

# PSI 协议的简单实现

姓 名:	柳致远
学 号:	2113683
专 业:	密码科学技术
指导老师:	邓琮弋
课 程:	密码学综合实验

## 目录

实验内容说明.....	2
基本内容: .....	2
协议流程: .....	2
实现方案: .....	2
实验代码分析.....	3
1、搭建运行环境 .....	3
2、代码实现过程 .....	3
3、代码流程图 .....	7
实验步骤及截图.....	8
1、 配置环境 .....	8
2、 代码运行结果截图 .....	8
实验结果分析.....	9

## 实验内容说明

### 基本内容：

用 python 实现基于 RSA 盲签名的 PSI 协议

### 协议流程：

- a) 客户端和服务端约定使用的 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  $\{q_1, q_2, \dots, q_y\}$ ，客户端数据集为  $\{p_1, p_2, \dots, p_x\}$ ；服务器端生成私钥  $d$
- b) 客户端离线准备，随机生成随机数（盲因子） $r_i \bmod n$ ，并对自己的数据进行盲化处理，得到  $p_i(r_i)^e \bmod n$ ，并计算盲化因子对应的逆元  $r_i^{-1}$
- c) 服务器计算  $BF((q_i)^d \bmod n)$
- d) 客户端发送盲化后的数据  $p_i(r_i)^e \bmod n$
- e) 服务器对客户端的数据进行盲签名，发送盲签名后得到的  $(p_i)^d r_i \bmod n$  以及第三步中计算所得数据  $BF((q_i)^d \bmod n)$
- f) 客户端计算  $BF((p_i)^d r_i r_i^{-1} \bmod n)$  获得  $BF((p_i)^d \bmod n)$ ，并与从服务器端接收到的数据  $BF((q_i)^d \bmod n)$  进行比较匹配获得最终结果

### 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。

# 实验代码分析

## 1、搭建运行环境

```
pip install gmpy2
pip install bitarray==1.7.1
pip install pycryptodome
pip install pybloom_live
```

## 2、代码实现过程

```
import secrets
import gmpy2
import pybloom_live
from Crypto.PublicKey import RSA

RSA_BITS = 1024
RSA_EXPONENT = 65537
RT_COUNT = 0
```

**import** 语句导入了需要的模块和库。

- **secrets:** 用于生成随机数，这在安全敏感的场景中是更好的选择。
- **gmpy2:** 提供了高精度数学运算的功能。
- **pybloom\_live:** 可能是用于布隆过滤器的库。
- **Crypto.PublicKey:** 用于处理 RSA 密钥对的库。

`RSA_BITS = 1024` 和 `RSA_EXPONENT = 65537` 定义了 RSA 密钥的位数和指数。

`RT_COUNT` 此处初始化为 0，后续会将其赋值为客户端数据集的长度，同时也将作为生成随机盲因子的个数。

```
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
```

**generate\_private\_key** 函数用于生成 RSA 密钥对，接受两个参数 **bits** 和 **e** 分别为 RSA 密钥的位数和指数。将生成的私钥存入 **private\_key**，并将生成的公钥存在 **public\_key** 中。

```
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
```

**generate\_random\_factors** 用于生成随机因子（盲因子）。接受公钥为参数，循环 RF\_COUNT 次（RF\_COUNT 为客户端数据集的长度），每次生成一个随机因子 **r**，并使用 **gmpy2.invert()** 函数计算 **r** 的模反元素，即 **r\_inv**。使用 **gmpy2.powmod()** 函数对随机数 **r** 进行模幂运算，得到 **r** 的公钥加密结果 **r\_encrypted**，用于对数据进行盲化操作。将 **(r\_inv, r\_encrypted)** 添加到 **random\_factors** 列表中，以元组形式存储。

```
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
```

`blind_data` 函数，用于执行盲化数据集。`blind_data` 函数接受三个参数：分别是 `my_data_set`：此处为客户端的数据集的列表。`random_factors`：包含随机因子的列表，每个随机因子是一个元组 (`r_inv`, `r_encrypted`)。`n`：RSA 公钥 `n`，为盲化的模数。在循环中，对于每个数据元素 `p` 和对应的随机因子 (`r_inv`, `r_encrypted`)，提取 `r_encrypted`。对数据元素 `p` 进行盲化运算，计算  $(p * r\_encrypted) \% n$ 。将盲化结果添加到列表 `A` 中。

```
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
```

