Predicate Private Set Intersection With Linear Complexity

Yaxi Yang[1], Jian Weng[1], Yufeng Yi[1], Changyu Dong[2], Leo Yu Zhang[3], and Jianying Zhou[4]

[1] Jinan University, Guangzhou, China
[2] Guangzhou University, Guangzhou, China
[3] Griffith University, Queensland, Australia
[4] Singapore University of Technology and Design, Singapore

Abstract. Private Set Intersection (PSI) enables two parties to learn the intersection of their input sets without exposing other items that are not within the intersection. However, real-world applications often require more complex computations than just obtaining the intersection. In this paper, we consider the setting where each item in the input set has an associated payload, and the desired output is a subset of the intersection obtained by evaluating certain conditions over the payload. We call this new primitive Predicate Private Set Intersection (PPSI) and show its applicability in many different scenarios. While a PPSI protocol can be obtained by combining existing circuit-PSI and generic circuit-based secure computation, this naive approach is not efficient. Therefore, we also provide a specially designed PPSI protocol with linear complexity and good concrete efficiency. We implemented the protocol and evaluated it with extensive experiments. The results validated the efficacy of our PPSI protocol.

Keywords: Private set intersection · Secure comparison · Secure two-party computation.

1 Introduction

Private set intersection (PSI) enables two parties $\mathcal{C}$ and $\mathcal{S}$ who have private input sets $\mathbf{x}$ and $\mathbf{y}$ to get items that they have in common (i.e., $\mathbf{x} \cap \mathbf{y}$) without revealing other information. This technique can be applied to many applications, such as matching in mobile apps [7], threat detection [30] and document search [11]. Due to its versatility, many PSI protocols have been investigated [19,7,6,5]. Recent research on PSI focuses on improving its efficiency and the currently most efficient PSI protocols are KKRT [19] and CM20 [5]. Even though PSI can solve many real-world problems, in many other scenarios, a standard PSI protocol is not sufficient because the application's need may not be simply the intersection of two sets. For example:

Advertising Campaigns. A Transaction Data Provider (TP) has a database of the ids of customers and the amount they spend on transactions, denoted as

$(id, spending)$ [22]. An Advertisement Company (AdC) only has a database of customer ids of which some have watched its advertisements. Rather than only privately computing the intersection of customer ids to analyze the advertisement conversion rates, AdC also wants to find among those click-through customers, the high-value customers who spent above a threshold. This would allow the AdC to better negotiate the commission and improve its advertisement strategy. In this application, TP's database needs to be kept private, and AdC's customers and the threshold are also private information since they relate to the company's commercial confidentiality.

Database Join. Join is a fundamental operation in databases. With the prevalence of distributed databases and vertical federated learning [24], secure join over distributed data tables becomes notable [20,26]. Specifically, two database owners each owns a table X and Y, respectively. They want to perform join operations on the primary keys of X and Y to align data in the two tables and filter the results with certain conditions. If we use X_j to denote the j-th column of the table X, an example of an SQL-style join query is given as:

$$\textbf{select } X_1 \textbf{ from } X \textbf{ inner join } Y \textbf{ on } X_1 = Y_1 \textbf{ where } Y_2 > 23.3$$

As we can see, in those two applications, we are not interested in outputting the intersection set, but rather a subset of the intersection that meets certain conditions specified as a predicate over the payload associated with the element in the intersection. In the first example, AdC wants to find the customers who have watched the advertisement and also made a purchase on TP, and the amount of purchase is higher than a pre-defined threshold. Similarly, in the second example, X_1 and Y_1 can be seen as two sets, and items in Y_2 are corresponding payloads. The intended output is the set of items in Y, whose primary keys are also in X_1 and payloads satisfy the predicate $Y_2 > 23.3$ (or other arbitrary predicates). Consequently, existing PSI/PSI with payload computation works [19,7,6,5,21,31,3,40] are not enough to meet the requirements of the above-mentioned applications.

Motivated by this, we bring up a new primitive called *Predicate Private Set Intersection* (PPSI). PPSI allows each item on an input set of one party to have an associated payload, and another party can define a predicate over the payload. Then, only those items that are in the intersection of the input sets and whose corresponding payloads meet the constraints will be revealed. A naive way to obtain a PPSI protocol is to use circuit-PSI proposed in [31,3]. In a circuit-PSI protocol, two parties will receive shares of the intersection set instead of learning the intersection in clear. Then, we can add a predicate computation on the payloads and then feed the computation results and the shares to a circuit, thus achieving the desired functionality of PPSI. However, this naive solution is not efficient. Therefore we need a new design rather than trivially applying circuit-PSI.

Our contribution in this paper can be summarized as follows.

- We present the definition of *Predicate Private Set Intersection* (PPSI) and a concrete construction of the protocol. To avoid costly circuit-based compu-

tation, we design a sub-protocol, which we call predicate masking. At a high level, for two items each from a different input set, the two parties jointly evaluate the predicate over the payload, if the result is true, the same random value will be added to the two set items; if the evaluation result is false, different random values will be added to the two set items. Hence, after that, the two parties only need to perform a plain PSI protocol (e.g., KKRT & CM20) and the intersection computed will contain only the items that meet the condition.
- We demonstrate PPSI with two exemplar applications. Given the applications, we can apply further optimizations. We use a batched secure comparison protocol (BatchComp) as a building block to achieve better efficiency for predicate computation. This sub-protocol may be of independent interest in its own right.
- We evaluate the performance of our PPSI protocol in two applications with extensive experiments. The experimental results demonstrate that PPSI is practical and scalable. We also compare our protocols with the state-of-the-art circuit-PSI protocol CGS [3], and PPSI protocol is $3.9 \sim 10\times$ faster and about $3.1\times$ more communication efficient under different network settings.

The rest of the paper is organized as follows. In Sec. 2, we introduce PSI and its various variants. In Sec. 3, we formally define PPSI and its goals in security and efficiency. Sec. 4 and Sec. 5 present the technical foundation and details of our PPSI protocol, respectively. Sec. 6 theoretically proves the security of PPSI and Sec. 7 assesses its efficiency. Sec. 8 draws the conclusion of this paper.

2 Related Works

PSI was first introduced in [25]. Since then, lots of research about PSI has been carried out [10,15,19,7,6,33,31,5,8,37,3].

PSI. The aforementioned long line of works [10,19,33,29,5] are based on oblivious transfer to achieve PSI. And those protocols outperform other generic solutions. The first work in this line is [10], which is based on OT extension and a specially designed data structure called garbled Bloom filter. It achieves linear computational/communication complexity, and requires only a few public key operations to bootstrap OT extension. In KKRT [19], the authors constructed a randomized encoding protocol based on OT extension [17]. Based on the same techniques in [19], Pinkas et al. [33] showed a detailed analysis of how different parameters affect the communication cost. Next, Pinkas et al. [29] achieved a half communication cost than that of KKRT, but roughly $6 - 7$ times computation overhead compared to KKRT. Then, in CM20 [5], the authors designed a PSI protocol that is the fastest in a network with moderate bandwidth (e.g., 30–100 Mbps) and outperforms [29] in both computation and computation cost. Nevertheless, KKRT can still be viewed as a protocol optimized for the LAN setting, where bandwidth is not a bottleneck, and it achieves better computation/communication trade-offs. And CM20 targets and achieves better in the middle range bandwidth (e.g., 30–100 Mbps) setting. Some other PSI works [7,6,8] in this category are

based on Homomorphic Encryption (HE) methods. Chen et al. [7] introduced a PSI protocol based on Fan-Vercauteren leveled fully homomorphic encryption scheme [12] that has a communication complexity logarithmic in the larger input set size. Then, the security model of their work is strengthened in a follow-up paper [6]. Those HE-based protocols focus on computing the intersection of the unbalanced input sets and reducing communication overheads.

Circuit-PSI. The other line of work is circuit-PSI [15,31,37,3]. Huang et al. [15] put forward a notion of circuit-PSI protocol based on garbled circuits, which enables the secure computation of arbitrary functions f over the intersection. Subsequently, some enhanced 2-party protocols [31,3] for circuit-PSI have been proposed. In circuit-PSI [31,3], the intersection results are secret-shared between two parties, then these two parties can take the results as inputs of circuits, which achieve the functionality of f. The inputs of those circuits cannot only be the intersection results, but also some associated payloads. Therefore, [31,3] can compute on the associated payloads of the intersection items without leaking the intersection. And [31,3] both claim $O(n)$ complexity in the semi-honest setting for PSI with computation. But the circuit-PSI proposed in [3] is state-of-the-art and is 2.3× more communication efficient and around 2.3× faster than the protocol in [31]. We can use the circuit-PSI protocol in [3] to perform the same functionality as PPSI. However, our PPSI protocol achieves much better efficiency than [3]. Besides, in [37], the author proposed a novel protocol, named Conditional Private Set Intersection (CPSI), which can enforce additional conditions for PSI. The authors utilized Trusted Execution Environments (TEEs) and HE to construct their protocol. However, their protocol is subject to the security issues of TEE [36] and the efficiency limitation of HE.

PSI with payload computation. Some other works have also considered the application of PSI with payload computation [21,31,3,40]. Those works aim to securely compute some desired functions on the payloads associated with items that are in the intersection of the two input sets, and only reveal the computation results of payloads to parties. For example, PSI-Sum, which has been considered in many works [21,31,3,40], aims to compute the sum of the payloads for the items in the intersection set and only return the sum result to parties. Therefore, in PSI with payload computation, the parties are interested in the payload computation results rather than the items in input sets, which is different from PPSI.

3 Problem Formulation

Before discussing the definition of our focused problem, we first introduce the definition of Predicate Private Set Intersection (PPSI). Next, we identify the design goal of our protocol.

