数据安全实验报告

Lab2 半同态加密应用实践

网络空间安全学院 信息安全专业

2112492 刘修铭 1036

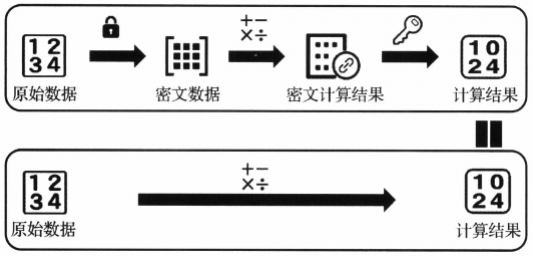
一、实验要求

- 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。
- 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

（一）半同态加密

同态加密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。



同态加密主要基于公钥密码体制构建，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。半同态加密仅支持单一类型的密文域同态运算（加或乘同态）。

（二）隐私信息获取

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。

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_1*enc_list[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

隐私信息获取的初衷是为了保护客户端的隐私，避免服务器端掌握客户端的信息。在这个设计中，密文操作是在服务器端完成的，服务器端收到客户端要读取的信息后，进行同态运算，然后将结果返回给客户端，客户端再使用自己的密钥进行解密，从而得到最后的信息。

（三）移位密码

移位密码又称凯撒密码，是一个比较简单的对称密码。

- 加密时，使用明文加上密钥
- 解密时，使用密文减去密钥

如此即可完成对信息的简单加解密。

三、实验过程（含主要源代码）

（一）实验环境安装

1. 安装 python 环境

本人已提前安装好 python 环境，在此仅展示安装后的运行效果。可以看到如下输出，说明安装成功并成功进入 python 运行环境。

```
✘ lxmliu2002@MMacBook-Pro ~ python3
Python 3.11.7 (main, Dec  4 2023, 18:10:11) [Clang 15.0.0 (clang-1500.1.0.2.5)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. 安装 phe 库

选定 python 环境后，执行 `pip install phe` 命令完成 phe 库的安装。

```
lxmliu2002@MMacBook-Pro ~/Desktop/Data_Security/lab2  main pip install phe
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: phe in /opt/homebrew/lib/python3.11/site-packages (1.5.0)

[notice] A new release of pip is available: 23.3.1 -> 24.0
[notice] To update, run: python3.11 -m pip install --upgrade pip
```

3. 验证环境正确性

导入 phe 库，可以看到没有出现错误信息，说明环境安装成功。

```
✘ lxmliu2002@MMacBook-Pro ~ python3
Python 3.11.7 (main, Dec  4 2023, 18:10:11) [Clang 15.0.0 (clang-1500.1.0.2.5)]
on darwin
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

> 为了更好掌握半同态加密的相关知识，本人一同完成了实验 2.1

实验参考中已经给出演示代码，故而此处对其思路进行简单分析。

- 导入 phe 库，为代码提供了加密和性能测试所需的功能，并使用 `paillier.generate_paillier_keypair()` 生成公钥和私钥
- 接着调用加解密库对信息进行加解密
- 对密文进行同态运算
- 最后打印输出相关运算结果

（三）隐私信息获取

实验参考中已经给出演示代码，故而此处对其思路进行简单分析。

- 首先设置一定参数，包括服务器端要保存的数值、客户端的公私钥以及要读取的信息的位置
- 接着客户端生成密文选择向量，只有对应位置为 true，其余均为 false
- 然后交由服务器端进行同态运算，并返回给客户端进行解密
- 客户端解密后即可得到自己想要的信息

（四）拓展实验

本人选择使用移位密码进行加解密，并设置密钥 `key = 666666`

- 首先设置一定参数，包括服务器端保存的数值、使用移位加密后得到的密文数值、客户端的公私钥以及要读取的信息的位置
- 接着客户端生成密文选择向量，交由服务器端进行运算
- 客户端对服务器端返回的数值进行解密

具体代码实现如下：

```python
from phe import paillier # 开源库
import random # 选择随机数
```

```python

def shift_encrypt_decrypt(data, key, encrypt = True):
    data_bytes = data.to_bytes((data.bit_length() + 7) // 8, byteorder = 'big')
    key_bytes = key.to_bytes((key.bit_length() + 7) // 8, byteorder = 'big')
    data_bytearray = bytearray(data_bytes)
    key_bytearray = bytearray(key_bytes)

    if encrypt:
        for i in range(len(data_bytearray)):
            data_bytearray[i] = (data_bytearray[i] + key_bytearray[i % len(key_bytearray)]) % 256
    else:
        for i in range(len(data_bytearray)):
            data_bytearray[i] = (data_bytearray[i] - key_bytearray[i % len(key_bytearray)]) % 256

    return int.from_bytes(data_bytearray, byteorder = 'big')

# key 为加解密密钥
key = 666666

###################### 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]
# 服务器要保存的加密后数值
server_list = []
for i in message_list:
    encrypted_data = shift_encrypt_decrypt(i, key, encrypt = True)
    server_list.append(encrypted_data)
    # print(encrypted_data)
# 加密后的数组
print("密钥 k 加密后的数组:", server_list)
length = len(server_list)
# 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]))

###################### 服务器端进行运算
c = 0
for i in range(length):
    c = c + server_list[i] * enc_list[i]
print("产生密文：", c.ciphertext())

```

```python
###################### 客户端进行解密
m = private_key.decrypt(c)
print("得到用密钥 k 加密后的数值:", m)
decrypted_data = shift_encrypt_decrypt(m, key, encrypt = False)
print("得到原始数值：", decrypted_data)
```

四、实验结果及分析

（一）基于 Python 的 phe 库完成加法和标量乘法的验证

运行程序后，得到如下的运行结果，与讲授内容一致。说明验证成功。

```
默认私钥大小:  3072
加密耗时s:  1.5536248683929443
加密数据（3.1415926）: 9313480816693073847248920384667110117929375865628896156077584885494331883640681
解密耗时s:  0.42659592628479004
原始数据（3.1415926）: 3.1415926
a+5 密文: 9313480816693073847248920384667110117929375865628896156077584885494331883640681
a+5= 8.1415926
a-3 0.14159260000000007
b*6= 600
c/-10.0= 4.6e-13
True
```

（二）隐私信息获取

运行程序后，得到如下的运行结果。

```
要读起的数值位置为:  8
产生密文:  6331363163564321979919930514601761
得到数值:  900
```

其中，要读取的数值位置 8 设置的信息为 900，与得到的数值一致，说明实验成功。

（三）拓展实验

运行程序后，得到如下的运行结果。

```
密钥 k 加密后的数组: [110, 210, 2904, 3004, 2848, 3204, 3304, 3404, 3504, 3348]
要读起的数值位置为:  1
产生密文:  756768519579433205786095134242169306976838740435160101453034802340308
得到用密钥 k 加密后的数值: 210
得到原始数值:  200
```

其中，要读取的数值位置 1 设置的信息为 200，与解密得到的数值一致，说明实验成功。

五、参考

本次实验主要参考教材内容完成。