`setup_bloom_filter` 函数用于设置布隆过滤器。首先初始化参数，使用 `for` 循环遍历数据集中的每个元素 `q`。在循环内部，对每个元素 `q` 使用 RSA 私钥进行签名，即计算 `gmpy2.powmod(q, private_key.d, private_key.n)`，并将对服务器的签名结果存储在 `sign` 中。将签名结果 `sign` 添加到布隆过滤器 `bf` 中。

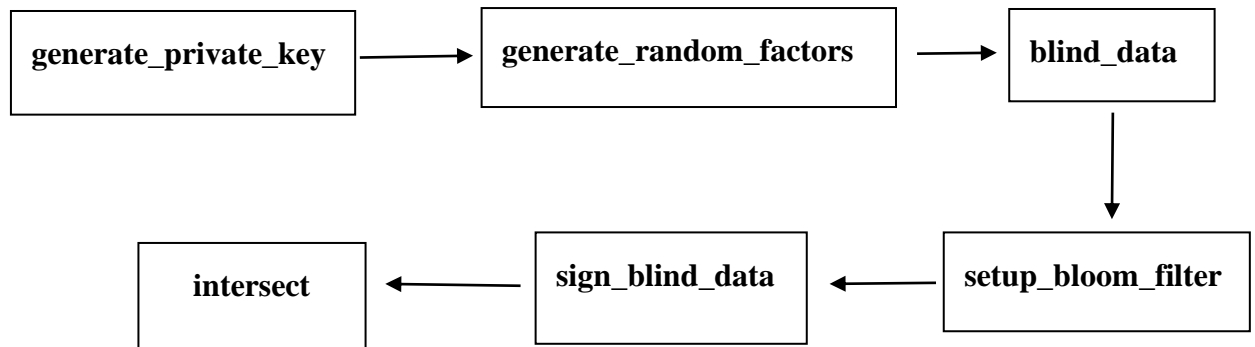
```
def sign_blind_data(private_key, A):
    B = []
    sbdf = open('signedblinddata.raw', 'w')
    for a in A:
        sign = gmpy2.powmod(a, private_key.d, private_key.n)    #盲签名
        B.append(sign)
        sbdf.writelines(f"{sign.digits()}\n")
    sbdf.close()
    return B
```

**sign\_blind\_data** 函数，用于对盲化数据进行盲签名。接受两个参数，一个是客户端数据集的盲化数据集 **A**，另一个为 RSA 的私钥。使用 **for** 循环遍历盲化数据集 **A** 中的每个元素 **a**。在循环内部，对每个盲化数据元素 **a** 使用 RSA 私钥进行盲签名，即计算 **gmpy2.powmod(a, private\_key.d, private\_key.n)**，并将签名结果存储在 **sign** 中。将盲签名结果 **sign** 添加到列表 **B** 中。

```
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
```

使用 **for** 循环同时遍历 **my\_data\_set**、**signed\_blind\_data** 和 **random\_factors** 中的元素，使用 **zip()** 函数将它们合并为一个可迭代对象。在循环内部，对于每个数据元素 **p**、盲签名数据元素 **b** 和对应的随机因子 (**r\_inv**, \_)，提取 **r\_inv**。计算要检查的值 **to\_check**，即 **(b \* r\_inv) % n**。使用布隆过滤器 **bloom\_filter** 检查 **to\_check** 是否存在于布隆过滤器中。如果 **to\_check** 存在于布隆过滤器中，则将原始数据元素 **p** 添加到结果列表 **result** 中。