3.1 Defining PPSI

We first present the definition of PPSI. Suppose one client $\mathcal{C}$ has a set $\mathbf{x} = \{x_1, ..., x_m\}$ with the constraining value α, and one server $\mathcal{S}$ provides a set $\mathbf{y} =$

$\{y_1, ...y_n\}$ with the associate payloads $\mathbf{p}_{\mathcal{S}} = \{p_{\mathcal{S},1}, ..., p_{\mathcal{S},n}\}$. Rather than merely computing the intersection set $\mathbf{x} \cap \mathbf{y}$, we wish to further confine this result subject to some constraints. Therefore, we define a predicate $\mathbf{P} : (\alpha, p_{\mathcal{S},i}) \rightarrow \{0, 1\}$, for $i \in [1, n]$. This predicate represents the constraint agreed upon by the client $\mathcal{C}$ and the server $\mathcal{S}$. And it can be replaced by any secure two-party computation protocol. After executing our PPSI protocol, $\mathcal{C}$ gets the results $\mathbf{r} = \{r_1, ...r_m\}$. If an item $r_{j,j\in[1,m]} = 1$, there exists an item $y_{i,i\in[1,n]} \in \mathbf{y}$ and $x_j = y_i$, and the associated payloads are subject to the predicate $\mathbf{P}(\alpha, p_{\mathcal{S},i}) = 1$, otherwise $r_j = 0$ and $\mathbf{P}(\cdot, \cdot) = 0$. The ideal functionality $\mathcal{F}_{\text{PPSI}}$ is shown in Figure 1.

$\underline{\mathcal{F}_{\text{PPSI}}}$

Parameters: Number of items m for a receiver, n for a sender, and ℓ for the length of each item.
Inputs: The client party $\mathcal{C}$ provides a set $\mathbf{x} = \{x_1, ..., x_m\}$ and a constraining value α for a predicate $\mathbf{P}$, the server $\mathcal{S}$ provides a set $\mathbf{y} = \{y_1, ..., y_n\}$ and the associated payload vector $\mathbf{p}_{\mathcal{S}} = \{p_{\mathcal{S},1}, ..., p_{\mathcal{S},n}\}$.

1. If there exist duplicated elements in sets $\mathbf{x}$ and $\mathbf{y}$, send abort to both $\mathcal{C}$ and $\mathcal{S}$.
2. Define a predicate $\mathbf{P}$ operating on each item of α and $\mathbf{p}_{\mathcal{S}}$, and then getting a boolean value as the operating result.
3. Define a vector $\mathbf{r} = \{r_1, ..., r_m\}$, for $j \in [1, m]$, where $r_j = 1$ if $\exists\ i \in [1, n]$, $\{x_j = y_i \in (\mathbf{x} \cap \mathbf{y})\} \wedge \{\mathbf{P}(\alpha, p_{\mathcal{S},i}) = 1\}$, else $r_j = 0$.

Outputs: The receiver outputs $\mathbf{r}$.

Fig. 1. Ideal functionality $\mathcal{F}_{\text{PPSI}}$.

3.2 Design Goal

In this work, our goal is to design an efficient predicate private set intersection protocol in the semi-honest setting, the following objectives need to be achieved.

- *Privacy preserving:* The inputs of one party in our protocol are kept private from the other party. Meanwhile, the size of the intersection set and access pattern also should be preserved and not revealed to any party.
- *Efficiency:* We also focus on reducing computation and communication costs to make this protocol as efficient as possible, as complex computation inevitably results in higher overhead. For computing set intersection with further computation, it is better to make the computation and communication overhead better than the generic circuit-PSI protocol.

4 Preliminaries

In this section, we will review the cryptographic techniques utilized in this paper, including secret sharing, cuckoo hashing, Oblivious Transfer (OT), secure comparison and the circuit-PSI protocol, and we will give the blueprint of the protocol [3].

4.1 Secret Sharing

Throughout our paper, we use 2-out-of-2 addictive secret sharing scheme over the ring $\mathbb{Z}_{2^\ell}$. $\langle x\rangle_{\mathcal{C}}$ and $\langle x\rangle_{\mathcal{S}}$ are used to denote the addictive shares that belong to party $\mathcal{C}$ and $\mathcal{S}$ respectively. And $\langle x\rangle_{\mathcal{C}}^{B}$ and $\langle x\rangle_{\mathcal{S}}^{B}$ represent boolean shares of a binary value x for $\mathcal{C}$ and $\mathcal{S}$, respectively. Then, we use Share(x) to denote the algorithm that takes x in $\mathbb{Z}_{2^\ell}$ as input and outputs the two shares $\langle x\rangle_{\mathcal{C}}$ and $\langle x\rangle_{\mathcal{S}}$ over $\mathbb{Z}_{2^\ell}$, and $\langle x\rangle_{\mathcal{C}} + \langle x\rangle_{\mathcal{S}} = x$ (where + denotes addition in $\mathbb{Z}_{2^\ell}$). In terms of security, the shares $\langle x\rangle_{\mathcal{C}}$ and $\langle x\rangle_{\mathcal{S}}$ completely hide the secret x. Someone who only has one share cannot infer any information about the secret, since the shares are totally random [9].

4.2 Hashing Techniques

Cuckoo hashing. Cuckoo hashing [28] is used in our protocol to align all items $\mathcal{C}$ and $\mathcal{S}$ owned and reduce the computation cost. In detail, cuckoo hashing uses some universal hash functions to map n items in a set $\mathbf{x} = \{x_1, ..., x_n\}$ to a cuckoo hash table $\mathbf{T}$ with b bins. Each bin only contains at most one item. In our protocol, we choose a variant of cuckoo hashing [32], which uses three hash functions h_1, h_2, h_3. To insert an item x_i into a cuckoo hash table $\mathbf{T}$, the brief procedure is as follows: 1) Check whether three bins indexed by $h_1(x_i)$, $h_2(x_i)$ and $h_3(x_i)$ have existing items or not; 2) If at least one of those bins is empty, insert x_i into the empty bin with the smallest index; 3) Otherwise, randomly select one bin in $h_1(x_i)$, $h_2(x_i)$ or $h_3(x_i)$, and evict the prior item in the selected bin and place x_i in it. 4) Recursively execute $1) - 3)$ to insert the evicted item from the previous step into the table $\mathbf{T}$. This procedure is repeated until all items in $\mathbf{x}$ are inserted and no more evictions are needed. After a certain number of iterations, if there are still some items that cannot be inserted into bins, it will cause failure and abortion to this process. Similar to [33,3], we set $b = 1.27n$ and achieve a failure probability of less than 2^{-40}. We utilize the formalization in [13]:

$$\mathbf{T_x} \leftarrow \text{Cuckoo}^{b}_{h_1,h_2,h_3}(\mathbf{x}), \tag{1}$$

where h_1, h_2, h_3: $\{0,1\}_l \rightarrow [b]$, and all items in $\mathbf{x}$ are inserted into a table $\mathbf{T_x}$ with b bins.

Simple Hashing. In our paper, we use the same hashing functions used in cuckoo hashing (i.e., h_1, h_2, and h_3) to achieve simple hashing. To map an item x to a hash table with simple hashing, this item will be put into three bins indexed with $h_1(x)$, $h_2(x)$, and $h_3(x)$. Then, for mapping m items to a hash

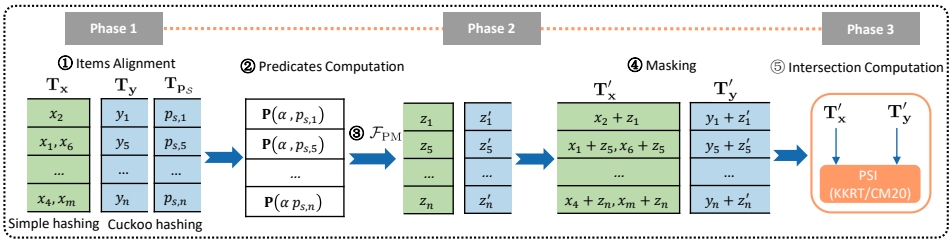

Fig. 2. A toy example of our PPSI protocol.

table with b bins, each item will be put into three bins, and each bin may have multiple items since collisions will happen. We assume the maximum number of items in one certain bin is $maxb$. As suggested in [33], $maxb$ is determined by the size of the input set m, the number of bins b and the statistical parameters via

$$P(\text{“}\exists \text{ bin with } \geq maxb \text{ items”}) \leq b \cdot [\sum_{\beta=maxb}^{m} \binom{m}{\beta} (\frac{1}{b})^{\beta} (1-\frac{1}{b})^{m-\beta}], \quad (2)$$

where P represents the possibility and it should be no greater than 2^{-40}.

4.3 Oblivious Transfer

Oblivious Transfer (OT) is an essential primitive in cryptography and can be used for constructing secure computation protocols. It is a two-party protocol between a sender and a receiver. The sender can transfer some of the potentially many pieces of messages to the receiver but remain oblivious as to which pieces have been transferred. We use different types of OT functionalities in this paper. The basic one is 1-out-of-2 OT $\binom{2}{1}$-OT_ℓ, where a sender inputs two strings (x_0, x_1) with length ℓ and a receiver inputs a single bit σ. And the receiver receives the string x_σ and learns nothing about $x_{1-\sigma}$ [17]. Then, we use $\binom{n}{1}$-OT_ℓ to denote 1-out-of-n OT, which is extended from the 1-out-of-2 OT. The sender's inputs are n strings $(x_0, ..., x_n)$ and the receiver receives the exact string x_σ [18]. And $\binom{2}{1}$-ROT_ℓ is the random OT. The sender receives two random elements x_0 and x_1 with length ℓ and the receiver outputs x_σ according to the chosen bit σ [18].

Notably, a large number of OT instances can be efficiently implemented by a few base OT instances with asymmetric key operations and symmetric key operations [17,18], denoted as IKNP-style OT. As suggested in [16], the general secure two-party computation can be upgraded by using VOLE-style OT extension, so the IKNP-style OT primitives in PPSI can be substituted by VOLE-style OT. We will discuss this in Section 7.

4.4 Secure Comparison

A secure comparison protocol takes x and y from two parties as inputs and returns the boolean shares of $\mathbf{1}\{x < y\}$ to each party. Notably, Rathee et al. [34] propose a secure comparison protocol based on a recursive problem-solving approach based on $\binom{n}{1}$-OT$_\ell$ and the functionality $\mathcal{F}_{\text{AND}}$. $\mathcal{F}_{\text{AND}}$ take boolean shares of values $x, y \in \{0, 1\}$ as inputs and returns boolean shares of the comparison result $x < y$. Then the core idea of the secure comparison protocol in CrypTFlow2 [34] is built on an observation

$$\mathbf{1}\{x < y\} = \mathbf{1}\{x^1 < y^1\} \oplus (\mathbf{1}\{x^1 = y^1\} \wedge \mathbf{1}\{x^0 < y^0\}), \tag{3}$$

where $x, y \in \mathbb{Z}_{2^\ell}$, $x = x^1||x^0$ and $y = y^1||y^0$. The basic OT protocol used in CrypTFlow2 is IKNP-style OT protocol [18]. For more details and the implementation of this protocol can refer to Appendix 1.

