Fast Unbalanced Private Set Union from Fully Homomorphic Encryption

Binbin Tu[1], Yu Chen[1], Qi Liu[1], and Cong Zhang[2,3]

[1] School of Cyber Science and Technology, Shandong University
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
[3] School of Cyber Security, University of Chinese Academy of Sciences
{tubinbin,liuqicst}@mail.sdu.edu.cn;yuchen@sdu.edu.cn;zhangcong@iie.ac.cn

Abstract. Private set union (PSU) allows two parties to compute the union of their sets without revealing anything else. It has found numerous applications in practice. Recently, some computationally efficient PSU protocols have been designed for the balanced case, but a limitation with these protocols is the communication complexity, which scales (super)-linearly with the size of the larger set. This is of particular concern when performing PSU in the unbalanced case, where one party is a constrained device holding a small set, and another is a large service provider holding a large set.

In this work, we propose a generic construction of unbalanced PSU from leveled fully homomorphic encryption (FHE) and a newly introduced protocol called permuted matrix Private EQuality Test (pm-PEQT). By instantiating the generic construction, we obtain two secure and fast unbalanced PSU protocols, whose communication complexity is linear in the size of the smaller set, and logarithmic in the larger set.

We implement our protocols. Experiments show that our protocols are more efficient than all previous protocols in the unbalanced case. Especially, the larger difference between the size of two sets, the better our protocols perform. For input sets of size 2^{10} and 2^{19} with 128-bit length items, our PSU takes 2.242 MB of communication to compute the union. Compared with the state-of-the-art PSU proposed by Jia et al. (Usenix Security 2022), there are 300× reduction in communication and roughly 30 - 120× reduction in computational overhead in WAN/LAN settings.

1 Introduction

PSU is a cryptographic protocol that allows two parties, a sender and a receiver with respective input sets X and Y, to compute the union $X \cup Y$, without revealing anything else. It has become considerably efficient and has been deployed in practice, such as cyber risk assessment [29,22,28], privacy-preserving data aggregation [4], and private ID [20] etc. However, most PSU [26,19,1,13,28,20] are designed in the balanced case. These protocols typically perform only marginally better when one of the sets is much smaller than the other. In particular, their communication cost scales at least linearly with the size of the larger set. In most real world applications, the sender's set might be much smaller than the receiver's, such as the sender (client) might be a mobile device with limited battery, computing power, and storage, whereas the receiver (server) is a high-end computing device. Moreover, the bandwidth between two parties might be limited. Most existing PSU protocols are not very efficient in dealing with the above unbalanced case.

Over the last decade, there has been a significant amount of work on private set intersection (PSI) including both balanced [32,34,11,16,5,21,41,37,36,15,23,27] and unbalanced case [9,35,8,12,40], but little attention has been paid on PSU, especially in unbalanced case. Recently, Jia et al. [25] propose an unbalanced PSU*[4] with shuffling technique, but their PSU* suffers the following three drawbacks. First, their PSU* is not fully secure, since it *leaks the information of the intersection size* to the sender. Such information leakage could be critical for PSU[5]. Second, the communication complexity of their PSU* is *linearly with the size of the larger set*. Finally, the communication copmplexty of their protocol is heavily dependent on the item

[4] In this paper, we use PSU* to indicate a PSU protocol with information leakage.
[5] Consider the following extreme case, the sender can get the intersection item if it inputs a one-element set.

length. Chen et al. [10] show how to tweak FHE-based PSI [9] to an unbalanced PSU protocol. As noted by the authors, their PSU protocol only serves as a proof of concept since it reveals intersection size to the sender, and straightforward applying the optimization tricks due to [9] will compromise the semi-honest security of the receiver. They left the fully secure and efficient FHE-based PSU protocol in the unbalanced setting as a challenging problem.

Motivated by the above discussions, we ask the following question:

Is it possible to design a secure and fast unbalanced PSU protocol which has a communication overhead linear in the smaller set and logarithmic in the larger set?

1.1 Contributions

In this paper, we give an affirmative answer to the above question. We summarize our contributions as follows:

1. We first propose a basic unbalanced PSU protocol based on leveled FHE. Then, we use an array of optimization techniques following [9,8,12] to optimize the basic protocol, while the optimization might leak some information of the intersection.
2. We introduce a new cryptographic protocol named permuted matrix private equality test (pm-PEQT) to avoid the information leakage. Then, we give two constructions of pm-PEQT. The first is based on $\mathsf{Permute} + \mathsf{Share}$ and multi-point oblivious pseudorandom function (mp-OPRF). The second is based on the decisional Diffie-Hellman (DDH) assumption.
3. we present a generic construction of unbalanced PSU in the semi-honest model from leveled FHE and pm-PEQT. By instantiating the generic construction, we obtain two *secure* and *fast* unbalanced PSU protocols which have communication complexity *linear* in the size of the smaller set, and *logarithmic* in the larger set. In particular, except for the OT phase, the communication of our PSU protocols are independent of the item length. Our protocols are particularly powerful when the set size of one party is much larger than that of the other.
4. We implement our PSU protocols. Experiments show that our protocols are more efficient than all previous protocols in the unbalanced case. For unbalanced sets size ($|X| = 2^{10}$, $|Y| = 2^{19}$) with 128-bit length items, our PSU protocol takes 2.242 MB of communication and 12 seconds of computation to compute the union with a single thread in LAN settings. Compared with the state-of-the-art PSU [25], there are roughly 300× reduction in communication and 30× reduction in computational overhead. In particular, the performance of our PSU protocols improve significantly in the case of low bandwidth. Our PSU requires 7.79 seconds which is about 120× faster than PSU [25] in 10 Mbps bandwidth.

1.2 Related Works

We revisit recent PSU protocols [28,20,25,42] with good efficiency. Table 1 provides a brief comparison of our protocols to the prior highest-performing PSU protocols. We report in detail the performance results and comparisons in Section 7.

Kolesnikov et al. [28] propose a PSU protocol based on the reverse private membership test (RPMT). In RPMT, the sender $\mathcal{S}$ with input x interacts with the receiver $\mathcal{R}$ holding a set Y, and $\mathcal{R}$ can learn a bit indicating whether $x \in Y$, while $\mathcal{S}$ learns nothing. Then, $\mathcal{R}$ runs OT with $\mathcal{S}$ to obtain $\{x\} \cup Y$. For $n = |X| = |Y|$, the protocol runs RPMT n times independently and requires $O(n^2)$ communication and $O(n^2 \log^2 n)$ computation. By using the bucketing technique, two parties hash their sets in m bins and each bin consists of ρ items. A large (n, n)-PSU[6] is divided into m small (ρ, ρ)-PSU. The complexity is reduced to $O(n \log n)$ communication and $O(n \log n \log \log n)$ computation. However, [25] points out that the bucketing technique leaks the information to $\mathcal{R}$. More precisely, $\mathcal{R}$ learns that some subsets (size ρ) hold the intersection items with high probability.

[6] In this paper, we use (m, n)-PSU to indicate a PSU protocol where the sender's set size is m and the receiver's set size is n.

