Efficient Unbalanced Quorum PSI from Homomorphic Encryption

Xinpeng Yang
Zhejiang University
yangxinpeng@zju.edu.cn

Liang Cai
Zhejiang University
leoncai@zju.edu.cn

Yinghao Wang
Zhejiang University
asternight@zju.edu.cn

Keting Yin
Zhejiang University
yinkt@zju.edu.cn

Lu Sun
Zhejiang University
sunlu__xx@126.com

Jingwei Hu*
Nanyang Technological University
davidhu@ntu.edu.sg

ABSTRACT

Multiparty private set intersection (mPSI) protocol is capable of finding the intersection of multiple sets securely without revealing any other information. However, its limitation lies in processing only those elements present in every participant's set, which proves inadequate in scenarios where certain elements are common to several, but not all, sets.

In this paper, we introduce an innovative variant of the mPSI protocol named unbalanced quorum PSI to fill in the gaps of the mPSI protocol. Unlike the previous quorum-PSI proposals which detect elements present in at least k out of n equal sets, our protocol is particularly tailored for unbalanced cases where the size of the receiver's set is much smaller than the size of the senders' sets. Our work achieves logarithmic communication complexity in the semi-honest setting, significantly surpassing previous work in efficiency.

The benchmarks show that it takes 22.7 seconds in WAN and 14.7 seconds in LAN for online computation, and only 87.8 MB of total communication to intersect 5535 elements across 15 sets, each containing 2^{24} elements. Compared to prior work, this is roughly an 87× reduction in communication and a 31× reduction in online time. Our protocols can be easily extended to the larger set with 2^{28} elements which is nearly impractical for prior work.

KEYWORDS

Private Set Intersection; Homomorphic Encryption; MPC

1 INTRODUCTION

This paper focuses on a specialized multiparty computation (MPC) protocol known as Private Set Intersection (PSI). The PSI protocol takes set $X$and set Y as inputs and outputs the intersection of the two sets $X \cap Y$. The party that receives the result gets nothing about the set of another party except for the intersection. The PSI protocol is widely used in many scenarios such as private contact discovery (e.g. Signal or WhatsApp) [22, 33, 41] and private biometric identification [6, 19, 40].

Most of previous work focused on two-party PSI protocols and Freedman et al. [18] proposed the first multiparty PSI (mPSI). Following that, Kolesnikov et al. [24] proposed the first practical efficient multiparty PSI protocol relying on fast oblivious transfer (OT). Our work is an extension of the multiparty PSI protocol.

We have deeply considered the existing multiparty PSI schemes and raised two observations which are not well studied in the open literature. First, the goal of the multiparty PSI protocol is to securely detect the elements that are common to all sets. Secondly, we noted that in practical applications, participants' data sets often vary significantly in size, ranging from a few hundred to several million entries.

*The corresponding author.

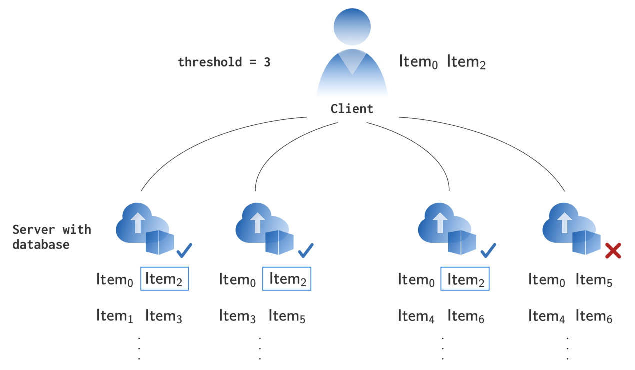
threshold = 3
Client
Item_0 Item_2
Server with database
Item_0 Item_2
Item_1 Item_3
Item_0 Item_2
Item_3 Item_5
Item_0 Item_2
Item_4 Item_6
Item_0 Item_5
Item_4 Item_6

Figure 1: The client interacts with multiple servers with query Item_2. The threshold is 3 and Item_2 is a match of quorum PSI protocol.

In this work, we propose an innovative mPSI variant called unbalanced quorum PSI protocol. The goal of the quorum PSI protocol is to find the elements that present in at least k out of n sets. In some special applications, a particular client has only a small list of data, but the server has a large database. Here the client acts as an enquirer, interacting with n servers to find out which elements its set appears in at least k servers' databases.

Our protocol broadens the scope of multiparty PSI beyond the standard computation of intersections. While conventional multiparty PSI focuses on finding common elements across all parties' sets, our protocol generalizes this approach. It specifically targets identifying elements that appear in a portion of the parties' sets. For instance, if a certain entity appears several times in different blacklists, we can say that the entity is a fraud with high probability, but it is not in the output of the multiparty PSI protocol. As is illustrated in Figure 1, where the client queries Item_2 and the threshold is set to 3, our protocol can find it but the multiparty PSI protocol cannot since the last set doesn't contain Item_2. Here are two interesting applications:

- *Identifying possible bad creditors*: An agency might wish to figure out if individuals appear on multiple banks' blacklists or high-risk lists, or a lender may need to verify if a borrower has simultaneous debts across different banks. Our protocol can help them find out these suspicious individuals without revealing any other private information.

- *Password leak check*: Clients can verify if their passwords have been compromised by running the unbalanced quorum PSI protocol with multiple password databases. The threshold is set to be 1 which means the password appears in at least one database. Our protocol preserves the privacy of these databases while informing the user if its password appears in any of them.

Our protocol focuses on unbalanced applications. The state-of-the-art quorum PSI [9] works well in the balanced setting, where the size of each set is the same. Their protocol is mainly based on generic multiparty communication and realized the balanced quorum PSI with a communication complexity $O(nm\kappa(\lambda+\kappa\log n))$ where n is the number of participants and m is the set size, κ and λ are the secure parameters. However, there is a limitation in their protocol that the communication cost scales linearly with the size of the larger set. This is a major concern, since in unbalanced scenarios, the larger set is often much larger than the smaller set. As a result, their protocol performs poorly when we extend it to unbalanced scenarios while our protocol works well achieving a logarithmic communication complexity with the size of the larger set. The experiment results show that our new construction reduces the communication costs and running time in WAN significantly.

1.1 Contribution

In this paper, we propose a new variant of quorum PSI protocol that focuses on unbalanced cases. We suppose there are n servers of set size n_s and one client of set size n_c. The inputs of the client are a few queries while the servers hold large sets such as databases that contain millions or billions of data i.e. $n_c \ll n_s$. Our protocol has the lowest communication complexity $O(n_c \log n_s)$ compared to previous work. Most of the previous related protocols [2, 9] only consider the balanced case and the state-of-art work [9] has the best communication complexity is $O(n_c + n_s)$. For example, when $n_c = 5535$ and $n_s = 2^{24}$ for 15 servers, [9] needs about 7.5 GB communication and 714 seconds while our protocol only needs 87.8 MB total communication and 22.7 seconds in WAN which reduces 87× communication and 31× computation.

1.2 Related Work

Research on the PSI protocol has been going on for many years and improved the performance of the PSI protocol dramatically. The prevailing approaches include Diffie-Hellman(DH) [15] and Oblivious Transfer(OT) [36]. The DH-based method [26] has a low communication cost while the OT-based [23, 31] method has a low computational cost. Most of the work focused on two-party PSI protocols and Kolesnikov et al. [24] proposed the first practical efficient multiparty PSI Protocol.

CLR17 [11] and CHLR18 [10] studied the PSI in the unbalanced scenarios where one of the two sets is much smaller than the other and Cong et al. [13] further reduced the concrete communication and computation. They constructed the efficient protocols based on fully homomorphic encryption (FHE) with the communication complexity $O(n_x \log n_y)$ where $n_x \ll n_y$. The FHE-based PSI protocol is more efficient than the generic PSI protocol in terms of communication cost and computational cost when there is hundreds of times difference in the set size. The generic PSI protocol is mainly aimed at cases where the set of participants is of the same size. But this is not applicable in unbalanced scenarios which is always at a cost of $O(n_x + n_y)$. When the size of the larger set is huge, the unbalanced PSI has a significant advantage over communication cost.

Bay et al. [2] also studied the quorum PSI and realized their protocol based on Paillier additive homomorphic encryption and bloom filter. The communication cost of this protocol is $O(n \cdot m \cdot l)$ where n is the number of parties, m is the size of the largest set and l is the threshold of dishonest parties. In the semi-honest and honest majority setting, the maximum dishonest parties can be up to $n/2$, which means the worst communication cost is $O(n^2 \cdot m)$. Due to the public-key encryption scheme, this protocol does not perform well in practice, and it takes about 45s for the 8 participants and 2^6 set size case.

1.3 Organization

First, we provide some preliminaries in section 2. Then in section 3, we first review the previous HE-based unbalanced PSI protocol and then, describe our unbalanced quorum PSI protocol. Finally, we present evaluation results and comparisons in section 4. We conclude the whole paper in section 5.

2 PRELIMINARIES

2.1 Notation

Throughout this paper, we will use the notations in Table 1.

Nation	Description
n_c	size of client's set
n_s	size of servers' sets
n	number of servers
k	threshold
T_i	cuckoo hash table or simple hash table
κ	statistical security parameter
λ	computational security parameter
$[\![x]\!]$	homomorphic encryption of a plaintext x
$[n]$	set of integers from 1 to n
$[a, b]$	set of integers from a to b
$\|X\|$	cardinality of set X
α	number of partitions
k_s	number of slices
σ	bit length of a slice
$\mathcal{W}$	base powers for DAG
$\langle x\rangle^B$	boolean share of x
$\langle x\rangle^A$	arithmetic share of x
$[x]$	threshold share of x

Table 1: Summary of frequent notations

2.2 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) is a type of encryption that allows users to evaluate arithmetic circuits directly on ciphertexts without decryption. In consideration of performance, the encryption parameters are often chosen to support circuits with a bounded multiplicative depth. Such schemes that only support circuits with bounded multiplicative depth are also named leveled fully homomorphic encryption, as is used in our scheme.

