南開大學
Nankai University

计算机学院

数据安全实验报告

秘密共享实践

姓名：李欣

学号：2011165

专业：计算机科学与技术

2023 年 4 月 23 日

目录

1 实验环境

- Ubuntu22.04
- Python

2 实验要求

借鉴实验 3.2，实现三个人对于他们拥有的数据的平均值的计算。

3 理论基础

3.1 基本概念

3.1.1 定义

秘密共享（Secret Sharing）是一种将秘密分割存储的密码技术，目的是阻止秘密过于集中，以达到分散风险和容忍入侵的目的，是信息安全和数据保密中的重要手段。目前，秘密共享已成为一种重要密码学工具，在诸多多方安全计算协议中被使用，例如拜占庭协议、多方隐私集合求交协议、阈值密码学等。

秘密分享方案的安全性有很多不同的定义方法。下面的定义是在 Beimel 和 Chor（Beimel and Chor，1992）定义的基础上修改而来的。

定义： 令 D 为秘密值所在域，令 D_1 为秘密份额所在域。令 $Shr : D \to D_1^n$ 为秘密分配算法（可能是随机性算法），$Rec : D_1^n \to D$ 为秘密重构算法，其中 $D_1 n$ 表示 n 个秘密份额、$D_1 k$ 表示 k 个秘密份额。 t, n - 秘密分享方案包含一对算法（Shr，Rec），满足下述**两个性质**

- 正确性

  令 $(s_1, s_2, \cdots, s_n) = Shr(s)$, 则：$Pr[\forall k \geq t, Rec(s_{i_1}, \cdots, s_{i_k}) = s] = 1$

- 完美隐私性

  任意包含少于 t 个秘密份额的集合都不会在信息论层面上泄漏与秘密值相关的任何信息。

我们在多数情况下使用的都是 $(n, n) -$ 秘密分享方案，即拥有全部 n 个秘密份额是重建出秘密值的充分必要条件。

3.1.2 分类

基于秘密分配和运算的形式，我们将秘密共享分为基于位运算的加性秘密共享和基于线性代数的线性秘密共享。

依据秘密重构的条件或者秘密分享的份额数量等，又可以将秘密共享分为如下类别：

- 门限秘密共享：任意大于等于阈值的参与方集合可重构出秘密。
- 多重秘密共享：参与方的子秘密可以多次使用，分别恢复多个共享秘密
- 多秘密共享：一次共享，共享多个秘密，且子秘密可以重复使用
- 可验证秘密共享：可通过公共变量验证自己子秘密的正确性

- 动态秘密共享：允许添加或删除参与方，定期或不定期更新参与方的子秘密，还允许在不同的时间恢复不同的秘密。

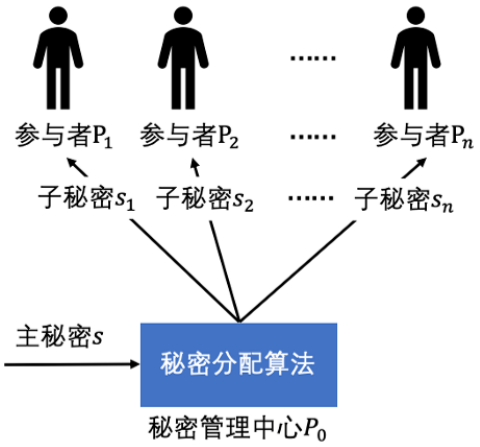
3.1.3 门限秘密共享

在 1979 年，Shamir 提出了门限秘密共享算法。

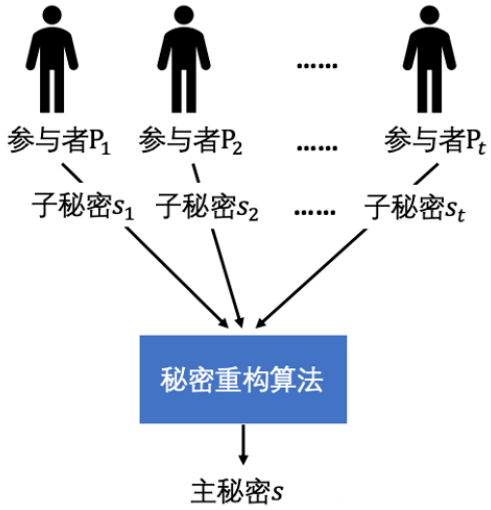
设和为两个正整数，且 $t \leq n$，n 个需要共享秘密的参与者集合为 $P = \{P_1, ..., P_n\}$。一个 (t, n) 门限秘密共享体制是指：假设 $P_1, ..., P_n$ 要共享同一个秘密，将 s 称为主秘密，至少 t 个参与者才可以共同恢复主秘密 s。

