic Codebook）。
# 在 ECB 模式下，每个分组都独立地进行加密，相同的明文分组将得到相同的
密文分组
cipher = DES3.new(sym_key, DES3.MODE_ECB)
for i in range(length):

    ciphertext=cipher.encrypt(message_list[i].to_bytes(length=24,byteorde
r='big',signed=False))
        enc_message_list.append(ciphertext)

# 客户端生成公私钥，选择读取的位置
public_key,private_key = paillier.generate_paillier_keypair()
# 随机选择一个位置读取
pos = random.randint(0,length-1)
print("本次要读取的数值位置是：",pos)

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

##### 服务器端进行运算

```

```

c = 0
for i in range(length):

    trans_message_list=int().from_bytes(enc_message_list[i],byteorder='big',signed=False)

    c = c + trans_message_list * enc_list[i]
# print("产生密文: ",c.ciphertext())

##### 客户端进行解密
m = private_key.decrypt(c)
print("未解密的密文为: ",m)
m = cipher.decrypt(m.to_bytes(24, 'big', signed=True))
print("解密后的明文为:
",int().from_bytes(m,byteorder='big',signed=True))

```

具体思路:

客户端

- 设置对称密钥 k

服务器端:

- 产生数据列表 $data_list=\{m_1, m_2, \dots, m_n\}$
- 存储用对称密钥 k 加密的密文

客户端:

- 生成公钥私钥
- 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:

- 把 byte 转为 int 进行计算

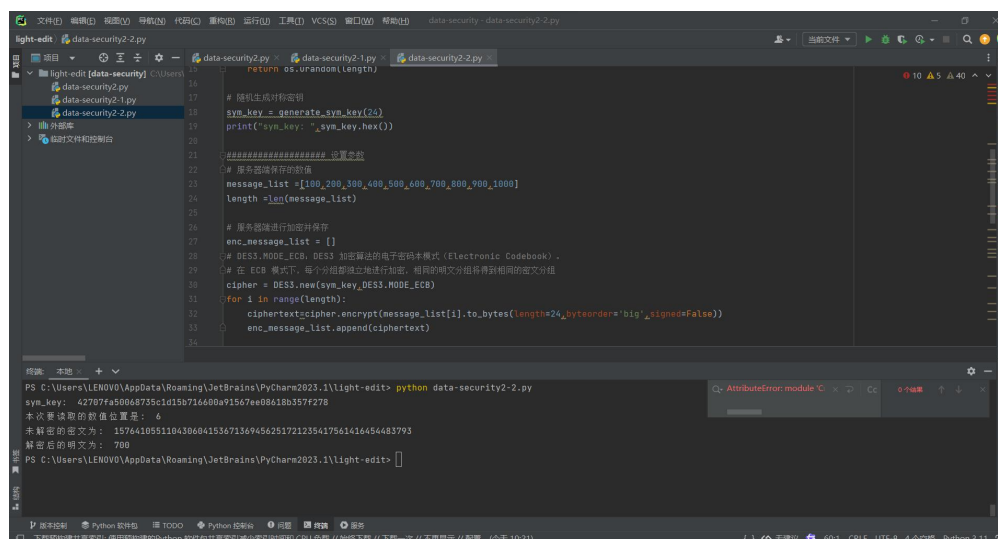
- 将数据与对应的向量相乘后累加得到密文

$$c = \text{trans_message_list}[1] * \text{enc_list}[1] + \dots + [n] * \text{enc_list}[n]$$

- 返回密文 c 给客户端

客户端：通过 DES，获取的 byte 解密得到 int

运行结果如下：



```
15 def generate_key():
16     return os.urandom(LENGTH)
17
18 # 随机生成对称密钥
19 sym_key = generate_sym_key(16)
20 print("sym_key: ", sym_key.hex())
21
22 ##### 设置参数
23 # 服务端保存的数据
24 message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
25 length = len(message_list)
26
27 # 服务端进行加密解密
28 enc_message_list = []
29 # DES3.MODE_ECB: DES3 加密算法的电子密码本模式 (Electronic Codebook),
30 # 在 ECB 模式下, 每个分组都独立地进行加密, 相同的明文分组将得到相同的密文分组
31 cipher = DES3.new(sym_key, DES3.MODE_ECB)
32 for i in range(length):
33     ciphertext = cipher.encrypt(message_list[i].to_bytes(length=24, byteorder='big', signed=False))
34     enc_message_list.append(ciphertext)
```

PS C:\Users\LENOVO\AppData\Roaming\JetBrains\PyCharm2023.1\light-edit> python data-security2-2.py

sym_key: 429b74a80a68725c1d13b716680a91567ee0861b3357278

本次要加密的数值位置是: 6

未加密的密文为: 157441095110438684153671369456251721235417561416454483793

解密后的明文为: 700

选取位置 6，得到密文及解密后数值 700

四、实验心得体会

本次实验，首先在给出的参考代码中熟悉 phe 的用法，将课上学习的理论与课下实验的实操相结合，并进一步掌握半同态加密的基本原理。拓展部分通过查找相关资料，学习对称加密算法的相关知识，从中深化对密码学的认识。