Fast Unbalanced Private Computing on (Labeled) Set Intersection with Cardinality

Binbin Tu, Xiangling Zhang, Yujie Bai, and Yu Chen(✉)

School of Cyber Science and Technology, Shandong University
{tubinbin,xianglingzhang,baiyujie}@mail.sdu.edu.cn;yuchen@sdu.edu.cn

Abstract. Private computation on (labeled) set intersection (PCSI/PCLSI) is a secure computation protocol that allows two parties to compute fine-grained functions on set intersection, including cardinality, cardinality-sum, secret shared intersection, and arbitrary functions. Recently, some computationally efficient PCSI protocols have emerged, but a limitation of these protocols is the communication complexity, which scales (super)-linear with the size of the large set. This is of particular concern when performing PCSI in the unbalanced case, where one party is a constrained device with a small set, and the other is a service provider holding a large set.

In this work, we first formalize a new ideal functionality called shared characteristic and its labeled variety called shared characteristic with labels, from which we propose the frameworks of PCSI/PCLSI protocols. By instantiating our frameworks, we obtain a series of efficient PCSI/PCLSI protocols, whose communication complexity is linear in the size of the small set, and logarithmic in the large set.

We demonstrate the practicality of our protocols with implementations. Experiment results show that our protocols outperform previous ones and the larger difference between the sizes of two sets, the better our protocols perform. For input set sizes 2^{10} and 2^{22} with items of length 128 bits, our PCSI requires only 4.62MB of communication to compute the cardinality; 4.71MB of communication to compute the cardinality-sum. Compared with the state-of-the-art PCSI proposed by Chen et al. [1], there are 58$\times$ and 77$\times$ reductions in the communication cost of computing cardinality and cardinality-sum.

Keywords: private computation on (labeled) set intersection; shared characteristic (with labels)

1 Introduction

Private Set Intersection (PSI) allows two parties, the sender and the receiver, to compute the intersection of their private sets X and Y with pre-determined sizes, such that the receiver only learns the interaction $X \cap Y$ and the sender learns nothing. Certain real-world applications are closely related to PSI but in fact, require only partial/aggregate information about the intersection to be revealed. In a notable real-world deployment, both Google [2,3] and Facebook [4] have implemented PCSI/PCLSI that allows them to compute some functions of the intersection (labels), where only the results and the intersection size are revealed, but not the intersection itself. Therefore, more fine-grained PCSI/PCLSI are required, including PCSI-card for intersection cardinality [5], PCSI-card-sum for intersection cardinality and sum [6,2,7], PCSI-secret-sharing for secret shared intersection [6,7,1], etc. However, many PCSI/PCLSI are designed for balanced inputs, resulting in a suboptimal performance for constrained hardware whose input set size is significantly smaller than the other. Specifically, the communication cost of these protocols increases at least linearly with the size of the large set. In many real-world scenarios, such as Client-Server cases, the input sets of two parties differ a lot in size, for instance, one party that can be seen as a client, is a mobile device with limited resources (e.g., battery, computing power, storage), while the other party is a high-performance server, meanwhile, the available bandwidth between two parties may be limited. However, most existing PCSI/PCLSI are not efficient in dealing with the unbalanced case.

There has been a significant amount of work on PSI [8,9,10,11,12] in recent years. The current efficient PSI [12] is almost as fast as the naive insecure hash-based protocol. In contrast to PSI, most PCSI/PCLSI realize specific functionalities, such as cardinality, cardinality and sum, etc., based on the general circuit constructions, while their efficiency is less satisfactory due to the complexity of the corresponding circuit. In the balanced setting, there are PSI [13,10] and PCSI [2,3,1] with linear complexity. In the unbalanced setting, there are PSI [14,6,15] with sublinear complexity in the size of the large set, but no such PCSI/PCLSI are known. Chen et al. [6] first present unbalanced PSI-with-computation by extending their labeled PSI to output secret shares of the intersection and combining with a secondary MPC protocol to compute cardinality and cardinality-sum functionalities. However, their protocols are described theoretically for giving circuit constructions is difficult and not accompanied by detailed experiments. Recently, a new framework of PCSI has been proposed by [1] from multi-query reverse private membership test (mqRPMT). Then, they develop mqRPMT with cardinality (mqRPMT*), from which they construct an unbalanced PCSI*, while mqRPMT* leaks the cardinality to the sender causing that PCSI* from mqRPMT* also leaks the information and cannot achieve standard security. As noted by the author [1], their unbalanced protocols cannot combine with optimizing techniques [14,6,15] causing their instantiations with low efficiency and only serving as a proof of concept. Furthermore, their protocols consider unbalanced scenarios that are not comprehensive, missing cases where the sender has the small set, and computing the receiver's labels. Therefore, how to realize efficient PCSI/PCLSI in the unbalanced setting and achieve standard security remains open questions. Motivated by the above discussions, we ask the following question:

Is it possible to design the frameworks of PCSI/PCLSI that enjoy efficient instantiations in the unbalanced setting, with communication linear in the size of the small set, and logarithmic in the large set?

1.1 Contributions

In this paper, we give an affirmative answer to the above question. Our contributions are summarized as follows:

1. We formalize a new ideal functionality called shared characteristic and its labeled variety called shared characteristic with labels, from which we present the frameworks of PCSI/PCLSI protocols in the semi-honest model by combining with some other primitives such as permuted matrix private equality test (pm-PEQT), additively homomorphic encryption (AHE) and oblivious transfer (OT).
2. We consider more comprehensive constructions of PCSI/PCLSI including two unbalanced cases, where one is that the receiver holds the small set, and the other is that the sender holds the small set. Especially, for PCLSI, each unbalanced case consists of two types of protocols: one is to compute the sender's labels and the other is to compute the receiver's labels.
3. By instantiating our frameworks based on fully homomorphic encryption (FHE), pm-PEQT, AHE, and OT, we obtain a series of efficient PCSI/PCLSI protocols whose communication complexity is linear in the size of the small set, and logarithmic in the large set. Thus, our protocols are very efficient when the set size of one party is much larger than that of the other.
4. We implement our PCSI/PCLSI and compare them with the state-of-the-art protocols. To the best of our knowledge, this is the first specialized implementation of PCSI/PCLSI for unbalanced scenarios. For set sizes ($|X| = 2^{10} \ll |Y| = 2^{22}$) with 128-bit length items, in a single thread and LAN settings: PCSI-card (PCSI-card-sum) takes 4.62 (4.71) MB of communication and 79.9 (80.03) seconds of computation. Compared with the PCSI [1], for PCSI-card (PCSI-card-sum), there are roughly 58$\times$ (77$\times$) reductions in the communication cost and 2.64$\times$ (2.76$\times$) speedup in the runtime. In particular, the performance of our PCSI/PCLSI improves significantly in the case of low bandwidth. Our PCSI-card (PCSI-card-sum) requires 121.09 (122.18) seconds in 1Mbps bandwidth, which is about 20.48$\times$ (26.98$\times$) faster than that of PCSI [1].

1.2 Technical Overview

Now, we overview our frameworks as follows. First, we introduce shared characteristic (SC) functionality and its labeled variety called shared characteristic with labels (SCwL) (depicted in Figure 1). We defer the functionalities $\mathcal{F}_{\mathrm{SC}}$ and $\mathcal{F}_{\mathrm{SCwL}}$ to Section 3. Next, we present the frameworks of PCSI/PCLSI from SC/SCwL

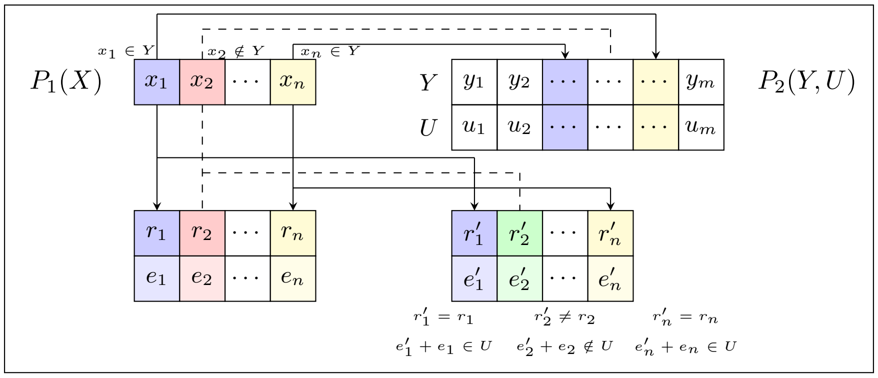
$x_1 \in Y$
$x_2 \notin Y$
$x_n \in Y$
$P_1(X)$
x_1 x_2 $\cdots$ x_n
Y y_1 y_2 $\cdots$ $\cdots$ $\cdots$ y_m
U u_1 u_2 $\cdots$ $\cdots$ $\cdots$ u_m
$P_2(Y, U)$
r_1 r_2 $\cdots$ r_n
e_1 e_2 $\cdots$ e_n
r'_1 r'_2 $\cdots$ r'_n
e'_1 e'_2 $\cdots$ e'_n
$r'_1 = r_1$
$r'_2 \neq r_2$
$r'_n = r_n$
$e'_1 + e_1 \in U$
$e'_2 + e_2 \notin U$
$e'_n + e_n \in U$

Fig. 1: Shared characteristic (with labels).

by combining with pm-PEQT, AHE, and OT (depicted in Figure 2). By instantiating our frameworks, we obtain a series of efficient PCSI/PCLSI protocols in the unbalanced setting.

Frameworks of PCSI from SC. SC implies PCSI-card by combining with pm-PEQT. This is because both parties input the shares from SC into pm-PEQT, which can check whether the corresponding shares are equal and output a permutated indication vector to the receiver $\mathcal{R}$, and then $\mathcal{R}$ reveals the cardinality by outputting the Hamming weight of the indication vector. PCSI-card-sum and PCSI-secret-sharing can be constructed by additionally coupling with AHE and OT. We give comprehensive constructions to cover all possible scenarios in the unbalanced setting, including both cases where the sender $\mathcal{S}$ owns the larger set or the receiver $\mathcal{R}$ owns the larger set. We defer the details to Section 4.
Frameworks of PCLSI from SCwL. SCwL is an extension of SC, outputting not only the same shares for intersection items but also additive shares of the labels of the intersection items. Therefore, following the frameworks of PCSI from SC, PCLSI protocols including card-sum, secret-sharing, and card-inner-product, can be constructed based on SCwL, pm-PEQT, AHE, and OT by computing the shares of labels instead of intersection items. We propose two types of PCLSI protocols including computing $\mathcal{S}'$s ($\mathcal{R}'$s) labels corresponding to the intersection items, and each type consists of the two cases that the sender holds the larger set or the receiver holds a larger set. We defer the details to Section 6.

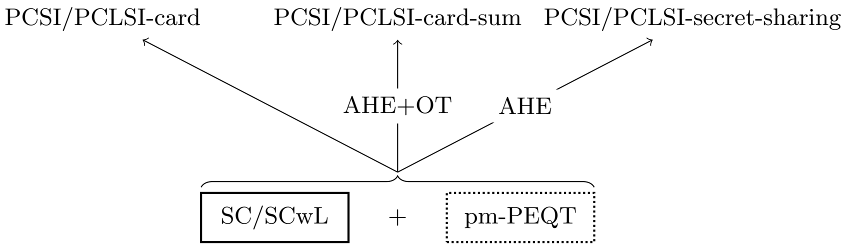

Fig. 2: Technical overview of our frameworks. The rectangles with solid lines denote new notions. The rectangle with dotted lines denotes the previous notion.