Protocols	Communication	Computation	Security
PSU* [28]	$O(n \log n)$	$O(n \log n)$ sym	Leaky
PSU [20]	$O(n \log n)$	$O(n \log n)$ sym	Full
PSU [42]	$O(n)$	$O(n)$ sym/pub	Full
PSU [25]	$O(n \log n)$	$O(n \log n)$ sym	Full
PSU* [25]	$O(n + m \log m)$	$O(n)$ sym	Leaky
Our PSU	$O(m \log n)$	$O(n)$ pub	Full

Table 1: Comparisons of PSU in the semi-honest setting. n denotes both sets size in balanced case. n and m denote the size of the large set and the small set, respectively, in unbalanced case. Pub: public-key operations; sym: symmetric cryptographic operations. We ignore the pub-key cost of κ base OTs where κ is computational security parameter. PSU* [28] leaks the information to the receiver ($\mathcal{R}$ learns some subsets have the intersection items). PSU* [25] leaks the information to the sender ($\mathcal{S}$ learns the intersection size).

Garimella et al. [20] give a PSU protocol based on permuted characteristic functionality which in turn can be built from oblivious switching. Simply speaking, the sender $\mathcal{S}$ holding a set X interacts with the receiver $\mathcal{R}$ holding a set Y. As a result, $\mathcal{S}$ gets a random permutation π and $\mathcal{R}$ gets a vector $\mathbf{e} \in \{0,1\}^n$, where if $e_i = 1$, $x_{\pi(i)} \in Y$, else $x_{\pi(i)} \notin Y$. Then, $\mathcal{R}$ runs OT protocol with $\mathcal{S}$ to obtain the set union. Their protocol requires $O(n \log n)$ communication and $O(n \log n)$ computation.

Zhang et al. [42] recently give a generic framework of PSU based on the multi-query reverse private membership test (mq-RPMT). In mq-RPMT, the sender $\mathcal{S}$ holding a set X interacts with the receiver $\mathcal{R}$ holding a set Y. As a result, $\mathcal{S}$ gets nothing and $\mathcal{R}$ gets $\mathbf{b} \in \{0,1\}^n$, satisfying $b_i = 1$ if and only if $x_i \in Y$. Then, two parties runs OT protocol to let $\mathcal{R}$ get the set union. To construct mq-RPMT, they combine the oblivious key-value store (OKVS) and vector decryption-then-matching (VODM). By instantiating OKVS and VODM, they obtain two concrete mq-RPMT. The first is based on symmetric-key encryption and general 2PC. The second is based on re-randomizable public-key encryption. Both constructions achieve linear computation $O(n)$ and communication $O(n)$.

Jia et al. [25] propose a PSU with the shuffling technique. Simply speaking, the receiver $\mathcal{R}$ hashes a set Y into Y_c by Cuckoo hash and the sender $\mathcal{S}$ hashes a set X by simple hash. $\mathcal{R}$ shuffles Y_c by a permutation π chosen by $\mathcal{S}$. $\mathcal{S}$ and $\mathcal{R}$ get shuffled shares $\{s_{\pi(i)}\}$ and $\{s'_{\pi(i)}\}$, where $Y_c[\pi(i)] = s_{\pi(i)} \oplus s'_{\pi(i)}$, respectively. Two parties run mp-OPRF to compute all PRF values and $\mathcal{S}$ sends its PRF values to $\mathcal{R}$. $\mathcal{R}$ tests which items belong to the union and runs OT with $\mathcal{S}$ to get the union. Their PSU requires $O(n \log n)$ communication and $O(n \log n)$ computation. They also consider the unbalanced case and give an unbalanced PSU which requires $O(n + m \log m)$ communication and $O(n)$ computation.

2 Overview of Our Techniques

We provide the high-level intuition for our unbalanced PSU protocol. First, we propose a basic PSU protocol based on leveled FHE. Our basic protocol is easy to understand, but it is not efficient due to the depth of homomorphic circuits is deep. Then, we try to improve the basic PSU by applying optimization techniques following [9,8,12] to reduce the depth of homomorphic circuits. However, straightforward application leaks the information of the intersection. To remedy the leakage, we introduce a new cryptographic protocol called permuted matrix private equality test (pm-PEQT). Finally, we manage to give a generic construction of fully secure unbalanced PSU from leveled FHE and pm-PEQT. By instantiating the generic construction, we obtain a secure and fast unbalanced PSU protocol. We describe the ideal functionality of PSU in Figure 1.

2.1 Our Basic PSU Protocol

Our starting point is the FHE-based basic PSI protocol [9] and we review the protocol as follows.

Parameters: Two parties: the sender $\mathcal{S}$ with set X and receiver $\mathcal{R}$ with set Y.
Functionality:

1. Wait for an input $X = \{x_1, x_2, \cdots, x_m\} \subset \{0,1\}^*$ from $\mathcal{S}$, and an input $Y = \{y_1, y_2, \cdots, y_n\} \subset \{0,1\}^*$ from $\mathcal{R}$.
2. Give output $X \cup Y$ to $\mathcal{R}$.

Fig. 1: Ideal functionality $\mathcal{F}_{\text{PSU}}^{m,n}$ for private set union

Basic PSI protocol. Chen et al. [9] first give a basic PSI protocol. Informally, the receiver $\mathcal{R}$ encrypts its items y_i, $i \in [|Y|]$ and sends $c_i \leftarrow \text{FHE.}\mathsf{Enc}(y_i)$ to the sender $\mathcal{S}$; $\mathcal{S}$ chooses random non-zero plaintexts r_i and homomorphically computes $c_i' \leftarrow \text{FHE.}\mathsf{Enc}(r_i \cdot f(y_i))$, where the polynomial $f(x) = \Pi_{x_i \in X}(x - x_i)$, and then returns c_i' to $\mathcal{R}$; $\mathcal{R}$ decrypts c_i': if $r_i f(y_i) = 0$, it knows $y_i \in X \cap Y$, else, it gets a random item. The protocol requires communication linear in the smaller set, but it has high computational costs and deep homomorphic circuits, because the degree of $f(x)$ is related to the large set size.

Basic PSU protocol. The functionality adjustment (PSI $\rightarrow$ PSU) doesn't seem to be straightforward, since the randomized product $rf(y) = 0$ leaks the information of the intersection to the receiver. The main challenge is to find a new randomization method that hides the information of the intersection and admits to check which items belong to the union. We solve the problem by adding a random value r to randomize the polynomial value. In this way, the randomized value $r + f(y)$ leaks nothing to $\mathcal{R}$. Meanwhile, $\mathcal{R}$ sends the result $r + f(y)$ to $\mathcal{S}$ and $\mathcal{S}$ can check whether the item y belongs to the union by verifying $r \stackrel{?}{=} r + f(y)$. In order to let the receiver output the results, we consider the opposite case of [9], in which the sender holds a small set and the receiver holds a large set.

We start with a special case. Suppose that the sender $\mathcal{S}$ has only one item x and the receiver $\mathcal{R}$ holding a large set Y gets the resulting union $\{x\} \cup Y$. We show our basic unbalanced PSU based on leveled FHE as follows: $\mathcal{S}$ uses its public key to encrypt x and sends $c = \text{FHE.}\mathsf{Enc}(x)$ to $\mathcal{R}$; $\mathcal{R}$ chooses random non-zero value r, and homomorphically computes $c' = \text{FHE.}\mathsf{Enc}(r + \Pi_{y_i \in Y}(x - y_i))$, and returns the new ciphertext to $\mathcal{S}$; $\mathcal{S}$ decrypts c' and gets the plaintext $r' = r + \Pi_{y_i \in Y}(x - y_i)$, then it returns r' back to $\mathcal{R}$; $\mathcal{R}$ checks $r' \stackrel{?}{=} r$, if $r' = r$, it sets $b = 0$ indicating $x \in X \cap Y$, else $b = 1$ indicating $x \notin X \cap Y$. Finally, $\mathcal{R}$ invokes the OT protocol with $\mathcal{S}$ to obtain the union $\{x\} \cup Y$.