有一个秘密管理中心 P_0 来负责对 s 进行管理和分配，P_0 掌握有秘密分配算法和秘密重构的算法，这两个算法均满足重构要求和安全性要求。

秘密分配。秘密管理中心 P_0 首先通过将主秘密 s 输入秘密分配算法，生成 n 个值，分别为 $s_1, ..., s_n$，称 $s_1, ..., s_n$ 为子秘密。然后秘密管理中心 P_0 分别将秘密分配算法产生的子秘密 $s_1, ..., s_n$ 通过 P_0 与 P_i 之间的安全通信信道秘密地传送给参与者 P_i，参与者 P_i 不得向任何人泄露自己所收到的子秘密。



秘密重构。门限值 t 指的是任意大于或等于 t 个参与者 P_i，将各自掌握的子秘密 s_i 进行共享，任意的一个参与者 P_i 在获得其余 $t-1$ 个参与者所掌握的子秘密后，都可独立地通过秘密重构算法恢复出主秘密 s。而即使有任意的 $n-t$ 个参与者丢失了各自所掌握的子秘密，剩下的 t 个参与者依旧可以通过将各自掌握的子秘密与其他参与者共享，再使用秘密重构算法来重构出主秘密 s。安全性要求任意攻击者通过收买等手段获取了少于 t 个的子秘密，或者任意少于 t 个参与者串通都无法恢复出主秘密 s，也无法得到主秘密 s 的信息。



3.2 Shamir 方案

构造思路:

- 一般的，设 $\{(x_1, y_1), ..., (x_k\ y_k)\}$ 是平面上 k 个不同的点构成的点集，那么在平面上存在唯一的 $k-1$ 次多项式 $f(x) = a_0 + a_0x + ...a_{k-1}x^{k-1}$ 通过者 k 个点。
- 若把秘密 s 取做 $f(0)$，n 个份额取做 $f(i)(i = 1, , n)$，那么利用其中任意 k 个份额就可以重构 $f(x)$，从而得到秘密 $s = f(0)$。

3.2.1 秘密分配算法

假设 F_q 为 q 元有限域，q 是素数且 $q < n$。而 $P = \{P_1, ..., P_n\}$ 是参与者集合，P 共享主秘密 $s, s \in F_q$，秘密管理中心 P_0 按如下所述的步骤对主秘密 s 进行分配，为了可读性起见，以下公式均略去了模 q 操作:

1. P_0 秘密地在有限域 F_q 中随机选取 $t-1$ 个元素，记为 $a_1, ..., a_{t-1}$，并取以 x 为变元的多项式 $f(x) = a_{t-1}x^{t-1} + ... + a_1x^1 + s = s + \sum_{i=1}^{t-1} a_ix^i$
2. 对于 $1 \le i \le n$，P_0 秘密计算 $y_i = f(i)$。
3. 对于 $1 \le i \le n$，P_0 通过安全信道秘密地将 (i, y_i) 分配给 P_i。

3.2.2 秘密重构算法

依据 (t, n) 门限共享体制，秘密重构需要 t 个参与方都将所拥有的秘密分享出来，以便求解。通常有两类方法：解方程法和多项式插值法。

3.3 插值原理

对某个多项式函数，已知有给定的 $k+1$ 个取值点:$(x_0, y_0), ..., (x_k, y_k)$，其中 x_j 对应着自变量的位置，而 y_j 对应着函数在这个位置的取值。假设任意两个不同的 x_j 都互不相同，那么应用拉格朗日插