Instantiations. Following the extension of (labeled) PSI [14,6], we informally show basic constructions of SC based on the FHE as follows. The sender $\mathcal{S}$ holding a small set $X = \{x_i\}_{i\in[n]}$ interacts the receiver $\mathcal{R}$ holding a large set $Y = \{y_j\}_{j\in[m]}$. $\mathcal{S}$ uses its public key to encrypt the item x_i and sends the ciphertext $c = \text{FHE}.\mathsf{Enc}(x_i)$ to $\mathcal{R}$; $\mathcal{R}$ chooses random non-zero value r_i, and homomorphically computes $c' = \text{FHE}.\mathsf{Enc}(r_i + f(x_i))$, where the polynomial $f(x) = \Pi_{y_j \in Y}(x - y_j)$ and returns the new ciphertext to $\mathcal{S}$; $\mathcal{S}$ decrypts c' and gets the plaintext $r'_i = r_i + f(x)$. Therefore, if $x_i \in Y, i \in [n]$ both parties get $r'_i = r_i$, $r'_i \neq r_i$ otherwise. Similar to [14], the basic protocol achieves communication linear in the small set and is independent of the large set, but the deep depth of the homomorphic circuit leads to high computational costs since the degree of $f(x)$ is related to the large set size. SCwL can be constructed following labeled PSI [6], and we defer the detail in Figure 9.

1.3 Related Works

Here, we review PCSI frameworks in the balanced/unbalanced setting as follows. Table 1 provides a comparison of our protocols with the state-of-the-art protocols in the unbalanced setting. n and m denote the size of the small set and the large set, respectively.

Protocols	Communication	Computation	No generic 2PC
PCSI/PCLSI [7]	$O(m + n \log n)$	$O(m + n \log n)$	✓
PCSI/PCLSI [1]	$O(m + n)$	$O(m + n)$	✓
Circuit-PCSI/PCLSI [6]	$O(n \log m)$	$O(m + n \log m)$	×
Our PCSI/PCLSI	$O(n \log m)$	$O(m + n \log m)$	✓

Table 1: Comparisons of PCSI/PCLSI in the semi-honest setting.

Balanced case. A functionality for computing arbitrary functions of intersection can be implemented using *generic 2PC* protocols by expressing the functionality as a circuit. Huang et al. [16] show a sort-compare-shuffle circuit for using either GMW or Yao's protocol. Furthermore, Pinkas et al. [17,18,19] develop circuit-based PCSI by using a special-purpose preprocessing phase before performing general-purpose 2PC and achieve linear communication complexity.

Garimella et al. [7] propose the permuted characteristic functionality $\mathcal{F}_{\text{PC}}$ based on oblivious switching network (OSN), from which they give a PCSI framework including PCSI-card, PCSI-card-sum and PCSI-secret-sharing. However, the core construction of $\mathcal{F}_{\text{PC}}$ requires an OSN subprotocol [20], all protocols lead to super-linear communication. After that, a new framework to perform PCSI from multi-query reverse private membership test (mqRPMT) is given by [1], and they present two constructions of mqRPMT from commutative weak pseudorandom function and permuted oblivious pseudorandom function, respectively. Both constructions can be realized from DDH-like assumptions and achieve linear communication and computation complexity with the size of the large set.

Unbalanced case. Chen et al. [6] first propose an efficient labeled PSI whose communication is sublinear in the size of the large set. They show how to extend this construction to enable each party to obtain secret shared labels associated with intersection items. After that, they show PSI-with-computation in the unbalanced setting by feeding shares into a downstream *generic 2PC* to compute any functions over these labels. However, since it is difficult to design a circuit for computing PCSI-card/card-sum, their extended protocol is described only theoretically and not accompanied by detailed experiments.

2 Preliminaries

Notation. We denote two parties in our protocols as sender ($\mathcal{S}$) and receiver ($\mathcal{R}$), and their respective input sets as X and Y with sizes n and m. $X = \{(x_i, v_i)\}_{i \in [n]}$ denotes a labeled set with size n where v_i is a label of the item x_i. We write ($\mathcal{S} \gg \mathcal{R}$) to denote that $\mathcal{S}$ holds the larger set and ($\mathcal{S} \ll \mathcal{R}$) to denote that $\mathcal{R}$ holds the larger set. We use κ and λ to indicate the computational and statistical security parameters, respectively. $[n]$ denotes the set $\{1, 2, \cdots, n\}$. We denote vectors by lower-case bold letters, e.g., $\mathbf{s}$, and matrices by upper-case bold letters, e.g., $\mathbf{S}$. We write $\mathbf{s} = [s_i]_{i \in [n]}$ to denote a vector $[s_1, \cdots, s_n]$. For a permutation π over n items, we write $\{s_{\pi(1)}, \cdots, s_{\pi(n)}\}$ to denote $\pi(\{s_1, \cdots, s_n\})$, where $s_{\pi(i)}$ indicates the i-th element after the permutation. We use $m \boxplus c$ to denote the "addition" of the plaintext in c with the plaintext m and $a \boxtimes c$ to denote the "multiplication" of the plaintext in c by scalar a.

2.1 PCSI/PCLSI Functionalities

We review the functionalities of private computing on (labeled) set intersections (PCSI/PCLSI) [7,1] in Figure 3 and 4.

Functionality $\mathcal{F}_{\text{PCSI}}$: On input $X = \{x_i\}_{i\in[n]}$ from $\mathcal{S}$ and $Y = \{y_j\}_{j\in[m]}$ from $\mathcal{R}$, where each $x_i, y_j \in \mathbb{Z}_p$ for some integer modular p:

- **card**: Give output k to $\mathcal{R}$, where $k = |X \cap Y|$.
- **card-sum**: Give output (k, s) to $\mathcal{R}$, where $k = |X \cap Y|, s = \sum_{x_i \in Y} x_i$.
- **secret-shares**[1]: Give $\mathbf{a} = [a_i]_{i\in[k]}$ to $\mathcal{S}$ and $\mathbf{b} = [b_i]_{i\in[k]}$ to $\mathcal{R}$, where $z_i = a_i + b_i$, $i \in [k]$, $[z_i]_{i\in[k]}$ is a random permutation of $X \cap Y$.
- **card-function**: Give output (k, z) to $\mathcal{R}$, where $k = |X \cap Y|, z = g(X \cap Y)$, g can be any function.

Fig. 3: Ideal functionality for private computing on set intersection

Functionality $\mathcal{F}_{\text{PCLSI}}$: On input labeled sets $X = \{(x_i, v_i)\}_{i\in[n]}$ from $\mathcal{S}$ and $Y = \{(y_j, u_j)\}_{j\in[m]}$ from $\mathcal{R}$, where $x_i, y_j \in \{0,1\}^*$, $v_i, u_j \in \mathbb{Z}_p$ for some integer modular p:

- **card-sum with $\mathcal{S}'$s ($\mathcal{R}'$s) labels**: Give output (k, s) to $\mathcal{R}$, where $k = |X \cap Y|, s = \sum_{x_i \in Y} v_i$ $(s = \sum_{y_j \in X} u_j)$.
- **secret-shares with $\mathcal{S}'$s ($\mathcal{R}'$s) labels**: Give additive secret shares $\mathbf{a} = [a_i]_{i\in[k]}$ to $\mathcal{S}$ and $\mathbf{b} = [b_i]_{i\in[k]}$ to $\mathcal{R}$, where $z_i = a_i + b_i, i \in [k]$, $[z_i]_{i\in[k]}$ is a random permutation of $\mathcal{S}'$s ($\mathcal{R}'$s) labels corresponding to $X \cap Y$.
- **card-function with $\mathcal{S}'$s ($\mathcal{R}'$s) labels**: Give output (k, z) to $\mathcal{R}$, where $k = |X \cap Y|, z = g(z_1, \cdots, z_k)$, where g can be any function, $[z_i]_{i\in[k]}$ is a random permutation of $\mathcal{S}'$s ($\mathcal{R}'$s) labels corresponding to $X \cap Y$.
- **card-inner-product**: Give output (k, d) to $\mathcal{R}$, where $k = |X \cap Y|, d = \sum_{x_i = y_j, i\in[n], j\in[m]} v_i \cdot u_j$.

Fig. 4: Ideal functionality for private computing on labeled set intersection

2.2 Labeled PSI

We now recall the labeled PSI [6], which is following the architecture of [17,14]. Specifically, $\mathcal{R}$ holding a small set $Y = \{y_i\}_{i\in[m]}$ interacts with $\mathcal{S}$ holding a large labeled set $X = \{(x_i, v_i)\}_{i\in[n]}$ and outputs the labels of intersection.

First, $\mathcal{R}$ inserts Y[2] into a hash table Y_c by Cuckoo hashing and each bin $Y_c[i]$ consists of at most *one* item. $\mathcal{S}$ inserts X into a hash table X_b by simple hashing, where the i-th bin is indicated as $X_b[i]$ and each

[1] Both parties obtain the intersection cardinality by calculating the number of shares in secret-sharing functionality, so the "card" symbol is omitted.

[2] In [6], both parties first run mp-OPRF functionality to compute all PRF values of two sets X and Y, and all subsequent steps operate on the PRF values, which we omit for convenience.

bin consists of at most B items[3]. That is, $X \cap Y = \bigcup_i (Y_c[i] \cap X_b[i]) = \bigcup_i (\{y_i\} \cap X_b[i])$, where y_i is the sole item in the bin $Y_c[i]$. Then, both parties encode their respective bins into the plaintext field $\mathbb{F}$, and compute the label of $\{y_i\} \cap X_b[i]$ by using fully homomorphic encryption (FHE): $\mathcal{R}$ sends an encryption of y_i, denoted as $[\![y_i]\!]$, to $\mathcal{S}$; $\mathcal{S}$ computes two polynomials f_i, g_i such that for all $x_j \in X_b[i]$, $f_i(x_j) = 0$ and if $\exists\ x_j = y_i$, $g_i(y_i) = v_j$ where v_j is a label corresponding to the item x_j, otherwise, $g_i(y_i)$ is a random element in $\mathbb{F}$; $\mathcal{S}$ chooses two random values r_i, r'_i and homomorphically computes $[\![z_i]\!] = (r_i f_i([\![y_i]\!]))$ and $[\![z'_i]\!] = (r'_i f_i([\![y_i]\!]) + g_i([\![y_i]\!]))$. Therefore, if $y_i \in X_b[i]$, $[\![z_i]\!]$ is an encryption of 0 and $[\![z'_i]\!]$ is an encryption of v_j, otherwise, $[\![z_i]\!]$ and $[\![z'_i]\!]$ are encryptions of two random values. Finally, $[\![z_i]\!]$ is returned to $\mathcal{R}$, who concludes that if $z_i = 0$, $y_i \in X$ and obtains the label $z'_i = v_j$, otherwise, $y_i \notin X$ and gets a random value z'_i.

After that, Chen et al. [6] also consider PSI-with-computation by extending labeled PSI to output shares[4] of intersection and additive secret shares of corresponding labels, which can be forwarded as input to a secondary MPC protocol. For simplicity, instead of returning $[\![z_i]\!]$ and $[\![z'_i]\!]$, $\mathcal{S}$ returns ciphertexts $[\![z_i + e_i]\!]$ and $[\![z'_i + e'_i]\!]$, where e_i and e'_i are two random values. When $\mathcal{R}$ decrypts the ciphertexts, it will hold shares of z_i and z'_i. That is, if two shares of z_i are equal, both parties hold the shares of $z'_i = v_j$ which is a label of x_j. Otherwise, both parties hold the shares of a random value.

2.3 Building Blocks

We briefly review the main technical tools as follows.

Simple hashing. In the simple hashing [21], the hash table consists of m bins $B_1, \cdots, B_m$. Hashing is done by mapping each element x to a bin $B_{h(x)}$ using a hash function $h : \{0,1\}^* \to [m]$ that was chosen uniformly at random and independently of the input elements. According to the following inequality [22], the maximum bin size B can be set to ensure that no bin will contain more than B items except with probability $2^{-\lambda}$ when hashing n items into m bins.

$$\Pr[\exists \text{ bin size} \geq B] \leq m \left[\sum_{i=B}^{n} \binom{n}{i} \cdot \left(\frac{1}{m}\right)^i \cdot \left(1 - \frac{1}{m}\right)^{n-i} \right]$$

Cuckoo hashing. Cuckoo hashing can be used to build dense hash tables by many hash functions [23,24,25,26]. Following [14,6], we use three hash functions and adjust the number of items and table size to reduce the stash size to 0 while achieving a hashing failure probability of $2^{-\lambda}$. For a detailed explanation, we refer the reader to [14].

