Private Set Intersection for Unequal Set Sizes

Author*: dev-goodata*

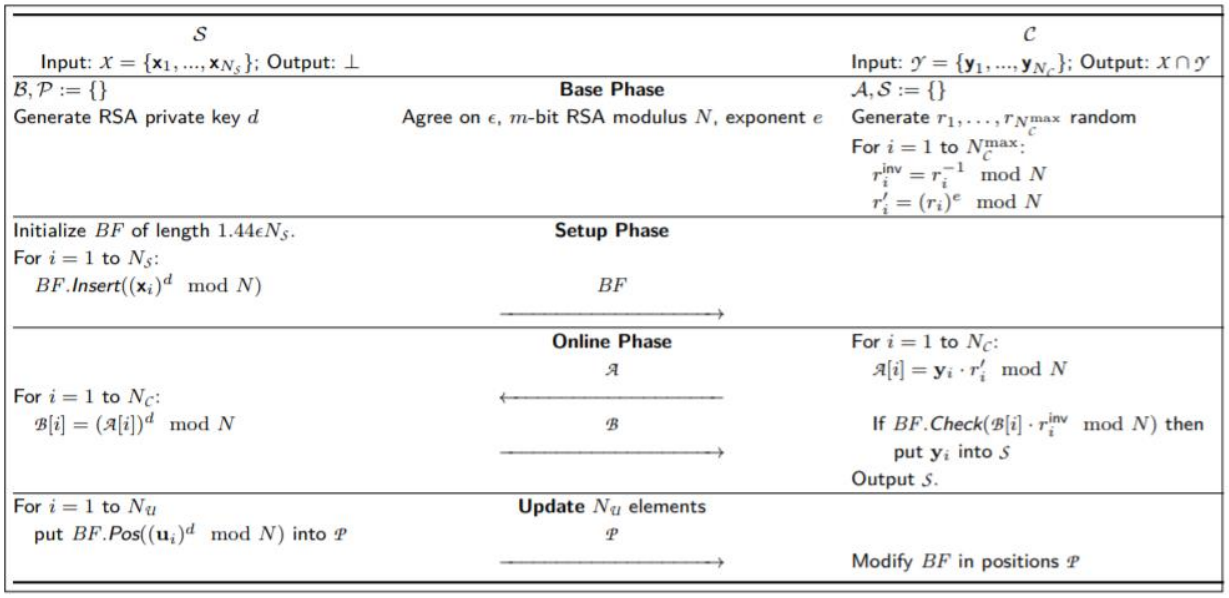
Summary

The private set intersection(PSI) in the main net is used for the DOs to get the intersection of their data without leaking other data and preserves the data privacy during data exchange. To achieve these goals, we have 3 steps to do.

1. Two DOs connect then get the intersection by PSI.
2. More than 3 DOs connect then get the intersection by PSI.
3. DOs compute the intersection under the assistance of mpc.

Literature review

https://encrypto.de/papers/KLSAP17.pdf
The algorithm I use is the RSA Blind Signature-based PSI(RSA-PSI) as described in the paper. The core of the algorithm is the RSA and Blind-Signature.



$\mathcal{S}$		$\mathcal{C}$
Input: $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_\mathcal{S}}\}$; Output: $\perp$		Input: $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_{N_\mathcal{C}}\}$; Output: $\mathcal{X} \cap \mathcal{Y}$
$\mathcal{B}, \mathcal{P} := \{\}$	**Base Phase**	$\mathcal{A}, \mathcal{S} := \{\}$
Generate RSA private key d	Agree on ϵ, m-bit RSA modulus N, exponent e	Generate $r_1, \ldots, r_{N_\mathcal{C}^{\max}}$ random
		For $i = 1$ to $N_\mathcal{C}^{\max}$:
		$r_i^{\mathsf{inv}} = r_i^{-1} \mod N$
		$r'_i = (r_i)^e \mod N$
Initialize BF of length $1.44\epsilon N_\mathcal{S}$.	**Setup Phase**	
For $i = 1$ to $N_\mathcal{S}$:		
$BF.\mathit{Insert}((\mathbf{x}_i)^d \mod N)$	$\xrightarrow{BF}$	
	Online Phase	For $i = 1$ to $N_\mathcal{C}$:
	$\xleftarrow{\mathcal{A}}$	$\mathcal{A}[i] = \mathbf{y}_i \cdot r'_i \mod N$
For $i = 1$ to $N_\mathcal{C}$:		
$\mathcal{B}[i] = (\mathcal{A}[i])^d \mod N$	$\xrightarrow{\mathcal{B}}$	If $BF.\mathit{Check}(\mathcal{B}[i] \cdot r_i^{\mathsf{inv}} \mod N)$ then
		put $\mathbf{y}_i$ into $\mathcal{S}$
		Output $\mathcal{S}$.
For $i = 1$ to $N_\mathcal{U}$	**Update** $N_\mathcal{U}$ **elements**	
put $BF.\mathit{Pos}((\mathbf{u}_i)^d \mod N)$ into $\mathcal{P}$	$\xrightarrow{\mathcal{P}}$	Modify BF in positions $\mathcal{P}$