The key different step between our basic PSU and the basic PSI [9] is using different randomization methods. We compute the sum of a random value r and the polynomial value $f(x)$, where $f(x) = \Pi_{y_i \in Y}(x - y_i)$. $\mathcal{S}$ obtains the random plaintext $r' = r + f(x)$ which leaks nothing and $\mathcal{R}$ getting $r' = r + f(x)$ can checks $r' = r$ or not. If $r' = r =$, $x \in Y$, else $x \notin Y$. This will leak some information of $x \notin Y$, but this leakage does not cause any harm to the PSU, since the PSU protocol releases that value at last.

2.2 Optimized PSU with Leakage

We first review optimized unbalanced PSI as follows. Chen et al. [9] use an array of optimization techniques such as hashing, batching, windowing, partitioning, modulus switching, etc, to optimize their basic protocol and obtain a fast unbalanced PSI. Informally, the receiver $\mathcal{R}$ inserts a small set Y into Y_c by Cuckoo hash. The sender $\mathcal{S}$ inserts a large set X into X_b by simple hash, where the i-th bin indicates as $X_b[i]$ and each bin consists of B items. $\mathcal{S}$ partitions each bin $X_b[i]$ into α subsets and each subset consists of $B' = B/\alpha$ items. Therefore, the large (n, m)-PSI is divided into many small $(B', 1)$-PSI. For each small PSI, $\mathcal{S}$ encodes each subset (B' items) into a polynomial and randomizes it by multiplying a random value, then it homomorphically computes and sends new ciphertexts to $\mathcal{R}$. $\mathcal{R}$ decrypts the ciphertexts and gets the set intersection. Since the degree of the polynomial is related to the small subset size B', each small PSI has a low homomorphic circuit. We review the basic PSI protocol and its optimizations in Figure 2.

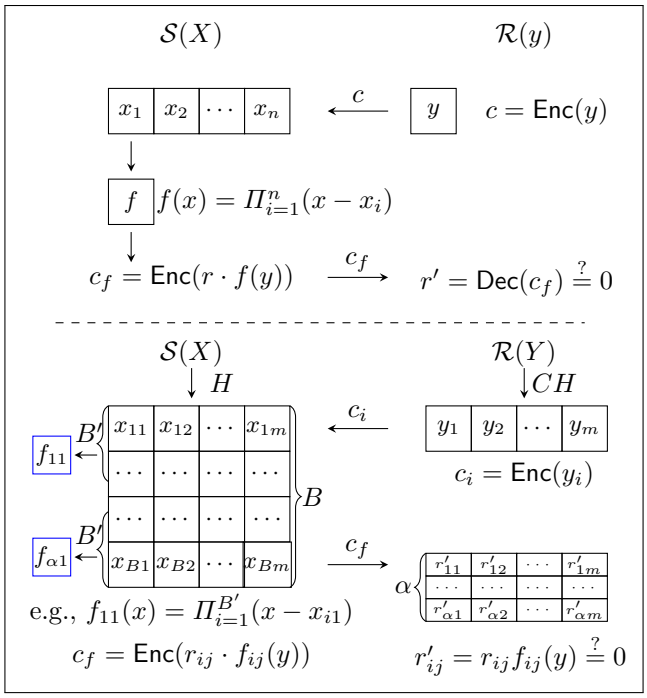

Fig. 2: The basic PSI and its optimizations [9]

It is tempting to use the same optimization techniques [9] to improve our basic PSU. Roughly, the sender $\mathcal{S}$ hashes the small set X into X_c by Cuckoo hash, and the receiver $\mathcal{R}$ hashes the large set Y into Y_b and then partitions each bin $Y_b[i]$ into α subsets and each subset consists of $B' = B/\alpha$ items. The large (m, n)-PSU is divided into many small $(1, B')$-PSU. For each small PSU, $\mathcal{R}$ encodes each subset (B' items) into a polynomial and randomizes it by adding a random value, then it homomorphically computes and sends new ciphertexts to $\mathcal{S}$. $\mathcal{S}$ decrypts the ciphertexts and sends the plaintexts back. $\mathcal{R}$ checks which items belong to the set union, and invokes OT with $\mathcal{S}$ to get them. We show our basic PSU and its optimizations (omit OT) in Figure 3.

We emphasize that, unlike PSI [9], the optimization techniques for PSI is not suitable for our PSU. This is because a large PSI can be divided into many small PSI, and the receiver can combine all small set intersections into the output securely. However, if we divide a large (m, n)-PSU into many small $(1, B')$-PSU directly, this causes information leakage about the intersection. We show the comparison of PSI [9] and our optimized PSU (omit OT) with leakage in Figure 4.

Note that in the fully secure (n, m)-PSU, from the view of $\mathcal{R}$, any item in the set Y could be an item in $X \cap Y$. However, in our optimized PSU*, $\mathcal{R}$ learns some subsets with size B' have the item in $X \cap Y$. Moreover, if $\mathcal{S}$ returns its decrypted results r' to $\mathcal{R}$ directly. $\mathcal{R}$ can check which items of X belong to the set union. This leaks the information of $X \cap Y$. Because there are α subsets with size B' in one bin, if $f(x) = 0$ in one subset, $\mathcal{R}$ gets $f'(x) \neq 0$ in other subsets, which causes $\mathcal{R}$ could compute the intersection items with sufficient polynomial values. For example, in Figure 4 (right), in the first column, if $r_{11} = r'_{11}$, this means $x_1 \in Y$[7] and $x_1 \in \{y_{11}, \cdots, y_{B'1}\}$, but $x_1 \notin \{y_{(B'+1)1}, \cdots, y_{B1}\}$. $\mathcal{R}$ gets the rest nonzero polynomial values $f_{21}(x_1), \cdots, f_{\alpha 1}(x_1)$ and it could compute x_1 from them.

Based on above analysis, the main challenge is how to optimize our basic PSU without causing information leakage. More precisely, we need to overcome the following two difficulties:

– The receiver is able to check $r_{ij} \stackrel{?}{=} r_{ij} + f_{ij}(x)$ at all positions without knowing $r_{ij} + f_{ij}(x)$.

[7] In j-th column, as long as there is a position i, such that $r_{ij} = r'_{ij}$, $x_j \in Y$. Meanwhile, at most one position is equal in each column.

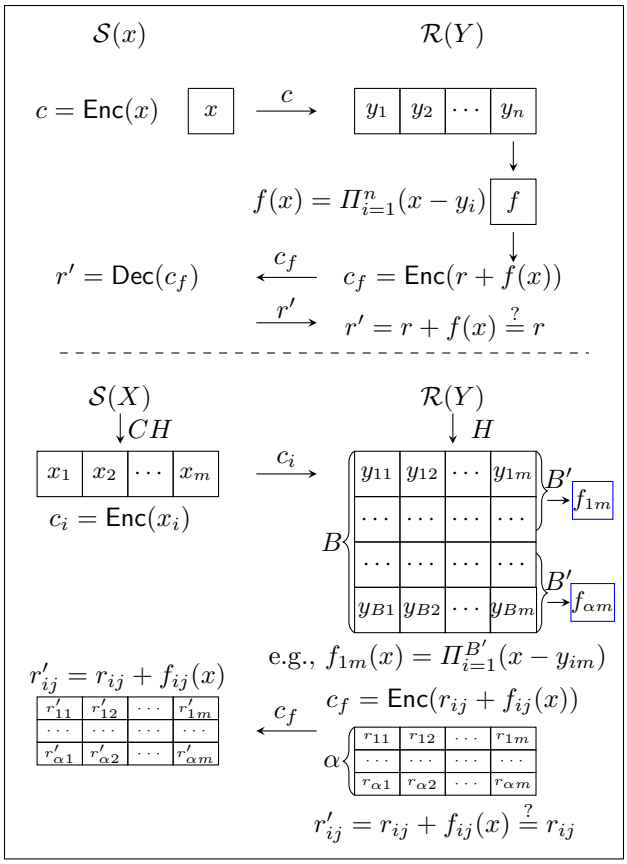

Fig. 3: The basic PSU (omit OT) and its optimizations

– The receiver is able to check $r_{ij} = r_{ij} + f_{ij}(x)$ in same positions without knowing the positions i, j.

