南　　开　　大　　学

计　算　机　学　院

数据安全

秘密共享实践

穆禹宸　2012026

年级：2020 级

专业：信息安全、法学双学位班

指导教师：刘哲理

2023 年 4 月 19 日

目录

一、 实验名称

秘密共享实践

二、 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。

三、 实验过程

(一) 实验原理

Shamir 的 (t，n) 门限方案基于 Lagrnage (拉格朗日) 插值法来实现，简单地说，设秘密通过秘密共享算法分发给个成员共享，每一个成员持有一个子密钥也称为 shadow 或 Secret debris, 如果满足:

1. 任何不少于 t 个的有效成员使用他持有的正确的碎片都可以恢复秘密。
2. 任何 t 个以下的成员集都无法恢复秘密。

我们称这种方案为 (t ,n) 门限秘密共享方案，简称为门限方案，t 称为方案的门限值。

1. 简单介绍

为了分享一个秘密 s，(t,n) Shamir 门限秘密共享首先选择一个 $t-1$ 阶的多项式 $p=s+a_1x+a_2x^2+\cdots+a_{t-1}x^{t-1}$，$p$ 的常数项为秘密 s。n 个参与者分别拥有多项式 p 上的 n 个不同的点作为秘密份额，如 1 $p(1)$,$(2,p(2),\cdots(n,p(n))$，则其中的任意 t 个参与者出示自己拥有的秘密份额，即可恢复出多项式 p，从而得到秘密 s。或者根据拉格朗日插值法直接恢复出 $s=p(0)$。

2. 具体步骤

我们假定 n 是参与者的数目，n 是门限值，p 是一个大素数要求 p > n 并且大于 p 秘密 s 的可能的最大取值; 秘密空间与份额空间均为有限域 GF(p)。因此我们得到如下操作，如图所示:

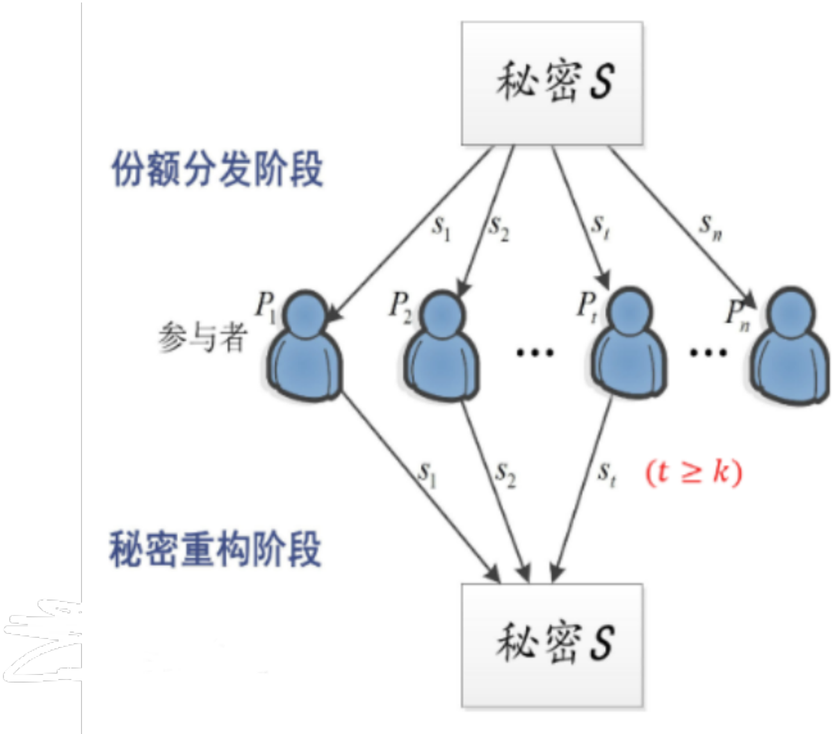


图 1: 整体流程

秘密分发

秘密分发者 D 给 n 个参与者 Pi(0 i n) 分配份额的过程，即方案的分配算法如下:

(1) 随机选择一个 GF(p) 上的 k - 1 次多项式使得 f(0) = a0 = s 要在个参与者中分享的秘密 D 对 f(x) 保密。

(2) D 在 Z_p 中选择 n 个互不相同的非零元素 x1, x2, ⋯, xn，计算 (0 i n)。

(3) 将 (xi , yi) 分配给参与者 Pi(0 i n)，值 xi 是公开的，yi 作为的秘密份额，不公开。

秘密重构

给定任何 t 个点, 不妨设为前 t 个点（x1，y1)，(x2, y2)，⋯, (xt, yt)。由插值公式:

(1) 多项式 h(x):

$$h(x) = \sum_{i=1}^{t} s_i \prod_{j=1, j \neq i} \frac{x - x_j}{x_i - x_j}$$

(2) 共享秘密 s 为:

$$S = h(0) = \sum_{i=1}^{t} s_i \prod_{j=1, j \neq i} \frac{x - x_j}{x_i - x_j}$$

因此可得简单的实验代码。

（二） 实验代码

实验代码主要有如下几个部分：

快速幂计算 $a^b \mod p$

```python
# 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
```

注意，这里是解决 `GF(p)` 的部分。这段代码实现了快速幂算法，用于计算 $a^b\ mod\ p$ 的值。其中，a 、b 、p 分别为底数、指数和模数。算法的核心思想是将指数 b 用二进制表示，然后根据二进制位上的值来进行幂运算，从而减少了运算次数，提高了计算效率。具体实现过程如下：

1. 首先将底数 a 对模数 p 取余，避免出现大数运算。
2. 初始化答案为 1 。
3. 当指数 b 不为 0 时，进行循环。
4. 判断当前二进制位上的值是否为 1 ，如果是，则将答案乘上当前底数 a 对模数 p 取余的结果。
5. 将指数 b 右移一位，相当于将当前二进制位舍去。
6. 将底数 a 平方，相当于将指数除以 2 。
7. 循环结束后，返回答案即可。

然后是**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**

```python
# 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=fname+'='+str(f[0])
    for i in range(1,T+1):
        f_print+='+'+str(f[i])+'x^'+str(i)
    print(f_print)
    return f
```

具体过程为：

1. 首先定义一个空列表 f ，用于存储多项式的系数。

2. 将常数项系数 x_0 添加到列表 f 中。

3. 使用循环，从 0 到 T-1 ，生成 T 个随机数，作为多项式的系数，添加到列表 f 中。

4. 构建多项式的字符串表示，将多项式名和常数项系数拼接起来，然后使用循环，将每一项的系数和次数拼接起来，形成完整的多项式字符串。

5. 输出多项式字符串。

6. 返回多项式系数列表 f 。

然后为一个很简答的函数用于**计算多项式值**

```python
# 计算多项式值
def count_polynomial(f,x,p):
    ans=f[0]
    for i in range(1,len(f)):
        ans=(ans+f[i]*quickpower(x,i,p))%p
    return ans
```

计算多项式在 x 处的值，其中 f 为多项式系数列表， p 为模数。

1. 首先将多项式的常数项系数 f_0 赋值给答案 ans 。

2. 使用循环，从 1 到 T ，遍历多项式系数列表 f ，计算每一项的值，并将其加到答案 ans 中。

3. 对每一项的值进行快速幂运算，使用 quickpower 函数计算 $x^i \ mod \ p$ 的值，然后将其乘上对应的系数 f_i 。

4. 将每一项的值加到答案 ans 中，并对答案取模，避免出现大数运算。

5. 循环结束后，返回答案 ans 。

最后是**重构函数 f 并返回** f(0)

```python
# 重构函数 f 并返回 f(0)
def restructure_polynomial(x,fx,t,p):
    ans=0
    # 利用多项式插值法计算出 x=0 时多项式的值
    for i in range(0,t):
        fx[i]=fx[i]%p
        fxi=1
        # 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
        for j in range(0,t):
            if j !=i:
                fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p
        fxi=(fxi*fx[i])%p
        ans=(ans+fxi)%p
    return ans
```

1. 首先定义一个变量 ans，用于存储多项式在 x=0 处的值。
2. 使用循环，从 0 到 t-1，遍历多项式的系数列表 fx。
3. 对每一个系数 fx_i 进行取模运算，避免出现大数运算。
4. 使用多项式插值法，计算出多项式在 x=0 处的值。具体实现过程如下：
   (a) 定义一个变量 fxi，用于存储插值多项式的系数。
   (b) 使用循环，从 0 到 t-1，遍历多项式的系数列表 fx。
   (c) 如果 $j \neq i$，则计算 x_j 在 x_i 处的插值多项式系数，使用逆元计算，避免出现除法运算。具体实现过程如下：
      i. 定义一个变量 c，用于存储 x_j 在模 p 意义下的逆元，即 $c = x_j^{p-2}\ mod\ p$。
      ii. 计算插值多项式的系数 fxi ,使用公式 fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p 。
5. d. 将插值多项式的系数 fxi 乘上 fx_i，得到 x_i 在 x=0 处的值。
6. e. 将 x_i 在 x=0 处的值加到 ans 中。
7. 对 ans 取模，避免出现大数运算。
8. 返回 ans。