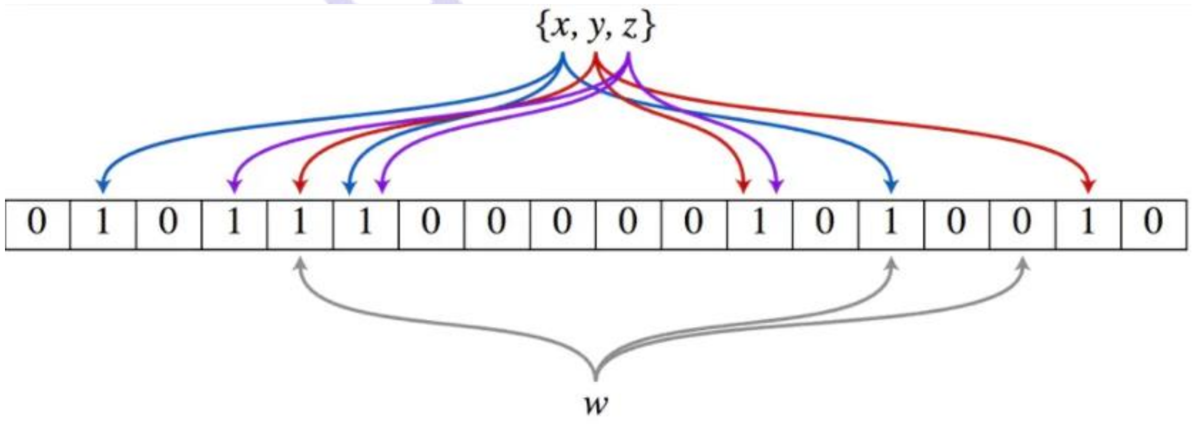
Base Phase:

1. Server generate RSA private key d(For Server to sign data), public key e and RSA modulus N(product of two big prime, p*q) , then send e and N to Client.
2. Client generates m(m is the size of Client's set) random number [r1,r2…,rm] (r is less than N), then generates r_inv and r' for blinding and unblinding Client's set.

Online Phase:

1. Server use d(private key) to sign its own set, then insert it into a new bloom filter. Then send the bloom filter to Client
2. Client blind its data by r' and N, then sends it to Server.
3. Server uses d(private key) to sign the data received from Client then send it back to Client.
4. Client receives the signed data from Server and unblind it using r_inv and N, finally check which element is in the bloom filter by the bloom filter's function Check().The elements that exist in the bloom filter are the intersection of Server with Client.

Bloom Filter:



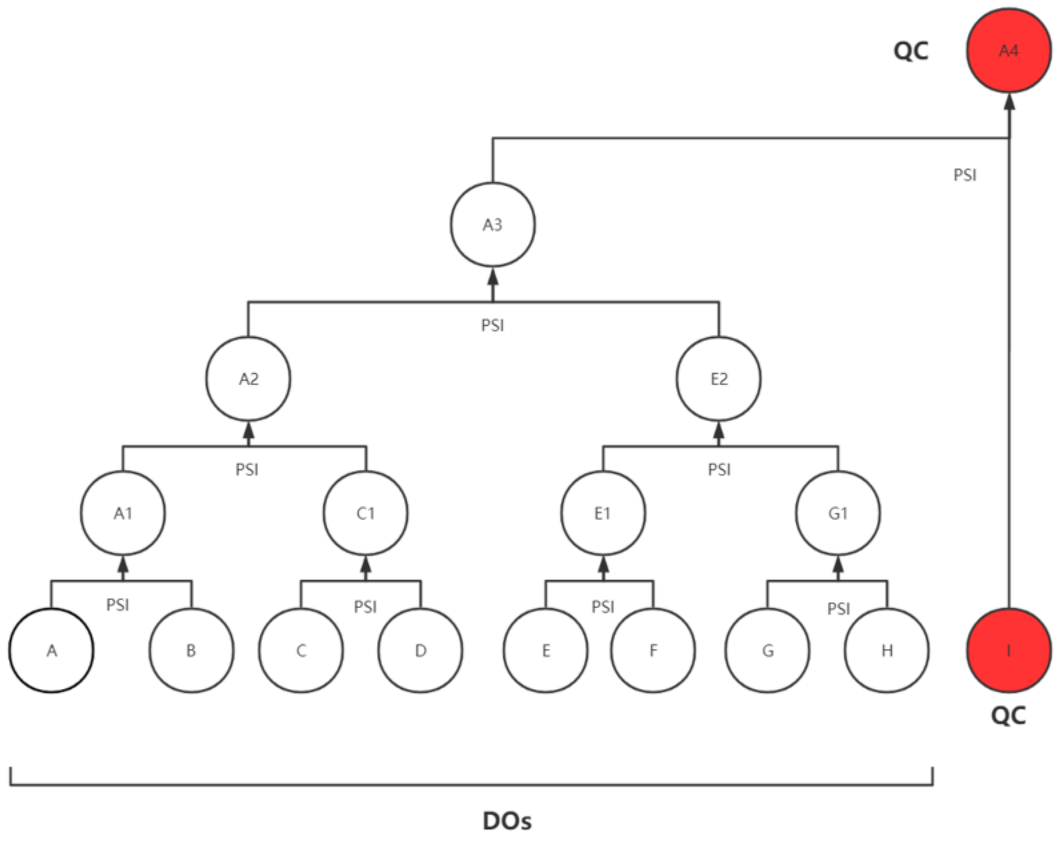
The picture above is a simple bloom filter that has 3 elements passed by the hash functions. Then judge the new element w whether it exists in the bf or not. The result is no because the indexes w passed by the hash functions are not 1 completely.