We address the difficulties by introducing a cryptographic protocol named permuted matrix private equality test (pm-PEQT) that enables the receiver check whether the values in permuted positions are equal without knowing the values and permutation.

2.3 Permuted Matrix Private Equality Test

We introduce a new cryptographic protocol named permuted matrix private equality test (pm-PEQT) which can be seen as an extension of private equality test (PEQT). In the PEQT, a receiver who has an input string x interacts with a sender holding an input string y, and the result is that the receiver learns a bit indicating whether $x = y$ and nothing else, whereas the sender learns nothing. In our pm-PEQT, the sender holding a matrix $\mathbf{R}'_{\alpha \times m}$ and a matrix permutation $\pi = (\pi_c, \pi_r)$ interacts with a receiver holding a matrix $\mathbf{R}_{\alpha \times m}$. As a result, the receiver learns (only) the bit matrix $\mathbf{B}_{\alpha \times m}$ indicating that if $b_{ij} = 1$, $r_{\pi(ij)} = r'_{\pi(ij)}$, else, $r_{\pi(ij)} \neq r'_{\pi(ij)}$, $i \in [\alpha], j \in [m]$, while the sender learns nothing about $\mathbf{R}$. Compared with PEQT, pm-PEQT admits a matrix private equality test with *positions permutation*. We show the ideal functionality of pm-PEQT in Figure 5.

Constructions of pm-PEQT. pm-PEQT can not be easily built from PEQT by running many PEQT instances in parallel. This is because it is difficult to shuffle the receiver's items without knowing the permutation of the sender. We give two constructions of pm-PEQT as follows.

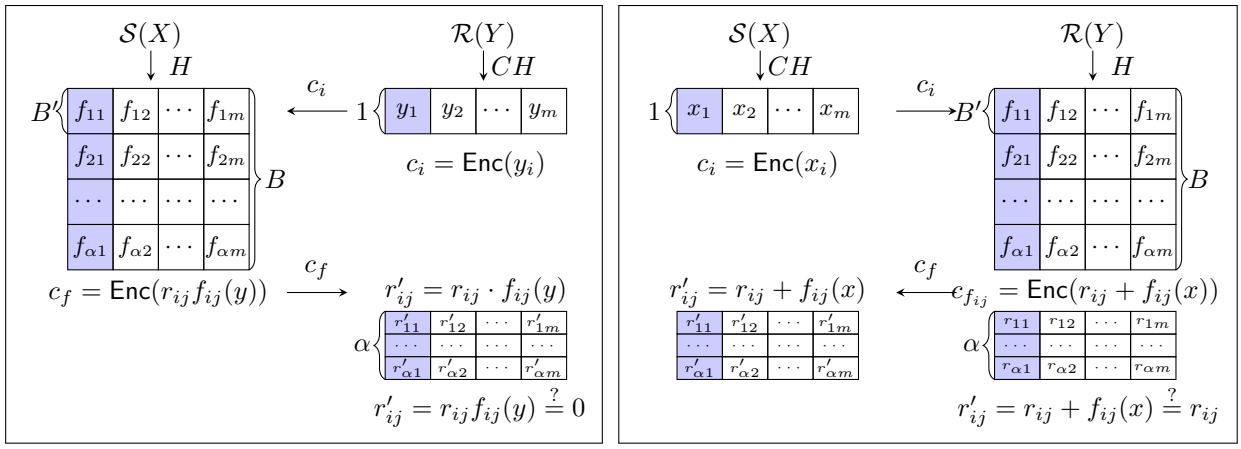

Fig. 4: Comparison of PSI [9] (left) and optimized PSU (omit OT) with leakage (right)

Parameters: Two parties: The receiver with a matrix $\mathbf{R}_{\alpha\times m}$. The sender with a matrix $\mathbf{R}'_{\alpha\times m}$ and a matrix permutation $\pi = (\pi_c, \pi_r)$, where π_c (over $[m]$) is a column permutation and π_r (over $[\alpha]$) is a row permutation.
Functionality:

1. Wait for an input $\mathbf{R}' = [r_{ij}]$, $i \in [\alpha], j \in [m]$ and a permutation $\pi = (\pi_c, \pi_r)$ from the sender, and an input $\mathbf{R} = [r_{ij}]$, $i \in [\alpha], j \in [m]$ from the receiver.
2. Give the bit matrix $\mathbf{B}_{\alpha\times m}$ to the receiver, if $r_{\pi(ij)} = r'_{\pi(ij)}$, $b_{ij} = 1$, else, $b_{ij} = 0$, for all $i \in [\alpha], j \in [m]$.

Fig. 5: Permuted matrix private equality test $\mathcal{F}_{\text{pm-PEQT}}$

The first construction is based on Permute + Share [20,25] and mp-OPRF [33,7]. Informally, $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal Permute + Share functionality $\mathcal{F}_{\text{PS}}$ twice: First, both parties permute and share the columns of $\mathbf{R}$. $\mathcal{R}$ inputs each column of $\mathbf{R}$ and $\mathcal{S}$ inputs a permutation π_c over $[m]$. As a result, $\mathcal{R}$ gets shuffled share $\mathbf{S}_{\pi_c} = [s_{\pi_c(ij)}]$ and $\mathcal{S}$ gets $\mathbf{S}'_{\pi_c} = [s'_{\pi_c(ij)}]$, where $s_{\pi_c(ij)} \oplus s'_{\pi_c(ij)} = r_{\pi_c(ij)}$. Then both parties permute and share the rows of $\mathbf{S}_{\pi_c}$. $\mathcal{R}$ inputs each row of $\mathbf{S}_{\pi_c}$ and $\mathcal{S}$ inputs a permutation π_r over $[\alpha]$. As a result, $\mathcal{R}$ gets $\mathbf{S}_{\pi_r} = [s_{\pi_r(ij)}]$ and $\mathcal{S}$ gets $\mathbf{S}'_{\pi_r} = [s'_{\pi_r(ij)}]$, where $s_{\pi_r(ij)} \oplus s'_{\pi_r(ij)} = s_{\pi_c(ij)}$. $\mathcal{R}$ gets the shuffled matrix shares $\mathbf{S}_\pi = \mathbf{S}_{\pi_r}$ and $\mathcal{S}$ gets the shuffled matrix shares $\mathbf{S}'_\pi = \pi_r(\mathbf{S}'_{\pi_c}) \oplus \mathbf{S}'_{\pi_r}$, where $s_{\pi(ij)} \oplus s'_{\pi(ij)} = r_{\pi(ij)}$, $i \in [\alpha], j \in [m]$. Then, both parties invoke mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$. $\mathcal{R}$ inputs shuffled shares $\mathbf{S}_\pi$ and obtains the outputs $F_k(s_{\pi(ij)})$, $i \in [\alpha], j \in [m]$, and $\mathcal{S}$ gets the key k of a PRF. Furthermore, $\mathcal{S}$ permutes the matrix $\mathbf{R}'$ by $\pi = (\pi_c, \pi_r)$ and gets $\mathbf{R}'_\pi = [r'_{\pi(ij)}]$, and then computes all PRF values $F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, $i \in [\alpha], j \in [m]$ and sends them to $\mathcal{R}$. Finally, $\mathcal{R}$ sets $b_{ij} = 1$, if $F_k(s_{\pi(ij)}) = F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, else, sets $b_{ij} = 0$, and gets a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha], j \in [m]$. The Permute + Share [20,25] and mp-OPRF [33,7] are fast cryptographic tools. The communication complexity of our pm-PEQT based on Permute + Share and mp-OPRF is $O(\alpha m \log \alpha m)$.