In this work, we use RLWE-based cryptosystems BFV [17] implemented in SEAL [38]. The scheme is defined over a plaintext ring $\mathcal{Z}_t[X]/(x^n+1)$ and ciphertext ring $\mathcal{Z}_p[X]/(x^n+1)$, where n is a power of 2 and $t \ll q$ are integers. It's customary to define $R = \mathcal{Z}[X]/(x^n+1)$ so the ciphertext ring and plaintext ring can be denoted as $R_q = R/qR$ and $R_t = R/tR$. The ciphertext space is $R_q^2 = R/qR \times R/qR$. Thus, the size of each ciphertext is $2n\log(q)$. BFV is a noise-based scheme, which means that a ciphertext contains noise and the noise grows as the circuit is evaluated. There is a limited noise budget for each ciphertext, beyond which the decryption will fail.

The plaintext modular t is often chosen to be a prime number that satisfies $t \equiv 1 \; mod \; 2n$. Such t enables us to obtain a ring isomorphism from the plaintext space R_t to $\mathcal{Z}_t^n$. The isomorphism translates polynomial additions and multiplications into slot-wise additions and multiplications in each of the n fields $\mathcal{Z}_t$. This technique is often referred to as SIMD(Single Instruction, Multiple Data). We refer readers to [4, 20] for more details.

SIMD allows us to evaluate a circuit on a batch of inputs efficiently. In practice, the selection of parameter (n, q, t) determines the security level and the maximum depth of the circuit being evaluated. The security level is estimated using the LWE estimator [1].

In our scheme, we will be evaluating homomorphic addition and multiplication on ciphertexts. The costs of one step of evaluation can be viewed as the noise growth in the ciphertext and the computation time. In BFV, the cost of homomorphic multiplication is much greater than homomorphic addition, both in noise growth and computation time. We refer readers to [17] for a detailed analysis. In such consideration, it is vital to keep the multiplication gates in the circuit as few as possible.

2.3 Estimation of the max load of hash table

When hashing elements into a hash table, there could be multiple items hashed to the same table entry. Upon the procedure finishes, the distribution of the elements may not be even. Denote the number of max elements in one entry as K. For security, common PSI protocols fill in dummy items in those entries with fewer elements until they all contain K element. It is vital to estimate the maximum load of the hash entries which determines the efficiency of a scheme. For ease of estimation, we assume the hash function is a random function. From a balls-into-bins arguments in [35], putting m balls into n bins (corresponding to hashing m elements into n entries) when $m \gg n\log^3 n$ yields the following result with high probability:

$$K \le \frac{m}{n} + \sqrt{\frac{2m\log n}{n}}$$

When $m \ll n$ there is no concrete estimation of K. The distribution of bound can be computed as follows:

$$\Pr[\text{maximum height} > K] \le n \sum_{i=K+1}^{m} \binom{m}{i} \left(\frac{1}{n}\right)^i \left(1 - \frac{1}{n}\right)^{m-i} \quad (1)$$

2.4 Cuckoo Table

Cuckoo hashing [29] utilizes K different hash functions $h_1, \dots, h_K : \{0,1\}^\sigma \to [B]$ which map m elements into B bins. The mapping is done in the following way: For an element x, if any of the bins with index $h_i(x)$ is empty, the element x is placed in that bin. If multiple bins are empty, the one with the smallest index is chosen. If all bins are occupied, a random $i \in [K]$ is selected, and the element in bin $h_i(x)$ is evicted and replaced by x. The evicted element is then reinserted using the same procedure. This process is repeated until all elements are inserted or a predefined number of iterations is reached.

In the case that an upper limit of iterations is reached and still we cannot find a vacancy for the element, we say that the cuckoo hashing fails. The probability of failure is analyzed by Pinkas et al. [34], who show that choosing $K = 3$ and $B = 1.27m$ yields a failure probability of 2^{-40}. The same parameter is adopted in this work.

2.5 Two-party Functionalities

Two two-party functionalities are utilized in our scheme.

Equality Test. The equality test functionality $\mathcal{F}_{\mathsf{EQ}}^{l}$ takes as private input one l-bit string a and b from each of the two parties P_1, P_2 respectively. The two parties receive boolean shares of bit 1 if $a = b$, and bit 0 otherwise as output.

Boolean to Arithmetic Share Conversion. The boolean to arithmetic share conversion functionality $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}}$ converts boolean shares of a bit b to additive arithmetic shares of b over a field $\mathbb{F}$. Concretely speaking, the functionality takes as private input from parties P_1 and P_2 boolean shares $\langle b\rangle_1^b$ and $\langle b\rangle_2^b$ of a bit b and outputs additive arithmetic shares over $\mathbb{F}$ $\langle b\rangle_1^a$ and $\langle b\rangle_2^a$ of b. We implement the functionality with CrypTFlow2 [37] using correlated OT.

2.6 Multiparty Functionalities

Our scheme assumes an honest majority setting. We utilize the following multiparty functionalities, implemented with the protocols from [16, 25].

Let $\mathbb{F}(+,\cdot)$ be a finite field. Let n be the number of parties and $t < n/2$ be the corruption threshold. Abusing notations, we use $[a]$ to denote an (n,t)-linear secret sharing of element $a \in \mathbb{F}$ where the i-th party P_i holds the share $[a]_i$. The linearity of the sharing scheme enables each party to locally calculates the shares of any linear combination of the shared elements.

- $\mathcal{F}_{\mathsf{Random}}^{n,t}(l)$: Generates $[r_1], \dots, [r_l]$ for uniform random $r_1, \dots, r_l \in \mathbb{F}$.
- $\mathcal{F}_{\mathsf{Mult}}^{n,t}([a],[b])$: Takes as input $[a],[b]$ and outputs $[a \cdot b]$.
- $\mathcal{F}_{\mathsf{Reveal}}^{n,t}([a])$: Takes as input $[a]$ and outputs a.
- $\mathcal{F}_{\mathsf{Reshare}}^{n}(d, \langle a\rangle^a)$: Takes additive shares $\langle a\rangle^a$ where $a \in \mathbb{F}$ and outputs a (n,d)-sharing of a.
- $\mathcal{F}_{\mathsf{DoubleRandom}}^{n,t}(l)$: Generates $[r_1], \dots, [r_l]$ and $\langle r_1\rangle^a, \dots, \langle r_l\rangle^a$ for uniform random $r_1, \dots, r_l \in \mathbb{F}$.

Functionality	Communication	Rounds
$\mathcal{F}^{n,t}_{\text{Random}}$	$\lceil \frac{l}{n-t} n(n-1) \lceil \log \lvert\mathbb{F}\rvert \rceil$	1
$\mathcal{F}^{n,t}_{\text{Mult}}$	$2\left(\frac{2n}{n-t}+3\right)(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	5
$\mathcal{F}^{n,t}_{\text{Reveal}}$	$(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	1
$\mathcal{F}^{n}_{\text{Reshare}}$	$2(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	2
$\mathcal{F}^{n,t}_{\text{DoubleRandom}}$	$2\left\lceil \frac{l}{n-t} \right\rceil n(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	1

Table 2: Communication costs of n-party functionalities.

The communication cost of each of the functionalities is listed in Table 2. For simplicity, we assume in $\mathcal{F}^{n,t}_{\text{Random}}$ and $\mathcal{F}^{n,t}_{\text{DoubleRandom}}$ that $l \gg n$. We let $\lceil j/(n-t) \rceil$ to be $j/(n-t)$ and approximate t/n with $1/2$.

3 PREVIOUS HE-BASED UNBALANCED PSI

In this section, we will review the previous HE-based PSI protocol and then illustrate how to use it in our protocol.

CLR17 [11] focused on the PSI in unbalanced scenarios with the communication complexity $O(n_x \log n_y)$ where $n_x \ll n_y$. The generic PSI protocol is mainly aimed at cases where the set of participants is of the same size. Nevertheless, this is not applicable in unbalanced scenarios which is always at a cost of $O(n_x + n_y)$. When the size of the larger set is huge, the unbalanced PSI has a significant advantage over communication cost.

The first step is data preprocessing, which is to map the elements to fixed-length bits. As mentioned in CLR17, we usually use an OPRF approach instead of a naive hash to avoid the noise flooding of homomorphic encryption. We denote $X' = \{\mathsf{OPRF}(x)\}_{x \in X}$ and $Y' = \{\mathsf{OPRF}(y)\}_{y \in Y}$. Using a suitable collision-resistant hash parameter, we can say that $X \cap Y = X' \cap Y'$ with a false positive probability of no more than $1 - 2^{-\kappa}$. Following that, we use X' to replace the elements in X.