Leveled fully homomorphic encryption. The leveled fully homomorphic encryption supports circuits of a certain bounded depth. Following [14], our protocols require that the leveled FHE satisfies IND-CPA secure with circuit privacy [27]. We use an array of optimization techniques of FHE as [14,6,15], such as batching, windowing, partitioning, and modulus switching to significantly reduce the depth of the homomorphic circuit. For the implementation, we use the homomorphic encryption library SEAL which implements the BFV scheme [28] following [14,6,15].

Oblivious transfer. Oblivious transfer (OT) [29] is a two-party protocol between $\mathcal{S}$ and $\mathcal{R}$. $\mathcal{S}$ with two input strings (x_0, x_1) interacts with $\mathcal{R}$ holding a choice bit b. The result is that $\mathcal{R}$ learns x_b, while $\mathcal{S}$ learns nothing about b. Ishai et al. [30] introduce an OT extension that allows for many OT executions at the cost of computing a small number of public-key operations. We recall 1-out-of-2 OT functionality $\mathcal{F}_{\mathrm{OT}}$ in Figure 5.

Permuted matrix private equality test. The permuted matrix private equality test (pm-PEQT) [31] is an extension of private equality test with permutation. The pm-PEQT can be used in vectors by removing the operations of shuffling rows, in which $\mathcal{S}$ holding a vector $\mathbf{r} = [r_i]_{i\in[n]}$ and a permutation π over $[n]$ interacts with $\mathcal{R}$ holding a vector $\mathbf{r}' = [r'_i]_{i\in[n]}$. As a result, $\mathcal{R}$ learns $\mathbf{b} = [b_i]_{i\in[n]}$ which is an indication bit vector for shuffled vectors $\pi(\mathbf{r})$ and $\pi(\mathbf{r}')$, where if $r'_{\pi(i)} = r_{\pi(i)}$, $b_i = 1$, else $b_i = 0$. We review the ideal functionality of pm-PEQT with vectors in Figure 6.

[3] In the PSI [14,6], they use cuckoo hashing with no stash and three simple hash functions. For same i-bin in the hash tables Y_c and X_b, if an item $y_i \in Y_c[i] \cap X_b[i]$, it belongs to the intersection.

[4] This is a special sharing because two shares are equal for intersection item and not equal for non-intersection item.

Parameters: Two parties: $\mathcal{S}$ and $\mathcal{R}$. Message length κ.
Functionality $\mathcal{F}_{\text{OT}}$:

- Wait for input $\{x_0, x_1\}$ from $\mathcal{S}$. Wait for input $b \in \{0,1\}$ from $\mathcal{R}$.
- Give x_b to $\mathcal{R}$.

Fig. 5: 1-out-of-2 oblivious transfer functionality

Parameters: Two parties: $\mathcal{S}$ with a vector $\mathbf{r} = [r_i]_{i\in[n]}$ and a permutation π over $[n]$; $\mathcal{R}$ with a vector $\mathbf{r}' = [r'_i]_{i\in[n]}$.
Functionality $\mathcal{F}_{\text{pm-PEQT}}$:

- Wait for an input $\mathbf{r} = [r_i]_{i\in[n]}$ and a permutation π from $\mathcal{S}$, and an input $\mathbf{r}' = [r'_i]_{i\in[n]}$ from $\mathcal{R}$.
- Give a bit vector $\mathbf{b} = [b_i]_{i\in[n]}$ to $\mathcal{R}$, where for all $i \in [n]$, if $r_{\pi(i)} = r'_{\pi(i)}$, $b_i = 1$, else, $b_i = 0$.

Fig. 6: Permuted matrix private equality test

3 Shared Characteristic (with Labels)

In this section, we formalize the ideal functionalities named shared characteristic (with labels) (depicted in Figure 1), show efficient constructions following the extension of (labeled) PSI [14,6], and give the security proofs.

Shared characteristic. Here, we define the shared characteristic (SC) functionality $\mathcal{F}_{\text{SC}}$ in Figure 7. Roughly speaking, a party P_1 holding a set $X = \{x_i\}_{i\in[n]}$ interacts with a party P_2 holding a set $Y = \{y_j\}_{j\in[m]}$, and the result is that for each $x_i \in X, i \in [n]$ if $x_i \in Y$, P_1 and P_2 learn the same shares $r_i = r'_i$, otherwise, they learn random values $r_i \neq r'_i$.

Functionality $\mathcal{F}_{\text{SC}}$: On input $X = \{x_i\}_{i\in[n]}$ from P_1; $Y = \{y_j\}_{j\in[m]}$ from P_2:

- Give a random vector $\mathbf{r} = [r_i]_{i\in[n]}$ to P_1 and give a random vector $\mathbf{r}' = [r'_i]_{i\in[n]}$ to P_2, where if $x_i \in Y$, $i \in [n]$, $r_i = r'_i$, else $r_i \neq r'_i$.

Fig. 7: Shared characteristic

Shared characteristic with labels. Considering the labeled case, we extend shared characteristic into shared characteristic with labels (SCwL). The ideal functionality $\mathcal{F}_{\text{SCwL}}$ is defined in Figure 8, where a party P_1 with a labeled set $X = \{x_i\}_{i\in[n]}$ interacts with a party P_2 holding a set $Y = \{(y_j, u_j)\}_{j\in[m]}$, and the result is that for each $x_i \in X$, $i \in [n]$, if $\exists\ j \in [m]$ s.t. $y_j = x_i$, P_1 and P_2 learn secret shares (r_i, e_i) and (r'_i, e'_i), where $r_i = r'_i$ and $e_i + e'_i = u_j$[5], otherwise, P_1 and P_2 learn random values (r_i, e_i) and (r'_i, e'_i), where $r_i \neq r'_i$ and $e_i + e'_i \neq u_j$.
Constructions of SC and SCwL. Same to the extension of (labeled) PSI [14,6], we show the constructions of SC/SCwL in Figure 9. Note that we assume that P_1 holds the small set, and if P_2 holds the small set, both parties perform the protocols with their roles switched. Additionally, the efficiency of SC/SCwL protocols in

[5] The addition operation "+" refers to modular addition, and we omit modular operation in this paper for convenience.

Functionality $\mathcal{F}_{\text{SCwL}}$: On input $X = \{x_i\}_{i\in[n]}$ from P_1; $Y = \{(y_j, u_j)\}_{j\in[m]}$ from P_2:

– Give two random vectors $\mathbf{r} = [r_i]_{i\in[n]}$ and $\mathbf{e} = [e_i]_{i\in[n]}$ to P_1 and give two random vectors $\mathbf{r}' = [r'_i]_{i\in[n]}$ and $\mathbf{e}' = [e'_i]_{i\in[n]}$ to P_2, where for all $i \in [n]$, if $\exists\ j \in [m]$ s.t. $y_j = x_i$, there is $r_i = r'_i$ and $e_i + e'_i = u_j$, otherwise, $r_i \neq r'_i$ and $e_i + e'_i \neq u_j$.

Fig. 8: Shared characteristic with labels

Figure 9 can be improved by optimizing techniques used in [14,6,15], such as batching, windowing, partitioning, modulus switching, etc. For a detailed explanation, we refer the reader to [14,6,15].
Communication. The windowing technique results in a significant reduction in the homomorphic circuit depth, at the cost of increasing the communication in step 5 from $O(n)$ to $O(n \log m)$. The communication of the other step is at most $O(n)$. In summary, the communication of SC (SCwL) is $O(n \log m)$.

We prove security in the standard semi-honest simulation-based paradigm. For a formal definition of simulation-based security in the semi-honest setting, we refer the reader to [32]. For all proofs, we exhibit simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ for simulating corrupt $\mathcal{S}$ and $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Input: P_1 inputs $X = \{x_i\}_{i\in[n]}$ and P_2 inputs $Y = \{(y_j, u_j)\}_{j\in[m]}$, where $x_i, y_j \in \{0,1\}^*$, $u_j \in \mathbb{Z}_p$ for some integer modular p, $n \ll m$:
Output: P_1 outputs two vectors $(\mathbf{r}, \mathbf{e})$. P_2 outputs two vectors $(\mathbf{r}', \mathbf{e}')$.

1. [**Setup**] Both parties agree on the hashing, mp-OPRF, and FHE parameters.
2. [**Hashing**] P_1 hashes X into table $\mathbf{x}'_c$ by Cuckoo hashing, where $\mathbf{x}'_c$ consists of n_c bins and each bin has one item. P_2 uses the same hash functions to insert Y into table $\mathbf{Y}'_{B\times n_c}$, where $\mathbf{Y}'_{B\times n_c}$ consists of n_c bins and each bin has B items.
3. [**mp-OPRF**] Both parties input their items in hash tables and invoke mp-OPRF functionality. The result is P_1 obtains PRF values of $\mathbf{x}'_c$ denoted as $\mathbf{x}_c$, and P_2 obtains a PRF key k. Then, P_2 computes PRF values of $\mathbf{Y}'_{B\times n_c}$ denoted as $\mathbf{Y}_{B\times n_c}$.
4. [**Computing coefficients** from $\mathbf{Y}_{B\times n_c}$] P_2 chooses random vectors $\mathbf{r}' = [r'_j]_{j\in[n_c]}$ and $\mathbf{e}' = [e'_j]_{j\in[n_c]}$. For j-th bin $\mathbf{y}_j = [y_{j,i}]_{i\in[B]}$, $j \in [n_c]$, P_2 computes polynomials $F_j(x) = f_j(x) + r'_j$ and $G_j(x) = f_j(x) + h_j(x) - e'_j$, where for all $i \in [B]$, $f_j(y_{j,i}) = 0$ and $h_j(y_{j,i}) = u_{j,i}$[6]. Thus, P_2 obtains two coefficient matrices $\mathbf{A}$ and $\mathbf{L}$, where j-th column of $\mathbf{A}$ and $\mathbf{L}$ are the coefficients of F_j and G_j.
5. [**Encrypt** $\mathbf{x}_c$] P_1 uses its FHE public key to encrypt each element in $\mathbf{x}_c = [x_j]_{j\in[n_c]}$ and sends all ciphertexts $[\![x_j]\!]$, $j \in [n_c]$ to P_2.
6. [**Homomorphically Computing**] For each $[\![x_j]\!]$, P_2 homomorphically computes encryptions of all powers $\mathbf{C}_j = [[\![x_j^1]\!], \cdots, [\![x_j^B]\!]]$. Then, P_2 homomorphically evaluates $\mathbf{C}'_j = \mathbf{C}_j\mathbf{A}_j$ and $\mathbf{C}^*_j = \mathbf{C}_j\mathbf{L}_j$, $j \in [n_c]$, and sends all ciphertexts to P_1.
7. [**Decrypt** and **Output**] P_1 decrypts the ciphertexts into $\mathbf{r} = [r_j]_{j\in[n_c]}$ and $\mathbf{e} = [e_j]_{j\in[n_c]}$, and then outputs them. P_2 outputs $\mathbf{r}' = [r'_j]_{j\in[n_c]}$ and $\mathbf{e}' = [e'_j]_{j\in[n_c]}$.

Fig. 9: Shared characteristic (with labels) protocols. The marked parts are only needed for shared characteristic with labels protocol.

Theorem 1. *The protocols in Figure 9, are secure for $\mathcal{F}_{\text{SC/SCwL}}$ in the $\mathcal{F}_{\text{mp-OPRF}}$-hybrid model, in the presence of semi-honest adversaries, provided that the fully homomorphic encryption scheme is IND-CPA secure.*

Proof. We prove the security of SC/SCwL protocols and give the simulators Sim_{P_1} and Sim_{P_2} as follows.