The bloom filter(BF) may get false positives because of the hash-collision.BF has a bit array and uses the hash functions to set the indexes in the bit array. When you want to judge whether a element is in the BF, the BF will input the element(bytes) into hash functions and get some indexes, if these indexes had been set as 1, it means the element exists in the BF.

Advantage and disadvantage:

The originally proposed protocol uses a cryptographic hash function H which the paper substitutes with a Bloom filter BF in order to achieve better communication complexity and lower client storage. The BF is implemented by a binary array and some hash function; it's used for judging whether an element is a member of a set or not. But there may be some loss(an element is not a member of A but the BF gets the wrong result, false positives).

Procedure of more than 3 DOs' RSA-PSI



1. (MPC)Divide DOs to couples, each couple includes two DOs. Then every couple gets intersection by RSA-PSI. In the start and end of a PSI procedure, DO should write event in chain. It is decided by GoodDataServer that which DO is Client and which DO is Server in PSI.
2. (MPC)Choose one DO from every couple in 1. as the new DOs, then divide them to couples(SS to do), do the same things as 1.Untill only one DO.
3. The DO get in 2. communicates with QC by PSI, QC gets the intersection of all DOs and QC.

In Particular, the mpc chosen by GoodDataService makes the plan of all DOs' PSI including how to divide couples, which DO is server or client.And the planning is decided by mpc at first and don't change, if error happens ,for example, mpc or some DO dies, it will lead to fail of all PSI. In the future, mpc may design the planning of PSI in every round by estimating each DO's calculation and network.

Implementation

I am prepared to implement The Blind RSA-based PSI by python3 based on the rsa library.

The main class and function is listed.

```
# PsiGrpc server. Receive requests ,handle then and return responses to client

class DoPsiServer:
    # init the server, load data ,create rsa key and add_log to chain
    def __init__(self, keyword_strlist: List[str], key_bit_length: int = 512)

    # Server receives and sends requests(responses) here with streaming grpc function
    def DoPsiConnection(self, request_iterator, context)

    # handle each request in request_iterator and yield response
    def handle_one_request(self, request, context)

    # start a new threading for preparing work. When finish, send client a response
    def prepare_for_next_round(self, status: PsiStatus)



#  PSI server including the functions that use for handling data

class PsiServer:


    # generate rsa key by self.generate_key
    def __init__(self,  key_length)

    # create Pubkey and Privkey according to the length
    def generate_key(self, bit_length)

    # sign data with server’s private key
    def server_sign(self, bytes_list, privkey)

    # build bloom filter from server’s signed data
    def build_bloom_filter(self, signed_list: List[bytes], capacity=10000, fp_prob=None)

    # server sign the bytes data and return signed bytes data
    def sign_data(bytes_list):
            return signed_data
```

```
# Psi Grpc client,send requests and receive response

class DoPsiClient:
    # init the client, set psi_server’s addr, load data, create channel..
    def __init__(channel):
                monitor()

    # get request from the request_queue and yield it
    def _process_events(self)

    # get request from ‘_process_events’ and send it to Server, then receive the responses
    # and save them in response_queue

    # start the client
    def start(self)

    # clean up and close the client
    def close(self)

    # put request in request_queue
    def send_request(self, request)

    # get request_result(response) from the response_queue
    def get_request_result(self)
```