Then they store the elements in the hash table. Let h_1, h_2, h_3 be the hash functions used in the mapping process since using two hash functions is not friendly to stash-less cuckoo hash. The client stores its elements into a cuckoo hash table C of size β and the server i stores its elements $y \in Y$ into a simple hash table $\mathcal{S}$ where each element will be stored in 3 bins $\mathcal{S}[h_j(x)]$ for $j = 1, 2, 3$. There is exactly one element in each slot of C and may be several elements in each slot of $\mathcal{S}$. For each element in the bin of C, all possible matching elements can be found inside the corresponding bin of T_i because they are stored in the same bin. The intersection can then be derived from all the bin-wise intersections of C and $\mathcal{S}$:

$$X' \cap Y' = \bigcup_{k \in [\beta]} (C[k] \cap \mathcal{S}[k])$$

The server represents its elements in $\mathcal{S}[k]$ as a polynomial:

$$F_k(x) = \prod_{y \in \mathcal{S}[k]} (x - y) = \sum a_i \cdot x^i$$

This polynomial equals zero only when $x \in \mathcal{S}[k]$ and random number otherwise. The server evaluates the polynomial $F_k(x)$ using homomorphic encryption and the client receives $F_k(\llbracket x \rrbracket)$ for $x \in C[k]$. The client only knows the value of $F_k(x)$ and nothing about the server's set. This is known as the private membership test (PMT), which effectively determines whether an element belongs to a set. We denote this functionality as:

$$\mathsf{PMT}(x, Y) = \begin{cases} 1 & \text{if } x \in Y \\ 0 & \text{if } x \notin Y \end{cases}$$

where x is an element and Y is a set.

3.1 Trade off of Computation and Communication

The server needs to evaluate the PMT polynomials while the server has the coefficients of the polynomials a_i and the client has the power x^i. If the client computes all the powers needed by the polynomials and sends to the server, the server only needs to perform the scalar multiplication and addition without fully homomorphic encryption but additive homomorphic encryption. In the other case, the server computes all the power itself and then evaluates the polynomials. The computation cost is relatively high, but the communication cost is low. We need to find a trade-off between computation and communication. Previous work [13] utilizes the Directed Acyclic Graphs (DAG) and Paterson-Stockmeyer algorithm [30] to reduce the computation and communication overhead. As mentioned in the preliminaries, the cost of scalar multiplication is much less than non-scalar multiplication. For a polynomial $f(x) = \sum_{i=0}^{n} a_i \cdot x^i$, we need n scalar multiplications and n non-scalar multiplications. However, we can use the Paterson-Stockmeyer algorithm to reduce the cost to $O(\sqrt{n})$ non-scalar multiplications. The server chooses two parameters such that $n = L \cdot H - 1$ and then computes the powers $y^2, y^3, \cdots, y^{L-1}$ and $y^L, y^{2L}, \cdots, y^{(H-1)L}$. The server evaluates the polynomial in the following way:

$$f(y) = \sum_{j=0}^{H-1} \left(y^{j*L} \cdot \sum_{i=0}^{L-1} a_{j*L+i} \cdot y^i \right)$$

where the total cost is $L + 2H - 4$ non-scalar multiplications and the minimum cost is $O(\sqrt{n})$ when $L \approx \sqrt{2(n+1)}$.

Since we are using leveled fully homomorphic encryption, we can only compute circuits of a limited depth. To compute powers $y^2, y^3, \cdots, y^{L-1}$, we need a circuit of depth $O(\log L)$ which is still unacceptable for a large L. One way to reduce the depth is that the client computes base powers $y^2, y^4, \cdots, y^{\lceil \log_2 L \rceil}$ in advance and then sends to the server. Following that, the server just needs to evaluate the polynomial at depth $O(\log \log L)$. However, this is not enough, and we need to reduce the number of base powers. For example, to compute a polynomial of degree 28 at a circuit depth of 3, the client has to send base powers y^1, y^2, y^4. But we can use base powers y^1, y^5 instead which reduces about 33% communication cost. This problem can be viewed as a variant of *Global postage-stamp problem* [7, 8] and we refer to the parameter of [8] to optimize the number of base powers.

3.2 PSI with Computation

The traditional private set interaction protocol, often referred to as plain PSI, directly computes the intersection of sets and usually only one party will get the result. However, there is a need for additional computation on the intersection[12, 14, 32], e.g., by transferring them to an MPC circuit. The participants will only get a share of the intersection result. So we call it PSI with computation or circuit PSI. Similarly, our protocol ensures that the result is also secret shared after the intersection

HE-base PSI protocol can also be turned into the circuit PSI protocol. Instead of returning the result directly, the server sends the value with a mask and the client decrypts it to get the secret sharing. The server chooses a random number r and sends the ciphertext $[\![\boldsymbol{F_k(x) + r}]\!]$ to the client. The client decrypts the ciphertext and gets $F_k(x)+r$. The client gets the secret sharing of $F_k(x)+r$ and the server gets r. Only when the element belongs to the set, $F_k(x)$ will be $\mathbb{0}$, that is $F_k(x) + r = r$. Following this, both client and servers run the equality test protocol and get the secret sharing of $\mathbb{1}$ or $\mathbb{0}$. The client and server can get the secret sharing of $\mathbb{1}$ if the element x belongs to the server's set and $\mathbb{0}$ otherwise.

4 UNBALANCED QUORUM PRIVATE SET INTERSECTION

In this section, we elaborate on the details of our protocol, building upon the foundational knowledge previously discussed. We begin by formally defining the quorum PSI protocol and its objectives.

The goal of quorum PSI is to compute all the elements that are present in the client's set and at least k times in the servers' sets. In other words, an element appears in the quorum private set intersection if and only if it is found in the client's set and at least k servers' sets.

Suppose the sets of the n servers is $S_1, S_2, \cdots, S_n$. A trivial method to compute the quorum private set intersection is to search all subsets of $\{1, 2, \cdots, n\}$ whose size is k and compute the multi-party private set intersection of C and $S_{d_1}, S_{d_2}, \cdots, S_{d_k}$ where set $d = \{d_1, d_2, \cdots, d_k\} \subseteq \{1, 2, \cdots, n\}$. We can get the result by

$$I = \bigcup_{|d|=k, d\subseteq[n]} \left(C \cap (\bigcap_{j\in d} S_j) \right)$$

However, this method is neither efficient nor secure, as it requires iterating through $\binom{n}{k}$ subsets, leading to exponential computational complexity.

The ideal functionality of unbalanced quorum PSI is illustrated in Figure 2. Our protocol is a multiparty protocol and it is not allowed to reveal any partial intersection results. In essence, the client is prohibited from disclosing any information regarding the intersection between its set and the sets of other servers. To realize our protocol, we need to protect the following private information:

- Can not reveal any additional information in the semi-honest setting
- Can not reveal the total times an element appears in all sets
- Can not reveal which k servers have the same element that is in the intersection I

4.1 An Overview of Unbalanced Quorum PSI

Our building block is the *unbalanced circuit PSI protocol* and it is the combination of unbalanced PSI protocol and circuit PSI protocol. A key observation is that it is possible to turn unbalanced PSI protocol

There are n servers $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n$ and a client C. The threshold is $k \in [1, n]$.
Input: For each $i \in [n]$, $\mathcal{S}_i$ inputs a set S_i of size n_s. The client inputs a set C of size n_c. Here $n_c \ll n_s$.
Output: Denote the times that the element $C[j]$ appears in all servers' sets as:

$$cnt_j = |\{i : C[j] \in S_i \text{ for } i \in [n]\}|$$

then output

$$I = \{C[j] : cnt_j \geq k \ \text{ for } j \in [n_c]\}$$

to the client. Nothing to the servers.

Figure 2: Unbalanced Quorum PSI Functionality

into circuit PSI protocol. In our scenario, the size of the client's set is significantly smaller compared to the size of the servers' sets. Consequently, our protocol leverages the advantages offered by the unbalanced PSI protocol such as CLR17 [11] and WT23 [41]. Based on previous work, we turn the unbalanced PSI protocol into an unbalanced circuit PSI protocol. We illustrate our new idea by providing a simpler example involving the interaction between the client and a single server. As is mentioned in circuit PSI, we shall not get the matching results directly but compute the secret sharing of the intersection. Assume the elements in the client's set are $\{x_1, x_2, \cdots, x_{n_c}\}$. The client and server will get a random vector $\vec{s}_1$ and $\vec{s}_2$ separately. It satisfies that

$$\vec{s}_1[j] \oplus \vec{s}_2[j] = \begin{cases} 1 & \text{if} \quad x_j \in Y \\ 0 & \text{otherwise} \end{cases}$$

After that, we will take the matching results as the inputs to a MPC circuit. Here $\vec{s}_1$ and $\vec{s}_2$ are both boolean shares but it is not friendly to arithmetic computations while most operations of our MPC circuits are additions and multiplications.

To facilitate arithmetic operations in our MPC circuit, we convert these boolean shares into arithmetic shares over a finite field $\mathbb{F}_p$. That is:

$$\vec{s}_1[j] + \vec{s}_2[j] = \begin{cases} 1 & \text{if} \quad x_j \in Y \\ 0 & \text{otherwise} \end{cases}$$

where the components of vector $\vec{s}_1$ and $\vec{s}_2$ are elements over a finite field $\mathbb{F}_p$. Denote the elements-wise sum of the two vectors of client and server i as $\vec{v}_i$, that is $\vec{s}_1[j]+\vec{s}_2[j] = \vec{v}_i[j]$ for $j \in [n_c]$ and $\vec{s}_1$ and $\vec{s}_2$ is the arithmetic share of $\vec{v}_i$. The client repeats the above process for every server to obtain the shares of $\vec{v}_1, \vec{v}_i, \cdots, \vec{v}_n$. It is easily to find that $cnt_j = \sum_{i=1}^{n} \vec{v}_i[j]$ and if $\sum_{i=1}^{n} \vec{v}_i[j] \geq k$, the element x_j appears at least k times in servers' set and therefore $x_j \in I$. So the last step for us is to sum all the shares and perform a comparison protocol to get the final result.