– **Corrupt P_1.** $\mathsf{Sim}_{P_1}(X, (\mathbf{r}, \mathbf{e}))$ simulates the view of corrupt P_1 as follows. Sim_{P_1} hashes X into $\mathbf{x}'_c$ as the real protocol, and randomly selects PRF values $\mathbf{x}_c$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{mp-OPRF}}(\mathbf{x}'_c, \mathbf{x}_c)$ and appends the output to the view. Then, it encrypts $(\mathbf{r}, \mathbf{e})$ to simulate the ciphertexts in step 6. The security of mp-OPRF guarantees the view in simulation is computationally indistinguishable from the real view.
– **Corrupt P_2.** Sim_{P_2} $(Y, (\mathbf{r}', \mathbf{e}'))$ simulates the view of corrupt P_2 as follows. Sim_{P_2} hashes Y into $\mathbf{Y}'_{B\times n_c}$ as the real protocol, and randomly selects PRF key k, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{mp-OPRF}}(\mathbf{Y}'_{B\times n_c}, k)$ and appends the output to the view. Then, it computes all PRF values and coefficients as real protocol and encrypts n_c random values to simulate the ciphertexts in step 5. The view generated by Sim_{P_2} is indistinguishable from a real view of P_2 by the following hybrids:
  - Hyb_0: P_2's view in the real protocol.
  - Hyb_1: Same as Hyb_0 except that Sim_{P_2} runs the $\mathcal{F}_{\text{mp-OPRF}}$ simulator to produce the simulated view for P_2. The security of the mp-OPRF protocol guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. The hybrid is the view output by Sim_{P_2}.
  - Hyb_2: Same as Hyb_1 except that the ciphertexts are replaced by encrypting random values generated by Sim_{P_2}. Since the fully homomorphic encryption scheme is IND-CPA secure, the simulation is indistinguishable from the real view.

4 Private Computing on Set Intersection

In this section, we give the frameworks of PCSI protocols in two cases where one is that $\mathcal{S}$ holds the small set ($\mathcal{S} \ll \mathcal{R}$) and the other is $\mathcal{R}$ holds the small set ($\mathcal{S} \gg \mathcal{R}$).

4.1 PCSI ($\mathcal{S} \ll \mathcal{R}$)

Here, PCSI ($\mathcal{S} \ll \mathcal{R}$) is described in Figure 10, including PCSI-card, PCSI-card-sum and PCSI-secret-sharing.

Theorem 2. *PCSI ($\mathcal{S} \ll \mathcal{R}$) protocols in Figure 10, including PCSI-card in step 1, PCSI-card-sum in step 2, and PCSI-secret-sharing in step 3, are secure against semi-honest adversaries in the ($\mathcal{F}_{\text{SC}}$, $\mathcal{F}_{\text{pm-PEQT}}$, $\mathcal{F}_{\text{OT}}$)-hybrid model.*

Proof. We prove the security of PCSI-card ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X)$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}^{P_1}_{\text{SC}}(X, \mathbf{r})$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{r}, \pi)$ and appends the output to the view. Since the views of the underlying SC and pm-PEQT simulator are indistinguishable, the simulation is indistinguishable from the real view.
– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}$ (Y, k) chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}^{P_2}_{\text{SC}}$ $(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0,1\}^n$, s.t. $\sum_{i=1}^{n} b_i = k$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. The security of SC and pm-PEQT guarantees the view in simulation is indistinguishable from the real view.

We prove the security of PCSI-card-sum ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X)$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}^{P_1}_{\text{SC}}(X, \mathbf{r})$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it generates (v_i^0, v_i^1) like the real protocol. Finally, it invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{OT}}(v_i^0, v_i^1)$, $i \in [n]$ and appends the output to the view. The security of SC, pm-PEQT, and OT guarantees the view in simulation is indistinguishable from the real view.

Input: $\mathcal{S}$ inputs set $X = \{x_i\}_{i\in[n]}$ and $\mathcal{R}$ inputs set $Y = \{y_j\}_{j\in[m]}$, where $n \ll m$, $x_i, y_j \in \mathbb{Z}_p$ for some integer modular p.
Protocols:

1) $\mathcal{S}$ and $\mathcal{R}$ play as P_1 and P_2 to invoke $\mathcal{F}_{\text{SC}}$: both parties input X and Y. As a result, $\mathcal{S}$ outputs a vector $\mathbf{r} = [r_i]_{i\in[n]}$ and $\mathcal{R}$ outputs a vector $\mathbf{r}' = [r'_i]_{i\in[n]}$.
2) Both parties invoke $\mathcal{F}_{\text{pm-PEQT}}$: $\mathcal{S}$ inputs $\mathbf{r}$ and a random permutation π over $[n]$, and $\mathcal{R}$ inputs $\mathbf{r}'$. The result is that $\mathcal{R}$ gets a bit vector $\mathbf{b} = [b_i]_{i\in[n]}$, where $b_i = 1$ if $r_{\pi(i)} = r'_{\pi(i)}$, and $b_i = 0$ otherwise.

1. **PCSI-card**.
   3) $\mathcal{R}$ computes and outputs $k = \sum_{i=1}^{n} b_i$.
2. **PCSI-card-sum**.
   3) $\mathcal{S}$ permutes $x_i, i \in [n]$ with π to obtain $x_{\pi(i)}$, and then chooses n random values e_i such that $\sum_{i=1}^{n} e_i = 0$. $\mathcal{S}$ lets $v_i^1 = x_{\pi(i)} + e_i, i \in [n]$ and $v_i^0 = e_i$.
   4) $\mathcal{S}$ and $\mathcal{R}$ run $\mathcal{F}_{\text{OT}}$: for each $i \in [n]$, $\mathcal{S}$ inputs (v_i^0, v_i^1) and $\mathcal{R}$ inputs b_i. Thus, if $b_i = 1$, $\mathcal{R}$ obtains $u_i = v_i^1$, otherwise, gets $u_i = v_i^0$. Finally, $\mathcal{R}$ computes and outputs $s = \sum_{i=1}^{n} u_i$.
3. **PCSI-secret-sharing**.
   3) $\mathcal{S}$ shuffles $x_i, i \in [n]$ with π to obtain $x_{\pi(i)}$, and uses its public key $pk_{\mathcal{S}}$ to compute $c_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{S}}, x_{\pi(i)})$, $i \in [n]$ and sends $\mathbf{c} = [c_i]_{i\in[n]}$ to $\mathcal{R}$.
   4) $\mathcal{R}$ removes the ciphertexts when $b_i = 0$: for all $i \in [n]$, if $b_i = 1$, lets $c'_j = c_i$, $j \in [k]$, and then chooses a random permutation π' over $[k]$ to permute c'_j and obtains $c^*_{\pi'(j)}$. After that, $\mathcal{R}$ chooses a random vector $\mathbf{s}' = [s'_j]_{j\in[k]}$ and computes $c_{\pi'(j)} \boxplus (-s'_j)$ and sends all ciphertexts back to $\mathcal{S}$.
   5) $\mathcal{S}$ decrypts the ciphertexts to s_j, $j \in [k]$ and outputs the plaintexts $\mathbf{s} = [s_j]_{j\in[k]}$. $\mathcal{R}$ outputs $\mathbf{s}$.

Fig. 10: PCSI ($\mathcal{S} \ll \mathcal{R}$) protocols

– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}$ (Y, k, s) chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}_{\text{SC}}^{P_2}(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0,1\}^n$, s.t. $\sum_{i=1}^{n} b_i = k$, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it chooses a random values $v_i, i \in [n]$, s.t. $\sum_{i=1}^{n} v_i = s$. Finally, it invokes $\mathsf{Sim}_{\text{OT}}^{\mathcal{R}}(b_i, v_i)$, $i \in [n]$ and appends the output to the view. The views of the underlying SC, pm-PEQT, and OT simulator are indistinguishable.

We prove the security of PCSI-secret-sharing ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X, \mathbf{s})$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}_{\text{SC}}^{P_1}(X, \mathbf{r})$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}, \pi)$ and appends the output to the view. Finally, it encrypts $\mathbf{s}$ to simulate the ciphertexts in step 4). The views of the underlying SC and pm-PEQT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.
– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, \mathbf{s}')$ chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}_{\text{SC}}^{P_2}$ $(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0,1\}^n$, s.t. $\sum_{i=1}^{n} b_i$ is equal to the number of the elements in $\mathbf{s}'$. $\mathsf{Sim}_{\mathcal{R}}$ invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. Finally, it encrypts random values to simulate the ciphertexts in step 3). The view generated by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from a real view of $\mathcal{R}$ by the following hybrids:
  - Hyb_0: $\mathcal{R}$'s view in the real protocol.
  - Hyb_1: Same as Hyb_0 except that $\mathsf{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{SC}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of SC guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
  - Hyb_2: Same as Hyb_1 except that $\mathsf{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{pm-PEQT}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of pm-PEQT guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.

- Hyb_3: Same as Hyb_2 except that the ciphertexts in step 3) are replaced by encrypting random values generated by $\mathsf{Sim}_{\mathcal{R}}$. Since the AHE is IND-CPA secure, the simulation is indistinguishable from the real view.

PCSI-card-function. According to PCSI-secret-sharing, $\mathcal{S}$ and $\mathcal{R}$ get the secret-shared intersection. Thus, both parties can compute any function of intersection based on generic 2PC.

5 PCSI ($\mathcal{S} \gg \mathcal{R}$)

In this section, PCSI ($\mathcal{S} \gg \mathcal{R}$) is described in Figure 11, including PCSI-card, PCSI-card-sum and PCSI-secret-sharing.

Input: $\mathcal{S}$ inputs set $X = \{x_i\}_{i\in[n]}$ and $\mathcal{R}$ inputs set $Y = \{y_i\}_{i\in[m]}$, where $x_i, y_i \in \mathbb{Z}_p$ for some integer modular p, $n \gg m$.
Protocols:

1) $\mathcal{S}$ and $\mathcal{R}$ play as P_2 and P_1 to invoke $\mathcal{F}_{\text{SC}}$: Both parties input X and Y, and the result is that $\mathcal{S}$ outputs a vector $\mathbf{r} = [r_i]_{i\in[m]}$ and $\mathcal{R}$ outputs a vector $\mathbf{r}' = [r'_i]_{i\in[m]}$.
2) Both parties invoke $\mathcal{F}_{\text{pm-PEQT}}$: $\mathcal{S}$ inputs $\mathbf{r}$ and a permutation π over $[m]$, and $\mathcal{R}$ inputs $\mathbf{r}'$. As a result, $\mathcal{R}$ gets a bit vector $\mathbf{b} = [b_i]_{i\in[m]}$, where $b_i = 1$ if $r_{\pi(i)} = r'_{\pi(i)}$, and $b_i = 0$ otherwise.

