

《数据安全》实验报告

姓名： 费泽锟 学号： 2014074 班级： 信安班

实验名称：

半同态加密应用实践

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。

实验过程：

基础实验部分：

我们首先对半同态 Paillier 算法进行简单的介绍,在基础部分的实验之中我们需要对给出的 test.py 文件之中的客户端以及服务端代码进行理解与复现。

同态加密(homomorphic encryption)是一种加密算法,它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,这样我们就可以在不暴露原始数据的基础上将数据加密后给予合作伙伴,使其能够完成对加密后数据的运算,并且得到的结果与未加密前的结果一致。

根据支持的运算种类的不同以及运算次数的区别,半同态加密(partial homomorphic encryption)仅支持单一类型的密文域同态运算(加或乘同态)而本次实验中所复现的 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。

我们接下来通过实验指导进行基础实验内容的复现,并在复现过程之中对代码以及服务器保密流程进行讲解。

1. 安装 python 环境与安装 phe 库

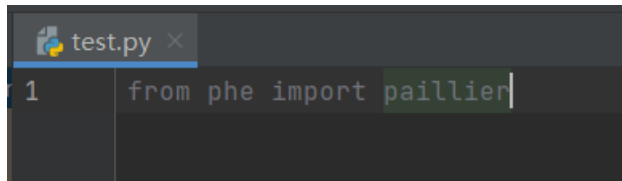
我们首先输入 python 命令对 phe 库尝试进行导入,发现确实没有这个库,这个时候我们就需要使用 pip install phe 命令完成 phe 库的安装。

这里我们可以直接使用 `pycharm` 顺便下载到本地即可。

~~~~~

## 2. 验证环境正确性

我们进入 `pycharm` 环境，尝试 `from phe import paillier`，可以看到：



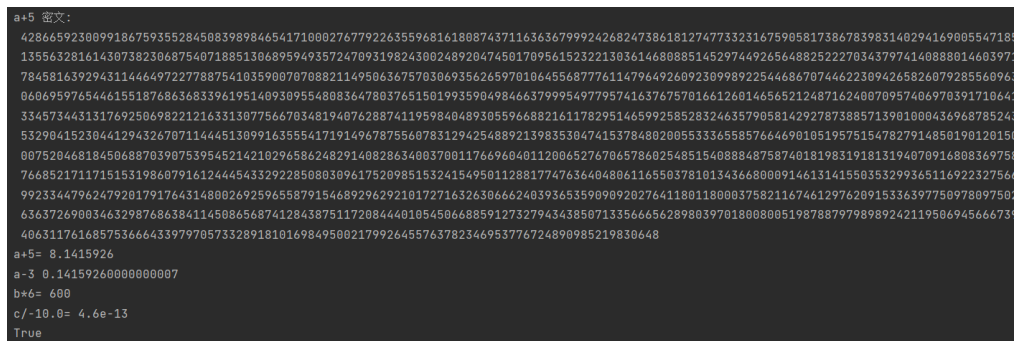
```
test.py x
1 from phe import paillier
```

就可以顺利地发现我们的实验环境就已经配置完成了。

~~~~~

3. 通过实验 2.1 来验证 `paillier` 算法能成功应用

我们首先通过给出的 `test1.py` 文件之中的代码，测试 `paillier` 算法能否被成功地运用到数据的加密与解密过程。运行结果如下：



```
a+5 密文:
4286659230899186759355284508398984654171000276779226355968161808743711636367999242682473861812747733231675905817386783983140294169005547181
135563281614307382306875407188513068959493572470931982430024892047450170956152322130361468088514529744926564882522270343797414088801460397
784581639294311466497227788754103590870708821149506367570306935626597010645568777611479649260923099892254468670744622309426582607928556096
060695976544615518768636833961951409309554808364780376515019935904984663799954977957416376757016612601465652124871624087095740697039171064
334573443131769250698221216331307756670348194076288741195984048930559668821611782951465992585283246357905814292787388571390100043696878524
532904152304412943267071144451309916355541719149678755607831294254889213983530474153784802005533365585766469010519575154782791485019012015
007520468184506087039075395452142102965862482914082863400370011766960401120065276706578602548515408884875874018198319181319407091480836975
766852171715153198607916124445433292285080309617520985153241549501128817747636404080611655037810134366800091461314155035329936511692232756
992334479624792017917643148002692596558791546892962921017271632630666240393653590909202764118011800037582116746129762091533639775097809750
636372690834632987686384114508656874128438751172084440105450668859127327943438507133566656289803970180088051987887979898924211950694566673
40631176168575366643397970573328918101698495082179926455763782346953776724890985219830648

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