至此，全部所需函数编写完成。

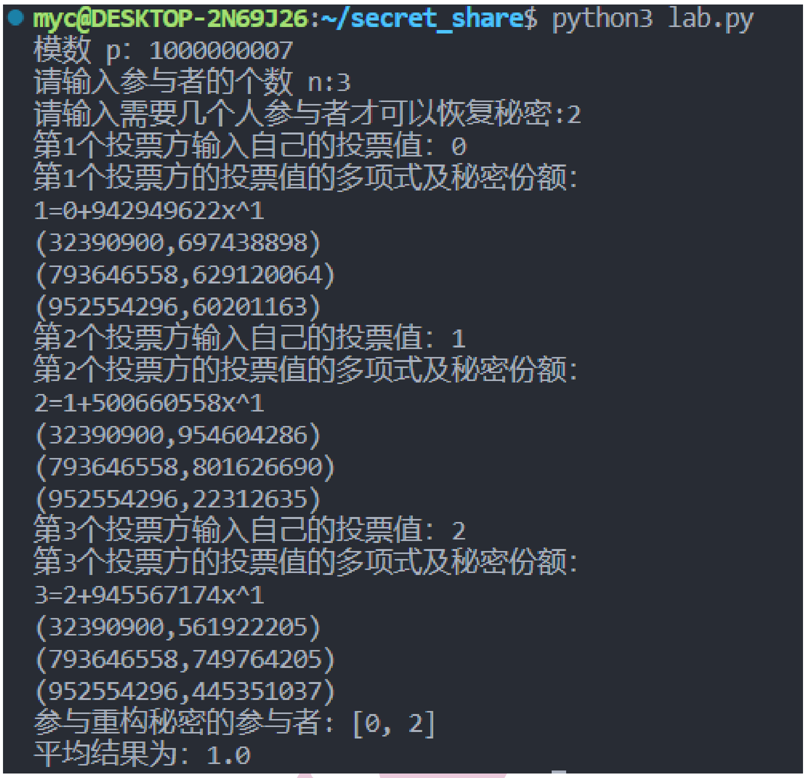
下面为代码流程

```python
if __name__ == '__main__':
    # 设置模数 p
    p=1000000007
    print(f'模数 p: {p}')

    # 输入门限 (t,n) 以及秘密值 s1,s2
    n=int(input(" 请输入参与者的个数 n:"))
    t=int(input(" 请输入需要几个人参与者才可以恢复秘密:"))
    s=[]# 三个投票方的秘密值
    f=[]# 三个秘密值对应的多项式
    shares_x=[]# 三个秘密值对应的秘密份额
    # 选择 n 个互不相同的随机值
    for i in range(0,n):
        temp=random.randrange(0,p)
        while temp in shares_x:
            temp=random.randrange(0,p)
        shares_x.append(temp)
    shares_y=[]
    for i in range(0,3):
        s.append(int(input(f'第{i+1}个投票方输入自己的投票值: ')))
        # 输出多项式及秘密份额
        print(f'第{i+1}个投票方的投票值的多项式及秘密份额: ')
        f.append(get_polynomial(s[i],t-1,p,str(i+1)))
        temp=[]
        for j in range(0,n):
            temp.append(count_polynomial(f[i],shares_x[j],p))
            print(f'({shares_x[j]},{temp[j]})')
        shares_y.append(temp)
    # 在 n 个参与者中任选 t 个人重构求和之后的值
    # 任意选取 t 个人
    Party=[]
    for i in range(0,t):
        temp=random.randint(0,n-1)
        while temp in Party:
            temp=random.randint(0,n-1)
        Party.append(temp)
    print('参与重构秘密的参与者:',Party)
    #t 个参与方分别将自己手中 s[0]、s[1] 和 s[2] 的秘密份额相加, 得到 s[0]+s[1]+s[2]
    ↪ 的秘密份额
    shares_s123_x=[]
    shares_s123_y=[]
    for i in range(0,t):
        shares_s123_x.append(shares_x[Party[i]])
        temp=0
        for j in range(0,3):
            temp+=shares_y[j][Party[i]]
        shares_s123_y.append(temp)
    s123=restructure_polynomial(shares_s123_x,shares_s123_y,t,p)/3
    print(f'平均结果为: {s123}')
```

四、 实验结果

运行代码，设置三人参与密钥分发，两人为门限值，具体过程和结果如下所示：



```
myc@DESKTOP-2N69J26:~/secret_share$ python3 lab.py
模数 p: 1000000007
请输入参与者的个数 n:3
请输入需要几个人参与者才可以恢复秘密:2
第1个投票方输入自己的投票值: 0
第1个投票方的投票值的多项式及秘密份额:
1=0+942949622x^1
(32390900,697438898)
(793646558,629120064)
(952554296,60201163)
第2个投票方输入自己的投票值: 1
第2个投票方的投票值的多项式及秘密份额:
2=1+500660558x^1
(32390900,954604286)
(793646558,801626690)
(952554296,22312635)
第3个投票方输入自己的投票值: 2
第3个投票方的投票值的多项式及秘密份额:
3=2+945567174x^1
(32390900,561922205)
(793646558,749764205)
(952554296,445351037)
参与重构秘密的参与者: [0, 2]
平均结果为: 1.0
```

