MPC多方安全计算

简介：安全多方计算是指在无可信第三方的情况下，多个参与方协同计算一个约定的函数，并且保证每一方仅获取自己的计算结果，无法通过计算过程中的交互数据推测出其他任意一方的输入和输出数据（除非函数本身可以由自己的输入推测出其他参与方的输入和输出）。

安全多方计算于1986 年由姚期智院士通过姚氏百万富翁问题提出：两个百万富翁街头邂逅，他们都想炫一下富，比一比谁更有钱，但是出于隐私，都不想让对方知道自己到底拥有多少财富，如何在不借助第三方的情况下，让他们知道谁更有钱。姚氏“百万富翁问题”后经发展，成为现代密码学中非常活跃的研究领域，即安全多方计算。

## **百万富翁问题**

**两个百万富翁都想比较到底谁更富有，但是有都不想让别人知道自己有多少钱。在没有可信的第三方的情况下如何进行？**

这个问题就是著名的姚式百万富翁问题。姚式，即大名鼎鼎的姚期智，我国唯一一个图灵奖获得者，此问题开创了安全多方计算领域。

为了说明，这里先简化问题，将两个富翁的财产限定在1千万到1亿之间，而且只想做千万级的比较。也就是说：两个富翁，分别为张三和李四。他们自己都清楚自己有几千万财产，也即，他们心里清楚 1～10中的一个数（代表自己千万级的财富）；他们想知道到底谁的数更大一些。

这里假定：

* 1、两人都值得信任，不会作假
* 2、两人都希望诚实地比较出谁更服务（即谁的数更大）
* 3、两人又都希望知道对方财产到底是多少，如果可能的话，拿到具体数字最好了

**一个”解决方案”**

有人提出这样一个解决方案：放一个天平，两边放上封闭的盒子，让两个富翁分别在两边放入质量相同的苹果，有几千万财富就放几个，最后看哪边重就可以了。就这么简单。

真的这么简单么？这里方案的提出者忽略了一个条件，也就是不存在可信的第三方，天平谁来提供？提供天平者是可以知道一切的。

这是一个看似简单实则非常复杂的问题。

**不经意传输的解决方案**

这个问题在数学上，必须借助于密码学。不经意传输是解决这个问题的绝妙方案。 那么什么是不经意传输呢？拿这个例子来说，就是张三给李四提供 n 个选择，这n个选择对李四而言都是无法分辨的（即无法获知原始内容的），李四从中选择一个并告诉张三。但有趣的是，张三不能知道李四选择的是哪一个。

回到百万富翁问题，一个简单的解决方案就是一下步骤：

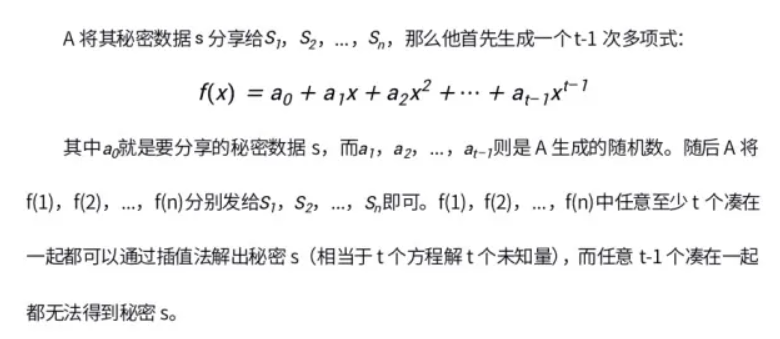
1. 1、张三找10个一模一样的箱子，按照1～10的顺序摆好，并按照自己的财富值分别往里面放入苹果梨和香蕉，具体放法为：如果序号小于自己的财富之，放入苹果，相等，则放入梨，大于自己的财富值，放入香蕉；
2. 2、把10个盒子都叫上锁；并叫李四过来（或者寄给李四）
3. 3、李四根据自己的财富值对相应的箱子再加一把锁。然后把其他所有箱子销毁。并把这个选择的箱子送给张三。
4. 4、张三看到送回来的箱子，但他不知道李四选择的是第几个箱子，因为每个箱子都是一样的
5. 5、张三李四分别开锁，看里面是什么水果： -- 如果是苹果，张三比李四富有； -- 如果是梨，两人一样有钱 -- 如果是香蕉，李四比张三富有