值公式所得到的拉格朗日插值多项式为

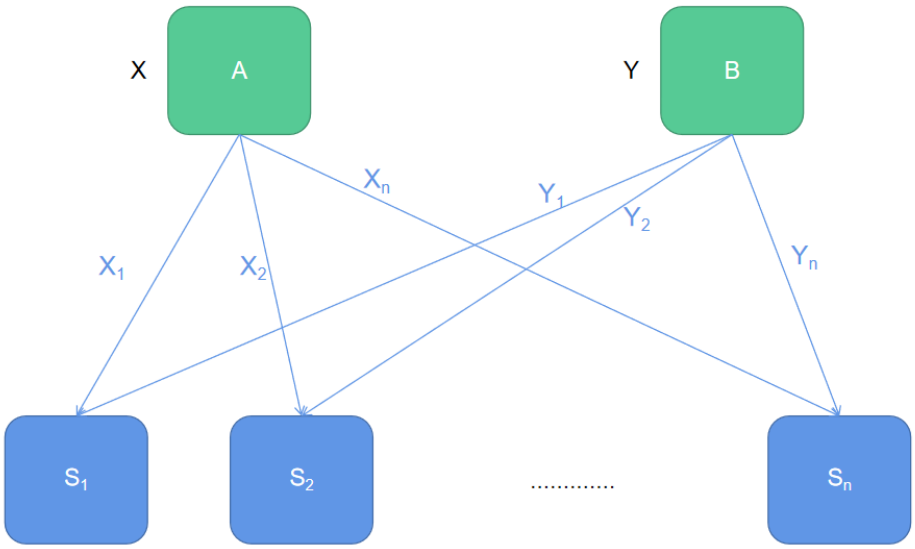
$$L(x) := \sum_{j=0}^{k} y_j l_j(x)$$

其中每个 $l_j(x)$ 为拉格朗日基本多项式 (或称插值基函数)，其表达式为：

$$l_j(x) := \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} ... \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} ... \frac{x - x_k}{x_j - x_k}$$

3.4 同态特性

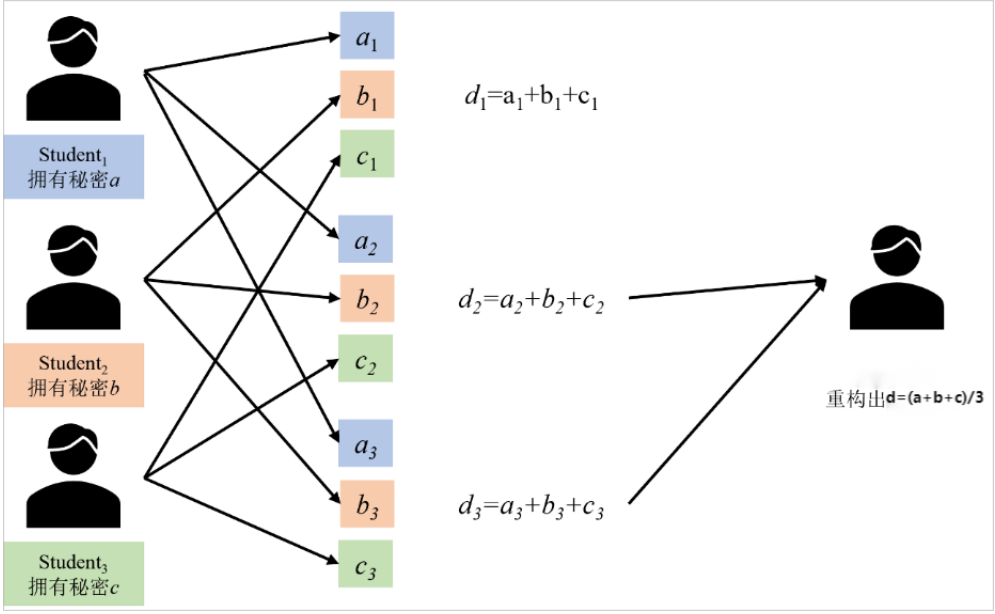
秘密共享体系还具有同态的特性。如下图所示有特征 A 和 B，他们的值被随机分成碎片 $(X_1, X_2, ..., X_n)$ 和 $(Y_1, Y_2, ..., Y_3)$，并分配到不同参与节点 $(S_1, S_2, ..., S_n)$ 中，每个节点的运算结果的加和能等同于原始 A 与 B 的加和。同样通过增加其他计算机制，也能满足乘积的效果，这就是秘密共享具备的 “同态性”，各参与者可以在不交换任何数据的情况下直接对密码数据求和，乘积。



在秘密共享系统中，攻击者必须同时获得一定数量的秘密碎片才能获得密钥，通过这样能提高系统的安全性。另一方面，当某些秘密碎片丢失或被毁时，利用其它的秘密份额仍让能够获得秘密，这样可提高系统的可靠性。

4 实验过程

基于 Shamir 门限秘密共享进行协议设计，具体过程如下图所示：



4.1 ss_function.py

ss_function.py 代码里面定义了一些秘密共享过程中三个拥有秘密值以及计算方会用到的函数。

ss_function.py

```python
import random

#快速幂计算 a^b%p
def quickpower(a,b,p):
    a=a%p
    ans=1
    while b!=0:
        if b&1:
            ans=(ans*a)%p
        b>>=1
        a=(a*a)%p
    return ans

#构建多项式: x0 为常数项系数, T 为最高次项次数, p 为模数,fname 为多项式名
def get_polynomial(x0,T,p,fname):
    f=[]
    f.append(x0)
    for i in range(0,T):
        f.append(random.randrange(0,p))
    #输出多项式
    f_print='f'+fname+'='+str(f[0])
    for i in range(1,T+1):
        f_print+='+'+str(f[i])+'x^'+str(i)
    print(f_print)
    return f