Our scheme protects the privacy of the client and servers. All of our intermediate computation data are secret-shared and only the client will get the answers in the final phase. The client only knows the interaction I and does not know any additional information about the sets of servers nor which servers' sets contain the elements in interaction.

4.2 Revisit Polynomial Link (PoL)

We use the state-of-the-art unbalanced PSI protocol WT23 [41] as our foundation. They proposed the polynomial link and reduced the communication cost significantly. In the following, we describe how PoL works and how we turn PoL into the unbalanced circuit-PSI protocol.

Wu et al. [41] gave a novel insight to reduce the total communication cost of HE-base unbalanced PSI protocol. They mainly optimized the communication from the client to the servers. They found that the first slice of an element is enough to represent the whole element and then construct special polynomials to build the link between the first slice and the other slices. Suppose an element is divided into k_s slices and the i-th slice of x is $x^{(i)}$, which is $x = x^{(1)}||x^{(2)}||\cdots||x^{(k_s)}$. We define the polynomials $f_j(x)$ for $j \in [k_s]$ which satisfy the property for each element $x_i \in X$:

$$\begin{aligned} f_1\left(x_i^{(1)}\right) &= 0 \\ f_2\left(x_i^{(1)}\right) &= x_i^{(2)} \\ &\vdots \\ f_{k_s}\left(x_i^{(1)}\right) &= x_i^{(k_s)} \end{aligned}$$

In other words, the polynomial $f_j(\cdot)$ maps the first slice of each element $x_i^{(1)}$ onto the other slices $x_i^{(j)}$ for all $x_i \in X$ and $j \in [2, k_s]$. We take $\left(x_1^{(1)}, x_1^{(j)}\right), \left(x_2^{(1)}, x_2^{(j)}\right), \cdots$ for $j \in [2, k_s]$ as points to build the polynomials $f_j(\cdot)$ for $j \in [2, k_s]$. And if the element y belongs to X which means there is an element $x_k \in X$ and $x_k^{(1)} = y^{(1)}$, the client only receives the matching result if and only if $f_1\left(y^{(1)}\right) = 0$ and $f_j\left(y^{(1)}\right) = y^{(j)}$ for $j \in [2, k_s]$.

If we find that there are two elements $x_i \neq x_j$ but $x_i^{(1)} = x_j^{(1)}$, PoL will fail. This leads to the polynomials $f_j(\cdot)$ for $j \in [2, k_s]$ mapping the same first slice of x_i and x_j onto the different slices. To avoid this problem, we need to split the set into smaller partitions and run PoL for each partition.

Based on the observations above, the client only needs to send the first slice of the element rather than the whole element. This saves k_s times the communication $Com_{c\to s}$ but increases the communication $Com_{s\to c}$ by k_s times because the server needs to reply with more polynomial evaluation results. However, in practical implementations like SEAL [38], usually use the modulus switching [5] to reduce the size of ciphertext after evaluation. Therefore, $Com_{c\to s}$ is much bigger than $Com_{s\to c}$ so that the total communication is even lower.

The false positive probability of PoL refers to the probability that a client's set element is incorrectly identified as an intersected element. Suppose the size of set X is β and the binary length of the slice is σ. So the collision probability of the first slice is $\beta \cdot \frac{1}{2^\sigma}$ and the collision probability of each of the rest $k_s - 1$ slice is $\frac{1}{2^\sigma}$ since all the first slices are mapped to the same value $\mathbf{0}$, but the remaining slices are all mapped to different values. As a result, the final collision probability of a mismatching element is $\beta \cdot \frac{1}{2^\sigma} \cdot \left(\frac{1}{2^\sigma}\right)^{k_s - 1}$ which is the same as the naive normal polynomial without splitting elements. We need to select appropriate β and $\sigma \cdot k_s$ such that the collision probability is lower than a secure parameter.

4.3 PoL Extension for Unbalanced Circuit PSI

PoL is still a plain unbalanced PSI protocol and the client can directly get the matching results. Here we propose a new PoL extension. As mentioned above, we add the matching result sent by the server with a random mask r, and the server keeps the r as its share.

Denote the DAG base powers is $\mathcal{W}$ and the degree of the polynomials is β. The client sends $[\![(\boldsymbol{y}^{(1)})^j]\!]$ for $j \in \mathcal{W}$ as the DAG base powers. Upon receiving the base powers, the server expands the base powers to get all the powers $[\![(\boldsymbol{y}^{(1)})^j]\!]$ for $j \in [\beta]$. And then the server evaluates the polynomials for $i \in [1, k_s]$:

$$[\![\boldsymbol{z_i}]\!] = f_i([\![\boldsymbol{y}^{(1)}]\!])$$

Before sending it back to the client, the server masks the polynomial evaluation results with random r_i. The share of server is r_i for $i \in [1, k_s]$. Upon receiving the ciphertexts, the client decrypts the ciphertext $[\![\boldsymbol{z_i} + \boldsymbol{r_i}]\!]$ to get the plaintext $z_i + r_i$ and we denote it as $\hat{z}_i$. Here the matching targets for the client are $0, y^{(2)}, y^{(3)}, \cdots, y^{(d)}$, but the polynomial evaluation results received by the client are masked with random value. Therefore, the client needs to compute the difference of $\hat{z}_i$ and $y^{(j)}$ for $j \in [2, k_s]$ and z'_i and $\mathbf{0}$. Then we obtain $z'_1 = z_1 - \mathbf{0}$ and $z'_i = \hat{z}_i - y^{(i)}$ for $i \in [2, k_s]$. If y is a matching result, i.e. all slices of y will equal to all slices of an element in the set, where $z'_i = f_i(y^{(1)}) + r_i - y^{(i)} = r_i$ for all $i \in [1, k_s]$.

Notice that the binary length of each element's slice is σ and the predicate holds:

$$y \in X \Leftrightarrow (z'_1 == r_1) \wedge (z'_2 == r_2) \wedge \cdots (z'_{k_s} == r_{k_s})$$

And we can further pack these k_s slices together to check whether $z'_1||z'_2||\cdots||z'_{k_s}$ equals to $r_1||r_2||\cdots||r_{k_s}$ which is both $k_s \cdot \sigma$ bits. We denote the equality test result of $z'_1||z'_2||\cdots||z'_{k_s}$ and $r_1||r_2||\cdots||r_{k_s}$ as eq and the client and the server will receive boolean shares $\langle eq \rangle_0$ and $\langle eq \rangle_1$ separately which satisfies $\langle eq \rangle_0 \oplus \langle eq \rangle_1 = eq$.

The above procedure simply checks whether an individual element y belongs to set X. Concretely, combined with SIMD and cuckoo hash table, the matching process is always paralleled. We need to deal with a vector instead of an element. In addition, we split the set into α partitions to make sure that all the duplications are divided into different partitions. We are supposed to repeat the above procedure for all the partitions of the same set. But the element y appears in at most one of the α partitions such that partitioning will not affect the correctness of the protocol. Based on the above analysis, our protocol works well in combination with SIMD and partitioning. The detailed protocol will be described in Section 4.6.

4.3.1 Optimization for unbalanced circuit PSI. The first step of the PSI protocol is to hash the elements through OPRF instead of using the original items directly. This approach has several advantages against previous schemes. Firstly, it reduces the computational and communication overhead of the protocol by compressing the length of an element to a fixed length under some security parameter when we are handling longer items. Secondly, it provides security against malicious client, even though the hash of an element not in the intersection is leaked, the client can't know any information about the original element. This avoids the use of noise flooding in homomorphic encryption. However, our protocol does not require the use of OPRF.

As mentioned in protocol [39], the use of OPRF does not provide the same benefits in our circuit PSI protocol and instead, we use a universal hash function, which is agreed upon in advance, to map the elements. In circuit PSI protocol, the client will not get the intersection hence it is unnecessary to use OPRF to hide the elements against malicious client and the simpler universal hashing suffices to compress the elements.

4.4 Convert 2-PC share into multiparty share

After performing the extended PoL discussed in Section 4.3, the client and the server have the boolean share of matching results. The next step is to compute the multiparty comparison function based on the boolean share. Notice that all the current shares are two-party shares, which means that the secret is divided into two parts. We use the additive secret sharing scheme to convert the two-party shares into multiparty shares. And, the arithmetic share is more suitable for the arithmetic circuits in the MPC, which consist of multiplication and addition.

Suppose that the secret share of PMT results for the client and server i is $\langle eq^i_j \rangle^B_0$ and $\langle eq^i_j \rangle^B_1$ for $j \in [1, n_c]$. Here $\langle eq^i_j \rangle^B_0 \oplus \langle eq^i_j \rangle^B_1 = eq^i_j$ is the PMT result of element $C[j]$. We first convert it into arithmetic shares using $\mathcal{F}^{\mathbb{F}}_{\mathsf{B2A}}$ functionality, that is $\langle eq^i_j \rangle^A_0$ and $\langle eq^i_j \rangle^A_1$ for $j \in [1, n_c]$ and $\langle eq^i_j \rangle^A_0 + \langle eq^i_j \rangle^A_1 = \langle eq^i_j \rangle^B_0 \oplus \langle eq^i_j \rangle^B_1$. We find that the target cnt_j equals the sum of all the PMT results of the client and servers for element $C[j]$. That is $cnt_j = \sum_{i=1}^{n} \mathsf{PMT}(C[j], S_i)$. So we can compute the sum of all the shares of eq^i_j and get the arithmetic share of cnt_j:

$$cnt_j = \sum_{i=1}^{n} \left(\langle eq^i_j \rangle^A_0 + \langle eq^i_j \rangle^A_1 \right) = \left(\sum_{i=1}^{n} \langle eq^i_j \rangle^A_0 \right) + \sum_{i=1}^{n} \langle eq^i_j \rangle^A_1$$

Here the $\sum_{i=1}^{n} \langle eq^i_j \rangle^A_0$ can be computed by the client and the server i just keeps $\langle eq^i_j \rangle^A_1$ for its share of cnt_j. In this way, the client and the servers can get the arithmetic share of cnt_j which is truly shared between multiple parties.