4.5 Technique Overview of Circuit-PSI

Circuit-PSI is a protocol that can be used as a useful implementation of PPSI. To illustrate the difference between our construction and circuit-PSI, we give a general description of the state-of-the-art circuit-PSI protocol [3] in this section.

First, a party $\mathcal{C}$ will use cuckoo hashing to map all items of $\mathcal{C}$'s input set $\mathbf{x}$ to a hash table $\mathbf{T_x}$. Then another party $\mathcal{S}$ uses a simple hashing method to hash $\mathcal{S}$'s set $\mathbf{y}$ to another hash table $\mathbf{T_y}$ with the same hash functions as used in cuckoo hashing. For security purposes, $\mathcal{C}$ will use dummy items to pad each bin to make sure each bin has the same number of items. Next, $\mathcal{C}$ and $\mathcal{S}$ need to generate random values for each item in the bins of their hash tables. Therefore, if an item is in the intersection of $\mathbf{x} \cap \mathbf{y}$, then this item will exist in the same bin of $\mathbf{T_x}$ and $\mathbf{T_y}$. Then, the authors in [3] proposed an Oblivious Programmable Pseudorandom Function (OPPRF) protocol and built a private set membership protocol based on OPPRF. If the table $\mathbf{T_x}$ and $\mathbf{T_y}$ both have b bins, for bin $\tau \in [1, b]$, $\mathcal{C}$ and $\mathcal{S}$ perform a private set membership protocol over the bins with the same indexes. $\mathcal{C}$ and $\mathcal{S}$ compare all items in $\mathbf{T_x}[\tau]$ and $\mathbf{T_y}[\tau]$ to get whether the item in $\mathbf{T_x}[\tau]$ is exists in $\mathbf{T_y}[\tau]$. After this protocol, $\mathcal{C}$ and $\mathcal{S}$ will get the secret-shared boolean results, which can be used as input for other subsequent computations (e.g., circuit-based computation, associated payloads computation).

5 The Construction of PPSI

In this section, we present our PPSI protocol. Before delving into the details, we first give a high-level overview of PPSI, which achieves the ideal functionality of PPSI. Then we introduce the technical description of PPSI and the details of our main sub-protocols. Finally, we discuss the applications of PPSI.

5.1 High-level Overview

In our PPSI protocol, first, the party $\mathcal{S}$ will use cuckoo hashing to map all items of $\mathcal{S}$'s input set $\mathbf{y}$ and payload set to a hash table. Then the other party $\mathcal{C}$ uses a simple hashing method to hash all items in $\mathbf{x}$ to another hash table with the same hash functions as used in cuckoo hashing. The process can align all items of $\mathbf{x}$ and $\mathbf{y}$. Then, $\mathcal{C}$ and $\mathcal{S}$ perform a secure two-party computation according to the predicate $\mathbf{P}$. The inputs of $\mathbf{P}$ are the constraining value α from $\mathcal{C}$ and payloads from $\mathcal{S}$. The outputs of $\mathbf{P}$ are boolean values and secret-shared to $\mathcal{C}$ and $\mathcal{S}$. If the boolean value is equal to 1, $\mathcal{C}$ gets a random value z and $\mathcal{S}$ will get z', where $z = z'$. If the boolean value is equal to 0, $\mathcal{C}$ and $\mathcal{S}$ also get z and z' respectively, while $z \neq z'$. Next, $\mathcal{C}$ and $\mathcal{S}$ can add those random values z and z' to the items in their sets. Therefore, if an item s is in the intersection set $\mathbf{x} \cap \mathbf{y}$, and the corresponding payload of the item is p and $\mathbf{P}(\alpha, p) = 1$, then $\mathcal{C}$ and $\mathcal{S}$ will add z and z' to this item in their sets to get new sets. It is obvious that $s + z = s + z'$ if $z = z'$. Finally, $\mathcal{C}$ and $\mathcal{S}$ will perform a plain PSI protocol on their new sets. Therefore, this process can be seen as a filter in PPSI, and it can filter items that are in the intersection but do not satisfy the predicate.

5.2 Technical Description

In this section, we present a description of our PPSI protocol. We assume two parties ($\mathcal{C}$ and $\mathcal{S}$) need to perform PPSI, the process of PPSI can be divided into three phases as follows.

Phase 1) Items Alignment. In this phase, two parties use hashing techniques to align their input items. Similar to circuit-PSI protocols, two parties first map their items to hash tables, which consist of multiple bins. If an input item is in the intersection result, both parties map it to the same bin. This process can reduce the computation cost of subsequent phases. To be specific, $\mathcal{S}$ uses cuckoo hashing to hash the items in $\mathbf{y}$ into a hash table $\mathbf{T_y}$, i.e., $\mathbf{T_y} \leftarrow \text{Cuckoo}^{b}_{h_1,h_2,h_3}(\mathbf{y})$. And the table $\mathbf{T_y}$ has b bins. According to the definition of cuckoo hashing, there is only one item in each bin of $\mathbf{T_y}$. Then, $\mathcal{C}$ uses simple hashing functions h_1, h_2 and h_3 to hash the items in $\mathbf{x}$ into a two-dimension hash table $\mathbf{T_x}$ with b bins. That is, an item $x_j \in \mathbf{x}$ will exist in three hash bins indexed by $h_1(x_j)$, $h_2(x_j)$ and $h_3(x_j)$. In each bin of $\mathbf{T_x}$, there will be multiple items since collisions will happen. In terms of privacy concerns, if any two or three of $h_1(x_j)$, $h_2(x_j)$ and $h_3(x_j)$ collide and are mapped to one bin, then $\mathcal{C}$ needs to map one or two dummy items any other bin to ensure that $\mathcal{C}$ maps $3m$ items in all bins. Otherwise, $\mathcal{S}$ would know that a collision has happened when $\mathcal{C}$ aligns his/her items. We assume the bin $\mathbf{T_x}[\tau]$ has $maxb_\tau$ items, where $\tau \in [1, b]$. Next, $\mathcal{S}$ also maps the associated payloads to a hash table $\mathbf{T_{p_S}}$ with the same mapping method. That is, if items $y_j \in \mathbf{y}$ appear in the bin $h_2(y_j)$, then the associated payload $p_{\mathcal{S},j}$ is also in the same bin $h_2(y_j)$ of $\mathbf{T_{p_S}}$. At the end of this phase, $\mathcal{C}$ and $\mathcal{S}$ have aligned all items they own. For an item in the intersection $\mathbf{x} \cap \mathbf{y}$, this item will be mapped into the same bins of $\mathbf{T_x}$ and $\mathbf{T_y}$. Therefore, $\mathcal{C}$ and $\mathcal{S}$ only need

to compare the items in bins with the same indexes in $\mathbf{T_x}$ and $\mathbf{T_y}$.

Phase 2) Predicate Masking. In this phase, a predicate $\mathbf{P}$ operating on the constraining value α and payloads $\mathbf{p}_{\mathcal{S}}$ is computed between $\mathcal{C}$ and $\mathcal{S}$. The predicate $\mathbf{P}$ takes α from $\mathcal{C}$ and a payload from $\mathbf{T}_{\mathbf{p}_{\mathcal{S}}}$ as inputs, and outputs boolean results for this payload, indicating whether this payload meets the constraints or not. We assume $\tau \in [1, b]$, and compute $\mathbf{P}(\alpha, \mathbf{T}_{\mathbf{p}_{\mathcal{S}}}[\tau]) \rightarrow p^*_\tau$ and $p^*_\tau \in \{0, 1\}$. And $\mathbf{P}$ could be any secure two-party computation protocol executed between $\mathcal{C}$ and $\mathcal{S}$, as long as the final results are boolean values and secret-shared to $\mathcal{C}$ and $\mathcal{S}$. We will give three applications of different secure two-party protocols in Section. 5.5.

After $\mathcal{C}$ and $\mathcal{S}$ have executed the predicate $\mathbf{P}$, it will generate a boolean value for the payload in each bin of $\mathbf{T}_{\mathbf{p}_{\mathcal{S}}}$. And this boolean value is secret-shared between $\mathcal{C}$ and $\mathcal{S}$. Then, $\mathcal{C}$ and $\mathcal{S}$ need to call a predicate masking functionality $\mathcal{F}_{\text{PM}}$ that we designed to securely compute a random value for masking the items in each bin of $\mathbf{T_x}$ and $\mathbf{T_y}$ according to the result of $\mathbf{P}$. The functionality of $\mathcal{F}_{\text{PM}}$ is formalized as Figure 3. Our $\mathcal{F}_{\text{PM}}$ takes as input boolean shares of choice bits p^*_τ, and returns two random values z_τ and z'_τ to $\mathcal{C}$ and $\mathcal{S}$. If the corresponding choice bit is equal to 1, then $z_\tau = z'_\tau$, else $z_\tau \neq z'_\tau \xleftarrow{\$} \mathbb{Z}_{2^\ell}$. This random value leaks no information about the inputs. Next, for $\tau \in [1, b]$ and $\beta \in [1, maxb_\tau]$, $\mathcal{C}$ adds z_τ to the item $\mathbf{T_x}[\tau][\beta]$, and maps this new item to the bin $\mathbf{T'_x}[\tau]$ of a new two-dimension hash table $\mathbf{T'_x}$. At the same time, $\mathcal{S}$ adds z'_τ to the item $\mathbf{T_y}[\tau]$, and maps those new items into the same bin of a new hash table $\mathbf{T'_y}$.

The core idea of this phase is that, if an item $x_j \in \mathbf{x}$ is equal to the item $y_i \in \mathbf{y}$ and the associated payload meets the constraints $p^*_i = 1$, then we add the same random value to both x_j and y_i. Otherwise, we add different random values to x_j and y_i. After that, we can filter out the items that are in the intersection but whose payloads do not meet the constraint predicate $\mathbf{P}$.