上面的例子中蕴含了半诚实模型、不经意传输等安全多方计算中用到的一系列思想和方法。

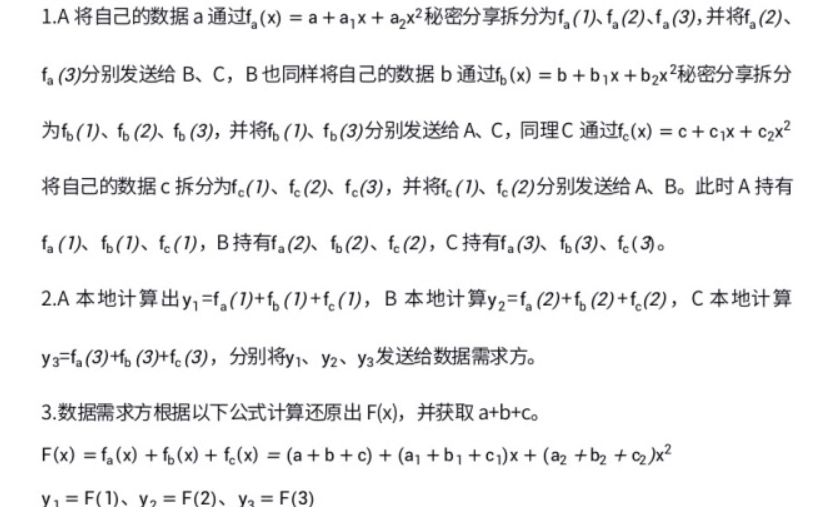
MPC拥有多种基础技术，比如密钥分享，随机预言机，不经意传输，姚氏混淆电路等。我这里主要研究密钥分享

**密钥分享**：秘密分享（SS，Secret Sharing）是指数据拆散成多个无意义的数，并将这些数分发到多个参与方那里。每个参与方拿到的都是原始数据的一部分，一个或少数几个参与方无法还原出原始数据，只有把各自的数据凑在一起时才能还原出真实数据。

秘密分享的一种经典方案，1979年Shamir提出了阈值秘密分享方案，该方案支持n个参与方中的任意t个可以联合解开秘密数据，具体方案如下：



假设三方A，B，C分别有数据a，b，c，有数据需求方想要计算a+b+c。

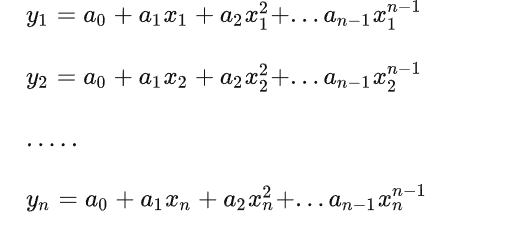


## **拉格朗日插值法**

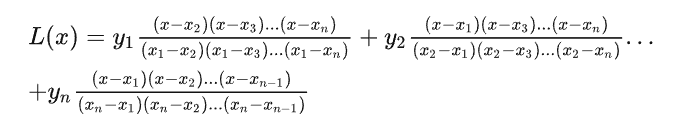
1. 已知 n个点的坐标 (x1,y1),(x2,y2),(x3,y3)....(xn,yn) ，求得一个 n-1 次多项式 过这些点。

2. 假设 n-1 次多项式为：

3. 将n个点代入多项式得：



4. 易解出拉格朗日插值多项式为：



下面是关于这个算法我写的一个demo

package com.MPC;  
  
import java.math.BigInteger;  
import java.util.Random;  
  