~~~~~

#### 4. 有了实验 2.1 中的内容后对实验 2.2 进行复现

对于 test.py 的复现是基于以下几个特点的：

1. 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。
2. 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。

接下来我们对代码进行分析：

```
##### 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
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)
```

在这一段代码之中，我们在服务器之中保存了一个数据列表，然后通过 API 函数 generate\_paillier\_keypair 函数来获取了一对公私密钥，并且在 0 —— length-1 之中生成了一个模拟客户端取出位置的索引。

```
##### 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]) )
```

接着在客户端会模拟对想要取出的位置的加密向量，如果  $i == pos$  时那么位置向量之中的值为 1，其余计算均为 0，然后对这一向量进行加密运算，将其发送给服务端。

```
##### 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)
```

然后在服务端时,因为在密文之中只有为1的密文在计算时,密文计算结果不为0,所以对每一个位置的数据进行运算相加后就能够得到最后想要的数据密文,只需对其再进行解密即能得到最终的明文结果了。

总结总体的流程如下:

客户端:

1. 设置要选择的数据位置为 pos
2. 生成选择向量  $\text{select\_list}=\{0,\cdots,1,\dots,0\}$ , 其中, 仅有 pos 的位置为 1
3. 生成密文向量  $\text{enc\_list}=\{E(0),\cdots,E(1),\dots,E(0)\}$
4. 发送密文向量 enc\_list 给服务器

服务端:

1. 将数据与对应的向量相乘后累加得到密文  $c=m_1\text{enc\_list}[1]+\cdots+m_n\text{enc\_list}[n]$
2. 返回密文 c 给客户端

这样就能完成从服务器给定的 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了。

~~~~~

扩展实验部分:

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。具体分析代码实现步骤如下:

1. 设置参数与生成公私钥

```
# 设置参数
message_list = [100,200,300,400,500,600,700,800,900,1000]
length = len(message_list)
pos = random.randint(0, length-1)
print("要读取的数值位置为: ", pos)

# 生成密钥
key = Fernet.generate_key()
cipher = Fernet(key)
```

这里我们可以使用 `cipher` 类进行加密，也可以使用 AES 或者 DES 进行对称加密进行实验，虽然 DES 在商业上已经不再安全了，但是可以运用 DES 进行实验。

2. 在客户端生成选择向量并通过对称加密进行运算

```
# 客户端生成选择向量
select_list = [i == pos for i in range(length)]
select_bytes = bytes(select_list)

# 对选择向量进行加密
encrypted_select_bytes = cipher.encrypt(select_bytes)

# 服务器端进行运算
encrypted_list = [cipher.encrypt(str(num).encode()) for num in message_list]
encrypted_sum = encrypted_list[0]
```

将整数列表转换为字节序列，并使用对称加密方案加密字节序列列表，这里我们需要注意一点就是对称加密所进行运算的对象均为字节序列，我们需要将整数与字节序列进行双向的转换。

3. 在客户端使用对称加密密钥 k 进行解密

```
# 客户端进行解密
decrypted_result = cipher.decrypt(encrypted_result)
# 先得到选择向量
decrypted_select_bytes = decrypted_result[:length]
# 再得到相应的数值
decrypted_sum = decrypted_result[length:]
# 由选择向量确定数值
selected_number = message_list[pos]

for i in range(length):
```

将上述结果与列表中的第三个数字解密后转换为整数的和转换为字节序列并加密，

再次相加得到一个新的密文结果,依此类推,最终得到整个列表中所有数字的加的密文。
再通过选择向量计算的结果与服务器存储的计算结果获得想要得到的密文,将其解密即可获得最终的明文结果。

心得体会:

通过本次实验,我们学习了半同态加密算法的原理,实验中给出了一个非常贴近现实的使用 paillier 的场景,我们完成了最终隐私获取实验的相关操作,客户端能从服务器给定的 m 个消息中获取其中一个,且不会向服务器泄露获取了哪一个消息,同时客户端也能完成获取消息的解密,拿到相应的数据。

经过学习与实践,将在密码学课上学习的知识和数据安全学习的内容结合在了一起,更好的理解了半同态加密算法的特性及其适合的应用场景。