数据安全实验报告

姓名：任薏霖 学号：2011897 专业：物联网工程

1 实验名称：半同态加密应用实践

2 实验要求

- 验证 Paillier 加密算法对标量四则运算的正确性；
- 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，同时客户端能完成或许消息的解密；
- 扩展要求：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，使用隐私信息获取的方法得到指定密文后用 k 解密。

3 实验过程

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。

3.1 环境配置

本实验环境主要基于 Windows 下的 python 开发环境，在完成基础 python 环境配置后，进行相关库函数的安装。在本次实验中，主要应用的库为 phe 库和 AES 库，其具体安装和调用如下：

```
# 环境安装
pip install phe
pip install pycryptodomex
# 库调用
from phe import paillier #开源库
import time # 做性能测试
from Cryptodome.Cipher import AES
```

3.2 基于 Python 的 phe 库完成加法和标量乘法的验证

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。

本实验模块通过复现 test1.py 的内容，实现对加法和标量乘法的验证。

3.2.1 加密与解密

首先定义需要加密的数据，并自动生成公钥与私钥，其中默认私钥大小为 3072。

```
print("默认私钥大小: ", paillier.DEFAULT_KEYSIZE)
# 生成公私钥
public_key, private_key = paillier.generate_paillier_keypair()
# 测试需要加密的数据
message_list = [3.1415926,100,-4.6e-12]
```

在完成数据测试后，首先通过公钥对数据”3.1415926” 进行加密，于输出端显示加密耗时时间及加密数据内容，其具体代码和输出结果如下:

```
# 加密操作
time_start_enc = time.time()
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())
```

```
加密耗时s:  0.9547238349914551
加密数据（3.1415926）: 5449261418288222239742181742601259323035085242114677632171850573
169949655523844045815505296860643982930212377332301398264128461603219577343070189699262
020155287798102197031035102182920445274177702263176962494365938716969188956034534256393
179662004674192503310095950911819589547682301884673705572036487800102050857255506886538
696113540554719331364365136887202851719174283275711005961338293512555635702588668888825
567849428669078010622663502180558204388520925354366796873887565812227831912920074464330
508892575538233688213048473417566546690725822711371405128489373563551836828709876367092
198523924796282168901054502358577424456214381577157435959770006547284166255327027194787
956823942401218168586955266056351513215503496899769129238841362737215697650677670833461
720474093128456089112295945918702226196924317260102296953334460733921980744430495378908
235611665240054894830482649901612624540976231444142748448611929129824737713975150345938
526646134179136755605913256447065345822113908847438977704976613295150804525100343359124
996811086209435420337342050649936114569196241820306566144259721238865370997480277065007
358300674388169117896917537105696845189170973829991723543568650008311280775303191514685
282619681032836801682373847663766618966226691919409592411461473162058238348800341053000
078679902366914370727293229957728901647018757138934977782782227018987500369007072909215
317916195540270274044436130009375531655859779658324554051657339551521843938082406944508
671736896075108344738973271289185888011793663629450706849832330116998906120403228669702
549361210943512137942214838043662340982735734324418489135648468869207166625324123578535
398443404776550262681772943920057968286808075052187978476261101908833520280748013610473
021375956884290930222118065739665151489315004793900295706993583576040853563399697434355
354186031494396961707313223396011177882926345
```

图 3.1: 加密输出结果

在完成加密操作后再根据私钥对上述密文进行解密，其具体代码和输出结果如下:

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

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

```
In [8]: # 解密操作
        time_start_dec = time.time()
        decrypted_message_list = [private_key.decrypt(c) for c in encrypted_message_list]
        time_end_dec = time.time()
        print("解密耗时s: ",time_end_dec-time_start_dec)
        print("原始数据（3.1415926）:",decrypted_message_list[0])

        解密耗时s: 0.255446195602417
        原始数据（3.1415926）: 3.1415926
```