Combined with partitioning. For the sake of simplicity, we did not introduce additional subscripts to distinguish between different partitions in the above scheme. However, the unbalanced PSI protocols [11, 13, 41] always use partitioning to lower the degree of the polynomials, and it's more efficient in the existing homomorphic encryption schemes such as SEAL [38]. Our scheme is also compatible with the partitioning technique. For a given element $C[j]$, we will get the PMT result for different partitions. We denote the PMT result of the client and server i for partition k as $\langle eq^i_{j,k} \rangle^B_0$ and $\langle eq^i_{j,k} \rangle^B_1$ for $j \in [1, n_c]$ and $k \in [1, k_s]$.

We first convert it into arithmetic shares using $\mathcal{F}^{\mathbb{F}}_{\mathsf{B2A}}$ functionality, that is $\langle eq^i_{j,k} \rangle^A_0$ and $\langle eq^i_{j,k} \rangle^A_1$. Likewise, we can still add the secret sharing of different partitions together due to the additive secret sharing scheme. An element exists in at most one partition simultaneously, so the sum of these secret shares is up to $\mathbb{1}$. If the sum is $\mathbb{0}$, it means that the element is not contained in any partition, i.e., it is not in the set. Both the servers and the client compute $\langle eq^i_j \rangle^A_1 = \sum_{k=1}^{k_s} \langle eq^i_{j,k} \rangle^A_1$ and $\langle eq^i_j \rangle^A_0 = \sum_{k=1}^{k_s} \langle eq^i_{j,k} \rangle^A_0$. To conclude, our scheme works even if the partitioning is applied.

Parameters: There are n servers $\mathcal{S}_1, \cdots, \mathcal{S}_n$ and a client C, with threshold secret shares $[x]$. The field is denoted as $\mathbb{F}_p$ for a prime p.
Define the polynomial (publicly known to all):

$$\psi_1(x) = \begin{cases} (x-k)\cdot(x-(k+1))\cdots(x-n) & k \geq \frac{n}{2} \\ (x-0)\cdot(x-1)\cdots(x-(k-1)) & k < \frac{n}{2} \end{cases}$$

Input: For each $i \in [n]$, P_i inputs its secret shares $[x]_i$ n_s. The client inputs its secret shares $[x]_0$.
Protocol:

(1) Pre-processing: $C, \mathcal{S}_1, \cdots, \mathcal{S}_n$ run:
$[s_1], \cdots, [s_J] \leftarrow \mathsf{Random}(J)$
(2) Evaluating the polynomial: $C, \mathcal{S}_1, \cdots, \mathcal{S}_n$ run:
- invoke $\mathcal{F}^{n,t}_{\mathsf{Mult}}$ to compute all the required powers $[x]^i$ and followed by local scalar multiplications and additions to get $[\psi_1(x)]$.
- for each $j \in [J]$, compute $[v_j] \leftarrow \mathcal{F}^{n,t}_{\mathsf{Mult}}([\psi_1(x)], [s_j])$ and send to the client to recover v_j.

Output: When $k \geq \frac{n}{2}$, if $v_j = 0$ holds for all $j \in [J]$, output **1** otherwise **0**. When $k < \frac{n}{2}$, if $v_j = 0$ holds for all $j \in [J]$, output **0** otherwise **1**.

Figure 3: Comparison Protocol $\mathcal{F}_{\mathsf{CMP}}$

4.5 Multiparty Comparison Protocol

The final step is to compare the cnt_j and the threshold k. If $cnt_j \geq k$ holds, the element $C[j]$ belongs to I. Now the client and the servers obtain its respective additive secret shares of cnt_j and the shares are distributed to the client and n servers. We employ the comparison protocol proposed in [9] to execute the threshold comparison for the secret shares distributed to the client and n servers.

Our protocol follows the widely used honest majority security which requires that most of the parties are not corrupted and keep their shares secret. Consequently, using the threshold secret share to replace the additive share is a better choice for the sake of efficiency. Here we use the Shamir secret sharing scheme to perform the comparison functionality in Figure 3.

The range of cnt_j is $[0, n]$. For $k \geq \frac{n}{2}$, we consider the polynomial $\psi_1(x) = (x-k)\cdot(x-(k+1))\cdots(x-n)$ which satisfies the following property $\psi_1(x) = 0$ if $x \geq k$. Similarly, $k < \frac{n}{2}$, we use the polynomial $\psi_1(x) = (x-0)\cdot(x-1)\cdots(x-(k-1))$ which satisfies the following property $\psi_1(x) = 0$ if $x < k$.

However, this polynomial will leak some additional information when x are not in the expected range. For example, when $k < \frac{n}{2}$

and $x \geq k$, it is easy to find the value of x from $\psi_1(x)$. The client will be able to know exactly how many times the element has appeared which violates the security of our protocol. As is shown in Figure 3 step (1), we generate some secret value s_i and multiply it with $\psi_1(x)$ to hide the original value of $\psi_1(x)$. In this case, the client only gets the value of $s_i \cdot \psi_1(x)$ and it is impossible to know the value of $\psi_1(x)$ without knowing s_i. But this will not affect the correctness of the protocol because the client can check whether $\psi_1(x) = 0$ with a probability $1 - 2^{-\kappa}$. Given that the s_i is a random value in the field $\mathbb{F}_p$, and the probability of $s_i = 0$ is $\frac{1}{p}$. This assertion will be checked for J times, and we need to choose an appreciate J to make sure $(\frac{1}{p})^J < 2^{-\kappa}$ which means $J = \lceil \frac{\kappa}{\log p} \rceil$.

We use the polynomial $\psi_1(x)$ in our implementation. This construction requires additional J multiplications and reduces the degree of polynomial to $min(n-k+1, k)$. However, this is not the best choice for all cases such as when $min(n-k+1,k) + J > n$. We can use another polynomial $\psi_2(x)$ to replace $\psi_1(x)$ which is more efficient for small n. The polynomial $\psi_2(x)$ is defined as:

$$\psi_2(x) = \begin{cases} 1 & x \geq k \\ 0 & x < k \end{cases}$$

which can be constructed using the polynomial interpolation algorithm and its degree is $n+1$. This polynomial will not leak any additional information because its value keeps the same for any $x \geq k$ and vice versa. The polynomial $\psi_2(x)$ is more efficient when the number of servers is relatively small. And the polynomial $\psi_1(x)$ is better in the case of a larger number of servers or smaller k.

We use the comparison protocol to check whether cnt_j is beyond the threshold. However, it can also be generalized to check between-threshold and below-threshold. Therefore, we can just modify the polynomial ψ and use our protocol to find the elements that present in no more than k sets and so on.

4.6 Full Protocol

The main techniques used in this work are the unbalanced circuit PSI protocol, two-party protocol, and multiparty protocol. As shown in Figure 4, we first the unbalanced circuit PSI protocol to obtain the matching results of the client and servers. Then we perform the two-party protocol to convert the matching results into the equality test results. Finally, we invoke the multiparty protocol to compute the multiparty comparison function. Our protocol is separated into two phases. The first phase is offline and preprocessing the data. The second phase is online and the client interacts with servers. We detail the full protocol in the following.

First phase is to pre-process the data, where all participants hash the data using an appropriate universal hash function $\mathcal{H}$. And then they store the elements in the hash table. The pre-process is offline and can be done before interacting. Let h_1, h_2, h_3 be the hash functions used in the mapping process. The client stores its elements into a cuckoo hash table T_0 of size B and the server i stores its elements into a simple hash table T_i where each element will be stored in 3 bins. There is at most one element in each slot of T_0 and there may be several elements in each slot of T_i. For empty slot in the cuckoo hash table, the client places a dummy element. For each element in the bin of T_0, all possible matching elements can be found inside the corresponding bin of T_i because they are stored in the same bin. The intersection can then be derived from all the bin-wise intersections of T_0 and T_i. not solved

Second phase is to perform the unbalanced circuit PSI protocols. However, we can not obtain the PMT result directly because the bin size in T_i is quite big and also we need to divide a bin into several partitions to avoid the duplication of the first slice. The servers i divide a bin $T_i[j]$ into α partitions and the result of $\mathsf{PMT}(T_0[j], T_i[j][k])$ is $\langle eq^i_{j,k} \rangle^B_0$ and $\langle eq^i_{j,k} \rangle^B_1$.

An element belongs to at most one of the α partitions. The $\mathsf{PMT}(T_0[j], T_i[j][k])$ for $k \in [\alpha]$ is either all $\mathbb{0}$ or a $\mathbb{1}$ and the rest are $\mathbb{0}$. So the sum $\sum_{k=1}^{\alpha} \mathsf{PMT}(T_0[j], T_i[j][k])$ can determine whether element $T_0[j]$ belongs to partition $T_i[j][k]$, i.e., $\mathbb{0}$ for presence and $\mathbb{1}$ for absence.