  
*/\*\*  
 \** ***@author*** *lisihan  
 \** ***@Description*** *MPC秘密分享  
 \** ***@date*** *2023/5/18-18:54  
 \*/*public class SecretSharing *{  
  
 /\*\*  
 \* 生成密钥分享  
 \** ***@param*** *secret 秘密值  
 \** ***@param*** *n 分享数量  
 \** ***@param*** *k 重构秘密所需的最小分享数量  
 \** ***@param*** *prime 大素数  
 \** ***@return*** *\*/* public static BigInteger*[]* generateShares*(*BigInteger secret, int n, int k, BigInteger prime*) {* // 存储多项式系数  
 BigInteger*[]* coefficients = new BigInteger*[*k - 1*]*;  
 // 存储生成的分享  
 BigInteger*[]* shares = new BigInteger*[*n*]*;  
  
 // 生成多项式随机系数  
 for *(*int i = 0; i < k - 1; i++*) {* coefficients*[*i*]* = *getRandomBigInteger(*prime*)*;  
 *}* // 计算秘密分享  
 for *(*int i = 1; i <= n; i++*) {* BigInteger x = BigInteger.*valueOf(*i*)*;  
 BigInteger y = secret;  
  
 // 计算多项式的值  
 for *(*int j = 0; j < k - 1; j++*) {* BigInteger coefficient = coefficients*[*j*]*;  
 BigInteger power = x.pow*(*j + 1*)*.mod*(*prime*)*;  
 BigInteger term = coefficient.multiply*(*power*)*.mod*(*prime*)*;  
 y = y.add*(*term*)*.mod*(*prime*)*;  
 *}* shares*[*i - 1*]* = y;  
 *}* return shares;  
 *}* public static BigInteger reconstructSecret*(*BigInteger*[]* shares, int k, BigInteger prime*) {* BigInteger secret = BigInteger.*ZERO*;  
  
 // 拉格朗日插值  
 for *(*int i = 0; i < k; i++*) {* BigInteger xi = BigInteger.*valueOf(*i + 1*)*;  
 BigInteger numerator = BigInteger.*ONE*;  
 BigInteger denominator = BigInteger.*ONE*;  
  
 for *(*int j = 0; j < k; j++*) {* if *(*i != j*) {* BigInteger xj = BigInteger.*valueOf(*j + 1*)*;  
 numerator = numerator.multiply*(*xj.negate*())*.mod*(*prime*)*;  
 denominator = denominator.multiply*(*xi.subtract*(*xj*))*.mod*(*prime*)*;  
 *}  
 }* BigInteger inverse = denominator.modInverse*(*prime*)*;  
 BigInteger share = shares*[*i*]*;  
 BigInteger term = share.multiply*(*numerator*)*.multiply*(*inverse*)*.mod*(*prime*)*;  
 secret = secret.add*(*term*)*.mod*(*prime*)*;  
 *}* return secret;  
 *}* public static BigInteger getRandomBigInteger*(*BigInteger limit*) {* BigInteger random;  
 do *{* random = new BigInteger*(*limit.bitLength*()*, new Random*())*;  
 *}* while *(*random.compareTo*(*limit*)* >= 0*)*;  
 return random;  
 *}* public static void main*(*String*[]* args*) {* BigInteger secret = new BigInteger*(*"42"*)*;  
 int n = 5;   
 int k = 3;   
 BigInteger prime = BigInteger.*valueOf(*2*)*.pow*(*128*)*.subtract*(*BigInteger.*ONE)*;  
  
 BigInteger*[]* shares = *generateShares(*secret, n, k, prime*)*;  
 System.*out*.println*(*"Shares: " + java.util.Arrays.*toString(*shares*))*;  
  
 BigInteger reconstructedSecret = *reconstructSecret(*shares, k, prime*)*;  
 System.*out*.println*(*"Reconstructed secret: " + reconstructedSecret*)*;  
 *}  
}*