1. **PCSI-card**.
   3) $\mathcal{R}$ computes and outputs $k = \sum_{i=1}^{m} b_i$.
2. **PCSI-card-sum**.
   3) $\mathcal{R}$ uses its public key $pk_{\mathcal{R}}$ to compute c_i=AHE.$\mathsf{Enc}(pk_{\mathcal{R}},\, y_i)$, $i \in [m]$ and sends $\mathbf{c} = [c_i]_{i\in[m]}$ to $\mathcal{S}$.
   4) $\mathcal{S}$ permutes c_i, $i \in [m]$ with π to obtain $c_{\pi(i)}$, and then chooses m random values e_i such that $\sum_{i=1}^{m} e_i = 0$. $\mathcal{S}$ lets c_i^1= $c_{\pi(i)} \boxplus e_i$ =AHE.$\mathsf{Enc}(pk_{\mathcal{R}},\, y_{\pi(i)} + e_i)$ and $c_i^0 = e_i$, $i \in [m]$.
   5) $\mathcal{S}$ and $\mathcal{R}$ runs $\mathcal{F}_{\text{OT}}$: for each $i \in [m]$, $\mathcal{S}$ inputs (c_i^0, c_i^1) and $\mathcal{R}$ inputs b_i. Thus, if $b_i = 1$, $\mathcal{R}$ gets and decrypts c_i^1 to $m_i = y_{\pi(i)} + e_i$. Otherwise, $\mathcal{R}$ lets $m_i = c_i^0$. Finally, $\mathcal{R}$ computes and outputs $s = \sum_{i=1}^{m} m_i$.
3. **PCSI-secret-sharing**.
   2) $\mathcal{S}$ and $\mathcal{R}$ switch their roles in $\mathcal{F}_{\text{pm-PEQT}}$: $\mathcal{S}$ inputs $\mathbf{r}$ and $\mathcal{R}$ inputs $\mathbf{r}'$ and a permutation π over $[m]$. The result is that $\mathcal{S}$ gets a bit vector $\mathbf{b} = [b_i]_{i\in[m]}$.
   3) $\mathcal{R}$ shuffles Y with π and uses its public key $pk_{\mathcal{R}}$ to compute c_i = AHE.$\mathsf{Enc}(pk_{\mathcal{R}},\, y_{\pi(i)})$, $i \in [m]$ and sends $\mathbf{c} = [c_i]_{i\in[m]}$ to $\mathcal{S}$.
   4) $\mathcal{S}$ removes the ciphertexts when $b_i = 0$ and for all $i \in [m]$, if $b_i = 1$, lets $c'_j = c_i$, $j \in [k]$, then chooses a random permutation π' over $[k]$ to permute c'_j. After that, $\mathcal{S}$ chooses a random vector $\mathbf{s} = [s_j]_{j\in[k]}$ and computes $c^*_{\pi'(j)} \boxplus (-s_j)$ and sends all ciphertexts back to $\mathcal{R}$.
   5) $\mathcal{R}$ decrypts the ciphertexts to s'_j, $j \in [k]$ and outputs the plaintexts $\mathbf{s}' = [s'_j]_{j\in[k]}$. $\mathcal{S}$ outputs $\mathbf{s}$.

Fig. 11: PCSI ($\mathcal{S} \gg \mathcal{R}$) protocols

Theorem 3. *PCSI ($\mathcal{S} \gg \mathcal{R}$) protocols in Figure 11, including PCSI-card in step 1, PCSI-card-sum in step 2, and PCSI-secret-sharing in step 3, are secure against semi-honest adversaries in the ($\mathcal{F}_{\text{SC}}$, $\mathcal{F}_{\text{pm-PEQT}}$, $\mathcal{F}_{\text{OT}}$)-hybrid model.*

Proof. We prove the security of PCSI-card ($\mathcal{S} \gg \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X)$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}_{\text{SC}}^{P_2}(X,\, \mathbf{r})$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}, \pi)$ and appends the output to the view. Since the views of the underlying SC and pm-PEQT simulator are indistinguishable, the simulation is indistinguishable from the real view.

– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}$ (Y, k) chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}_{\mathrm{SC}}^{P_1}$ $(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[m]} \in \{0,1\}^m$, s.t. $\sum_{i=1}^{m} b_i = k$, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. The security of SC and pm-PEQT guarantees the view in simulation is indistinguishable from the real view.

We prove the security of PCSI-card-sum ($\mathcal{S} \gg \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X)$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}_{\mathrm{SC}}^{P_2}(X, \mathbf{r})$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it encrypts random values in place of the ciphertexts in step 3), and then generates (c_i^0, c_i^1) like the real protocol. Finally, it invokes $\mathsf{Sim}_{\mathrm{OT}}^{\mathcal{S}}(c_i^0, c_i^1)$, $j \in [m]$ and appends the output to the view. The security of AHE, SC, pm-PEQT, and OT guarantees the view in simulation is indistinguishable from the real view.
– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}$ (Y, k, s) chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}_{\mathrm{SC}}^{P_1}(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[m]} \in \{0,1\}^m$, s.t. $\sum_{i=1}^{m} b_i = k$, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it chooses random values $v_i, i \in [m]$, s.t. $\sum_{i=1}^{m} v_i = s$. If $b_i = 1$, it encrypts v_i to c_i, else, lets $c_i = v_i$. Finally, it invokes $\mathsf{Sim}_{\mathrm{OT}}^{\mathcal{R}}(b_i, c_i)$, $i \in [m]$ and appends the output to the view. The views of the underlying SC, pm-PEQT, and OT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

We prove the security of PCSI-secret-sharing ($\mathcal{S} \gg \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X, \mathbf{s})$ chooses a random vector $\mathbf{r}$, invokes $\mathsf{Sim}_{\mathrm{SC}}^{P_2}(X,\ \mathbf{r})$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[m]} \in \{0,1\}^m$, s.t. $\sum_{i=1}^{m} b_i$ is equal to the number of the elements in $\mathbf{s}$. $\mathsf{Sim}_{\mathcal{S}}$ invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}, \mathbf{b})$ and appends the output to the view. Finally, it encrypts random values to simulate the ciphertexts in step 3). The security of AHE, SC, and pm-PEQT guarantees the view in simulation is indistinguishable from the real view.
– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, \mathbf{s}')$ chooses a random vector $\mathbf{r}'$, invokes $\mathsf{Sim}_{\mathrm{SC}}^{P_1}$ $(Y, \mathbf{r}')$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}', \pi)$ and appends the output to the view. Finally, it encrypts $\mathbf{s}'$ to simulate the ciphertexts in step 4). The views of the underlying SC and pm-PEQT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

6 Private Computing on Labeled Set Intersection

In this section, we consider two types of PCLSI protocols: one is to compute $\mathcal{S}'$s labels and the other is to calculate $\mathcal{R}'$s labels. Each type further consists of two cases in the unbalanced setting where one is that $\mathcal{R}$ holds the large set ($\mathcal{S} \ll \mathcal{R}$) and the other is that $\mathcal{S}$ holds the large set ($\mathcal{S} \gg \mathcal{R}$).

6.1 PCLSI ($\mathcal{S} \ll \mathcal{R}$)

Here, we present two types of PCLSI (with $\mathcal{S}'$s or $\mathcal{R}'$s labels) protocols in the case that $\mathcal{R}$ holds the large set ($\mathcal{S} \ll \mathcal{R}$).

PCLSI with $\mathcal{S}'$s labels. We describe PCLSI with $\mathcal{S}'$s labels ($\mathcal{S} \ll \mathcal{R}$) protocols as follows:

1. PCLSI-card-sum with $\mathcal{S}'$s labels ($\mathcal{S} \ll \mathcal{R}$) is similar to PCSI-card-sum ($\mathcal{S} \ll \mathcal{R}$) in Figure 10 step 2, except that $\mathcal{S}$ generates the inputs of $\mathcal{F}_{\mathrm{OT}}$ by using the labels v_i instead of the items x_i, $i \in [n]$.
2. PCLSI-secret-sharing with $\mathcal{S}'$s labels ($\mathcal{S} \ll \mathcal{R}$) is similar to PCSI-secret-sharing ($\mathcal{S} \ll \mathcal{R}$) in Figure 10 step 3, except that $\mathcal{S}$ encrypts its labels v_i instead of the items x_i, $i \in [n]$.

Input: $\mathcal{S}$ inputs set $X = \{(x_i, v_i)\}_{i\in[n]}$ and $\mathcal{R}$ inputs set $Y = \{(y_j, u_j)\}_{j\in[m]}$, where each $u_j \in \mathbb{Z}_p$ for some integer modular p, $x_i, y_j \in \{0,1\}^*$, $n \ll m$:
Protocols:

1) $\mathcal{S}$ and $\mathcal{R}$ play as P_1 and P_2 to invoke $\mathcal{F}_{\text{SCwL}}$: both parties input X and Y. The result is that $\mathcal{S}$ outputs two vectors $(\mathbf{r}, \mathbf{e})$ and $\mathcal{R}$ outputs two vectors $(\mathbf{r}', \mathbf{e}')$.
2) Both parties invoke $\mathcal{F}_{\text{pm-PEQT}}$: $\mathcal{S}$ inputs $\mathbf{r}$ and a permutation π over $[n]$, and $\mathcal{R}$ inputs $\mathbf{r}'$. As a result, $\mathcal{R}$ gets a bit vector $\mathbf{b} = [b_i]_{i\in[n]}$, where $b_i = 1$ if $r_{\pi(i)} = r'_{\pi(i)}$, and $b_i = 0$ otherwise.

1. **PCLSI-card-sum.**
   3) $\mathcal{R}$ uses its public key $pk_\mathcal{R}$ to compute $c_i = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_i)$, $i \in [n]$ and sends the ciphertexts to $\mathcal{S}$.
   4) $\mathcal{S}$ permutes c_i with π to obtain permuted ciphertexts $c_{\pi(i)}$=AHE.$\mathsf{Enc}(pk_\mathcal{R},\, e'_{\pi(i)})$, then chooses a random vector $\mathbf{w} = [w_i]_{i\in[n]}$ s.t. $\sum_{i=1}^n w_i = 0$, computes c_i^1=AHE.$\mathsf{Enc}(pk_\mathcal{R},\, e'_{\pi(i)}) \boxplus (w_i + e_{\pi(i)}) = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_{\pi(i)} + w_i + e_{\pi(i)})$, and lets $c_i^0 = w_i$.
   5) $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\text{OT}}$: for $i \in [n]$, $\mathcal{S}$ inputs (c_i^0, c_i^1) and $\mathcal{R}$ inputs b_i. As a result, if $b_i = 1$, $\mathcal{R}$ obtains the ciphertext c_i^1 and decrypt it to $m_i = e'_{\pi(i)} + w_i + e_{\pi(i)}$, otherwise, $\mathcal{R}$ lets $m_i = c_i^0$. Finally, $\mathcal{R}$ computes and outputs $\sum_{i=1}^n m_i$.
2. **PCLSI-secret-sharing.**
   3) $\mathcal{R}$ uses its public key $pk_\mathcal{R}$ to compute and send $c_i = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_i)$ to $\mathcal{S}$.
   4) $\mathcal{S}$ randomly chooses $\mathbf{w} = [w_i]_{i\in[n]}$ and computes $c'_i = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_i) \boxplus (w_i + e_i)$. Then, $\mathcal{S}$ uses its public key $pk_\mathcal{S}$ to encrypt $-w_i$ and obtains $\hat{c}_i = \text{AHE.}\mathsf{Enc}(pk_\mathcal{S}, -w_i)$. $\mathcal{S}$ shuffles c'_i and $\hat{c}_i$ with π and sends $(c'_{\pi(i)}, \hat{c}_{\pi(i)})$ to $\mathcal{R}$.
   5) $\mathcal{R}$ lets $c''_j = c'_{\pi(i)}$ and $\hat{c}'_j = \hat{c}_{\pi(i)}$, $j \in [k]$, if $b_i = 1$, and removes the ciphertexts if $b_i = 0$. Then, $\mathcal{R}$ decrypts c''_j to m_j, chooses a random permutation π' over $[k]$, and shuffles m_j and $\hat{c}'_j$ with π'. $\mathcal{R}$ chooses a random vector $\mathbf{s}' = [s'_j]_{j\in[k]}$, and sends $\hat{c}'_{\pi'(j)} \boxplus (-s'_j + m_{\pi'(j)})$ to $\mathcal{S}$.
   6) $\mathcal{S}$ decrypts $\hat{c}'_{\pi'(j)} \boxplus (-s'_j + m_{\pi'(j)})$ to s_j and outputs $\mathbf{s} = [s_j]_{j\in[k]}$. $\mathcal{R}$ outputs $\mathbf{s}'$.
3. **PCLSI-card-inner-product.**
   3) $\mathcal{R}$ uses its public key $pk_\mathcal{R}$ to compute and send $c_i = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_i)$ to $\mathcal{S}$.
   4) $\mathcal{S}$ permutes c_i with π, chooses a random vector $\mathbf{w} = [w_i]_{i\in[n]}$ such that $\sum_{i=1}^n w_i = 0$, then lets $c_i^1 = (\text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_{\pi(i)}) \boxtimes v_{\pi(i)}) \boxplus (w_i + e_{\pi(i)} v_{\pi(i)}) = \text{AHE.}\mathsf{Enc}(pk_\mathcal{R}, e'_{\pi(i)} v_{\pi(i)} + w_i + e_{\pi(i)} v_{\pi(i)})$ and $c_i^0 = w_i$.
   5) $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\text{OT}}$: for $i \in [n]$, $\mathcal{S}$ inputs (c_i^0, c_i^1) and $\mathcal{R}$ inputs b_i. As a result, if $b_i = 1$, $\mathcal{R}$ obtains c_i^1 and decrypt it to $m_i = e'_{\pi(i)} v_{\pi(i)} + w_i + e_{\pi(i)} v_{\pi(i)}$, otherwise, $\mathcal{R}$ lets $m_i = c_i^0$. Finally, $\mathcal{R}$ computes and outputs $\sum_{i=1}^n m_i$.