### 3、代码流程图



## 实验步骤及截图

## 1、配置环境

```
C:\Windows\system32\cmd.exe > +
Microsoft Windows [版本 10.0.22631.3447]
(c) Microsoft Corporation。保留所有权利。

C:\Users\王磊>pip install gmpy2
Collecting gmpy2
  Obtaining dependency information for gmpy2 from https://files.pythonhosted.org/packages/3b/f9/eb3d3ef918d8b84085eb31a79224f38b94052a8b747f7f112a1889981/gmpy2-2.1.5-cp311-cp311-win_amd64.whl.metadata
    Downloading gmpy2-2.1.5-cp311-cp311-win_amd64.whl.metadata (3.0 kB)
    Downloading gmpy2-2.1.5-cp311-cp311-win_amd64.whl (1.1 MB)
  Installing collected packages: gmpy2
    Successfully installed gmpy2-2.1.5

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\王磊>pip install bitarray==1.7.1
Collecting bitarray==1.7.1
  Downloading bitarray-1.7.1.tar.gz (58 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheels for collected packages: bitarray
    Building wheel for bitarray (pyproject.toml) ... done
    Created wheel for bitarray: file:///C:/Users/王磊/AppData/Local/Temp/pip-build-cache/wheel/cc/6b/69/7f1e/7f13b3cae956c16b99492833ce79c9bd392ca98
    Stored in directory: C:\Users\王磊\AppData\Local\Pip\Cache\wheels/cc/6b/69/7f1e/7f13b3cae956c16b99492833ce79c9bd392ca98
  Successfully built bitarray
  Installing collected packages: bitarray
    Successfully installed bitarray-1.7.1

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

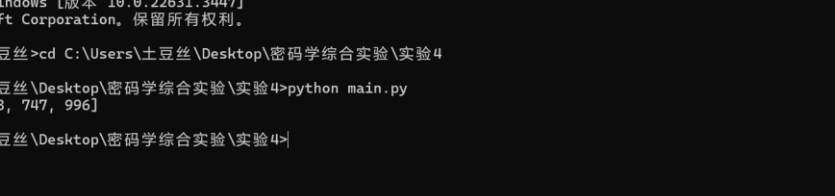
C:\Users\王磊>pip install pycryptodome
Collecting pycryptodome
  Obtaining dependency information for pycryptodome from https://files.pythonhosted.org/packages/1f/90/d131cbebd412902386fa8108b7cc135122288b71d4d24d77bebd7e2a76/pycryptodome-3.20.0-cp35-ab31-win_amd64.whl.metadata
    Downloading pycryptodome-3.20.0-cp35-ab31-win_amd64.whl.metadata (1.4 kB)
    Downloading pycryptodome-3.20.0-cp35-ab31-win_amd64.whl (1.8 MB)
  Installing collected packages: pycryptodome
    Successfully installed pycryptodome-3.20.0

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\王磊>pip install pybloom Live
Collecting pybloom Live
  Downloading pybloom_Live-4.0.0.tar.gz (10 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Requirement already satisfied: bitarray==0.3.4 in d:\python\lib\site-packages (from pybloom_Live) (1.7.1)
  Obtaining dependency information for xxhash==3.0.0 from https://files.pythonhosted.org/packages/b7/3a/714ae9796ef430fed6d413b6209882c150406b9373f62604dc6f3c/xxhash-3.4.1-cp311-cp311-win_amd64.whl.metadata
    Downloading xxhash-3.4.1-cp311-cp311-win_amd64.whl.metadata (12 kB)
    Downloading xxhash-3.4.1-cp311-cp311-win_amd64.whl (29 kB)
  Building wheels for collected packages: pybloom_Live
    Building wheel for pybloom_Live (pyproject.toml) ... done
    Created wheel for pybloom_Live: file:///C:/Users/王磊/AppData/Local/Temp/pip-build-cache/wheel/d6/f1/55/dbeaddc16a142880450dc659572d0281ba5d699a2b7f925657c
    Stored in directory: C:\Users\王磊\AppData\Local\Pip\Cache\wheels/d6/f1/55/dbeaddc16a142880450dc659572d0281ba5d699a2b7f925657c
  Successfully built pybloom_Live
  Installing collected packages: xxhash, pybloom_Live
    Successfully installed pybloom_Live-4.0.0 xxhash-3.4.1

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

## 2、代码运行结果截图



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\system32\cmd.exe". The window content displays the following text:

```
Microsoft Windows [版本 10.0.22631.3447]  
(c) Microsoft Corporation。保留所有权利。  
  
C:\Users\土豆丝>cd C:\Users\土豆丝\Desktop\密码学综合实验\实验4  
  
C:\Users\土豆丝\Desktop\密码学综合实验\实验4>python main.py  
[0, 249, 498, 747, 996]  
  
C:\Users\土豆丝\Desktop\密码学综合实验\实验4>
```



## 实验结果分析

由实验结果可知客户端和服务端数据集的交集为[0,249,498,747,996],该结果符合预期。在该实验中给客户端的数据集填充 **client\_data\_set = list(range(0, 1024, 249))**，而服务端的数据集填充为 **server\_data\_set = list(range(0, 1024))**，二者的交集正好为[0,249,498,747,996]，符合实验结果。