



南开大学
Nankai University

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	1
二、 实验要求	1
三、 实验过程	1
(一) 实验原理	1
1. 简单介绍	1
2. 具体步骤	1
(二) 实验代码	3
四、 实验结果	7
五、 心得体会	7
六、 附录：完整实验代码	8

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 Lagrange (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 n 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 $p = s + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$, 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 $(1, p(1)), (2, p(2)), \dots, (n, p(n))$, 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:

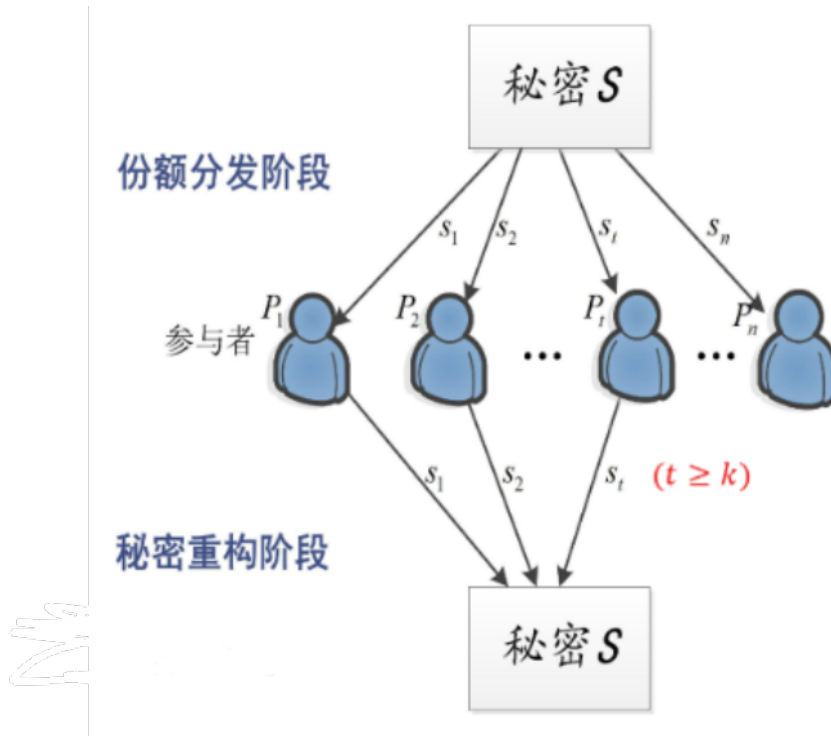


图 1: 整体流程

秘密分发

秘密分发者 D 给 n 个参与者 $P_i (0 \leq i \leq n)$ 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 $f(0) = a_0 = s$ 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 x_1, x_2, \dots, x_n ，计算 $(0 \leq i \leq n)$ 。
- (3) 将 (x_i, y_i) 分配给参与者 $P_i (0 \leq i \leq n)$ ，值 x_i 是公开的， y_i 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 $(x_1, y_1), (x_2, y_2), \dots, (x_t, y_t)$ 。由插值公式：

- (1) 多项式 $h(x)$ ：

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

- (2) 共享秘密 s 为：

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

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 $a^b \bmod p$

```

1 # 快速幂计算  $a^b \bmod p$ 
2 def quickpower(a,b,p):
3     a=a%p
4     ans=1
5     while b!=0:
6         if b&1:
7             ans=(ans*a)%p
8             b>>=1
9             a=(a*a)%p
10    return ans

```

注意，这里是解决 $\text{GF}(p)$ 的部分。这段代码实现了快速幂算法，用于计算 $a^b \bmod p$ 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**： x_0 为常数项系数， T 为最高次项次数， p 为模数， $fname$ 为多项式名

```

1 # 构建多项式:  $x_0$  为常数项系数,  $T$  为最高次项次数,  $p$  为模数,  $fname$  为多项式名
2 def get_polynomial(x0,T,p,fname):
3     f=[]
4     f.append(x0)
5     for i in range(0,T):
6         f.append(random.randrange(0,p))
7     # 输出多项式
8     f_print=fname+'='+str(f[0])
9     for i in range(1,T+1):
10        f_print+='+'+str(f[i])+'x'+str(i)
11    print(f_print)
12    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**

```

1 # 计算多项式值
2 def count_polynomial(f,x,p):
3     ans=f[0]
4     for i in range(1,len(f)):
5         ans=(ans+f[i]*quickpower(x,i,p))%p
6     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 $x^i \bmod p$ 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 $f(0)$**

```

1 # 重构函数 f 并返回 f(0)
2 def restructure_polynomial(x,fx,t,p):
3     ans=0
4     # 利用多项式插值法计算出 x=0 时多项式的值
5     for i in range(0,t):
6         fx[i]=fx[i]%p
7         fxi=1
8         # 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
9         for j in range(0,t):
10             if j !=i:
11                 fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p
12         fxi=(fxi*fx[i])%p
13         ans=(ans+fxi)%p
14     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 $c = x_j^{p-2} \bmod p$ 。
 - 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