Fig. 12: PCLSI with $\mathcal{R}'$s labels ($\mathcal{S} \ll \mathcal{R}$) protocols

PCLSI with $\mathcal{R}'$s labels. Here, we present the framework of PCLSI with $\mathcal{R}'$s labels ($\mathcal{S} \ll \mathcal{R}$) in Figure 12.

Theorem 4. *PCLSI with $\mathcal{R}'$s labels ($\mathcal{S} \ll \mathcal{R}$) in Figure 12, including PCLSI-card-sum with $\mathcal{R}'$s labels in step 1, PCLSI-secret-sharing with $\mathcal{R}'$s labels in step 2, PCLSI-card-inner-product in step 3 are secure against semi-honest adversaries in $(\mathcal{F}_{\text{SCwL}}, \mathcal{F}_{\text{pm-PEQT}}, \mathcal{F}_{\text{OT}})$-hybrid model, provided that AHE is IND-CPA secure.*

Proof. We prove the security of PCLSI-card-sum with $\mathcal{R}'$s labels ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_\mathcal{S}$ and $\mathsf{Sim}_\mathcal{R}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_\mathcal{S}(X)$ chooses random vectors $(\mathbf{r}, \mathbf{e})$, invokes $\mathsf{Sim}_{\text{SCwL}}^{P_1}(X, (\mathbf{r}, \mathbf{e}))$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 3), and then generates (c_i^0, c_i^1), $i \in [n]$ as the real protocol. Finally, it invokes $\mathsf{Sim}_{\text{OT}}^{\mathcal{S}}(c_i^0, c_i^1)$, $i \in [n]$ and appends the output to the view. The security of AHE, SCwL, pm-PEQT, and OT guarantees the view in simulation is indistinguishable from the real view.

- **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, k, s)$ chooses random vectors $(\mathbf{r}', \mathbf{e}')$, invokes $\mathsf{Sim}^{P_2}_{\text{SCwL}}\ (Y, (\mathbf{r}', \mathbf{e}'))$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0, 1\}^n$, s.t. $\sum_{i=1}^{n} b_i = k$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}\ (\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it chooses random values $v_i, i \in [n]$, s.t. $\sum_{i=1}^{n} v_i = s$. If $b_i = 0$, it defines $c_i = v_i$, else, it generates the ciphertext c_i by encrypting v_i. Finally, it invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{OT}}(b_i, c_i)$, $i \in [n]$ and appends the output to the view. The views of the underlying SCwL, pm-PEQT and OT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

We prove the security of PCLSI-secret-sharing with $\mathcal{R}'$s labels ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

- **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}\ (X, \mathbf{s})$ chooses random vectors $(\mathbf{r}, \mathbf{e})$, invokes $\mathsf{Sim}^{P_1}_{\text{SCwL}}\ (X, (\mathbf{r}, \mathbf{e}))$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 4). The views of the underlying SCwL and pm-PEQT simulator are indistinguishable. Since the AHE is IND-CPA secure, the simulation is indistinguishable from the real view.
- **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, \mathbf{s}')$ chooses random vectors $(\mathbf{r}', \mathbf{e}')$, invokes $\mathsf{Sim}^{P_2}_{\text{SCwL}}\ (Y, (\mathbf{r}', \mathbf{e}'))$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0, 1\}^n$, s.t. $\sum_{i=1}^{n} b_i$ is equal to the number of the elements in $\mathbf{s}'$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 3) and encrypts $\mathbf{s}$ to simulate the ciphertexts in step 5). The security of AHE, SCwL, and pm-PEQT guarantees the view in simulation is indistinguishable from the real view.

We prove the security of PCLSI-card-inner-product ($\mathcal{S} \ll \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

- **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}\ (X, \mathbf{s})$ chooses random vectors $(\mathbf{r}, \mathbf{e})$, invokes $\mathsf{Sim}^{P_1}_{\text{SCwL}}\ (X, (\mathbf{r}, \mathbf{e}))$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it generates (c_i^0, c_i^1) like the real protocol. Finally, it invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{OT}}(c_i^0, c_i^1)$, $i \in [n]$ and appends the output to the view. The security of SCwL, pm-PEQT, and OT guarantees the view in simulation is indistinguishable from the real view.
- **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, k, s)$ chooses random vectors $(\mathbf{r}', \mathbf{e}')$, invokes $\mathsf{Sim}^{P_2}_{\text{SCwL}}\ (Y, (\mathbf{r}', \mathbf{e}'))$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[n]} \in \{0, 1\}^n$, s.t. $\sum_{i=1}^{n} b_i = k$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it chooses random values $v_i, i \in [n]$, s.t. $\sum_{i=1}^{n} v_i = s$. If $b_i = 0$, it defines $c_i = v_i$, else, it generates the ciphertext c_i by encrypting v_i. Finally, it invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{OT}}(b_i, c_i)$, $i \in [n]$ and appends the output to the view. The views of the underlying SCwL, pm-PEQT and OT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

6.2 PCLSI ($\mathcal{S} \gg \mathcal{R}$)

In this section, we present two types of PCLSI (with $\mathcal{S}'$s or $\mathcal{R}'$s labels) protocols in the case that $\mathcal{S}$ holds the large set ($\mathcal{S} \gg \mathcal{R}$).

PCLSI with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$). Here, we present the framework of PCLSI with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$) in Figure 13.

Theorem 5. *PCLSI with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$) in Figure 13, including PCLSI-card-sum with $\mathcal{S}'$s labels in step 1, PCLSI-secret-sharing with $\mathcal{S}'$s labels in step 2, PCLSI-card-inner-product in step 3 are secure against semi-honest adversaries in ($\mathcal{F}_{\text{SCwL}}, \mathcal{F}_{\text{pm-PEQT}}, \mathcal{F}_{\text{OT}}$)-hybrid model, provided that AHE is IND-CPA secure.*

Input: $\mathcal{S}$ inputs set $X = \{(x_i, v_i)\}_{i\in[n]}$ and $\mathcal{R}$ inputs set $Y = \{(y_i, u_i)\}_{i\in[m]}$, where each $v_i, u_i \in \mathbb{Z}_p$ for some integer modular p, $x_i, y_i \in \{0,1\}^*$, $n \gg m$:
Protocols:

1) $\mathcal{S}$ and $\mathcal{R}$ play as P_2 and P_1 to invoke $\mathcal{F}_{\text{SCwL}}$: both parties input X and Y. The result is that $\mathcal{S}$ outputs two vectors $(\mathbf{r}, \mathbf{e})$ and $\mathcal{R}$ outputs two vectors $(\mathbf{r}', \mathbf{e}')$.
2) Both parties invoke $\mathcal{F}_{\text{pm-PEQT}}$: $\mathcal{S}$ inputs $\mathbf{r}$ and a permutation π, and $\mathcal{R}$ inputs $\mathbf{r}'$. As a result, $\mathcal{R}$ gets a bit vector $\mathbf{b} = [b_i]_{i\in[m]}$, where $b_i = 1$ if $r_{\pi(i)} = r'_{\pi(i)}$, and $b_i = 0$ otherwise.

1. **PCLSI-card-sum.**
   3) $\mathcal{R}$ uses its public key $pk_{\mathcal{R}}$ to compute and send $c_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, e'_i)$ to $\mathcal{S}$.
   4) $\mathcal{S}$ permutes c_i with π, chooses a random vector $\mathbf{w} = [w_i]_{i\in[m]}$ such that $\sum_{i=1}^m w_i = 0$, then lets c_i^1=AHE.$\mathsf{Enc}(pk_{\mathcal{R}},\, e'_{\pi(i)}) \boxplus (w_i + e_{\pi(i)})$=AHE.$\mathsf{Enc}(pk_{\mathcal{R}},\, e'_{\pi(i)} + w_i + e_{\pi(i)})$ and $c_i^0 = w_i$.
   5) $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\text{OT}}$: for $i \in [m]$, $\mathcal{S}$ inputs (c_i^0, c_i^1) and $\mathcal{R}$ inputs b_i. As a result, if $b_i = 1$, $\mathcal{R}$ obtains c_i^1 and decrypt it to $m_i = e'_{\pi(i)} + w_i + e_{\pi(i)}$, otherwise, $\mathcal{R}$ lets $m_i = c_i^0$. Finally, $\mathcal{R}$ computes and outputs $\sum_{i=1}^m m_i$.
2. **PCLSI-secret-sharing.**
   3) $\mathcal{R}$ uses its public key $pk_{\mathcal{R}}$ to compute and send $c_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, e'_i)$ to $\mathcal{S}$.
   4) $\mathcal{S}$ randomly chooses $\mathbf{w} = [w_i]_{i\in[m]}$ and computes $c'_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, e'_i) \boxplus (w_i + e_i)$. Then, $\mathcal{S}$ uses its public key $pk_{\mathcal{S}}$ to encrypt $-w_i$ and obtains $\hat{c}_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{S}}, -w_i)$. $\mathcal{S}$ shuffles c'_i and $\hat{c}_i$ with π and sends $(c'_{\pi(i)}, \hat{c}_{\pi(i)})$ to $\mathcal{R}$.
   5) $\mathcal{R}$ lets $c''_j = c'_{\pi(i)}$ and $\hat{c}'_j = \hat{c}_{\pi(i)}$, $j \in [k]$, if $b_i = 1$. Then, $\mathcal{R}$ decrypts c''_j to m_j, chooses a random permutation π' over $[k]$, and shuffles m_j and $\hat{c}'_j$ with π'. $\mathcal{R}$ chooses a random vector $\mathbf{s}' = [s'_j]_{j\in[k]}$, and sends $\hat{c}'_{\pi'(j)} \boxplus (-s'_j + m_{\pi'(j)})$ to $\mathcal{S}$.
   6) $\mathcal{S}$ decrypts $\hat{c}'_{\pi'(j)} \boxplus (-s'_j + m_{\pi'(j)})$ to s_j and outputs $\mathbf{s} = [s_j]_{j\in[k]}$. $\mathcal{R}$ outputs $\mathbf{s}'$.
3. **PCLSI-card-inner-product.**
   3) $\mathcal{R}$ uses its public key $pk_{\mathcal{R}}$ to compute and send $c_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, u_i \cdot e'_i)$, $c'_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, u_i)$, $i \in [m]$ to $\mathcal{S}$.
   4) $\mathcal{S}$ permutes c_i and c'_i with π. Then, $\mathcal{S}$ chooses random vector $\mathbf{w} = [w_i]_{i\in[m]}$, such that $\sum_{i=1}^m w_i = 0$, computes $c_i^1 = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, u_{\pi(i)} \cdot e'_{\pi(i)}) + (e_{\pi(i)} \boxtimes \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, u_{\pi(i)})) \boxplus w_i = \text{AHE}.\mathsf{Enc}(pk_{\mathcal{R}}, u_{\pi(i)} \cdot e'_{\pi(i)} + w_i + e_{\pi(i)} \cdot u_{\pi(j)})$, and lets $c_i^0 = w_i$.
   5) $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\text{OT}}$: for $i \in [m]$, $\mathcal{S}$ inputs (c_i^0, c_i^1) and $\mathcal{R}$ inputs b_i. As a result, if $b_i = 1$, $\mathcal{R}$ obtains the ciphertext c_i^1 and decrypt it to $m_i = u_{\pi(i)} \cdot e'_{\pi(i)} + w_i + e_{\pi(i)} \cdot u_{\pi(i)}$, otherwise, $\mathcal{R}$ lets $m_i = c_i^0$. Finally, $\mathcal{R}$ computes and outputs $\sum_{i=1}^m m_i$.