图 2: 实验结果

可以看到，我们的结果最后的 $1.0 = (1+2+3)/3$，因此可以证明，本次实验取得圆满成功！

五、 心得体会

在实验中，实现了多项式插值法，用于重构多项式。多项式插值法是一种基于拉格朗日插值公式的方法，可以通过已知的多项式函数值，计算出多项式的系数，从而重构多项式。在 Shamir 秘密分享方案中，我们使用多项式插值法来重构秘密。

接着，实现了 Shamir 秘密分享方案。在该方案中，我们首先随机生成一个 t-1 次多项式 f(x)，其中 t 为阈值，然后将秘密 s 分成 n 份，分配给 n 个人。每个人都会得到一个点 (i,f(i))，其中 i 为该人的编号，f(i) 为多项式在 i 处的函数值。只有当收集到至少 t 个点时，才能通过多项式插值法重构出原始秘密 s。

此外，我们还发现，在实现 Shamir 秘密分享方案时，需要注意多项式系数和秘密的取值范围，避免出现大数运算和精度误差。同时，我们还可以通过使用多项式求导和快速幂运算等技巧，进一步提高 Shamir 秘密分享方案的效率和安全性。如果仔细观察结果，我们取的模数 p 非常大，也是保障了安全。

总之，通过本次实验，我们深入了解了 Shamir 秘密分享方案的原理和实现方法，掌握了多项式插值法和 Python 编程技巧，同时也加深了对数据安全的理解。

六、 附录：完整实验代码

实验完整代码如下所示：

实验完整代码

```
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=fname+'='+str(f[0])
    for i in range(1,T+1):
        f_print+='+'+str(f[i])+'x^'+str(i)
    print(f_print)
    return f
#计算多项式值
def count_polynomial(f,x,p):
    ans=f[0]
    for i in range(1,len(f)):
        ans=(ans+f[i]*quickpower(x,i,p))%p
    return ans
#重构函数 f 并返回 f(0)
def restructure_polynomial(x,fx,t,p):
    ans=0
    #利用多项式插值法计算出 x=0 时多项式的值
    for i in range(0,t):
        fx[i]=fx[i]%p
        fxi=1
        #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
        for j in range(0,t):
            if j !=i:
                fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p
        fxi=(fxi*fx[i])%p
        ans=(ans+fxi)%p
    return ans
```

```python
if __name__ == '__main__':
    #设置模数 p
    p=1000000007
    print(f'模数 p: {p}')

    #输入门限(t,n)以及秘密值 s1,s2
    n=int(input("请输入参与者的个数 n:"))
    t=int(input("请输入需要几个人参与者才可以恢复秘密:"))
    s=[]#三个投票方的秘密值
    f=[]#三个秘密值对应的多项式
    shares_x=[]#三个秘密值对应的秘密份额
    #选择 n 个互不相同的随机值
    for i in range(0,n):
        temp=random.randrange(0,p)
        while temp in shares_x:
            temp=random.randrange(0,p)
        shares_x.append(temp)
    shares_y=[]
    for i in range(0,3):
        s.append(int(input(f'第{i+1}个投票方输入自己的投票值: ')))
        #输出多项式及秘密份额
        print(f'第{i+1}个投票方的投票值的多项式及秘密份额: ')
        f.append(get_polynomial(s[i],t-1,p,str(i+1)))
        temp=[]
        for j in range(0,n):
            temp.append(count_polynomial(f[i],shares_x[j],p))
            print(f'({shares_x[j]},{temp[j]})')
        shares_y.append(temp)
    #在 n 个参与者中任选 t 个人重构求和之后的值
    #任意选取 t 个人
    Party=[]
    for i in range(0,t):
        temp=random.randint(0,n-1)
        while temp in Party:
            temp=random.randint(0,n-1)
        Party.append(temp)
    print('参与重构秘密的参与者:',Party)
    #t个参与方分别将自己手中s[0]、s[1]和s[2]的秘密份额相加, 得到s[0]+s[1]+s[2]的秘密份额
    shares_s123_x=[]
    shares_s123_y=[]
    for i in range(0,t):
        shares_s123_x.append(shares_x[Party[i]])
        temp=0
        for j in range(0,3):
            temp+=shares_y[j][Party[i]]
        shares_s123_y.append(temp)
    s123=restructure_polynomial(shares_s123_x,shares_s123_y,t,p)/3
```

```
    print(f'平均结果为：{s123}')
```