```

```
27
28      #计算多项式值
29      def count_polynomial(f,x,p):
30          ans=f[0]
31          for i in range(1,len(f)):
32              ans=(ans+f[i]*quickpower(x,i,p))%p
33          return ans
34
35      #重构函数 f 并返回 f(0)
36      def restructure_polynomial(x,fx,t,p):
37          ans=0
38          #利用多项式插值法计算出 x=0 时多项式的值
39          for i in range(0,t):
40              fx[i]=fx[i]%p
41              fxi=1
42              #在模 p 下，(a/b)%p=(a*c)%p，其中 c 为 b 在模 p 下的逆元，c=b^(p-2)%p
43              for j in range(0,t):
44                  if j !=i:
45                      fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p
46              fxi=(fxi*fx[i])%p
47              ans=(ans+fxi)%p
48          return ans
```

4.2 ss_student.py

三个拥有秘密值的人分别执行 ss_student.py，将自己的秘密值共享给另外两个用户。

ss_student.py

```
1   import ss_function as ss_f
2   #设置模数 p
3   p=1000000007
4   print(f'模数 p: {p}')
5   #输入参与方 id 以及秘密 s
6   id=int(input("请输入参与方 id:"))
7   s=int(input(f'请输入 student_{id}拥有的数值 s:'))
8   #秘密份额为(share_x,share_y)
9   shares_x=[1,2,3]
10  shares_y=[]
11  #计算多项式及秘密份额(t=2,n=3)
12  print(f'Student_{id}拥有的数值的多项式及秘密份额: ')
13  f=ss_f.get_polynomial(s,1,p,str(id))
14  temp=[]
15  for j in range(0,3):
16      temp.append(ss_f.count_polynomial(f,shares_x[j],p))
17      print(f'({shares_x[j]},{temp[j]})')
18      shares_y.append(temp[j])
19  #Student_id 将自己的拥有的数值的秘密份额分享给两外两个学生
20  #将三份秘密份额分别保存到 student_id_1.txt,student_id_2.txt,student_id_3.txt
```

```
    #Student_i 获得 Student_id_i.txt
    for i in range(1,4):
        with open(f'student_{id}_{i}.txt','w') as f:
            f.write(str(shares_y[i-1]))
```

结果如下，在文件夹下会产生 9 个 txt 文件，分别保存三个秘密值的秘密份额：**用户 1：**

用户 1

```
python3 ss_student.py
1
3
```

用户 2：

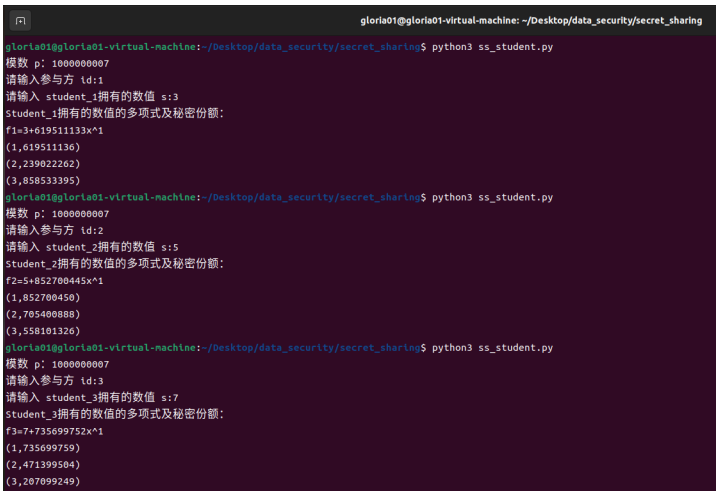
用户 2

```
python3 ss_student.py
2
5
```

用户 3：

用户 3

```
python3 ss_student.py
3
7
```



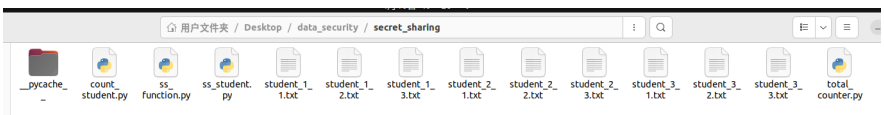


图 4.1: 9 个 txt 文件

4.3 count_student.py

将如下代码复制到 count_student.py，三个用户分别执行 count_student.py，获取另外两个用户的值的秘密份额，并将三个值的秘密份额相加。