$\underline{\mathcal{F}_{\text{PM}}}$

Inputs: On inputs $\langle p^* \rangle^B_{\mathcal{C}}$ from $\mathcal{C}$, and $\langle p^* \rangle^B_{\mathcal{S}}$ from $\mathcal{S}$, where $p^* \in \{0, 1\}$.

1. Define two random values $R^0, R^1 \xleftarrow{\$} \{0, 1\}^\ell$.
2. Define z and z', where $z = R^{\langle p^* \rangle^B_{\mathcal{C}}}$. If $p^* = 1$, then $z' = z$. Otherwise, $z' = R^{1-\langle p^* \rangle^B_{\mathcal{C}}}$.

Outputs: $\mathcal{C}$ outputs z, and $\mathcal{S}$ outputs R^0, R^1, z'.

Fig. 3. Ideal functionality $\mathcal{F}_{\text{PM}}$.

Phase 3) Intersection Computation. In this phase, $\mathcal{C}$ performs a plain PSI protocol (KKRT [19] or CM20 [5]) protocol with $\mathcal{S}$. $\mathcal{C}$ and $\mathcal{S}$ take the items in $\mathbf{T}'_{\mathbf{x}}$ and $\mathbf{T}'_{\mathbf{y}}$ as inputs to PSI. And there are n items in $\mathbf{T}'_{\mathbf{y}}$ and $3m$ items in $\mathbf{T}'_{\mathbf{x}}$. It is worth noting that, we consider the scenario of $\mathcal{C}$ being a receiver. However, our PPSI protocol is flexible. $\mathcal{S}$ can also be set as a receiver, who gets the final result. And $\mathcal{C}$ plays the role of a sender. Therefore, this flexibility enables our protocol to meet more practical application requirements.

A toy example. We give an example of PPSI in Figure 2. All green tables represent the hash tables of $\mathcal{C}$, and blue tables belong to $\mathcal{S}$. First, they map all the items and payloads in $\mathbf{T}_{\mathbf{x}}$, $\mathbf{T}_{\mathbf{y}}$, $\mathbf{T}_{\mathbf{p}_{\mathcal{C}}}$ and $\mathbf{T}_{\mathbf{p}_{\mathcal{S}}}$. Second, $\mathcal{C}$ and $\mathcal{S}$ perform a predicate computation for each payload and the constraining value α. Third, they invoke the PM protocol to get pairs of random values. If $\mathbf{P}(\alpha, p_{\mathcal{S},1}) = 1$, then $z_1 = z'_1$. Next, $\mathcal{C}$ and $\mathcal{S}$ mask their items with those random values, and input the items in $\mathbf{T}_{\mathbf{x}}$ and $\mathbf{T}_{\mathbf{y}}$ to a PSI protocol. Consequently, if $x_2 = y_1$ and $\mathbf{P}(\alpha, p_{\mathcal{S},1}) = 1$, we can get $x_2 + z_1 = y_1 + z'_1$ and this value is still in the intersection set.

$\underline{\mathcal{F}_{\text{BatchComp}}}$

Parameters: Size of a vector n.
Inputs: On inputs $\alpha \geq 0$ from $\mathcal{C}$, and $\mathbf{p} = \{p_1, ..., p_n\}$ and from $\mathcal{S}$.

1. Define a vector $\mathbf{p}^* = \{p_1^*, ..., p_n^*\}$, where $p_i^* = 1$ if $p_i > \alpha$ and $p_i^* = 0$ otherwise.
2. Get the boolean shared vectors $\langle \mathbf{p}^* \rangle_{\mathcal{C}}^B$ and $\langle \mathbf{p}^* \rangle_{\mathcal{S}}^B$ from Share($\mathbf{p}^*$).

Outputs: $\mathcal{C}$ outputs $\langle \mathbf{p}^* \rangle_{\mathcal{C}}^B$, and $\mathcal{S}$ outputs $\langle \mathbf{p}^* \rangle_{\mathcal{S}}^B$.

Fig. 4. Ideal functionality $\mathcal{F}_{\text{BatchComp}}$.

5.3 The Predicate Masking Protocol

In this section, we will introduce our Predicate Masking protocol (PM). It achieves the ideal functionality $\mathcal{F}_{\text{PM}}$ as shown in Figure 3. This protocol aims to generate random values for two parties according to a secret-shared boolean value. Then, two parties can use random values output by $\mathcal{F}_{\text{PM}}$ to mask their own items for other computation. Therefore, this protocol can be used as a building block in other secure two-party computation.

Next, we give a detailed description of our protocol for realizing $\mathcal{F}_{\text{PM}}$. When $\mathcal{C}$ has a boolean value $\langle p^* \rangle_{\mathcal{C}}$, and $\mathcal{S}$ gets $\langle p^* \rangle_{\mathcal{S}}$, they invoke the functionality $\mathcal{F}_{\text{PM}}$ as shown in Algorithm 1. As we can see, the main part of this algorithm is based on a ROT protocol. At the end of this algorithm, $\mathcal{C}$ outputs z, and $\mathcal{S}$

Algorithm 1 Predicate Masking, $\mathcal{F}_{\text{PM}}$

Input: $\mathcal{C}$ holds a boolean value $\langle p^* \rangle_{\mathcal{C}}^{B}$;
$\mathcal{S}$ holds a boolean vector $\langle p^* \rangle_{\mathcal{S}}^{B}$.

Output: $\mathcal{C}$ learns z, and $\mathcal{S}$ learns z', s.t., $z = z'$ if $p^* = 1$, else $z_\tau, z'_\tau \xleftarrow{\$} \{0,1\}^\ell$.

1: $\mathcal{C}$ and $\mathcal{S}$ invoke an instance of $\binom{2}{1}$-ROT where $\mathcal{C}$ is the receiver with a choose bit $\langle p^* \rangle_{\mathcal{C}}^{B}$ and $\mathcal{S}$ is the sender. Then, $\mathcal{S}$ gets $R^0, R^1 \xleftarrow{\$} \mathbb{Z}_{2^\ell}$, and $\mathcal{C}$ receives $R^{\langle p^* \rangle_{\mathcal{C}}^{B}}$.
2: $\mathcal{C}$ sets $z = R^{\langle p^* \rangle_{\mathcal{C}}^{B}}$.
3: **if** $\langle p^* \rangle_{\mathcal{S}}^{B} = 0$ **then**
4: $\quad\mathcal{S}$ sets $z' = R^1$.
5: **else**
6: $\quad\mathcal{S}$ sets $z' = R^0$.
7: **end if**
8: $\mathcal{C}$ outputs z; $\mathcal{S}$ outputs z'.

gets z'. If $\mathcal{C}$ and $\mathcal{S}$ want to call this functionality n times to generate n pairs of different random values, it can be efficiently achieved by one round of end-to-end communication. The security of this protocol is based on the ROT protocol.

5.4 The Batched Secure Comparison Protocol

The Batched Secure Comparison protocol (BatchComp) is to realize batch secure comparisons at one time. Consider a case where $\mathcal{C}$ inputs a variable α and $\mathcal{S}$ inputs a set $\{p_1, ..., p_n\}$. And they wish to obtain the boolean shared values of the comparison results $\{\mathbf{1}\{\alpha < p_1\}, ..., \mathbf{1}\{\alpha < p_n\}\}$. We present the ideal functionality of BatchComp protocol in Figure 4. Inspired from [17,27], instead of repeating n times secure comparison protocol in CrypTFlow2 [34], we propose an efficient secure comparison protocol (BatchComp) to do those secure comparisons at once.

As defined in Figure 4, after this protocol, $\mathcal{C}$ and $\mathcal{S}$ will get a boolean share of a vector, and each item of this vector indicates whether each item of $\mathcal{S}$'s set is bigger than α. Here, we give a detailed explanation of our BatchComp protocol as Algorithm 5.2. There are four stages for our BatchComp protocol:

Splitting stage: In step 1-2, $\mathcal{C}$ and $\mathcal{S}$ splitting their input values as q parts. And we assume $q = \ell/k$. Therefore, there are 2^k possibilities for each part of input values;
Masking stage: In steps 3-8, $\mathcal{S}$ chooses two random values for all the parts of its input sets $\mathbf{p}$, and masks the random values with a bit value;
Choosing stage: In steps 9-13, $\mathcal{C}$ and $\mathcal{S}$ invoke the $\binom{n}{1}$-OT$_\ell$ extension protocol, and $\mathcal{C}$ chooses corresponding random values according to his/her input. This stage can compute the shares of the inequalities $\langle lt^t_{0,j} \rangle_{\mathcal{C},\mathcal{S}}$ and equalities $\langle eq^t_{0,j} \rangle_{\mathcal{C},\mathcal{S}}$ of the values at the leaf level;
Merging stage: Based on the same idea in Equation 3 above, recursively compute the shares of the inequalities and equalities of each node (steps 14-19). Then the values of the roots indicate the final output (step 20). Therefore, $\mathcal{C}$ and $\mathcal{S}$

Algorithm 2 Batched Comparison, $\mathcal{F}_{\text{BatchComp}}$

Input: $\mathcal{C}$ and $\mathcal{S}$ hold variable $\alpha \in \{0,1\}^\ell$ and variables $\mathbf{p} = \{p_1, ..., p_n\} \in \{0,1\}^\ell$, respectively.

Output: $\mathcal{C}$ and $\mathcal{S}$ learn $\{\langle 1\{\alpha < p_1\}\rangle_\mathcal{C}, ..., \langle 1\{\alpha < p_n\}\rangle_\mathcal{C}\}$ and $\{\langle 1\{\alpha < p_1\}\rangle_\mathcal{S}, ..., \langle 1\{\alpha < p_n\}\rangle_\mathcal{S}\}$, respectively.