图 3.2: 解密输出结果

通过加密耗时和解密耗时的时间对比可知，解密耗时相较时间更短，这一结果与 Paillier 算法的性质相符，即在加密过程中需要做两次指数运算，而解密只需要一次。

3.2.2 测试加法和乘法同态

其具体代码如下所示：

```
# 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
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

print("a+5 密文:",a.ciphertext()) # 密文纯文本形式
print("a+5=",private_key.decrypt(a_sum))
print("a-3",private_key.decrypt(a_sub))
print("b*6=",private_key.decrypt(b_mul))
print("c/-10.0=",private_key.decrypt(c_div))

# 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
# print((private_key.decrypt(a)+private_key.decrypt(b))==private_key.decrypt(a*b))
```

其具体结果如下所示：

```
a+5 密文: 5449261418288222239742181742601259323035085242114677632171850573169949655523844045815505296860643982930212377332301398264128461603219577343070189699262020155287798102197031035102182920445274177702263176962494365938716969188956034534256393179662004674192503310095950911819589547682301884673705572036487800102050857255506886538696113540554719331364365136887202851719174283275711005961338293512555635702588668888825567849428669078010622663502180558204388520925354366796873887565812227831912920074464330508892575538233688213048473417566546690725822711371405128489373563551836828709876367092198523924796282168901054502358577424456214381577157435959770006547284166255327027194787956823942401218168586955266056351513215503496899769129238841362737215697650677670833461720474093128456089112295945918702226196924317260102296953334460733921980744430495378908235611665240054894830482649901612624540976231444142748448611929129824737713975150345938526646134179136755605913256447065345822113908847438977704976613295150804525100343359124996811086209435420337342050649936114569196241820306566144259721238865370997480277065007358300674388169117896917537105696845189170973829991723543568650008311280775303191514685282619681032836801682373847663766618966226691919409592411461473162058238348800341053000078679902366914370727293229957728901647018757138934977782782227018987500369007072909215317916195540270274044436130009375531655859779658324554051657339551521843938082406944508671736896075108344738973271289185888011793663629450706849832330116998906120403228669702549361210943512137942214838043662340982735734324418489135648468869207166625324123578535398443404776550262681772943920057968286808075052187978476261101908833520280748013610473021375956884290930222118065739665151489315004793900295706993583576040853563399697434355354186031494396961707313223396011177882926345
a+5= 8.1415926
a-3 0.14159260000000007
b*6= 600
c/-10.0= 4.6e-13
True
```

图 3.3: 测试加法和乘法同态输出结果

通过上述代码可以得到，python 程序对运算符进行了承载，已支持直接密文上的运算；其次，只支持明文的加法，不支持明文的乘法。

3.3 隐私信息获取

本模块实验要求从服务器 m 个消息中加密读取一个，并不得泄露读取的消息序号，其实现主要利用的是 Paillier 的加法同态性和标量乘法同态性原理。

在完成本模块实验内容时，首先应基于 Paillier 协议进行设计，对 Paillier 的标量乘的性质进行扩展，具体设计如下：

- 服务器端：产生数据列表 data-list = m_1,m_2,...,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 = $0, ..., 1, ..., 0$，其中，仅有 pos 的位置为 1
  - 生成密文向量 enc-list = $E(0), ..., E(1), ..., E(0)$
  - 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 $c = m_4 * enc - llist[1] + ... + m_n * enc - list[n]$
- 返回密文 c 给客户端

- 客户端：解密密文 c 得到想要的结果

接下来，按照上述流程展示各自部分的代码细节。

3.3.1 服务器端：产生数据列表

其具体代码如下：

```
# 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)
```

3.3.2 客户端

其具体代码如下：

```
# 客户端生成公私钥
public_key, private_key = paillier.generate_paillier_keypair()
# 客户端随机选择一个要读的位置
pos = random.randint(0,length-1)
print("要读起的数值位置为：",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]) )
```

3.3.3 服务器端

其具体代码如下：

```
# 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())
```

3.3.4 客户端：解密

```
# 客户端进行解密
m=private_key.decrypt(c)
print("得到数值：",m)
```

多次测试，可得到不同结果如下：

```
###################### 客户端进行解密
m=private_key.decrypt(c)
print("得到数值：",m)
```

```
要读起的数值位置为： 7
得到数值： 800
```