count_student.py

```
p=1000000007
#输入参与方 id
id=int(input("请输入参与方 id:"))
#Student_id
    读取属于自己的秘密份额student_1_id.txt,student_2_id.txt,student_3_id.txt
data=[]
for i in range(1,4):
    with open(f'student_{i}_{id}.txt', "r") as f: #打开文本
        data.append(int(f.read())) #读取文本
#计算三个秘密份额的和
d=0
for i in range(0,3):
    d=(d+data[i])%p
#将求和后的秘密份额保存到文件 d_id.txt 内
with open(f'd_{id}.txt','w') as f:
    f.write(str(d))
```

用户 1 执行如下命令，获得三个投票值的秘密份额相加的结果保存到 d_1.txt。

用户 1

```
python3 count_student.py
1
```

用户 2 执行如下命令，获得三个投票值的秘密份额相加的结果保存到 d_2.txt。

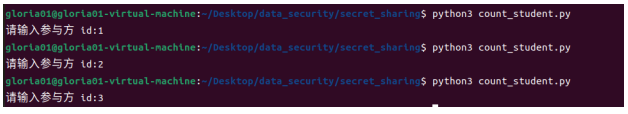
用户 2

```
python3 count_student.py
2
```

用户 3 执行如下命令，获得三个投票值的秘密份额相加的结果保存到 d_3.txt。

用户 3

```
python3 count_student.py
3
```



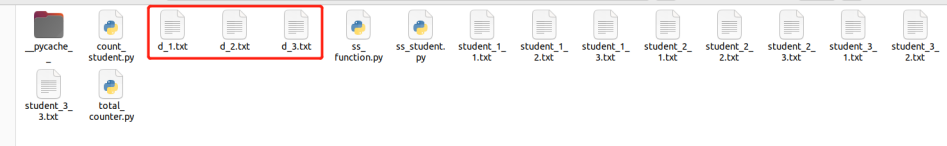


图 4.2: 3 个秘密份额相加的结果

4.4 total_counter.py

total_counter.py

```python
import ss_function as ss_f
#设置模数 p
p=1000000007
#随机选取两个参与方，例如 student2 和 student3，获得 d2,d3，从而恢复出 d=a+b+c
#读取 d2,d3
d_23=[]
for i in range(2,4):
    with open(f'd_{i}.txt', "r") as f: #打开文本
        d_23.append(int(f.read())) #读取文本
#加法重构获得 d
d=ss_f.restructure_polynomial([2,3],d_23,2,p)
print(f'三个人拥有数值的平均值：{d/3}')
```

计算方执行如下命令，得到三位用户秘密值之和后**除以**三。

Listing:

```
python3 vote_counter.py
```

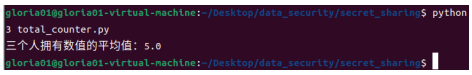


图 4.3: 9 个 txt 文件

5 心得体会

本次实验中，我们重点学习了 Shamir 方案，我了解了 Shamir 方案的原理，并进行了实验实践。Shamir 方案是一种多项式插值算法，能够将秘密信息分为多个部分，并分配给不同的个体进行储存。只有当多个个体共同协作才能恢复原始的秘密信息。该方案具有广泛的应用价值，能够解决许多秘密信息传输方面的安全问题。

Shamir 方案的通常思路是从一个秘密中生成多个部分，这些部分被称为共享（share），数量为 k 个，其中至少需要 n 个共享，才能恢复秘密。Shamir 方案基于公钥密码学中的多项式插值的概念和拉格朗日插值法。它将秘密信息拆分成多个份，并把每份进行加密处理，然后将加密后的密钥分配给多个受信任的个体。只有当这些个体共同拥有密钥才能重组出原始的秘密信息。

在数据安全方面，秘密共享是一种有效的方法，它将敏感数据拆分成多个部分，分别分配给不同的用户或实体，并确保只有在所有部分都被汇集后才能还原数据。这种方法具有以下几个特点：

1. **安全性**：由于数据被拆分成多个部分并分配给不同的实体，单个实体无法还原完整的数据，从而保护了数据的安全性。
2. **可靠性**：数据分散在多个实体中，即使某个实体出现问题，其他实体也能继续工作并还原数据。
3. **保护隐私**：通过秘密共享，数据被拆分并分配给多个实体，可以防止某个实体恶意使用或泄漏数据。