The second construction is based on the DDH assumption [2]. Let $\mathbb{G}$ be a cyclic group with order q, where the DDH problem is hard. Informally, $\mathcal{R}$ and $\mathcal{S}$ choose random value a, $b \leftarrow \mathbb{Z}_q$ and compute $v_{ij} = H(r_{ij})^a$, $v'_{ij} = H(r'_{ij})^b$ for $i \in [\alpha]$, $j \in [m]$, respectively, where $v = H(\cdot)$ are group elements in $\mathbb{G}$ output by hash functions H. $\mathcal{R}$ sends v_{ij} to $\mathcal{S}$. Then $\mathcal{S}$ computes $v''_{ij} = (v_{ij})^b$ and shuffles v''_{ij} and v'_{ij} by the permutation $\pi = (\pi_c, \pi_r)$ and gets $v''_{\pi(ij)} = \pi(v''_{ij})$, $v'_{\pi(ij)} = \pi(v'_{ij})$. $\mathcal{S}$ sends them to $\mathcal{R}$. Finally, $\mathcal{R}$ computes $b_{ij} = 1$, if $v''_{\pi(ij)} = v'^a_{\pi(ij)}$, else $b_{ij} = 0$, and gets a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha]$, $j \in [m]$. The communication complexity of our DDH-based pm-PEQT is $O(\alpha m)$.

2.4 Our Full PSU Protocol

We provide the high-level technical overview for our generic construction of PSU in Figure 6 and the details are as follows.

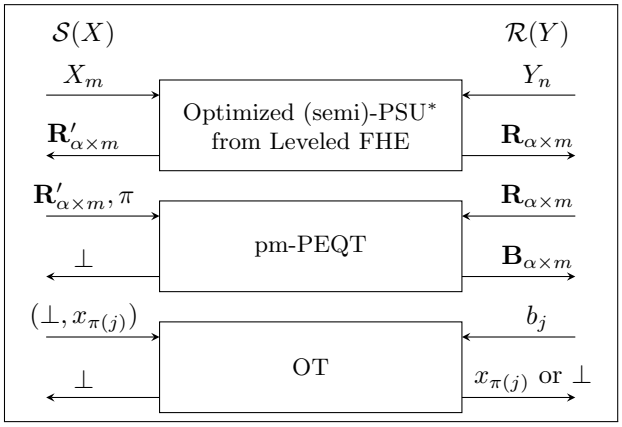

Fig. 6: Core design idea of our full PSU protocol

First, we construct a semi-finished FHE-based optimized PSU* in which the sender $\mathcal{S}$ dose not send its decrypted results $\mathbf{R}'_{\alpha\times m}$ to the receiver $\mathcal{R}$. $\mathcal{S}$ holding a set X interacts with $\mathcal{R}$ holding a set Y. The result is $\mathcal{R}$ outputs a matrix $\mathbf{R}_{\alpha\times m} = [r_{ij}]$, and $\mathcal{S}$ outputs a matrix $\mathbf{R}'_{\alpha\times m} = [r'_{ij}]$, where α denotes the number of partitions, m denotes the number of bins, r_{ij} denotes the random value used to hide each polynomial and r'_{ij} denotes the decrypted results. Note that for $i \in [\alpha]$ in same column, if all $r'_{ij} \neq r_{ij}$, $x_j \notin Y$, else, $x_j \in Y$.

Then, by using pm-PEQT, $\mathcal{S}$ inputs $\mathbf{R}'$ and a permutation $\pi = (\pi_c, \pi_r)$ and $\mathcal{R}$ inputs $\mathbf{R}$. As a result, $\mathcal{R}$ gets a bit matrix $\mathbf{B} = [b_{ij}]$, where if $b_{ij} = 1$, $i \in [\alpha], j \in [m]$, $r_{\pi(ij)} = r'_{\pi(ij)}$, else $r_{\pi(ij)} \neq r'_{\pi(ij)}$. $\mathcal{R}$ computes a bit vector $\mathbf{b} = [b_j]$, $j \in [m]$, for all $i \in [\alpha]$, if $b_{ij} = 0$, sets $b_j = 1$, else, sets $b_j = 0$. $\mathcal{S}$ permutes the Cuckoo hash table X_c by π_c and gets $\pi_c(X_c) = [x_{\pi_c(1)}, x_{\pi_c(2)}, \cdots, x_{\pi_c(m)}]$. We note that if $b_j = 1$, $x_{\pi_c(j)} \notin Y$, else $x_{\pi_c(i)} \in Y$, $j \in [m]$.

Finally, by using OT protocol, $\mathcal{S}$ inputs $(\perp, x_{\pi(j)})$, $j \in [m]$, and $\mathcal{R}$ inputs b_j, $j \in [m]$. $\mathcal{R}$ gets $x_{\pi_c(j)}$, if $b_j = 1$, else, gets $\perp$. After that, $\mathcal{R}$ outputs the union $Y \cup \{x_{\pi_c(j)}\}$.

In this way, we complete our construction of fully secure and fast unbalanced PSU protocol.

3 Preliminaries

3.1 Notation

We denote the parties as receiver $\mathcal{R}$ and sender $\mathcal{S}$. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \cdots, n\}$. 1^λ denotes the string of λ ones. We use κ and λ to indicate the computational and statistical security parameters, respectively. If S is a set, $s \leftarrow S$ indicates sampling s of S at random. 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. For a column permutation π_c (or, row permutation π_r) on a matrix $\mathbf{S} = [s_{ij}]$, we write $\mathbf{S}_{\pi_c}$ (or, $\mathbf{S}_{\pi_r}$) to denote $\pi_c(\mathbf{S}) = [s_{\pi_c(ij)}]$ (or, $\pi_r(\mathbf{S}) = [s_{\pi_r(ij)}]$) be the permuted matrix, where $s_{\pi_c(ij)}$ (or, $s_{\pi_r(ij)}$) indicates the i-th row and j-th column element after the permutation.

3.2 Building Blocks

We briefly review the main cryptographic tools including Cuckoo hashing, leveled fully homomorphic encryption, oblivious transfer, multi-point oblivious PRF, and Permute + Share.

Cuckoo hashing. Cuckoo hashing [31,14,18,38] can be used to build dense hash tables by many hash functions. Following [9], 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 $2^{-\lambda}$.

Leveled fully homomorphic encryption. The leveled fully homomorphic encryption supports circuits of a certain bounded depth. Following [9,8,12], we use an array of optimization techniques of FHE, such as batching, windowing, partitioning, modulus switching, etc, and our protocols require the leveled FHE satisfies IND-CPA secure with circuit privacy. We refer the reader to [9,8,12,3] for more details. For the implementation, we use the homomorphic encryption library SEAL which implements the BFV scheme [17].