Fig. 13: PCLSI with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$) protocols

Proof. We prove the security of PCLSI-card-sum with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

– **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}(X)$ chooses random vectors $(\mathbf{r}, \mathbf{e})$, invokes $\mathsf{Sim}_{\text{SCwL}}^{P_2}(X, (\mathbf{r}, \mathbf{e}))$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{S}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 3), and then generates (c_i^0, c_i^1), $i \in [m]$ as the real protocol. Finally, it invokes $\mathsf{Sim}_{\text{OT}}^{\mathcal{S}}(c_i^0, c_i^1)$, $i \in [m]$ and appends the output to the view. The security of AHE, SCwL, pm-PEQT and OT guarantees the view in simulation is indistinguishable from the real view.
– **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, k, s)$ chooses random vectors $(\mathbf{r}', \mathbf{e}')$, invokes $\mathsf{Sim}_{\text{SCwL}}^{P_1}(Y, (\mathbf{r}', \mathbf{e}'))$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i\in[m]} \in \{0,1\}^m$, s.t. $\sum_{i=1}^m b_i = k$, invokes $\mathsf{Sim}_{\text{pm-PEQT}}^{\mathcal{R}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it chooses random values $v_i, i \in [m]$, s.t. $\sum_{i=1}^m v_i = s$. If $b_i = 0$, it defines $c_i = v_i$, else, it generates the ciphertext c_i by encrypting v_i. Finally, it invokes $\mathsf{Sim}_{\text{OT}}^{\mathcal{R}}(b_i, v_i)$, $i \in [m]$ and appends the output to the view. The views of the underlying

Parameters		Protocols	Comm. (MB)	Total running time (s)															
				10Gbps				200Mbps				50Mbps				1Mbps			
$\|X\|$	$\|Y\|$			$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$
2^{10}	2^{10}	[1]	0.07	0.06	0.03	0.02	0.02	0.22	0.19	0.18	0.17	0.23	0.20	0.18	0.18	0.74	0.73	0.72	0.72
		Ours	1.72	0.94	0.87	0.86	0.84	4.72	4.62	4.59	4.58	4.85	4.80	4.74	4.67	19.07	19.02	19.01	19.00
	2^{12}	[1]	0.26	0.22	0.13	0.08	0.08	0.76	0.70	0.68	0.67	0.78	0.73	0.71	0.70	2.89	2.83	2.81	2.80
		Ours	1.72	0.93	0.88	0.88	0.85	4.73	4.68	4.66	4.60	4.86	4.84	4.78	4.76	19.38	19.18	19.03	19.01
	2^{14}	[1]	1.06	0.84	0.45	0.33	0.25	1.96	1.56	1.54	1.29	2.04	1.66	1.46	1.39	10.57	9.90	9.84	9.74
		Ours	1.72	1.02	0.92	0.92	0.91	4.74	4.73	4.69	4.62	4.87	4.84	4.84	4.81	20.12	19.40	19.24	19.06
	2^{16}	[1]	4.23	3.26	1.70	1.00	0.80	5.10	3.35	2.72	2.24	5.23	3.80	3.07	2.55	38.55	37.41	37.01	37.15
		Ours	2.08	1.65	1.48	1.46	1.28	5.53	5.52	5.46	5.48	5.71	5.69	5.60	5.58	23.03	22.98	22.90	22.67
	2^{18}	[1]	16.93	13.18	6.90	3.70	3.00	15.86	9.58	6.20	5.49	17.56	11.20	7.99	7.10	150.54	147.13	145.56	145.34
		Ours	2.28	3.81	2.93	2.47	2.30	7.77	6.60	6.10	5.86	7.96	7.09	6.46	6.09	26.90	26.73	25.07	25.03
	2^{20}	[1]	67.70	52.58	27.60	14.80	11.70	58.73	33.56	20.76	18.10	66.10	41	28.22	25.04	609.47	584.72	578.68	576.90
		Ours	2.67	18.83	12.10	8.55	6.65	22.48	16.10	12.81	11.37	22.89	16.30	13.25	11.59	44.57	37.95	34.77	33.17
	2^{22}	[1]	270.41	211.20	114.65	71.62	58.53	230.07	132.54	86.54	76.05	260.64	162.14	116.86	101.74	2480.26	2378.49	2328.75	2309.41
		Ours	4.62	79.90	47.89	32.46	23.34	83.81	52.09	36.34	29.08	84.09	52.31	36.72	29.40	121.09	91.29	76.14	67.97

Table 2: Comparisons of communication (in MB) and runtime (in seconds) between [1] and our PCSI-card for sets size ($|X| = 2^{10}$, $|Y| \in \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$) with threads $T \in \{1, 2, 4, 8\}$, and 10Gbps bandwidth, 0.2ms RTT; 200Mbps, 50Mbps and 1Mbps bandwidth, 80ms RTT. The best results are marked in cyan.

SCwL, pm-PEQT and OT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

We prove the security of PCLSI-secret-sharing with $\mathcal{S}'$s labels ($\mathcal{S} \gg \mathcal{R}$) and give the simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ as follows.

- **Corrupt Sender.** $\mathsf{Sim}_{\mathcal{S}}$ $(X, \mathbf{s})$ chooses random vectors $(\mathbf{r}, \mathbf{e})$, invokes $\mathsf{Sim}^{P_2}_{\text{SCwL}}$ $(X, (\mathbf{r}, \mathbf{e}))$ and appends the output to the view. Then, it chooses a random permutation π, invokes $\mathsf{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{r}, \pi)$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 3) and encrypts $\mathbf{s}$ to simulate the ciphertexts in step 5). The security of AHE, SCwL, and pm-PEQT guarantees the view in simulation is indistinguishable from the real view.
- **Corrupt Receiver.** $\mathsf{Sim}_{\mathcal{R}}(Y, \mathbf{s}')$ chooses random vectors $(\mathbf{r}', \mathbf{e}')$, invokes $\mathsf{Sim}^{P_1}_{\text{SCwL}}$ $(Y, (\mathbf{r}', \mathbf{e}'))$ and appends the output to the view. Then, it chooses a random vector $\mathbf{b} = [b_i]_{i \in [m]} \in \{0, 1\}^m$, s.t. $\sum_{i=1}^{m} b_i$ is equal to the number of the elements in $\mathbf{s}'$, invokes $\mathsf{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}(\mathbf{r}', \mathbf{b})$ and appends the output to the view. After that, it encrypts random values to simulate the ciphertexts in step 4). The views of the underlying SCwL and pm-PEQT simulator are indistinguishable. Since the AHE is IND-CPA secure, the simulation is indistinguishable from the real view.

The security proof of PCLSI-card-inner-product ($\mathcal{S} \gg \mathcal{R}$) is similar to the security proof of PCSI-card-sum ($\mathcal{S} \gg \mathcal{R}$) in the theorem 3, except that $\mathsf{Sim}_{\mathcal{S}}$ encrypts two random vectors in place of the ciphertexts of $\mathbf{e}$ and $\mathbf{u} + \mathbf{e}$ in the real view.

PCLSI with $\mathcal{R}'$s labels ($\mathcal{S} \gg \mathcal{R}$) We describe PCLSI with $\mathcal{R}'$s labels ($\mathcal{S} \gg \mathcal{R}$) protocols as follows:

1. PCLSI card-sum with $\mathcal{R}'$s labels ($\mathcal{S} \gg \mathcal{R}$) is similar to PCSI-card-sum ($\mathcal{S} \gg \mathcal{R}$) in Figure 11 step 2, except that $\mathcal{R}$ encrypts the labels u_i of Y, instead of encrypting the items y_i, $i \in [m]$.
2. PCLSI-secret-sharing with $\mathcal{R}'$s labels ($\mathcal{S} \gg \mathcal{R}$) is similar to PCSI-secret-sharing in Figure 11 step 3, except that $\mathcal{R}$ encrypts the labels u_i of Y, instead of encrypting the items y_i, $i \in [m]$.

Parameters		Protocols	Comm. (MB)	Total running time (s)															
				10Gbps				200Mbps				50Mbps				1Mbps			
$\|X\|$	$\|Y\|$			$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$	$T=1$	$T=2$	$T=4$	$T=8$
2^{10}	2^{10}	[1]	0.09	0.09	0.06	0.03	0.05	0.56	0.52	0.51	0.50	0.57	0.54	0.52	0.51	1.45	1.43	1.42	1.42
		Ours	1.81	1.08	1.05	1.01	1.01	5.38	5.33	5.32	5.30	5.49	5.49	5.41	5.39	20.61	20.52	20.44	20.43
	2^{12}	[1]	0.39	0.25	0.14	0.11	0.11	1.27	1.21	1.19	1.17	1.31	1.25	1.22	1.21	4.35	4.30	4.27	4.26
		Ours	1.81	1.08	1.06	1.06	1.05	5.38	5.35	5.33	5.31	5.51	5.49	5.45	5.42	20.91	20.70	20.56	20.51
	2^{14}	[1]	1.47	0.89	0.51	0.37	0.32	2.35	1.95	1.74	1.69	2.49	2.08	1.88	1.82	13.94	14.72	14.61	14.57
		Ours	1.81	1.31	1.29	1.13	0.94	5.43	5.32	5.29	5.27	5.74	5.55	5.46	5.46	22.02	21.14	20.80	20.80
	2^{16}	[1]	5.76	3.50	1.90	1.10	0.90	5.87	4.12	3.33	3.00	6.23	4.62	3.72	3.42	51.36	50.55	50.15	50.04
		Ours	2.18	1.75	1.68	1.57	1.31	6.10	5.96	5.63	5.79	6.25	6.19	5.74	5.77	24.61	24.53	24.39	24.31
	2^{18}	[1]	22.93	13.83	7.40	4.20	3.60	17.38	10.93	7.68	6.72	19.87	13.29	10.05	9.17	202.13	198.94	197.33	196.93
		Ours	2.37	4.00	3.06	2.66	2.47	8.15	7.29	6.98	6.32	8.25	7.53	7.05	6.58	28.51	27.19	26.66	26.49
	2^{20}	[1]	91.70	54.95	29.30	16.60	13.80	63.41	37.89	25.09	22.01	73.69	48.19	35.50	32.63	814.12	789.22	783.27	781.97
		Ours	2.77	18.90	12.21	8.64	6.75	23.06	16.20	12.99	11.64	23.22	16.56	13.55	11.72	46.00	39.39	35.71	34.03
	2^{22}	[1]	366.42	221.60	122.06	81.20	67.53	245.72	147.35	105.25	96.13	287.94	189.52	145.75	130.92	3296.22	3200.74	3149.41	3128.08
		Ours	4.71	80.03	48.30	32.86	24.64	84.28	52.89	37.93	30.26	84.39	53.10	38.27	30.53	122.18	92.72	77.00	69.31

Table 3: Comparisons of communication (in MB) and runtime (in seconds) between [1] and our PCSI-card-sum for sets size ($|X| = 2^{10}$, $|Y| \in \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$) with threads $T \in \{1, 2, 4, 8\}$, and 10Gbps bandwidth, 0.2ms RTT; 200Mbps, 50Mbps and 1Mbps bandwidth, 80ms RTT. The best results are marked in cyan.