However, we just get the boolean secret sharing which is not friendly with addition. We have two ways to solve this problem. One is to use **AND** gate to aggregate these boolean shares together and another is using $\mathcal{F}^{\mathbb{F}}_{B2A}$ to convert boolean share to arithmetic share and then addition can be applied. Considering that our subsequent circuits are all arithmetic circuits, we choose the second method of converting boolean share to arithmetic share. Here we get:

$$\langle eq^i_{j,k} \rangle^B_0 \oplus \langle eq^i_{j,k} \rangle^B_0 = \langle eq^i_{j,k} \rangle^A_0 + \langle eq^i_{j,k} \rangle^A_0$$

. And then both the client and server sum the result of each partition $\langle eq^i_j \rangle^A_0 = \sum_{k=1}^{\alpha} \langle eq^i_{j,k} \rangle^A_0$ and $\langle eq^i_j \rangle^A_1 = \sum_{k=1}^{\alpha} \langle eq^i_{j,k} \rangle^A_1$ which satisfies:

$$\begin{aligned} \langle eq^i_j \rangle^A_0 + \langle eq^i_j \rangle^A_1 &= \sum_{k=1}^{\alpha} \left(\langle eq^i_{j,k} \rangle^A_0 + \langle eq^i_{j,k} \rangle^A_1 \right) \\ &= \sum_{k=1}^{\alpha} eq^i_{j,k} = eq^i_j \end{aligned}$$

Here the equality test result eq^i_j indicates whether element $T_0[j]$ belongs to the set T_i. It will be $\mathbb{1}$ if element $T_0[j]$ is in the set T_i and $\mathbb{0}$ otherwise. Once obtaining the equality test results, we just need one step to compute the cnt_j that is $cnt_j = \sum_{i=1}^{n} eq^i_j$. The client sums all the share of k-th bin and gets $\langle cnt_j \rangle^A_0 = \sum_{i=1}^{n} \langle eq^i_j \rangle^A_0$. The server i sets $\langle cnt_j \rangle^A_i = \langle eq^i_j \rangle^A_1$. This fact holds because:

$$\begin{aligned} cnt_j &= \sum_{i=1}^{n} eq^i_j \\ &= \sum_{i=1}^{n} \left(\langle eq^i_j \rangle^A_0 + \langle eq^i_j \rangle^A_1 \right) \\ &= \left(\sum_{i=1}^{n} \langle eq^i_j \rangle^A_0 \right) + \langle eq^1_j \rangle^A_1 + \cdots + \langle eq^n_j \rangle^A_1 \\ &= \langle cnt_j \rangle^A_0 + \langle cnt_j \rangle^A_1 + \cdots + \langle cnt_j \rangle^A_n \end{aligned}$$

4.7 Security Analysis

We prove security in the standard semi-honest simulation-based paradigm.

Theorem 4.1. *The protocol described in Figure 4 is a correct and secure protocol against at most t corrupted parties among the senders*

Parameters: There are n servers $\mathcal{S}_1, \cdots, \mathcal{S}_n$ and a client C, with threshold k. The client and servers have agreed on hash functions $\mathcal{H}$ and h_1, h_2, h_3. $\mathcal{W}$ is the DAG base powers index negotiated in advance here $\mathcal{W} = \{w_1, w_2, \cdots\}$ and $w_1 = 1$. The number of partitions is α is and the size of the cuckoo hash table is β.
Input: Client inputs set $C \subset \{0,1\}^b$ of size n_c; Server i inputs set $S_i \subset \{0,1\}^b$ of size n_s.
Protocol:

(1) **Pre-processing**:
- **Hashing**: The client and servers hash their elements using a universal hash function $\mathcal{H}$. Denote $C' = \{\mathcal{H}(x) \mid \text{for } x \in C\}$ and $S_i' = \{\mathcal{H}(x) \mid \text{for } x \in S_i\}$. Perform the rest of the protocol with C' and S_i' replacing C and S_i, and output the original items in C and S_i as the result.
- **Hash to bins**: The client stores its elements in cuckoo hash table $\boldsymbol{T_0}$ using hash functions h_1, h_2, h_3 and the server i stores its elements in simple hash table $\boldsymbol{T_i}$ using the same hash functions h_1, h_2, h_3. Both the size of the cuckoo hash table and the simple hash table is β.
- **Prepare polynomial**: The client and servers break their elements into k_s slices and for an element we denote it as $x = x^{(1)}||x^{(2)}|| \cdot ||x^{(k_s)}$. Additionally, for each server, they need to cut the bins in the table into α partitions such the first slice of all elements in a bin has no duplicates. The table can be viewed as a 3-dimensional vector and $\boldsymbol{T_i}[j][k]$ $(j \in [\beta], k \in [\alpha])$ denotes the k-th sub-bins of the j-th bin in the table $\boldsymbol{T_i}$. Here the server constructs a polynomial through interpolation algorithm for every sub-bins in its table such that $f^i_{j,k,l}(x^{(1)}) = x^{(l)}$ for each $x \in \boldsymbol{T_i}[j][k], j \in [\beta], k \in [\alpha], l \in [k_s]$. The server stores the coefficients of the polynomials $\vec{\boldsymbol{c}}^i_{j,k,l}$ as the cache for its set.

(2) **Invoking PoL-based unbalanced PSI protocol**: The client prepares all the DAG base powers $[\![(\boldsymbol{y}^{(1)})^p]\!]$ for $p \in \mathcal{W}$ and sends to all servers. Each server first expands the based powers using DAG to get all the powers needed by $f^i_{j,k,l}(\cdot)$ and then evaluates the polynomial $[\![\boldsymbol{z}^i_{j,k,l}]\!] = f^i_{j,k,l}([\![\boldsymbol{y}^{(1)}]\!])$ using fully homomorphic encryption scheme. The last step before sending back ciphertexts is masking the result $[\![\boldsymbol{z}^i_{j,k,l}]\!]$ by a random value $r^i_{j,k,l}$ and the server holds $r^i_{j,k,l}$ as its share. Upon receiving the ciphertexts, the client decrypts the ciphertexts to compute its share $\hat{z}^i_{j,k,l} = z^i_{j,k,l} + r^i_{j,k,l} - y^{(l)}_j$.

(3) **Invoking two party computation protocol**:
- **Invoking EQ test protocol**: For each server i, the clients runs the equality test protocol $\mathcal{F}_{\mathsf{EQ}}$ by inputting bits pack $\hat{z}^i_{j,k,1}||\hat{z}^i_{j,k,2}|| \cdots ||\hat{z}^i_{j,k,l}$ and $r^i_{j,k,1}||r^i_{j,k,2}|| \cdots ||r^i_{j,k,l}$ to get the boolean share $\langle eq^i_{j,k}\rangle^B_0$ and $\langle eq^i_{j,k}\rangle^B_1$.
- **Convert boolean share to arithmetic share**: For each server i, the clients runs the boolean share to arithmetic share protocol $\mathcal{F}_{\mathsf{EQ}}$ with inputs $\langle eq^i_{j,k}\rangle^B_0$ and $\langle eq^i_{j,k}\rangle^B_1$ to get the arithmetic share $\langle eq^i_{j,k}\rangle^A_0$ and $\langle eq^i_{j,k}\rangle^A_1$.

(4) **Invoking multiparty computation protocol**:
- **Convert arithmetic share to threshold share**: The client and servers sum up all the share of equality test results for the same element to get $\langle \mathrm{cnt}^i_j\rangle^A_0 = \sum_{k=1}^{\alpha}\langle eq^i_{j,k}\rangle^A_0$. The server i computes $\langle cnt^i_j\rangle^A_1 = \sum_{k=1}^{\alpha}\langle eq^i_{j,k}\rangle^A_1$. And similarly, the client sums up all the share of different servers to get $\langle cnt_j\rangle^A_0 = \sum_{i=1}^{n}\langle cnt^i_j\rangle^A_0$. The server i sets $\langle cnt_j\rangle^A_i = \langle cnt^i_j\rangle^A_1$. The client and servers together run the resharing protocol $\mathcal{F}_{\mathsf{Reshare}}$ with inputs $\langle cnt_j\rangle^A_0$ and $\langle cnt_j\rangle^A_i$ to get the threshold share $[cnt_j]_0$ and $[cnt_j]_i$ separately.
- **Comparison with the threshold**: The client and servers run the comparison protocol $\mathcal{F}_{\mathsf{CMP}}$ with inputs $[cnt_j]_0$ and $[cnt_j]_i$ to get the threshold share $[cmp_j]_0$ and $[cmp_j]_i$.

Output: The servers send their shares to the client and the client reveals cmp_j. Compute $I = \{\ \boldsymbol{T_0}[j] \mid cmp_j = 1\}$.

Figure 4: Unbalanced Quorum PSI

and the receiver, with correctness error negligible in κ and security error negligible in λ.

Proof. For ease of analysis, we divide our protocol into two parts. The first part is composed of a PSI protocol and the two-party computation (step (1), (2) and (3) in Figure 4). The second part is the multi-party computation (step (4) in Figure 4) that occurs after the two-party computation.

Correctness. The protocol correctly computes the quorum PSI result, iff the following conditions hold:

(1) The **cuckoo hash table** procedure of the client does not abort.
(2) The **unbalanced PSI protocol** does not yield a false positive.
(3) The **comparison with the threshold** procedure does not yield a false positive.

By the analysis in the previous sections, all the above conditions hold with probability at least $1 - 2^{\kappa}$. By a union bound, the protocol is correct with probability at least $1 - 3 * 2^{\kappa}$.