图 3.4: 隐私信息获取：选取位置 7、得到数值 800

```
###################### 客户端进行解密
m=private_key.decrypt(c)
print("得到数值：",m)
```

```
要读起的数值位置为： 8
得到数值： 900
```

图 3.5: 隐私信息获取：选取位置 8，得到数值 900

3.4 扩展实验

扩展实验为上述实验的延伸，其主要内容为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，在通过隐私信息获取方法得到指定密文后，可通过解密得到对应明文。

该实验模块的本质是在上述实验的基础上增添对称加密方法，从而达到更优的安全性能。本实验模块选用 AES 分组密码来实现。

3.4.1 具体思路

- 设置对称密钥
- 服务器端：
  - 产生数据列表 data-list = m_1,m_2,...,m_n
  - 加密并保存；
- 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 = $0, ..., 1, ..., 0$，其中，仅有 pos 的位置为 1
  - 生成密文向量 enc-list = $E(0), ..., E(1), ..., E(0)$
  - 发送密文向量 enc-list 给服务器
- 服务器端：
  - 根据 paillier 算法的要求把 byte 转成 int 进行计算
  - 返回密文 c 给客户端

- 客户端：
  - 通过 AES，使获取的 byte 解密得到 int

3.4.2 具体代码

各部分代码如下所示：

设置对称密钥

```
# 提示：应设置为16位
password = b'abcdefghabcdefgh'
```

服务器端

```
# 产生数据列表
message_list = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
length = len(message_list)
# 加密并保存
encrypt_message_list = []
aes = AES.new(password, AES.MODE_ECB)
for i in range(length):
    # 加密时将int转成byte
    en_text = aes.encrypt(message_list[i].to_bytes(length = 16, byteorder = 'big',
        signed = False)) # 加密明文
    encrypt_message_list.append(en_text)
```

客户端

```
# 生成公私钥
public_key, private_key = paillier.generate_paillier_keypair()
# 客户端随机选择一个要读的位置
pos = random.randint(0, length - 1)
print("要读起的数值位置为：", 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

```
c = 0
# 将byte转换为int
for i in range(length):
    trans = int().from_bytes(encrypt_message_list[i], byteorder='big', signed=False)
```

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

客户端

```
# 客户端解密
m = private_key.decrypt(c)
print('未解密密文为: ', m)
m = aes.decrypt(m.to_bytes(length=16, byteorder='big', signed=True))
m = int().from_bytes(m, byteorder='big', signed=True)
print("得到数值: ", m)
```

3.4.3 结果

通过多次测试程序，其结果如下所示：

```
In [83]: print("要读起的数值位置为：", pos)
         m = private_key.decrypt(c)
         print("未经过AES解密的密文为:" , m)
         m = aes.decrypt(m.to_bytes(length=16, byteorder='big', signed=True))
         m = int().from_bytes(m, byteorder='big', signed=True)
         print("解密后数值：", m)

要读起的数值位置为： 2
未经过AES解密的密文为: 91888163111449298856610390251107207359
解密后数值： 300
```

图 3.6: 选取位置 2、得到数值 300

```
In [87]: print("要读起的数值位置为：", pos)
         m = private_key.decrypt(c)
         print("未经过AES解密的密文为:" , m)
         m = aes.decrypt(m.to_bytes(length=16, byteorder='big', signed=True))
         m = int().from_bytes(m, byteorder='big', signed=True)
         print("解密后数值：", m)

要读起的数值位置为： 1
未经过AES解密的密文为: 126417140509385135972455927348677837O4
解密后数值： 200
```

图 3.7: 选取位置 1、得到数值 200

4 心得体会

通过本次实验，首先在代码复现的相关内容中学习到了更深刻的有关半同态的知识，将课上学习的理论与课下实验的实际操作相结合，获得了更深刻的理解；除此，拓展部分是理论知识与基础实验的进一步延伸，通过查找相关资料，自行学习有关 DES 和 AES 的相关知识，及相关教程，实现了安全性更高的加密解密算法，从中收获到更丰富的有关密码学的知识，填补了我在密码学相关内容的空白。