$\mathcal{C}$ parses its input as $\alpha = \alpha^{q-1}||...||\alpha^0$, and for $j \in [1,n]$, $\mathcal{S}$ parses its inputs as $p_j = p_j^{q-1}||...||p_j^0$, where $\alpha^t, p_j^t \in \{0,1\}^k$, for $t \in [1, q-1]$.
$q = \ell/k$, $K = 2^k$.
for $t = \{0, ..., q-1\}$ **do**
  for $j = \{1, ..., n\}$ **do**
    $\mathcal{S}$ samples $\langle \text{lt}_{0,j}^t\rangle_\mathcal{S}, \langle \text{eq}_{0,j}^t\rangle_\mathcal{S} \xleftarrow{\$} \{0,1\}$.
    for $u = \{0, ..., K-1\}$ **do**
      $\mathcal{S}$ sets $\text{s}_{j,u}^t = \langle \text{lt}_{0,j}^t\rangle_\mathcal{S} \oplus 1\{u < y_j^t\}$.
      $\mathcal{S}$ sets $\text{v}_{j,u}^t = \langle \text{eq}_{0,j}^t\rangle_\mathcal{S} \oplus 1\{u = y_j^t\}$.
    end for
  end for
  $\mathcal{C}\&\mathcal{S}$ invoke an instance of $\binom{K}{1}$-OT where $\mathcal{S}$ is the sender with inputs $\{\text{s}_{1,u}^t||...||\text{s}_{n,u}^t\}_{u\in[0,K-1]}$, and $\mathcal{C}$ is the receiver with input α^t. Then, $\mathcal{C}$ receives $\{\text{s}_{1,\alpha^t}^t||...||\text{s}_{n,\alpha^t}^t\}$.
  $\mathcal{C}\&\mathcal{S}$ invoke an instance of $\binom{K}{1}$-OT where $\mathcal{S}$ is the sender with inputs $\{\text{v}_{1,u}^t||...||\text{v}_{n,u}^t\}_{u\in[0,K-1]}$ and $\mathcal{C}$ is the receiver with input α^t. Then, $\mathcal{C}$ receives $\{\text{v}_{1,\alpha^t}^t||...||\text{v}_{n,\alpha^t}^t\}$.
  for $j = \{1, ..., n\}$ **do**
    $\mathcal{C}$ sets $\langle \text{lt}_{0,j}^t\rangle_\mathcal{C} \leftarrow \text{s}_{j,\alpha^t}^t$.
    $\mathcal{C}$ sets $\langle \text{eq}_{0,j}^t\rangle_\mathcal{C} \leftarrow \text{v}_{j,\alpha^t}^t$.
  end for
end for
for $j = \{1, ..., n\}$ **do**
  for $i = \{1, ..., \log q\}$ **do**
    for $t = \{0, ..., (q/2^i) - 1\}$ **do**
      $\mathcal{C}$ invokes $\mathcal{F}_{\text{AND}}$ with inputs $\langle \text{lt}_{i-1,j}^{2t}\rangle_\mathcal{C}$ and $\langle \text{eq}_{i-1,j}^{2t+1}\rangle_\mathcal{C}$ to learn output $\langle \text{temp}\rangle_\mathcal{C}$.
      $\mathcal{S}$ invokes $\mathcal{F}_{\text{AND}}$ with inputs $\langle \text{lt}_{i-1,j}^{2t}\rangle_\mathcal{S}$ and $\langle \text{eq}_{i-1,j}^{2t+1}\rangle_\mathcal{S}$ to learn output $\langle \text{temp}\rangle_\mathcal{S}$.
      $\mathcal{C}$ sets $\langle \text{lt}_{i,j}^t\rangle_\mathcal{C} = \langle \text{lt}_{i-1,j}^{2t+1}\rangle_\mathcal{C} \oplus \langle \text{temp}\rangle_\mathcal{C}$.
      $\mathcal{S}$ sets $\langle \text{lt}_{i,j}^t\rangle_\mathcal{S} = \langle \text{lt}_{i-1,j}^{2t+1}\rangle_\mathcal{S} \oplus \langle \text{temp}\rangle_\mathcal{S}$.
      $\mathcal{C}$ invokes $\mathcal{F}_{\text{AND}}$ with inputs $\langle \text{eq}_{i-1,j}^{2t}\rangle_\mathcal{C}$ and $\langle \text{eq}_{i-1,j}^{2t+1}\rangle_\mathcal{C}$ to learn output $\langle \text{eq}_{i,j}^t\rangle_\mathcal{C}$.
      $\mathcal{S}$ invokes $\mathcal{F}_{\text{AND}}$ with inputs $\langle \text{eq}_{i-1,j}^{2t}\rangle_\mathcal{S}$ and $\langle \text{eq}_{i-1,j}^{2t+1}\rangle_\mathcal{S}$ to learn output $\langle \text{eq}_{i,j}^t\rangle_\mathcal{S}$.
    end for
  end for
  $\mathcal{C}$ sets $\langle p_j^*\rangle_\mathcal{C} = \langle 1\{\alpha < p_j\}\rangle_\mathcal{C} \leftarrow \langle \text{lt}_{\log q,j}^0\rangle_\mathcal{C}$, and $\mathcal{S}$ sets $\langle p_j^*\rangle_\mathcal{S} = \langle 1\{\alpha < p_j\}\rangle_\mathcal{S} \leftarrow \langle \text{lt}_{\log q,j}^0\rangle_\mathcal{S}$
end for

get the shared values of the comparison results $\mathbf{p}^* = \{1\{\alpha < p_1\}, ..., 1\{\alpha < p_n\}\}$.

The core idea behind our protocol is that we replace the $\binom{K}{1}$-OT$_\ell$ protocol in the **Choosing stage** with a batched OT protocol proposed in [27]. As we can see from this algorithm, in step 9, $\mathcal{C}$ has a choose bit α^t and $\mathcal{S}$ has a $n \times K$ matrix $\{s^t_{1,u}||...||s^t_{n,u}\}_{u\in[1,K-1]}$, then $\mathcal{C}$ and $\mathcal{S}$ need to perform the $\binom{K}{1}$-OT$_\ell$ protocol n times if we use the secure comparison protocol in CrypTFlow2 [34]. However, for the fixed choices, we only need to perform the $\binom{K}{1}$-OT$_\ell$ protocol once if we use the batch OT protocol [27]. Therefore, we can reduce half of the bandwidth requirement and make a 2.1x running time improvement than [34].

5.5 Applications of PPSI

In this section, we will give two general applications of our PPSI protocol as mentioned in Section 1. The first application is when the predicate **P** is a comparison computation, and the second is when **P** is a combination of multiple constraints.

Application 1: Advertising Campaigns. An AdC ($\mathcal{C}$), who has a database of customers id, wants to launch a query on a transaction database of a TP ($\mathcal{S}$). AdC wants to find those customers who have seen the advertisement and also purchased on TP and the purchased amount is greater than α. We modelize this problem as: $\mathcal{C}$ has a query set $\mathbf{x} = \{x_1, ..., x_m\}$, a payload value α and $\mathcal{S}$ provides a set $\mathbf{y} = \{y_1, ...y_n\}$, and for $i \in [1, n]$, each item y_i has an associate payload $\mathbf{p}_\mathcal{S} = \{p_{\mathcal{S},1}, ..., p_{\mathcal{S},n}\}$, and the predicate **P** is defined as a secure comparison computation. After executing our PPSI protocol, $\mathcal{C}$ gets the query result $\mathbf{r} = \{r_1, ...r_m\}$, where each item r_j indicates whether x_j is in the intersection set $\mathbf{x} \cap \mathbf{y}$ and the associate payload of it is bigger than the constraining value α. For $j \in [1, m]$ and $i \in [1, n]$, then we enumerate all possible cases:

$$\begin{cases} \textbf{if } \exists\ y_i:\ (x_j = y_i) \wedge (\alpha < p_{\mathcal{S},i}), \textbf{then } r_j = 1; \\ \textbf{if } \exists\ y_i:\ (x_j = y_i) \wedge (\alpha \geq p_{\mathcal{S},i}), \textbf{then } r_j = 0; \\ \textbf{if } \forall\ y_i:\ (x_j \neq y_i) \wedge (\alpha < p_{\mathcal{S},i}), \textbf{then } r_j = 0; \\ \textbf{if } \forall\ y_i:\ (x_j \neq y_i) \wedge (\alpha \geq p_{\mathcal{S},i}), \textbf{then } r_j = 0. \end{cases}$$

An Optimization. We propose an optimization method when $\mathcal{C}$ and $\mathcal{S}$ perform the PPSI protocol in this application. After $\mathcal{C}$ and $\mathcal{S}$ align all items and payloads, they need to perform a secure comparison protocol with their inputs α and $p_{\mathcal{S},i}$. We can use the state-of-the-art secure comparison protocol proposed in CrypTFlow2 [34]. Therefore, $\mathcal{C}$ and $\mathcal{S}$ invokes the secure comparison protocol of CrypTFlow2 b times. As we can see, α stays unchanged in those b times comparisons. Then, instead of running b times secure comparison protocol, we propose a Batched Secure Comparison protocol (BatchComp), which can do b times secure comparison in one time. The ideal functionality of BatchComp $\mathcal{F}_{\text{BatchComp}}$ is shown as Figure 4. And we give the details of $\mathcal{F}_{\text{BatchComp}}$ in Section 5.4.

Application 2: Database Join. In this application, we give an example of database join. To demonstrate different situations, this example is a combination of multiple predicates. For example, an SQL-styled query is as follows:

$$\textbf{select } X_1 \textbf{ from } X \textbf{ inner join } Y \textbf{ on } X_1 = Y_1 \textbf{ where } \alpha < Y_2 < \gamma$$

In this example, the predicates are defined as determining whether the payloads are in a certain range $[\alpha, \gamma]$. We can divide this constraining range into two separate predicates: the first is to compare α and Y_2, and the second is to compare γ and Y_2. Therefore, we can perform the **Phase 2** of PPSI two times to add different pairs of random values to the items of input sets. We formalize this problem as follows:

Suppose $\mathcal{C}$ has a query set $\mathbf{x} = \{x_1, ..., x_m\}$, two values α and γ, which defined a constraining range $[\alpha, \gamma]$, $\mathcal{S}$ provides a set $\mathbf{y} = \{y_1, ...y_n\}$ and an associated payload $\mathbf{p}_\mathcal{S} = \{p_{\mathcal{S},1}, ..., p_{\mathcal{S},n}\}$. $\mathcal{C}$ aims to get the result $\mathbf{r} = \{r_1, ...r_m\}$, where each item r_j indicates whether x_j is in the intersection set $\mathbf{x} \cap \mathbf{y}$ and the associate payload of it is between the constraining range $[\alpha, \gamma]$. Then $\mathcal{C}$ and $\mathcal{S}$ run PPSI. The same as depicted in Section 5, in **Phase 1**, $\mathcal{C}$ and $\mathcal{S}$ will align all items with simple hashing and cuckoo hashing to get $\mathbf{T_x}$, $\mathbf{T_y}$ and $\mathbf{T}_{\mathbf{p}_\mathcal{S}}$ (b bins), respectively. The only difference is in **Phase 2**, $\mathcal{C}$ and $\mathcal{S}$ will take secure comparison protocol as predicate $\mathbf{P}$ and perform BatchComp to securely compare the value α and the item in $\mathbf{p}_\mathcal{S}$ to get the secret-shared values of boolean results $\mathbf{p}_1^* = \{p_{1,\tau}^*\}_{\tau=1}^b$, where $p_{1,\tau}^* = \mathbf{1}\{\alpha < \mathbf{T}_{\mathbf{p}_\mathcal{S}}[\tau]\}$. Next, $\mathcal{C}$ and $\mathcal{S}$ invoke the PM protocol to get b random values $z_{1,\tau}$ and $z'_{1,\tau}$ ($\tau \in [1, b]$) respectively, and add those random values to their corresponding items in $\mathbf{T_x}[\tau]$ and $\mathbf{T_y}[\tau]$. Then, $\mathcal{C}$ and $\mathcal{S}$ will perform BatchComp again with different inputs to check whether all the payload values are less than γ. That is, $\mathcal{C}$ and $\mathcal{S}$ securely compute $p_{2,\tau}^* = \mathbf{1}\{\gamma > \mathbf{T}_{\mathbf{p}_\mathcal{S}}[\tau]\}$ to get $\mathbf{p}_2^* = \{p_{2,\tau}^*\}_{\tau=1}^b$, and then perform PM protocol with the input $\mathbf{p}^*$. After $\mathcal{C}$ gets b random values $z_{2,\tau}$ and $\mathcal{S}$ gets $z'_{2,\tau}$ ($\tau \in [1, b]$), $\mathcal{C}$ and $\mathcal{S}$ add those random values to the corresponding items in $\mathbf{T_x}[\tau]$ and $\mathbf{T_y}[\tau]$ again. At last, $\mathcal{C}$ and $\mathcal{S}$ perform PSI with inputs $\mathbf{T_x}$ and $\mathbf{T_y}$ in **Phase 3**.

As we can see from this application, the different predicates are constraints for those items in the intersection set. Therefore, we only need to perform all protocols in **Phase 2** multiple times to filter those items whose payloads cannot meet the constraints.

6 Security Analysis

In this section, we analyze the security of the proposed sub-protocols. Under the assumption that all parties are semi-honest, we use simulation-based to prove the security of PPSI. Next, we show that our protocols satisfy our design goals. Under the honest-but-curious assumption, the adversary is allowed to corrupt the client $\mathcal{C}$ or the server $\mathcal{S}$. To prove that a protocol is secure, the view of the corrupted party is simulatable by given its input and output of this protocol [23,14].

Definition 1 *A protocol is secure and can achieve the functionality $\mathcal{F}$ if there exists a probabilistic polynomial-time simulator Sim that can generate a view for the adversary Adv in the real world and the view is computationally indistinguishable from its view in the ideal world.*

To prove the security of our protocols, we then give the important lemma used in our security analysis.

Lemma 1. *For a random element $r \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ and any independent element $r' \in \mathbb{Z}_{2^\ell}$, $r \pm r'$ is uniformly random and independent from the element r'.*

The complete proof of this lemma can refer to [14,2]. Based on Definition 1 and Lemma 1, we prove that our sub-protocols can be perfectly simulated as follows:

Theorem 1. *The Predicate Masking protocol is secure against semi-honest adversaries.*

The proof of this theorem is presented in Appendix 2.

Theorem 2. *The Batched Secure Comparison protocol is secure against semi-honest adversaries.*

The proof of this theorem is shown in Appendix 2.

Privacy preserving: As we can deduce from the security proof, the inputs of $\mathcal{C}$ and $\mathcal{S}$ are kept private from each other in our protocol. Meanwhile, the size of intersection set $\mathbf{x} \cap \mathbf{y}$ are kept private from $\mathcal{C}$ and $\mathcal{S}$, since all intermediate results are secure-shared between $\mathcal{C}$ and $\mathcal{S}$. Besides, when $\mathcal{C}$ launches a query in the database of $\mathcal{S}$, all access pattern are kept private from $\mathcal{S}$ since each item in $\mathcal{S}$ are involved in the computation and $\mathcal{S}$ can not know which items that $\mathcal{C}$ gets.

Table 1. The running time in s and communication in MB of the protocol PM (IKNP-style OT based).

n		2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Time	LAN	0.003	0.012	0.026	0.092	0.29
	WAN	0.32	0.44	0.82	2.34	8.5
Comm.		0.5	1.5	5.25	20.5	81.5

Table 2. The running time in s and communication in MB of the protocol PM (VOLE-style OT based).

n		2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Time	LAN	0.001	0.004	0.001	0.044	0.157
	WAN	0.081	0.082	0.088	0.113	0.255
Comm.		0.002	0.009	0.04	0.16	0.64

7 Performance Evaluation

In this section, we first give the details about the environment of our experiments and the parameters used in our protocol. Then, we evaluate the building blocks of PPSI. Final, we evaluate the efficiency of PPSI and compare it with the state-of-art circuit-PSI protocol and other PSI-related protocols from different aspects.

n		2^{10}	2^{15}	2^{20}
Time	CrypTFlow2	0.014	0.21	6.81
	BatchComp (IKNP)	0.006	0.1	3.32
	BatchComp (VOLE)	0.004	0.058	2.24
Comm.	CrypTFlow2	0.47	15.05	481.5
	BatchComp (IKNP)	0.23	7.05	225.5
	BatchComp (VOLE)	0.04	1.12	36.57

Table 3. The running time in s and communication in MB of BatchComp and the secure comparison protocol of CrypTFlow2 in the LAN setting, and the items for comparison are 32-bit long. IKNP represents the IKNP-style OT based BatchComp protocol, and VOLE represents the VOLE-style OT based BatchComp protocol.

m vs n		2^{12} vs 2^{14}	2^{13} vs 2^{14}	2^{14} vs 2^{14}	2^{12} vs 2^{18}	2^{13} vs 2^{18}	2^{18} vs 2^{18}	2^{12} vs 2^{22}	2^{13} vs 2^{22}	2^{22} vs 2^{22}
Breakdown										
LAN	Alignment	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	0.023	0.024	0.024	1.18	1.18	1.22
	$\mathbf{P} + \mathcal{F}_{\text{PM}}$	0.007			0.087			1.39		
	PSI(KKRT)	0.037	0.046	0.062	0.2	0.2	0.69	2.98	3.03	14.1
	PSI(CM20)	0.23	0.31	0.33	0.98	0.97	4.52	13.6	97.7	54.94
WAN	Alignment	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	0.023	0.024	0.024	1.18	1.18	1.22
	$\mathbf{P} + \mathcal{F}_{\text{PM}}$	0.4			0.77			3.91		
	PSI(KKRT)	1.12	1.27	1.41	1.87	2.1	8.1	12.72	12.86	113.63
	PSI(CM20)	1.17	1.3	1.58	2.14	2.09	6.39	14.95	17.03	102.27

Table 4. The running time in s of the first application of PPSI, and the BatchComp and PM protocols are based on VOLE-style OT.

m vs n			2^{12} vs 2^{14}	2^{13} vs 2^{14}	2^{14} vs 2^{14}	2^{12} vs 2^{18}	2^{13} vs 2^{18}	2^{18} vs 2^{18}	2^{12} vs 2^{22}	2^{13} vs 2^{22}	2^{22} vs 2^{22}
Time	LAN	KKRT	0.052	0.06	0.077	0.38	0.38	0.86	6.78	6.83	17.91
		CM20	0.24	0.32	0.35	1.16	1.15	4.7	17.35	17.44	110.16
	WAN	KKRT	2.1	2.25	2.71	4.23	4.47	10.46	35.54	27.18	136.49
		CM20	2.15	2.28	2.89	4.5	4.44	8.75	27.77	31.36	125.13
Comm.		KKRT	2.69	3.97	5.21	20.88	22.91	81.31	317.87	319.15	1303.99
		CM20	2.31	3.5	4.65	16.31	17.78	73.01	237.87	239.1	1181.13

Table 5. The running time in s and communication in MB of the first application of PPSI, and the BatchComp protocol and PM protocol are based on IKNP-style OT.

7.1 Implementation Details

Our PPSI protocol is implemented in C++ and the code is available at https://www.ppsi.cn. For the cuckoo hashing method used in our protocol, we adopt the parameter used in [31,3]. When mapping n items to a hash table with b bins via three hash functions, we set $b = 1.27n$ for the stash-less setting. In addition, we test two different OT protocols to implement our protocols. One is IKNP-style OT protocol [18], and another is VOLE-style OT protocol [39], which achieves better performance in some circumstances. The statistical security parameter in our implementation is $\sigma = 40$, and the computational security parameter is $\lambda = 128$.

7.2 Experimental Environment

All the following experiments are conducted on virtual Linux machines running with AMD Ryzen 5 3600 3:60GHz CPU and 16GB of memory. All our programs

Network Setting	LAN					WAN				
$m = n$	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
CGS-PSM1	0.7	1.65	6.07	24.78	100.12	7.73	16.49	42.85	162.61	652.35
CGS-PSM2	0.95	1.68	5.22	20.29	80.65	9.27	13.55	32.97	116.02	456.59
PPSI (Ours)	0.077	0.23	0.86	4.35	17.91	2.71	3.21	10.46	31.62	136.49

Table 6. Running time in seconds of PPSI and CGS (CGS-PSM1, CGS-PSM2) [3]. We set the sizes of sets are equal and items in sets are of 64-bit length.

$m = n$	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
CGS-PSM1	24.33	99.48	397.65	1700.82	6824.49
CGS-PSM2	17.24	68.9	273.3	1155.7	4637.68
PPSI (Ours)	5.21	20.65	81.31	326.14	1303.99

Table 7. Communication in MB of PPSI and CGS.

are implemented in C++. And we ran our protocols in two network settings. The bandwidth between the two virtual Linux machines was about 10 GBps (LAN setting) and 100 Mbps (WAN setting), respectively. The round-trip time was about 0.02ms (LAN setting) and 80ms (WAN setting), respectively. The network setting is simulated based on the Cheetah framework [16,1]. Our implementation is built on top of the open-source Cheetah framework [1] provided by the authors of [16], the source code of CM20 [4], a library for private set intersection [35], and the EMP toolkit [38]. Besides, we evaluate the work [3] under the same environment as ours to show the efficiency of our protocols.