Security. Most of the transcripts in one instance of our protocol are secret shares. From the security of the secret sharing scheme, these shares are indistinguishable from random as long as the size of the colluding party is no larger than t. Thus, these shares are simulatable without any auxiliary information. Hereafter, our focus is on the remaining transcripts, specifically the input from the receiver $[\![(\boldsymbol{y}^{(1)})^p]\!]$ and the input set of the sender S_i during the execution of our protocol.

We discuss the simulation in two scenarios. The difference between them is whether the receiver is a member of the corrupted parties or not. We name the simulator in these two scenarios Sim_R and Sim_S respectively. Aside from all input from corrupted senders, Sim_R has extra access to the input of the receiver and the result of the protocol. Denote the set of honest senders as $\mathcal{HS}$ and the corrupted senders $\mathcal{CS}$.

In the first scenario, the receiver is corrupted. Denote the result of the protocol as I. For each element e in the receiver's set C, the simulator Sim_R counts the occurrences cnt_e for all elements e that occur in the corrupted senders' sets $S_i, i \in \mathcal{CS}$ and C. The input of the honest senders is then simulated in the following way:

(1) For all honest sender $j \in \mathcal{HS}$, denote the set of their input as $\{S_j\}$. Initialize all these sets to be empty.
(2) For each element $e \in C, e \in I$, the simulator Sim_R select $max(0, k - cnt_e)$ sets from $\{S_j\}$ and add e to each of them. This does not cause any set to have more than n_s elements because $|I| \le n_c < n_s$
(3) Fill the rest of the sets with random elements until all these sets have n_s elements. It could be done in such way: Sample a random element and add it to a random set. When a random element e in C is sampled, if $e \notin I$ and the occurrence of e in $\{S_j\}$ is no less than $k - cnt_e - 1$, resample e. There is no risk that no such e can be sampled since the element domain $|F|$ is much larger than $|C|$.

For Sim_S, as the honest senders have no interaction (aside from secret shares) with the corrupted ones, the only thing Sim_S has to simulate is the input from the receiver to corrupted senders. By the CPA-security of homomorphic encryption schemes, Sim_S can simulate the input of the receiver by sending the encryption of a random vector to the corrupted senders.

5 IMPLEMENTATION AND EVALUATION

We implement our protocol based on Microsoft APSI library [13]. We first realize the PoL based on APSI and then convert it to circuit-PSI protocol. For the 2PC phase, we utilize the EQ test in Cheetah [21] and B2A in Cryptflow2 [37]. Additionally, we make use of the multiparty computation functionalities available in CryptFlow2. We carried out our experiments on a single machine with Intel(R) Xeon(R) Platinum 8255C CPU @2.50GHz and 256 GB RAM. For communication, we simulate the network via the localhost network as APSI [1]. The bandwidth is controlled by wondershaper [2] and and we set the bandwidth of LAN and WAN to 30 Gbps and 100 Mbps respectively.

5.1 Optimization of PoL

In the Homomorphic Encryption (HE)-based Unbalanced PSI protocol, the total communication cost is a summation of client-to-server and server-to-client communications, denoted as $Com_{c\to s}$ and $Com_{s\to c}$ respectively. To leverage homomorphic encryption, the data is always processed in chunks. A chunk can be viewed as a vector and all the operations are vectorized. Combined with SIMD, we encode the data in a plaintext and then encrypt it to get a ciphertext. The communication cost is related to the number of ciphertexts transferred and the concrete communication cost is:

$$w \times Size_{c\to s} + \alpha \times Size_{s\to c}$$

Here w is the size of DAG base powers $\mathcal{W}$ and α is the number of slices. $Size_{c\to s}$ is the size of ciphertext from client to server and $Size_{s\to c}$ is the size of ciphertext from server to client. Combined with the modulus switching [5], $Size_{s\to c}$ is several times smaller than $Size_{c\to s}$, highlighting the importance of minimizing the number of client-to-server ciphertexts to reduce overall communication costs. This is a key aspect addressed by the PoL.

Reduce the offline time. We also optimized the polynomial interpolation process of [41]. The author used the MB algorithm proposed by Moenck and Borodin [28] with $O(b \log b)$ to interpolate a polynomial of degree b. When $b < 500$, the Newton method [3] has a better performance. However, We notice that PoL needs to interpolate k_s polynomials use the same x-coordinate of the points. This process can be optimized to use batch processing. Here we use the Lagrange interpolation algorithm and NTT method to reduce the offline time. Recall the Lagrange interpolation algorithm that constructs a polynomial passing through $(x_1, y_1), \cdots, (x_n, y_n)$, the polynomial is:

$$f(x) = \sum_{i=1}^{n} y_i \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

where we denote $L_i(x) = \prod_{j=1,j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$ as the Lagrange basis polynomial and $f(x) = \sum_{i=1}^{n} y_i \cdot L_i(x)$.

The Lagrange basis polynomial is independent of the value y_i and only depends on the x-coordinate of the points. We can reuse the Lagrange basis polynomial and then interpolate the polynomial with different y_i in batch, which saves the time for duplicate computation. We combined this optimization with the NTT method and the offline time is reduced by $\mathbf{49\% \sim 73\%}$. The benchmarks are shown in Figure 5 suggesting that it works better when the set size is 16 million.

Reduce the first polynomial. We find the first polynomial of PoL is redundant. Like the permutation-based hash, we can save the storage for the first slice of an element. If we find $f_2(x^{(1)}) = y^{(2)}$ and $f_2(y^{(1)}) = y^{(2)}$ and we can say that $y^{(1)} = x^{(1)}$ with probability $1 - \frac{1}{2^\sigma}$. In our implementation, we set $k_s = 4$ and $\sigma = 20$ for PoL where a slice can be placed into a slot of SIMD. By removing the first polynomial, we effectively reduce the communication cost by $1/4$. The false positive probability is $\left(\frac{1}{2^{20}}\right)^3$ and still satisfies the statistical secure parameter.

Estimation of the number of partitions. We compute the number of slices needed by PoL in advance. We observe that the number of slices corresponds to the maximum potential duplication of the first partition of elements within the same bin, and this number is

[1]https://github.com/microsoft/APSI
[2]https://github.com/magnific0/wondershaper

n_s	n_c	Protocol	Comm. (MB)	Online time		Offline time (s)
				WAN (s)	LAN (s)	
2^{28}	11041	Quorum PSI	—	—	—	—
		Ours	174.6	28.14	10.95	1908
	5535	Quorum PSI	—	—	—	—
		Ours	114.8	20.74	10.49	2307
2^{24}	11041	Quorum PSI	7802.68	725.42	63.25	205
		Ours	131.7	30.11	16.22	967
	5535	Quorum PSI	7691.4	714.05	54.44	210
		Ours	87.8	22.72	14.69	1024
2^{20}	11041	Quorum PSI	728.7	67.29	3.50	10
		Ours	118.6	17.29	4.64	58
	5535	Quorum PSI	617.4	55.99	3.41	10
		Ours	79.9	10.75	3.90	59
2^{16}	11041	Quorum PSI	286.6	26.99	1.80	1.20
		Ours	106.86	15.78	4.20	2.7
	5535	Quorum PSI	175.3	15.76	1.29	1.44
		Ours	66.6	8.45	2.72	2.8

Table 3: The communication cost and running time of our protocol and Quorum PSI [9] in WAN and LAN settings. The statistical parameter $\kappa = 40$ and computational parameters $\lambda = 128$. Offline time refers to the running time of servers manipulating the sets before connecting with the client. Online time refers to the running time of the interacting process of the servers and client. All experiments are with a single thread except $n_s = 2^{28}$. — indicates out of memory or too long time.

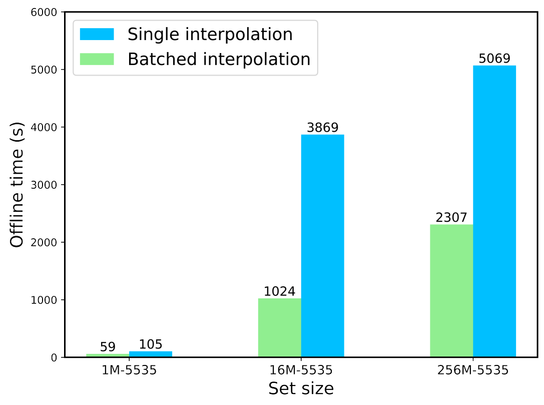

Figure 5: The offline time costs for different set sizes of batched and single interpolation.

related to the size of the bin. Suppose the number of elements in a bin is n and the bit-length of the first slice is σ. This question is equivalent to the hash-to-bins problem where we throw n balls into 2^{σ} bins and observe the distribution of the balls. We can find that the maximum duplication equals the maximum height of the balls stored in the same bin. Therefore, we can compute the number of slices in advance as the formula in Section 2.3 Equation 1 such that the elements are distributed as evenly as possible in each slice.

5.2 Comparison to Quorum PSI in [9]

We compare our protocol with [9] which focuses on balanced quorum PSI. The protocol has a theoretical communication complexity $O(nm\kappa(\lambda + \kappa \log n))$ where n is the number of servers, and m is the set size. Their communication cost is linear with the size of the set which is unacceptable in practice when the larger set grows. We analyzed their protocol and found that it can be extended to the unbalanced setting. Their protocol needs a communication cost $O(n_s + n_c)$ and the main cost is the obviously programmable pseudo-random function (OPPRF).

