**PSI协议的简单实现**

**一、基础知识**

**1. 密文集合交集运算**

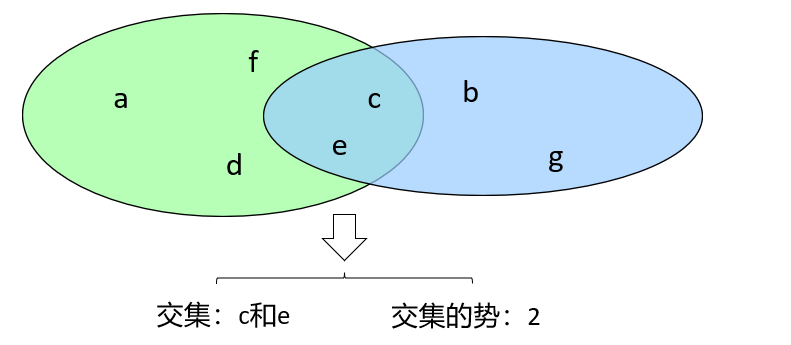
1. PSI：Privacy Set Intersect, PSI是安全多方计算的一个子类
2. 概念：两方隐私保护集合交集协议允许持有各自集合的两方来共同进行两个集合的交集计算。 协议最后的结果是一方或是两方得到正确的交集内容， 并且保证不会得到交集以外另一方集合中的任何信息。
3. PSI-CA：交集集合元素的个数
4. 场景：

条件：Alice拥有一个集合A={a1,a2,…,am}

Bob拥有一个集合B={b1,b2,…bn}

问题：求得交集I={x;x∈A∩B}

要求：对于不属于交集I的元素，Alice不能获得集合B-(A ∩ B)的元素，Bob也不能获得集合 A-(A ∩ B)的元素。



1. PSI的两类场景

* 平衡PSI：两方集合数量大体相等。
* 非平衡PSI：两方集合元素个数差异很大，具体表现在一大一小。

本实验要求基于公钥加密的原理，实现一个两方半诚实安全的PSI协议。其中使用的哈希单向函数选择布隆过滤器BF，BF是一种多哈希函数映射的快速查找算法，可以判断出某个元素肯定不在或者可能在集合中，即不会漏报但可能会误报，通常应用在一些需要快速判断某个元素是否属于集合，但不严格要求正确的场合。

**2. 协议流程**

1. 客户端和服务器端约定使用的RSA算法的模n、公钥e、比特位数m以及布隆过滤器函数BF，服务器端数据集为，客户端数据集为；服务器端生成私钥d
2. 客户端离线准备，随机生成随机数（盲因子），并对自己的数据进行盲化处理，得到，并计算盲化因子对应的逆元
3. 服务器计算
4. 客户端发送盲化后的数据
5. 服务器对客户端的数据进行盲签名，发送盲签名后得到的以及第三步中计算所得数据
6. 客户端计算获得，并与从服务器端接收到的数据进行比较匹配获得最终结果

**3. 实现方案**

使用python实现。为了简化，通过range函数取range(0, 1024)作为服务器端的数据集合，取range(0, 1024, 249)（即0、249、498、747、996）作为客户端的数据集合。

基于gmpy2库实现RSA算法，主要使用其中的invert函数进行乘法逆元的计算，使用powmod函数进行公钥加密和私钥签名；基于pycryptodome库生成密钥。哈希单向函数使用的BF基于pybloom\_live库实现。

**二、实验步骤**

**1. 搭建运行环境**

|  |
| --- |
| pip install gmpy2  pip install bitarray==1.7.1  pip install pycryptodome  pip install pybloom\_live |

**2. 编写程序实现**

|  |
| --- |
| import secrets  import gmpy2  import pybloom\_live  from Crypto.PublicKey import RSA  RSA\_BITS = 1024  RSA\_EXPONENT = 65537  RT\_COUNT = 0  def generate\_private\_key(bits=RSA\_BITS, e=RSA\_EXPONENT):  private\_key = RSA.generate(bits=bits, e=e)  public\_key = private\_key.publickey()  return private\_key  def generate\_random\_factors(public\_key):  random\_factors = []  rff = open('randomfactors.raw','w')  for \_ in range(RF\_COUNT):  r = secrets.randbelow(public\_key.n)  r\_inv = gmpy2.invert(r, public\_key.n)  r\_encrypted = gmpy2.powmod(r, public\_key.e, public\_key.n)  random\_factors.append((r\_inv, r\_encrypted))  rff.writelines(f"{r\_inv.digits()}\n")  rff.writelines(f"{r\_encrypted.digits()}\n")  rff.close()  return random\_factors  def blind\_data(my\_data\_set, random\_factors, n):  A = []  bdf = open('blinddata.raw','w')  for p, rf in zip(my\_data\_set, random\_factors):  r\_encrypted = rf[1]  blind\_result = (p \* r\_encrypted) % n  A.append(blind\_result)  bdf.writelines(f"{blind\_result.digits()}\n")  bdf.close()  return A  def setup\_bloom\_filter(private\_key, data\_set):  mode = pybloom\_live.ScalableBloomFilter.SMALL\_SET\_GROWTH  bf = pybloom\_live.ScalableBloomFilter(mode=mode)  for q in data\_set:  sign = gmpy2.powmod(q, private\_key.d, private\_key.n)  bf.add(sign)  bff = open('bloomfilter.raw','wb')  bf.tofile(bff)  bff.close()  return bf  def sign\_blind\_data(private\_key, A):  B = []  sbdf = open('signedblinddata.raw','w')  for a in A:  sign = gmpy2.powmod(a, private\_key.d, private\_key.n) #盲签名  B.append(sign)  sbdf.writelines(f"{sign.digits()}\n")  sbdf.close()  return B  def intersect(my\_data\_set, signed\_blind\_data, random\_factors, bloom\_filter, public\_key):  n = public\_key.n  result = []  for p, b, rf in zip(my\_data\_set, signed\_blind\_data, random\_factors):  r\_inv = rf[0]  to\_check = (b \* r\_inv) % n  if to\_check in bloom\_filter:  result.append(p)  return result  if \_\_name\_\_ == '\_\_main\_\_':  client\_data\_set = list(range(0, 1024, 249))  server\_data\_set = list(range(0, 1024))  RF\_COUNT = len(client\_data\_set)  private\_key = generate\_private\_key()  public\_key = private\_key.public\_key()  random\_factors = generate\_random\_factors(public\_key)  A = blind\_data(client\_data\_set, random\_factors, public\_key.n)  bf = setup\_bloom\_filter(private\_key, server\_data\_set)  B = sign\_blind\_data(private\_key, A)    result = intersect(client\_data\_set, B, random\_factors, bf, public\_key)  print(result) |

**3. 运行**

|  |
| --- |
| python main.py |

**三、实验要求**

**实验时间：**1周

**报告要求：**

1. 实验内容说明（简述实验做了些什么）

2. 实验代码分析（针对实验给出的代码，具体分析各关键函数的作用及如何实现的，给出代码流程图）

2. 实验步骤及截图

3. 实验结果分析

**提交方式：**通过雨课堂提交