```

1  if __name__ == '__main__':
2      # 设置模数 p
3      p=1000000007
4      print(f'模数 p: {p}')
5
6      # 输入门限 (t,n) 以及秘密值 s1,s2
7      n=int(input(" 请输入参与者的个数 n:"))
8      t=int(input(" 请输入需要几个人参与者才可以恢复秘密:"))
9      s=[] # 三个投票方的秘密值
10     f=[] # 三个秘密值对应的多项式
11     shares_x=[] # 三个秘密值对应的秘密份额
12     # 选择 n 个互不相同的随机值
13     for i in range(0,n):
14         temp=random.randrange(0,p)
15         while temp in shares_x:
16             temp=random.randrange(0,p)
17         shares_x.append(temp)
18     shares_y=[]
19     for i in range(0,3):
20         s.append(int(input(f'第{i+1}个投票方输入自己的投票值: ')))
21         # 输出多项式及秘密份额
22         print(f'第{i+1}个投票方的投票值的多项式及秘密份额: ')
23         f.append(get_polynomial(s[i],t-1,p,str(i+1)))
24         temp=[]
25         for j in range(0,n):
26             temp.append(count_polynomial(f[i],shares_x[j],p))
27             print(f'({shares_x[j]}, {temp[j]})')
28         shares_y.append(temp)
29     # 在 n 个参与者中任选 t 个人重构求和之后的值
30     # 任意选取 t 个人
31     Party=[]
32     for i in range(0,t):
33         temp=random.randint(0,n-1)
34         while temp in Party:
35             temp=random.randint(0,n-1)
36         Party.append(temp)
37     print('参与重构秘密的参与者:',Party)
38     #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
39     shares_s123_x=[]
40     shares_s123_y=[]
41     for i in range(0,t):
42         shares_s123_x.append(shares_x[Party[i]])
43         temp=0
44         for j in range(0,3):
45             temp+=shares_y[j][Party[i]]
46         shares_s123_y.append(temp)
47     s123=restructure_polynomial(shares_s123_x,shares_s123_y,t,p)/3
48     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

```

1 import random
2 #快速幂计算  $a^b \bmod p$ 
3 def quickpower(a,b,p):
4     a=a%p
5     ans=1
6     while b!=0:
7         if b&1:
8             ans=(ans*a)%p
9             b>>=1
10            a=(a*a)%p
11    return ans
12 #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
13 def get_polynomial(x0,T,p,fname):
14     f=[]
15     f.append(x0)
16     for i in range(0,T):
17         f.append(random.randrange(0,p))
18     #输出多项式
19     f_print=fname+'='+str(f[0])
20     for i in range(1,T+1):
21         f_print+=' '+str(f[i])+'x^'+str(i)
22     print(f_print)
23     return f
24 #计算多项式值
25 def count_polynomial(f,x,p):
26     ans=f[0]
27     for i in range(1,len(f)):
28         ans=(ans+f[i]*quickpower(x,i,p))%p
29     return ans
30 #重构函数 f 并返回 f(0)
31 def restructure_polynomial(x,fx,t,p):
32     ans=0
33     #利用多项式插值法计算出 x=0 时多项式的值
34     for i in range(0,t):
35         fx[i]=fx[i]%p
36         fxi=1
37         #在模 p 下,  $(a/b) \bmod p = (a*c) \bmod p$ , 其中 c 为 b 在模 p 下的逆元,  $c=b^{(p-2)} \bmod p$ 
38         for j in range(0,t):
39             if j !=i:
40                 fxi=(-1*fxi*x[j]*quickpower(x[i]-x[j],p-2,p))%p
41         fxi=(fxi*fx[i])%p
42         ans=(ans+fxi)%p
43     return ans

```

```

44 if __name__ == '__main__':
45     #设置模数 p
46     p=1000000007
47     print(f'模数 p: {p}')
48
49     #输入门限(t,n)以及秘密值 s1,s2
50     n=int(input("请输入参与者的个数 n:"))
51     t=int(input("请输入需要几个人参与者才可以恢复秘密:"))
52     s=[]#三个投票方的秘密值
53     f=[]#三个秘密值对应的多项式
54     shares_x=[]#三个秘密值对应的秘密份额
55     #选择 n 个互不相同的随机值
56     for i in range(0,n):
57         temp=random.randrange(0,p)
58         while temp in shares_x:
59             temp=random.randrange(0,p)
60         shares_x.append(temp)
61     shares_y=[]
62     for i in range(0,3):
63         s.append(int(input(f'第{i+1}个投票方输入自己的投票值: ')))
64         #输出多项式及秘密份额
65         print(f'第{i+1}个投票方的投票值的多项式及秘密份额: ')
66         f.append(get_polynomial(s[i],t-1,p,str(i+1)))
67         temp=[]
68         for j in range(0,n):
69             temp.append(count_polynomial(f[i],shares_x[j],p))
70             print(f'({shares_x[j]},{temp[j]})')
71         shares_y.append(temp)
72     #在 n 个参与者中任选 t 个人重构求和之后的值
73     #任意选取 t 个人
74     Party=[]
75     for i in range(0,t):
76         temp=random.randint(0,n-1)
77         while temp in Party:
78             temp=random.randint(0,n-1)
79         Party.append(temp)
80     print('参与重构秘密的参与者:',Party)
81     #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
82     # [2]的秘密份额
83     shares_s123_x=[]
84     shares_s123_y=[]
85     for i in range(0,t):
86         shares_s123_x.append(shares_x[Party[i]])
87         temp=0
88         for j in range(0,3):
89             temp+=shares_y[j][Party[i]]
90             shares_s123_y.append(temp)
91     s123=restructure_polynomial(shares_s123_x,shares_s123_y,t,p)/3

```

```
91 | print(f'平均结果为: {s123}')
```

NIKU