Oblivious transfer. Oblivious transfer [39] is a central cryptographic primitive in the area of secure computation. In the 1-out-of-2 OT, a sender with two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. The result is that the receiver learns x_b without learning anything about x_{1-b}, while the sender learns nothing about b. Ishai et al. [24] introduced the OT extension that allows for a large number of OT executions at the cost of computing a small number of public-key operations. We recall the 1-out-of-2 oblivious transfer functionality $\mathcal{F}_{\text{OT}}$ in Figure 7.

Parameters: Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
Functionality:

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

Fig. 7: 1-out-of-2 oblivious transfer functionality $\mathcal{F}_{\text{OT}}$

Multi-point oblivious pseudorandom function. An oblivious pseudorandom function (OPRF) allows the receiver to input x and learns the PRF value $F_k(x)$, where F is a PRF, and k is known to the sender. Pinkas et al. [33] propose multi-point OPRF (mp-OPRF) and realize efficient PSI protocols. Recently, Chase and Miao [7] propose a more efficient mp-OPRF based on oblivious transfer extension. In the mp-OPRF, the receiver inputs $\{x_1, x_2, \cdots, x_n\}$ and learns all PRF values $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_n)\}$, and the sender gets the key k. We recall the mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$ in Figure 8.

Parameters: A PRF F. Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$.
Functionality:

1. Wait for input $\{x_1, \cdots, x_n\}$ from $\mathcal{R}$.
2. Sample a random PRF seed k and give it to $\mathcal{S}$. Give $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_n)\}$ to $\mathcal{R}$.

Fig. 8: mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$

Permute + Share. We recall the $\mathsf{Permute} + \mathsf{Share}$ (PS) functionality $\mathcal{F}_{\text{PS}}$ defined by Chase et al. [6] in Figure 9. Roughly speaking, in the $\mathsf{Permute} + \mathsf{Share}$ protocol, P_0 inputs a set $X = \{x_1, \cdots, x_n\}$ of size n and P_1 chooses a permutation π on n items. The result is that P_0 learns the shuffled shares $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\}$ and P_1 learns the other shuffled shares $\{s'_{\pi(1)}, s'_{\pi(2)}, \cdots, s'_{\pi(n)}\}$, where $x_{\pi(i)} = s_{\pi(i)} \oplus s'_{\pi(i)}$, $i \in [n]$.

Parameters: Two parties: P_0 and P_1. Set size n for P_0;
Functionality:

1. Wait for input $X = \{x_1, \cdots, x_n\}$ from P_0, abort if $|X| \neq n$. Wait for input a permutation π from P_1, abort if π is not a permutation on n items.
2. Give output shuffled shares $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\}$ to P_0, and another shuffled shares $\{s'_{\pi(1)}, s'_{\pi(2)}, \cdots, s'_{\pi(n)}\}$ to P_1, where $x_{\pi(i)} = s_{\pi(i)} \oplus s'_{\pi(i)}$, $i \in [n]$.

Fig. 9: Permute + Share functionality $\mathcal{F}_{\text{PS}}$

4 The Basic PSU Protocol

We describe our basic PSU protocol in Figure 10 as a strawman protocol. In this protocol, if $r + f(x) \neq r$, the receiver can get $f(x)$ which leaks some information of $x \notin Y$, but this leakage does not cause any harm to the PSU, since the PSU protocol releases that value at last. We prove its semi-honest security in the following theorem.

Input: The sender $\mathcal{S}$ inputs set X of size $|X|$ and the receiver $\mathcal{R}$ inputs set Y of size $|Y|$.
Output: The receiver outputs $X \cup Y$. The sender outputs $\perp$.

1. [**Setup**] $\mathcal{S}$ generates a public-secret key pair for the scheme and keeps the secret key itself.
2. [**Set encryption**] $\mathcal{S}$ encrypts each item $x_i \in X$, $c_i = \text{FHE.Enc}(x_i), i \in [|X|]$ and sends $(c_1, \cdots, c_{|X|})$ to $\mathcal{R}$.
3. [**Computation**] For each c_i, $\mathcal{R}$
   (a) samples a random non-zero plaintext item r_i;
   (b) homomorphically computes $c'_i = \text{FHE.Enc}\ (f(x_i) + r_i)$, where $f(x) = \Pi_{y_i \in Y}(x - y_i)$.
   (c) sends $c'_i, i \in [|X|]$ to $\mathcal{S}$.
4. [**Decryption**] $\mathcal{S}$ decrypts $c'_i, i \in [|X|]$ to $r'_i = f(x_i) + r_i$ and sends them to $\mathcal{R}$.
5. [**Output**] $\mathcal{R}$ checks all plaintexts and sets a bit vector $\mathbf{b} = [b_i]$, $i \in [|X|]$. If $r'_i = r_i$, it sets $b_i = 0$, otherwise, sets $b_i = 1$. Then, both parties invoke OT protocol, $\mathcal{R}$ inputs the bit vector $\mathbf{b}$ and $\mathcal{S}$ inputs $(\perp, x_i)$, $i \in [|X|]$. For $i \in [|X|]$, $\mathcal{R}$ gets x_i, if $b_i = 1$, else gets $\perp$. Finally, $\mathcal{R}$ outputs $X \cup Y$.

Fig. 10: Basic PSU protocol

Theorem 1. *The PSU protocol described in Figure 10 is secure in the $\mathcal{F}_{OT}$-hybrid model, in the presence of semi-honest security adversaries, provided that the fully homomorphic encryption scheme is IND-CPA secure with circuit privacy.*

Proof. We construct $\text{Sim}_{\mathcal{S}}$ and $\text{Sim}_{\mathcal{R}}$ to simulate the views of corrupted sender $\mathcal{S}$ and corrupted receiver $\mathcal{R}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\text{Sim}_{\mathcal{S}}(X)$ simulates the view of corrupt $\mathcal{S}$ as follows: It encrypts $|X|$ random values. Then, it invokes $\text{Sim}^{\mathcal{S}}_{\text{OT}}(\perp, x_i)$, $i \in [|X|]$ and appends the output to the view. Now we argue that the view output by $\text{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. The plaintexts are randomized in the real view which is indistinguishable from the random values in the simulated view. The FHE satisfies the circuit privacy which hides the computational circuit in step 3. The view produced by the underlying OT simulator is indistinguishable from the real view. Thus, the simulation is indistinguishable from the real view.

Corrupt Receiver. $\text{Sim}_{\mathcal{R}}(Y, X \cup Y)$ simulates the view of corrupt $\mathcal{R}$ as follows: It simulates the ciphertexts by encrypting $|X|$ random value. $\text{Sim}_{\mathcal{R}}$ sets $\hat{X} = (X \cup Y) \backslash Y$ and pads $\hat{X}$ with $\perp$ into $|X|$ items

and permutes all items randomly. It computes the polynomial $f(y) = \Pi_{y_i \in Y}(y - y_i)$ and the random values $\mathbf{r} = [r_i]$, $i \in [|X|]$ used to randomize the polynomial. Then, for $\hat{x}_i \neq \perp$, $\mathrm{Sim}_{\mathcal{R}}$ defines $r'_i := f(\hat{x}_i) + r_i$, else, it defines $r'_i := r_i$, and appends $\mathbf{r}' = [r'_i], i \in [|X|]$ to the view. If $\hat{x}_i = \perp$, it sets $b_i = 0$, else, $b_i = 1$. Then $\mathrm{Sim}_{\mathcal{R}}$ invokes $\mathrm{Sim}^{\mathcal{R}}_{\mathrm{OT}}(b_i, \hat{x}_i)$ for $i \in [|X|]$ and appends the output to the view.