Setting	T	Large set size n						
		2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
LAN	1	0.94	0.93	1.02	1.65	3.81	18.83	79.90
	2	0.87	0.88	0.92	1.48	2.93	12.10	47.89
	4	0.86	0.88	0.92	1.46	2.47	8.55	32.46
	8	0.84	0.85	0.91	1.28	2.30	6.6	23.3
WAN	1	19.07	19.38	20.12	23.03	26.90	44.57	121.09
	2	19.02	19.18	19.4	22.98	26.73	37.95	91.29
	4	19.01	19.03	19.24	22.90	25.07	34.77	76.14
	8	19	19.01	19.06	22.67	25.03	33.17	67.97
Speedup		1.00-1.12×	1.01-1.09×	1.05-1.12×	1.02-1.28×	1.07-1.65×	1.34-2.85×	1.78-3.42×

Table 4: Scaling of our PCSI-card with set size and number of threads. Total runtime is in seconds. $n = \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$ for large set, 2^{10} for small set, and threads $T \in \{1, 2, 4, 8\}$. LAN setting with 10Gbps network bandwidth, 0.2ms RTT. WAN setting with 1Mbps bandwidth, 80ms RTT.

7 Implementations

In this section, we experimentally evaluate our PCSI/PCLSI protocols and compare with the state-of-the-art work [1][7] in terms of communication and runtime. Our source code is available upon request.

Experimental setup. We run our experiments on a single Intel Core i7-11700 CPU @ 2.50GHz with 16 threads and 16GB of RAM and simulate network latency and bandwidth by using the Linux `tc` command. The simulated network settings include typical LAN (10Gbps bandwidth and 0.2ms round-trip time (RTT)) and WAN (including 200Mbps, 50Mbps, and 1Mbps bandwidth, each with an 80ms RTT). The threads $T \in \{1, 2, 4, 8\}$. Following [1], we set computational security parameter $\kappa = 128$ and statistical security parameter $\lambda = 40$.

[7] PCSI [1] is more efficient than that of [7] (See [1] for a detailed comparison, where [1] achieves a $2.4 - 10.5\times$ speedup in the runtime and $10.9 - 14.8\times$ reductions in the communication cost) and the unbalanced PCSI [6] are described theoretically and do not provide source codes.

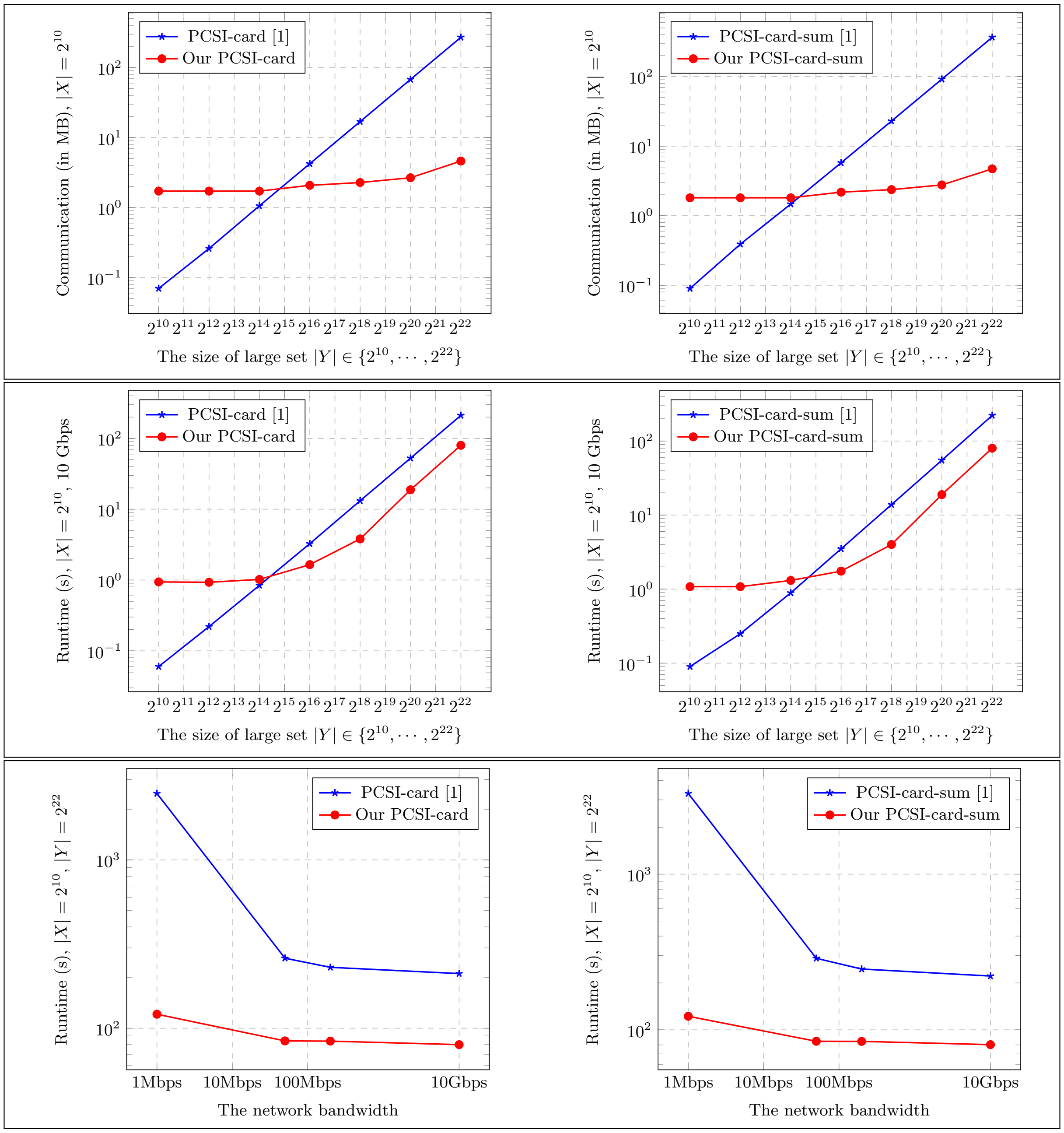

Fig. 14: Comparisons of communication (in MB) and runtime (in seconds) between [1] and our protocols. Both x-axis and y-axis are in log scale. The first two figures show the communication cost increases as the size of the large set increases. The middle two figures show the runtime increases as the size of the large set increases. The last two figures show the runtime decreases as the bandwidth increases.

7.1 Benchmark Comparison

In this section, we compare PCSI-card and PCSI-card-sum with that of [1] in terms of runtime and communication, and the results are reported in Table 2,3 and Figure 3. We stress that all reported costs are collected in the same environment.

Communication comparison. Our PCSI protocols achieve the lowest communication in the unbalanced setting. As shown in Figure 14, the larger difference between the two set sizes, the better our protocols perform. As shown in Table 2, for set sizes ($|X| = 2^{10}, |Y| = 2^{22}$), the communication of our PCSI-card requires 4.62MB, which is about 58× lower than PCSI-card [1] requiring 270.41MB; the communication of our PCSI-card-sum requires 4.71MB, which is about 77× lower than PCSI-card-sum [1] requiring 366.42MB.
Runtime comparison. As shown in Figure 14, the larger difference between the two set sizes, the better our protocols perform. As shown in Table 3, for set sizes ($|X| = 2^{10}$, $|Y| = 2^{22}$) with $T = 1$ in LAN setting, the runtime of our PCSI-card requires 79.9 seconds, while PCSI-card [1] requires 211.2 seconds, about 2.6× improvement; the runtime of our PCSI-card-sum requires 80.03 seconds, while PCSI-card-sum [1] requires 221.6 seconds, about 2.77× improvement. The performance of our protocols improves significantly in the case of low bandwidth. For set sizes ($|X| = 2^{10}$, $|Y| = 2^{22}$) with $T = 1$ in 1Mbps bandwidth, our PCSI-card requires 121.09 seconds, while PCSI-card [1] requires 2480.26 seconds, about 20.5× improvement; our PCSI-card-sum requires 122.18 seconds, while PCSI-card-sum [1] requires 3296.22 seconds, about 26.98× improvement.

7.2 Scalability and Parallelizability

We take PCSI-card as an example to demonstrate the scalability and parallelizability of our protocols in Table 4. PCSI-card scales well in the unbalanced setting. In single-thread, for set sizes $(2^{10}, 2^{10})$, $(2^{10}, 2^{16})$, and $(2^{10}, 2^{22})$, PCSI-card requires 0.94, 1.65 and 79.9 seconds, respectively. Benchmarking our implementation in the WAN setting, our PCSI-card also scales well due to the low communication cost: for $(2^{10}, 2^{22})$, PCSI-card requires 4.62MB of communication which is about 58× lower than PCSI-card [1]. Our PCSI-card is amenable to parallelization. Specifically, on the side of the large set, the server computes all polynomials and ciphertexts in parallel. For example, when increasing $T = 1$ to $T = 8$, our protocol shows a factor of 3.42× improvement as the running time reduces from 79.9 seconds to 23.3 seconds for set sizes $(2^{10}, 2^{22})$. We present the ratio between the runtime of $T = 1$ and $T = 8$ in the last row of Table 4. PCSI-card yields a better speedup when the large set becomes larger. Since for $n = 2^{10}$, the protocol achieves a moderate speedup of about 1.12, while considering $n = 2^{22}$, the speedup of about 3.42 is obtained at 8 threads.

8 Conclusion

This work formalizes shared characteristic and its labeled variety called shared characteristic with labels, from which we propose the frameworks of private computation on (labeled) set intersection (PCSI/PCLSI) protocols. By instantiating our frameworks in the unbalanced setting, we obtain a series of efficient PCSI/PCLSI protocols whose communication is linear in the size of the small set, and logarithmic in the large set. Our protocols are particularly attractive for private set operations in unbalanced scenarios, such as Client-Server cases (the input sets of two parties differ a lot in size), where they achieve very low communication costs: about 4.62MB (4.71MB) to compute intersection cardinality (cardinality-sum) for a set of one thousand items with a set of four million items, which is significantly lower than that of the state-of-the-art protocols.

References

1. Chen, Y., Zhang, M., Zhang, C., Dong, M.: Private set operations from multi-query reverse private membership test. IACR Cryptol. ePrint Arch. p. 652 (2022)
2. Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sum-with-cardinality. In: European Symposium on Security and Privacy (2020)
3. Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: CRYPTO (2020)
4. Buddhavarapu, P., Knox, A., Mohassel, P., Sengupta, S., Taubeneck, E., Vlaskin, V.: Private matching for compute. IACR Cryptol. ePrint Arch. p. 599 (2020)
5. Huberman, B.A., Franklin, M.K., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Conference on Electronic Commerce. pp. 78–86 (1999)

6. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: CCS. pp. 1223–1237 (2018)
7. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Public-Key Cryptography (2021)
8. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS. pp. 818–829 (2016)
9. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse OT extension. In: CRYPTO. pp. 401–431 (2019)
10. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: CRYPTO. pp. 34–63 (2020)
11. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: CRYPTO (2021)
12. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. In: CCS. pp. 2505–2517 (2022)
13. Meadows, C.A.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: IEEE Symposium on Security and Privacy. pp. 134–137 (1986)
14. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS. pp. 1243–1255 (2017)
15. 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: CCS. pp. 1135–1150 (2021)
16. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
17. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: USENIX Security Symposium (2015)
18. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: EUROCRYPT (2018)
19. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: EUROCRYPT (2019)
20. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: EUROCRYPT (2013)
21. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Proceedings of the 23rd USENIX Security Symposium. pp. 797–812 (2014)
22. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge university press (1995)
23. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: Algorithms - ESA 2001. pp. 121–133 (2001)
24. Devroye, L., Morin, P.: Cuckoo hashing: Further analysis. Inf. Process. Lett. 86(4), 215–219 (2003)
25. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. In: STACS 2003. pp. 271–282 (2003)
26. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. pp. 7:1–7:35 (2018)
27. Bourse, F., Pino, R.D., Minelli, M., Wee, H.: FHE circuit privacy almost for free. In: CRYPTO (2016)
28. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012)
29. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. p. 187 (2005)
30. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO (2003)
31. Tu, B., Chen, Y., Liu, Q., Zhang, C.: Fast unbalanced private set union from fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 653 (2022)
32. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346 (2017)