7.3 Evaluation for the Applications of PPSI

In this section, we will breakdown PPSI into individual components and give both theoretical and experimental analysis when applying our PPSI protocol in a scenario as depicted in Section 5.5. Then, we present the performance of our PM protocol and BatchComp protocol based on different network settings and OT protocols. Next, we combine those components together and evaluate the whole process.

In our first application, we assume the predicate $\mathbf{P}$ is a secure comparison protocol. We evaluate the running times of our PPSI based on different PSI protocols (KKRT [19] and CM20 [5]) in both LAN/WAN settings. we breakdown all individual components and present the execution times of the application of PPSI as shown in Table 4. Besides, we not only perform our PPSI for equal input sizes upto 2^{22} items, but also the unequal sizes. In **Phase 1**, $\mathcal{C}$ and $\mathcal{S}$ will align their items. We present the time for alignment with different input sizes. Then, in **Phase 2**, $\mathcal{C}$ and $\mathcal{S}$ perform a secure comparison in this application, which performs our BatchComp protocol. Besides, $\mathcal{C}$ and $\mathcal{S}$ use the results from BatchComp to perform the PM protocol. Because $\mathcal{C}$ and $\mathcal{S}$ need to execute b times of secure comparison and PM protocols, and b is only related to the size of

n, the running times of this phase are the same when n keeps unchanged. As we can see, if we use CM20 protocol in **Phase 3** of PPSI, it has less communication cost. In LAN setting, it can achieve better efficiency if we choose KKRT protocol as a component of PPSI.

Next, we can combine all individual components together and present the overall execution times of application 2 of PPSI as shown in Table 5. Similarly, we evaluate the running times of our PPSI based on KKRT and CM20 in both LAN/WAN settings. In the experiment of application 1, we use the VOLE-style OT to achieve BatchComp and PM protocols, which have better performance. However, we want to compare our PPSI with CGS [3], which is built based on IKNP-style OT. It would be unfair to use the VOLE-style OT to build our PPSI and compare it with CGS. Therefore, in this application, we evaluate the running time and communication cost of PPSI based on IKNP-style OT. Then, we show the comparison of PPSI and CGS in Section 7.5.

Performance of PM protocol. In PPSI, the number of times PM is run is related to the number of bins of the cuckoo hash tables b in our application. Therefore, the PM protocol is run $b = 1.27n$ times when $n = 2^{14}, 2^{16}, ..., 2^{22}$. In Table 1 and Table 2, we evaluate the performance of our PM protocol based on IKNP-style OT and VOLE-style OT protocols. As we can see, the VOLE-style OT based PM protocol has reduced communication costs and is much more efficient than the IKNP-style OT based PM protocol.

Performance of BatchComp protocol. For the BatchComp protocol, we show the results in Table 3. First, we compare BatchComp (IKNP-style OT based) with the comparison protocol in CrypTFlow2 in the LAN setting. As we can see, our protocol is about $2.1x$ better than CrypTFlow2 in both communication cost and running time. It is easy to learn that our BatchComp protocol is mainly relied on the OT protocol, therefore, except using IKNP-style OT protocol as a main building block of BatchComp, we also implement it based on VOLE-style OT protocol, which only has 1-bit communication cost for an instance of ROT protocol. And this VOLE-style OT based BatchComp protocol is about 1 order of magnitude efficient than IKNP-style OT based in terms of communication.

7.4 Theoretical Analyses

As described in Section 5.5, in **Phase 1**, $\mathcal{S}$ performs cuckoo hashing to map n items to a table with b bins and $b = 1.27n$. Then, $\mathcal{C}$ uses simple hashing to get hash tables with b bins. In **Phase 2**, $\mathcal{C}$ and $\mathcal{S}$ performing our BatchComp protocol for b times comparison instead of b instances of the secure comparison protocol in CrypTFlow2. The communication cost of b instances of $\binom{K}{1}-$OT is reduced from $b(2\lambda + K)$ bits to $(2\lambda + bK)$ bits, for $\lambda = 128$. And the communication cost for $\mathcal{F}_{\text{AND}}$ in step 14-22 in Algorithm 5.2 is $b(\lambda + 20)\lceil log\ q\rceil + b(2\lambda + 22)(q - 1 - \lceil log\ q\rceil)$ bits. Therefore, the communication for performing BatchComp in **Phase 2** is $2\lambda + bK + b(\lambda + 20)\lceil log\ q\rceil + b(2\lambda + 22)(q - 1 - \lceil log\ q\rceil)$

bits, where $q = 8$ in our application of PPSI. Then, the communication cost for performing b times of PM protocol is $b\lambda$ bits.

7.5 Performance Comparison With CGS

In Table 6 and Table 7, we compare PPSI with CGS [3], one of the state-of-art circuit-PSI protocols. CGS considers the situation that both parties have equal sizes of input sets, so we also perform experiments when $m = n$. There are two protocols proposed in CGS (CGS-PSM1 & CGS-PSM2). We use those two protocol in CGS to achieve the same functionality as shown in **Application 2**, and compare our PPSI protocol with those two protocols in CGS. And the running time and communication cost of PPSI shown in those two tables are IKNP-style OT based. Table 6 shows the running time of PPSI and CGS in the LAN and WAN settings. Under the LAN setting, our protocol is around $5-10$x faster than CGS. And for the WAN setting, the end-to-end running time of PPSI is up to 3.9x faster. And Table 7 presents the communication cost of both protocols. Our communication cost is about 3.1x than CGS. Overall, our protocol outperforms CGS in all network settings and different sizes.

Specifically, if a different predicate **P** is applied to PPSI and CGS, we only need to trivially substitute the running time of predicate computation in the final results to evaluate the performance. Therefore, the predicate computation would not affect the comparison results. As we can see, our PPSI protocol outperformance CGS in terms of running time and communication costs in different network settings.

8 Conclusion

We presented a two-party efficient PPSI protocol, which has substantial savings of computation cost and communication cost compared to circuit-PSI protocols. PPSI protocol is desirable in many real-world applications. According to different network settings, we have offered different methods to achieve the best performance of PPSI. As suggested in [16,3], we also tested PPSI based on the VOLE-style OT protocol to reduce communication costs. Based on performance analysis, we believe PPSI has a more practical use. For future work, we will try to build a PPSI protocol that is secure against malicious adversaries. One of the main challenges would be ensuring correctness when the adversaries become malicious.

Acknowledgments. This work was supported by Major Program of Guangdong Basic and Applied Research Project under Grant No. 2019B030302008, National Natural Science Foundation of China under Grant Nos. 61825203, U22B2028 and 62072132, National Key Research and Development Plan of China under Grant No. 2020YFB1005600, Guangdong Provincial Science and Technology Project under Grant No. 2021A0505030033, Science and Technology Major Project of

Tibetan Autonomous Region of China under Grant No. XZ202201ZD0006G, National Joint Engineering Research Center of Network Security Detection and Protection Technology, Guangdong Key Laboratory of Data Security and Privacy-Preserving, and Guangdong Hong Kong Joint Laboratory for Data Security and Privacy Protection. We would also thank the anonymous reviewers for their valuable comments.

Appendix 1: Secure Comparison

To compare $x \in \{0,1\}^\ell$ provided by $\mathcal{C}$ and $y \in \{0,1\}^\ell$ provided by $\mathcal{S}$, the secure comparison protocol performs the following four stages:
Splitting stage: $\mathcal{C}$ and $\mathcal{S}$ split their inputs x and y equally into q parts, and q is a power of 2. Each part has k bits, and assume k divides ℓ. Let K be a parameter and $K = 2^k$. That is $x = x^{q-1}||...||x^0$ and $y = y^{q-1}||...||y^0$, where $x^t, y^t \in \{0,1\}^k$, $t \in [0, q-1]$.
Masking stage: For each part $t \in [0, q-1]$, $\mathcal{C}$ prepares $\langle \mathrm{lt}_0^t \rangle_{\mathcal{C}}^B, \langle \mathrm{eq}_0^t \rangle_{\mathcal{C}}^B \xleftarrow{\$} \{0,1\}$. For all $u \in [0, K-1]$, $\mathcal{C}$ sets $s_u^t = \langle \mathrm{lt}_0^t \rangle_{\mathcal{C}}^B \oplus 1\{x^t < u\}$ and $v_u^t = \langle \mathrm{eq}_0^t \rangle_{\mathcal{C}}^B \oplus 1\{x^t = u\}$.
Choosing stage: $\mathcal{C}$ and $\mathcal{S}$ invoke an instance of $\binom{K}{1}$-OT_ℓ where $\mathcal{C}$ inputs $\{s_u^t\}_{u \in [0,K-1]}$ and $\mathcal{S}$ inputs the choose bit y^t. Then $\mathcal{S}$ gets the output $\langle \mathrm{lt}_0^t \rangle_{\mathcal{S}}^B$. Similarly, $\mathcal{C}$ and $\mathcal{S}$ invoke another instance of $\binom{K}{1}$-OT_ℓ where $\mathcal{C}$ inputs $\{v_u^t\}_{u \in [0,K-1]}$ and $\mathcal{S}$ inputs the choose bit y^t. Then $\mathcal{S}$ gets the output $\langle \mathrm{eq}_0^t \rangle_{\mathcal{S}}^B$.
Merging stage: $\mathcal{C}$ and $\mathcal{S}$ recursively compute the shares they have using the idea of Equation (3). For $i \in [1, \log q]$ and $t \in [1, (q/(2^i) - 1)]$, $\mathcal{C}$ and $\mathcal{S}$ invoke F_{AND} to compute $\langle \mathrm{lt}_i^t \rangle_{\mathcal{C}}^B = \langle \mathrm{lt}_{i-1}^{2t} \rangle_{\mathcal{C}}^B \wedge \langle \mathrm{eq}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B \oplus \langle \mathrm{lt}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B$ and $\langle \mathrm{lt}_i^t \rangle_{\mathcal{S}}^B = \langle \mathrm{lt}_{i-1}^{2t} \rangle_{\mathcal{S}}^B \wedge \langle \mathrm{eq}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B \oplus \langle \mathrm{lt}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B$. Then $\mathcal{C}$ and $\mathcal{S}$ can compute $\langle \mathrm{eq}_i^t \rangle_{\mathcal{C}}^B = \langle \mathrm{lt}_{i-1}^{2t} \rangle_{\mathcal{C}}^B \wedge \langle \mathrm{eq}_{i-1}^{2t+1} \rangle_{\mathcal{C}}^B$ and $\langle \mathrm{eq}_i^t \rangle_{\mathcal{S}}^B = \langle \mathrm{lt}_{i-1}^{2t} \rangle_{\mathcal{S}}^B \wedge \langle \mathrm{eq}_{i-1}^{2t+1} \rangle_{\mathcal{S}}^B$. Final, $\mathcal{C}$ gets $\langle \mathrm{lt}_i^0 \rangle_{\mathcal{C}}^B$ and $\mathcal{S}$ gets $\langle \mathrm{lt}_i^0 \rangle_{\mathcal{S}}^B$.
After the above four stages, $\mathcal{C}$ and $\mathcal{S}$ get the boolean shares of the comparison result of x and y.