We argue that the outputs of $\mathrm{Sim}_{\mathcal{R}}$ are indistinguishable from the 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 the ciphertexts in the step 2 are replaced by encrypting $|X|$ random values generated by $\mathrm{Sim}_{\mathcal{R}}$. Since the fully homomorphic encryption scheme is IND-CPA secure, the above simulation is indistinguishable from the real view.
Hyb_2: Same as Hyb_1 except that $\mathrm{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\mathrm{OT}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of OT protocol guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. The hybrid is the view output by $\mathrm{Sim}_{\mathcal{R}}$.

5 Permuted Matrix Private Equality Test

We give two efficient constructions of pm-PEQT in the semi-honest model. The functionality is specified in Figure 5.

5.1 pm-PEQT from Permute + Share and mp-OPRF

The first construction of pm-PEQT is based on the Permute + Share [20,25] and mp-OPRF [7] as described in Figure 11.

Input: The receiver $\mathcal{R}$ inputs a matrix $\mathbf{R}_{\alpha \times m}$. The sender $\mathcal{S}$ inputs a matrix $\mathbf{R}'_{\alpha \times m}$, and a permutation $\pi = (\pi_c, \pi_r)$, where π_c (over $[m]$) and π_r (over $[\alpha]$) are two sub-permutation for columns and rows.
Output: The receiver outputs a bit matrix $\mathbf{B}$. The sender outputs $\perp$.

1. $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal Permute + Share functionality $\mathcal{F}_{\mathrm{PS}}$ twice. First, both parties permute and share the columns of $\mathbf{R}$, where each column of $\mathbf{R}$ can be seen as an item. $\mathcal{R}$ inputs each column of $\mathbf{R}$ and $\mathcal{S}$ inputs the permutation π_c. As a result, $\mathcal{R}$ gets $\mathbf{S}_{\pi_c} = [s_{\pi_c(ij)}]$ and $\mathcal{S}$ gets $\mathbf{S}'_{\pi_c} = [s'_{\pi_c(ij)}]$, where $s_{\pi_c(ij)} \oplus s'_{\pi_c(ij)} = r_{\pi_c(ij)}$. Then, both parties permute and share the rows of $\mathbf{S}_{\pi_c}$, where each rows of $\mathbf{S}_{\pi_c}$ can be seen as an item. $\mathcal{R}$ inputs each rows of $\mathbf{S}_{\pi_c}$ and $\mathcal{S}$ inputs the permutation π_r. As a result, $\mathcal{R}$ gets $\mathbf{S}_{\pi_r} = [s_{\pi_r(ij)}]$ and $\mathcal{S}$ gets $\mathbf{S}'_{\pi_r} = [s'_{\pi_r(ij)}]$, where $s_{\pi_r(ij)} \oplus s'_{\pi_r(ij)} = s_{\pi_c(ij)}$. Finally, $\mathcal{R}$ gets the shuffled matrix shares $\mathbf{S}_\pi = \mathbf{S}_{\pi_r}$ and $\mathcal{S}$ gets the shuffled matrix shares $\mathbf{S}'_\pi = \pi_r(\mathbf{S}'_{\pi_c}) \oplus \mathbf{S}'_{\pi_r}$, where $s_{\pi(ij)} \oplus s'_{\pi(ij)} = r_{\pi(ij)}$, $i \in [\alpha], j \in [m]$.
2. Both parties invoke mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$. $\mathcal{R}$ inputs shuffled shares $\mathbf{S}_\pi$ and obtains the outputs $F_k(s_{\pi(ij)})$, $i \in [\alpha], j \in [m]$. $\mathcal{S}$ obtain the key k.
3. $\mathcal{S}$ computes $F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, $i \in [\alpha], j \in [m]$ and sends them to $\mathcal{R}$.
4. $\mathcal{R}$ sets $b_{ij} = 1$, if $F_k(s_{\pi(ij)}) = F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, else, $b_{ij} = 0$, and gets a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha], j \in [m]$.

Fig. 11: pm-PEQT from Permute + Share and mp-OPRF

Theorem 2. *The construction of Figure 11 securely implements functionality $\mathcal{F}_{pm\text{-}PEQT}$ in the ($\mathcal{F}_{PS}$, $\mathcal{F}_{mp\text{-}OPRF}$)-hybrid model, in the presence of semi-honest adversaries.*

Proof. We exhibit simulators $\mathrm{Sim}_{\mathcal{R}}$ and $\mathrm{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\mathrm{Sim}_{\mathcal{S}}(\mathbf{R}', \pi = (\pi_c, \pi_r))$ simulates the view of corrupt $\mathcal{S}$ as follows: $\mathrm{Sim}_{\mathcal{S}}$ randomly chooses $\mathbf{S}'_{\pi_c}$ and invokes $\mathrm{Sim}^{\mathcal{S}}_{\mathrm{PS}}(\pi_c, \mathbf{S}'_{\pi_c})$ and appends the output to the view. $\mathrm{Sim}_{\mathcal{S}}$ randomly chooses $\mathbf{S}'_{\pi_r}$

and invokes $\mathrm{Sim}^{\mathcal{S}}_{\mathrm{PS}}(\pi_r, \mathbf{S}'_{\pi_r})$ and appends the output to the view. Then, $\mathrm{Sim}_{\mathcal{S}}$ randomly selects a key k of PRF and invokes $\mathrm{Sim}^{\mathcal{S}}_{\text{mp-OPRF}}(k)$ and appends the output to the view.

We argue that the outputs of $\mathrm{Sim}_{\mathcal{S}}$ are indistinguishable from the real view of $\mathcal{S}$ by the following hybrids:

Hyb_0: $\mathcal{S}$'s view in the real protocol.
Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\mathrm{PS}}$ is replaced by $\mathbf{S}'_{\pi_c}$, $\mathbf{S}'_{\pi_r}$ chosen by $\mathrm{Sim}_{\mathcal{S}}$, and $\mathrm{Sim}_{\mathcal{S}}$ runs the $\mathcal{F}_{\mathrm{PS}}$ simulator to produce the simulated view for $\mathcal{S}$. The security of $\mathsf{Permute} + \mathsf{Share}$ guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
Hyb_2: Same as Hyb_1 except that the output key of $\mathcal{F}_{\text{pm-PEQT}}$ is replaced by the k chosen by $\mathrm{Sim}_{\mathcal{S}}$, and $\mathrm{Sim}_{\mathcal{S}}$ runs the $\mathcal{F}_{\text{pm-PEQT}}$ simulator to produce the simulated view for $\mathcal{S}$. The security of mp-OPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. The hybrid is the view output by $\mathrm{Sim}_{\mathcal{S}}$.

Corrupt Receiver. $\mathrm{Sim}_{\mathcal{R}}(\mathbf{R}, \mathbf{B} = [b_{ij}])$ simulates the view of corrupt $\mathcal{R}$ as follows: $\mathrm{Sim}_{\mathcal{R}}$ chooses $\mathbf{S}_{\pi_c}$ and invokes $\mathrm{Sim}^{\mathcal{R}}_{\mathrm{PS}}(\mathbf{R}, \mathbf{S}_{\pi_c})$ and appends the output to the view. $\mathrm{Sim}_{\mathcal{R}}$ chooses $\mathbf{S}_{\pi_r}$ and invokes $\mathrm{Sim}^{\mathcal{R}}_{\mathrm{PS}}(\mathbf{S}_{\pi_c}, \mathbf{S}_{\pi_r})$ and appends the output to the view. $\mathrm{Sim}_{\mathcal{R}}$ randomly selects u_{ij}, $i \in [\alpha]$, $j \in [m]$ and invokes $\mathrm{Sim}^{\mathcal{R}}_{\text{mp-OPRF}}(u_{ij})$ and appends the output to the view. Finally, for all $i \in [\alpha]$, $j \in [m]$, $\mathrm{Sim}_{\mathcal{R}}$ sets $v_{ij} = u_{ij}$ if $b_{ij} = 1$, else, it chooses v_{ij} randomly and appends all v_{ij} to the view.