They use an oblivious programmable pseudorandom function (OPPRF) [24] to realize the private membership test. They give three constructions: polynomial-based OPPRF, table-based OPPRF, and relaxed-batch OPPRF, each of which offers a different trade-off in parameters. The polynomial-based method is communication efficient but costs more time and performs well in the WAN setting due to its least concrete communication cost. And the table-based method is time efficient but costs more communication. The relaxed batch-OPPRF is a trade-off between the two methods and is adopted to implement OPPRF. The concrete communication of this case is

$$(8\lambda + 4\sigma)\beta + 1.31N\sigma$$

where the N is three times the size of the server's set and β is the batch size as known as the size of the cuckoo hash table. In the unbalanced setting, the theoretical communication complexity is

$$1.27n_c(8\lambda + 4\sigma) + 3.93n_s\sigma$$

and the communication cost of 2PC and MPC circuits which is linear with the size of the client set. So the total communication cost of this protocol is $O(n_s + n_c)$.

n_s	n_c	Protocol	Offline (s)	Online								
				Comm. (MB)			LAN (s)			WAN (s)		
				Total	HE	2PC	Total	HE	2PC	Total	HE	2PC
2^{24}	11041	Construction 1	320	16.4	14.6	1.76	24	25.8	1.85	30.4	25.9	4.56
		Construction 2	62.8	62.5	61.1	1.38	7.7	5.53	2.15	16.3	12.0	4.29
		Ours	861	6.24	4.99	1.25	13.92	12.92	1.00	13.85	12.78	1.07
	5535	Construction 1	351	11.7	10.3	1.45	20.92	18.9	2.02	22.2	17.8	4.39
		Construction 2	61.1	40.3	39.6	0.69	6.53	4.75	1.78	13.6	10.1	3.49
		Ours	890	4.22	3.34	0.88	12.39	11.47	0.92	12.44	11.49	0.95
2^{20}	11041	Construction 1	4.07	10.89	9.51	1.38	5.04	3.13	1.91	8.60	4.55	4.05
		Construction 2	4.27	19.27	17.43	1.84	3.44	1.39	2.1	9.48	5.14	4.34
		Ours	53	5.37	4.12	1.25	3.51	2.51	1.00	3.64	2.57	1.07
	5535	Construction 1	5.42	8.06	6.75	1.31	4.85	3.07	1.78	8.45	4.18	4.27
		Construction 2	3.79	12.15	11.23	0.92	2.59	0.98	1.61	9.28	4.30	3.98
		Ours	51	3.69	2.81	0.88	3.13	2.21	0.92	3.01	2.06	0.95

Table 4: The comparison of our unbalanced circuit PSI and [39] in WAN and LAN settings and the best results are marked with blue. The statistical parameter $\kappa = 40$ and computational parameters $\lambda = 128$. All executions are executed with a single thread.

We give the comparison results in Table 3 where we slightly modify their implementation to support unbalanced sets and compare with our protocol in different settings of set size and network. We set the size of servers' sets from 2^{16} to 2^{28} and the size of the client's set is 5535 and 11041. The max number of client's input in a ciphertext is 5535 so we always pad the set size to the multiples of 5535. The results show that our protocol is more efficient than [9] in the case of unbalanced set size. When $n_c = 5535$ and $n_s = 2^{24}$ for 15 servers, our protocol roughly achieves a 87× reduction in communication and 31× reduction in online time. Our protocol works well when $n_s = 2^{28}$ while [9] needs more than 100 GB communication.

While our protocol demonstrates significant improvements in communication and online computation time, it does present one limitation: the requirement for can not outperform for total time substantial offline computation, predominantly due to polynomial interpolation. However, this drawback is mitigated in practical applications. The sender's offline computation is a one-time process and can be performed independently by the sender. Once this initial computation is completed, the server can store the resulting data locally. This capability to retain precomputed data significantly reduces the overall computational burden in subsequent operations. Therefore, while the offline computational cost is relatively high, it can always be completed in advance and does not affect the online computation time. We believe it is an acceptable trade-off giving the protocol's practical efficiency in real-world scenarios.

5.3 Comparison to Unbalanced Circuit PSI in [39]

Furthermore, we also compare our protocol with the recent work [39] who introduced the first unbalanced circuit PSI protocol based on FHE. Their protocol is based on unbalanced PSI [13] and GMW protocol [27].

[39] includes two different constructions, referred to as Construction 1 and Construction 2 in Table 4. Construct 1 is communication efficient while Construction 2 is computation efficient. We compare our protocol with their protocol in different settings of set size and network. The implementation of [39] is not publicized, and we take some results in the original paper. To keep consistency with the experimental environment of [39], we ran our unbalanced circuit PSI protocol on a single machine with Intel Xeon Platinum (Cascade Lake) 8269 CPU @3.50GHz and 256G RAM. We found that our communication cost is $2 \sim 10\times$ better than their protocol. And our protocol will be more efficient in the multiparty setting where the client needs to interact with multiple servers. At this time, the bottleneck of a protocol is the bandwidth of the client. The online computation cost of our protocol is less than the Construction 1 in LAN and Construction 2 in WAN. The offline computation cost of our protocol is higher than their protocol. They utilize a larger α to reduce the size of the partitions, thereby decreasing the time required for polynomial interpolation. However, this comes at the cost of increased communication. The offline computation is a one-time process and does not impact the online phase, making it an acceptable trade-off for many practical applications.

6 CONCLUSION

In this paper, we propose the first unbalanced quorum PSI protocol which aims to compute the elements that appear at least k times in n sets. The core techniques are unbalanced PSI protocol and MPC functionalities protocols. We implemented our protocol based on Microsoft APSI library and the experimental results show that our protocol is more efficient than the previous work in the unbalanced setting. Compared to previous work, we reduce the communication and computation cost greatly in WAN and LAN settings.

REFERENCES

[1] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203. https://doi.org/doi:10.1515/jmc-2015-0016

[2] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. 2021. Practical multi-party private set intersection protocols. *IEEE Transactions on Information Forensics and Security* 17 (2021), 1–15.

[3] Ake Bjorck and Victor Pereyra. 1970. Solution of Vandermonde systems of equations. *Mathematics of computation* 24, 112 (1970), 893–903.

[4] Zvika Brakerski, Craig Gentry, and Shai Halevi. 2013. Packed ciphertexts in LWE-based homomorphic encryption. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*. Springer, 1–13.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.

[6] Anrin Chakraborti, Giulia Fanti, and Michael K Reiter. 2023. {Distance-Aware} Private Set Intersection. In *32nd USENIX Security Symposium (USENIX Security 23)*. 319–336.

[7] Michael F. Challis. 1993. Two new techniques for computing extremal h-bases Ak. *Comput. J.* 36, 2 (1993), 117–126.

[8] Michael F Challis and John P Robinson. 2010. Some extremal postage stamp bases. *Journal of Integer Sequences* 13, 2 (2010), 3.

[9] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1182–1204.

[10] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1223–1237.

[11] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1243–1255.

[12] Michele Ciampi and Claudio Orlandi. 2018. Combining private set-intersection with secure two-party computation. In *International Conference on Security and Cryptography for Networks*. Springer, 464–482.

[13] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1135–1150.

[14] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2012. Fast and private computation of cardinality of set intersection and union. In *International Conference on Cryptology and Network Security*. Springer, 218–231.

[15] Whitfield Diffie and Martin E. Hellman. 1976. New Directions in Cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638

[16] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 789–800.

[17] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144 https://eprint.iacr.org/2012/144.

[18] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*. Springer, 1–19.

[19] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. 2022. Structure-aware private set intersection, with applications to fuzzy matching. In *Annual International Cryptology Conference*. Springer, 323–352.

[20] Craig Gentry, Shai Halevi, and Nigel P Smart. 2012. Homomorphic evaluation of the AES circuit. In *Annual Cryptology Conference*. Springer, 850–867.

[21] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 809–826. https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[22] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*. 1447–1464.

[23] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 818–829.

[24] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1257–1272.

[25] Yehuda Lindell and Ariel Nof. 2017. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 259–276.

[26] Catherine Meadows. 1986. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*. IEEE, 134–134.

[27] Silvio Micali, Oded Goldreich, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM New York, NY, USA, 218–229.

[28] Robert Moenck and Allan Borodin. 1972. Fast modular transforms via division. In *13th Annual Symposium on Switching and Automata Theory (swat 1972)*. IEEE, 90–96.

[29] Rasmus Pagh and Flemming Friche Rodler. 2001. Cuckoo Hashing. In *Algorithms — ESA 2001*, Friedhelm Meyer auf der Heide (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 121–133.

[30] Michael S Paterson and Larry J Stockmeyer. 1973. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.* 2, 1 (1973), 60–66.

[31] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-light: lightweight private set intersection from sparse OT extension. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 401–431.

[32] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient circuit-based PSI via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 125–157.

[33] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security (TOPS)* 21, 2 (2018), 1–35.

[34] Benny Pinkas, T. Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *ACM Transactions on Privacy and Security (TOPS)* 21 (2018), 1 – 35. https://api.semanticscholar.org/CorpusID:3848716

[35] Martin Raab and Angelika Steger. 1998. "Balls into Bins" — A Simple and Tight Analysis. In *Randomization and Approximation Techniques in Computer Science*, Michael Luby, José D. P. Rolim, and Maria Serna (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 159–170.

[36] Michael O. Rabin. 1981. How to exchange secrets with oblivious transfer. In *Technical Report*.

[37] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 325–342. https://doi.org/10.1145/3372297.3417274

[38] SEAL 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[39] Yongha Son and Jinhyuck Jeong. 2023. PSI with computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*. 342–356.

[40] Erkam Uzun, Simon P. Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. 2021. Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search. 911–928.

[41] Mingli Wu and Tsz Hon Yuen. 2023. Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing. In *32nd USENIX Security Symposium (USENIX Security 23)*. 283–300.