Appendix 2: Security Proof

The proof of Theorem 1 is as follows.

Proof. As shown in Algorithm 1, for $\mathcal{C}$, the view during the protocol execution will be $\mathbf{view}_{\mathcal{C}} = (\langle p^* \rangle_{\mathcal{C}}^B, z)$. Since z is generated by ROT protocol and is a random value, according to Lemma 1, it is trivial to see that all values of $\mathcal{C}$'s view are uniformly random. $\mathcal{C}$ outputs nothing during this protocol. Therefore, $\mathbf{view}_{\mathcal{C}}$ and $\mathbf{output}_{\mathcal{C}}$ can be simulated by a simulator $Sim_{\mathcal{C}}$. Then $Sim_{\mathcal{C}}$ can generate a view for the adversary $Adv_{\mathcal{C}}$, who can not distinguish the generated view from its real view.

For $\mathcal{S}$, the view during the protocol execution will be $\mathbf{view}_{\mathcal{S}} = (\langle p^* \rangle_{\mathcal{S}}^B)$. Then $\mathcal{S}$ outputs R_0, R_1 during this protocol. Since R_0, R_1 is generated by ROT protocol and are random values, according to Lemma 1, all values of $\mathcal{S}$'s view are uniformly random. Therefore, $\mathbf{view}_{\mathcal{S}}$ and $\mathbf{output}_{\mathcal{S}}$ can be simulated by a simulator $Sim_{\mathcal{S}}$. Then $Sim_{\mathcal{S}}$ can generate a view for the adversary $Adv_{\mathcal{S}}$, who can not distinguish the generated view from its real view.

The proof of Theorem 2 is as follows.

Proof. As shown in Algorithm 5.2, for $\mathcal{C}$, the view in the protocol execution will be $\mathbf{view}_{\mathcal{C}} = (\alpha, \{\mathrm{s}_{1,\alpha^t}^t||...||\mathrm{s}_{n,\alpha^t}^t\}, \{\mathrm{v}_{1,\alpha^t}^t||...||\mathrm{v}_{n,\alpha^t}^t\})$. Since $\{\mathrm{s}_{1,\alpha^t}^t||...||\mathrm{s}_{n,\alpha^t}^t\}, \{\mathrm{v}_{1,\alpha^t}^t||...||\mathrm{v}_{n,\alpha^t}^t\}$ are generated from the random elements (step 7-8 in Algorithm 5.2), according

to Lemma 1, it is trivial to see that all values of $\mathcal{C}$'s view are uniformly random. The output of $\mathcal{C}$ is $\mathbf{output}_{\mathcal{C}} = \langle 1\{\alpha < p_i\}\rangle_{\mathcal{C}}$, which is generated from the random values $\{s^t_{1,\alpha^t}||...||s^t_{n,\alpha^t}\}, \{v^t_{1,\alpha^t}||...||v^t_{n,\alpha^t}\}$ (step 14-20 in Algorithm 5.2). Therefore, $\mathbf{view}_{\mathcal{C}}$ and $\mathbf{output}_{\mathcal{C}}$ can be simulated by a simulator $Sim_{\mathcal{C}}$. Then $Sim_{\mathcal{C}}$ can generate a view for the adversary $Adv_{\mathcal{C}}$, who can not distinguish the generated view from its real view.

For $\mathcal{S}$, the view in the protocol execution will be $\mathbf{view}_{\mathcal{S}} = (\mathbf{p}, \langle lt^t_{0,j}\rangle, \langle eq^t_{0,j}\rangle\})$. Since $\{\langle lt^t_{0,j}\rangle, \langle eq^t_{0,j}\rangle\}$ are generated from the random elements (step 5 in Algorithm 5.2), all values of $\mathcal{S}$'s view are uniformly random. The output of $\mathcal{S}$ is $\mathbf{output}_{\mathcal{S}} = \langle 1\{\alpha < p_i\}\rangle_{\mathcal{S}}$, which is generated from the random values $\{\langle lt^t_{0,j}\rangle, \langle eq^t_{0,j}\rangle\}$ (step 6-8 in Algorithm 5.2). Therefore, $\mathbf{view}_{\mathcal{S}}$ and $\mathbf{output}_{\mathcal{S}}$ can be simulated by a simulator $Sim_{\mathcal{S}}$. Then $Sim_{\mathcal{S}}$ can generate a view for the adversary $Adv_{\mathcal{S}}$, who can not distinguish the generated view from its real view.

References

1. Alibaba-Gemini-Lab: Opencheetah. `https://github.com/Alibaba-Gemini-Lab/OpenCheetah` (2022)
2. Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multiparty computation for data mining applications. International Journal of Information Security **11**(6), 403–418 (2012)
3. Chandran, N., Gupta, D., Shah, A.: Circuit-psi with linear complexity via relaxed batch opprf. Proceedings on Privacy Enhancing Technologies **1**, 353–372 (2022)
4. Chase, M., Miao, P.: Oprf-psi. `https://github.com/peihanmiao/OPRF-PSI` (2020)
5. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious prf. In: Annual International Cryptology Conference. pp. 34–63. Springer, Springer, Santa Barbara, CA, USA (2020)
6. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1223–1237. ACM, Los Angeles, CA, USA (2018)
7. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255. ACM, New York, NY, United States (2017)
8. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled psi from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1135–1150. ACM, New York, NY, United States (2021)
9. Demmler, D., Schneider, T., Zohner, M.: Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
10. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 789–800. ACM, Berlin, Germany (2013)
11. EdalatNejad, K., Raynal, M., Lueks, W., Troncoso, C.: Private set matching protocols. arXiv preprint arXiv:2206.07009 (2022)

12. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
13. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: IACR International Conference on Public-Key Cryptography. pp. 591–617. Springer (2021)
14. Huang, K., Liu, X., Fu, S., Guo, D., Xu, M.: A lightweight privacy-preserving cnn feature extraction framework for mobile sensing. IEEE Transactions on Dependable and Secure Computing **18**(3), 1441–1455 (2019)
15. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS. San Diego, California, USA (2012)
16. Huang, Z., Lu, W.j., Hong, C., Ding, J.: Cheetah: Lean and fast secure two-party deep neural network inference. Cryptology ePrint Archive (2022)
17. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Annual International Cryptology Conference. pp. 145–161. Springer, Santa Barbara, California, USA (2003)
18. Kolesnikov, V., Kumaresan, R.: Improved ot extension for transferring short secrets. In: Annual Cryptology Conference. pp. 54–70. Springer, Santa Barbara, CA, USA (2013)
19. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 818–829. ACM, New York, USA (2016)
20. Laur, S., Talviste, R., Willemson, J.: From oblivious aes to efficient and secure database join in the multiparty setting. In: International Conference on Applied Cryptography and Network Security. pp. 84–101. Springer (2013)
21. Le, P.H., Ranellucci, S., Gordon, S.D.: Two-party private set intersection with an untrusted third party. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2403–2420. ACM, New York, USA (2019)
22. Lepoint, T., Patel, S., Raykova, M., Seth, K., Trieu, N.: Private join and compute from pir with default. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 605–634. Springer, Singapore (2021)
23. Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography pp. 277–346 (2017)
24. Liu, Y., Zhang, X., Wang, L.: Asymmetrical vertical federated learning. arXiv preprint arXiv:2004.07427 (2020)
25. Meadows, C.: 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. pp. 134–134. IEEE (1986)
26. Mohassel, P., Rindal, P., Rosulek, M.: Fast database joins and psi for secret shared data. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1271–1287 (2020)
27. Mohassel, P., Rosulek, M., Trieu, N.: Practical privacy-preserving k-means clustering. Cryptology ePrint Archive (2019)
28. Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms **51**(2), 122–144 (2004)
29. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse ot extension. In: Annual International Cryptology Conference. pp. 401–431. Springer (2019)

30. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: fast, malicious private set intersection. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 739–767. Springer, Zagreb, Croatia (2020)
31. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based psi with linear communication. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 122–153. Springer, Darmstadt, Germany (2019)
32. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based psi via cuckoo hashing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–157. Springer, Israel (2018)
33. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) **21**(2), 1–35 (2018)
34. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342. ACM, New York, USA (2020)
35. Rindal, P.: libpsi. `https://github.com/osu-crypto/libPSI` (2020)
36. Taassori, M., Shafiee, A., Balasubramonian, R.: Vault: Reducing paging overheads in sgx with efficient integrity verification structures. In: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 665–678. ACM, New York, USA (2018)
37. Takeshita, J., Karl, R., Mohammed, A., Striegel, A., Jung, T.: Provably secure contact tracing with conditional private set intersection. In: International Conference on Security and Privacy in Communication Systems. pp. 352–373. Springer (2021)
38. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit` (2016)
39. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1607–1626. ACM, New York, USA (2020)
40. Ying, J.H., Cao, S., Poh, G.S., Xu, J., Lim, H.W.: Psi-stats: private set intersection protocols supporting secure statistical functions. In: International Conference on Applied Cryptography and Network Security. pp. 585–604. Springer, Rome, Italy (2022)