The view generated by $\mathrm{Sim}_{\mathcal{R}}$ 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 the output of $\mathcal{F}_{\mathrm{PS}}$ is replaced by $\mathbf{S}_{\pi_c}$, $\mathbf{S}_{\pi_r}$ chosen by $\mathrm{Sim}_{\mathcal{R}}$, and $\mathrm{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\mathrm{PS}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of $\mathsf{Permute} + \mathsf{Share}$ guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
Hyb_2: Same as Hyb_1 except that the output PRF values of $\mathcal{F}_{\text{mp-OPRF}}$ is replaced by u_{ij}, $i \in [\alpha]$, $j \in [m]$, and all PRF values in the last step is replaced by the v_{ij}, chosen by $\mathrm{Sim}_{\mathcal{R}}$ randomly, and $\mathrm{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{mp-OPRF}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of mp-OPRF and PRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.

5.2 pm-PEQT based on DDH

The second construction of pm-PEQT is based on DDH as described in Figure 12.

Input: The receiver $\mathcal{R}$ inputs a matrix $\mathbf{R}_{\alpha \times m}$. The sender $\mathcal{S}$ inputs a matrix $\mathbf{R}'_{\alpha \times m}$, and a permutation $\pi = (\pi_c, \pi_r)$ where π_c (over $[m]$) and π_r (over $[\alpha]$) are two sub-permutation for columns and rows.
Output: The receiver outputs a bit matrix $\mathbf{B}$. The sender outputs $\perp$.

1. $\mathcal{R}$ and $\mathcal{S}$ choose random value a, b and compute $v_{ij} = H(r_{ij})^a, v'_{ij} = H(r'_{ij})^b$ for $i \in [\alpha]$, $j \in [m]$, where $H(\cdot)$ denotes hash functions which output (multiplicative) group elements. $\mathcal{R}$ sends v_{ij} to $\mathcal{S}$.
2. $\mathcal{S}$ computes $v''_{ij} = (v_{ij})^b$ and shuffles $v''_{\pi(ij)} = \pi(v''_{ij})$, $v'_{\pi(ij)} = \pi(v'_{ij})$ with $\pi = (\pi_c, \pi_r)$ and sends them to $\mathcal{R}$.
3. $\mathcal{R}$ computes $b_{ij} = 1$, if $v''_{\pi(ij)} = v'^{a}_{\pi(ij)}$, else $b_{ij} = 0$, and gets a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha]$, $j \in [m]$.

Fig. 12: Instantiation of pm-PEQT based on DDH

Theorem 3. *The construction of Figure 12 securely implements functionality $\mathcal{F}_{pm\text{-}PEQT}$ based on DDH in the random oracle model, in the presence of semi-honest security adversaries.*

Proof. We exhibit simulators $\mathrm{Sim}_{\mathcal{R}}$ and $\mathrm{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\mathrm{Sim}_{\mathcal{S}}(\mathbf{R}', \pi = (\pi_c, \pi_r))$ simulates the view of corrupt $\mathcal{S}$ as follows: It chooses random group elements v_{ij}, $i \in [\alpha]$, $j \in [m]$ to simulate the view. We argue that the outputs of $\mathrm{Sim}_{\mathcal{S}}$ are indistinguishable from the real view of $\mathcal{S}$ by the following hybrids:

Hyb_0: $\mathcal{S}$'s view in the real protocol consists of $H(r_{ij})^a$, $i \in [\alpha]$, $j \in [m]$, where $a \leftarrow \mathbb{Z}_q$.
Hyb_1: Same as Hyb_0 except that $\mathrm{Sim}_{\mathcal{S}}$ chooses random group elements v_{ij}, $i \in [\alpha]$, $j \in [m]$ instead of $H(r_{ij})^a$, $i \in [\alpha]$, $j \in [m]$, where $a \leftarrow \mathbb{Z}_q$. The hybrid is the view output by $\mathrm{Sim}_{\mathcal{S}}$.

We argue that the views in Hyb_0 and Hyb_1 are computationally indistinguishable. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary against the DDH assumption. Given the DDH challenge $g^x, g^{y_{ij}}, g^{z_{ij}}$, where $x, y_{ij} \leftarrow \mathbb{Z}_q$, $\mathcal{A}$ is asked to distinguish if $z_{ij} = x \cdot y_{ij}$ or random values. $\mathcal{A}$ implicitly sets $a = x$, and simulates (with the knowledge of $\mathbf{R}$) the view as below:

- RO queries: $\mathrm{Sim}_{\mathcal{S}}$ honestly emulates random oracle H. For queries r_{ij}, if $r_{ij} \notin \mathbf{R}$, it picks a random group element to assign $H(r_{ij})$, otherwise, it assigns $H(r_{ij}) = g^{y_{ij}}$.
- Outputs $g^{z_{ij}}$, $i \in [\alpha]$, $j \in [m]$.

Clearly, if $z_{ij} = x \cdot y_{ij}$, $\mathcal{A}$ simulates Hyb_0. Else, it simulates Hyb_1 (without the knowledge of $\mathbf{R}$), because it responds to all RO queries with random group elements without knowing that the inputs belong to $\mathbf{R}$ or not. Therefore, the outputs of $\mathrm{Sim}_{\mathcal{S}}$ are computationally indistinguishable from the real view based on the DDH assumption.

Corrupt Receiver. $\mathrm{Sim}_{\mathcal{R}}(\mathbf{R}, \mathbf{B} = [b_{ij}])$ simulates the view of corrupt $\mathcal{R}$ as follows: $\mathrm{Sim}_{\mathcal{R}}$ chooses $a \leftarrow \mathbb{Z}_q$ randomly and simulates the first round message as real protocol. For $b_{ij} = 0$, $i \in [\alpha]$, $j \in [m]$, it chooses random group elements v_{ij} and u_{ij} to simulate the view. For $b_{ij} \neq 0$, $i \in [\alpha]$, $j \in [m]$, it chooses random group elements v_{ij} and sets $u_{ij} = v_{ij}^a$ to simulate the view.

We argue that the outputs of $\mathrm{Sim}_{\mathcal{R}}$ are indistinguishable from the real view of $\mathcal{R}$ by the following hybrids:

Hyb_0: $\mathcal{R}$'s view in the real protocol consists of $H(r'_{\pi(ij)})^b$ and $H(r_{\pi(ij)})^{ab}$, $i \in [\alpha]$, $j \in [m]$, where $a, b \leftarrow \mathbb{Z}_q$.
Hyb_1: Same as Hyb_0 except that for $b_{ij} = 0$, that is $r_{\pi(ij)} \neq r'_{\pi(ij)}$, $\mathrm{Sim}_{\mathcal{R}}$ chooses random group elements v_{ij} and u_{ij} instead of $H(r'_{\pi(ij)})^b$ and $H(r_{\pi(ij)})^{ab}$.
Hyb_2: Same as Hyb_1 except that for $b_{ij} = 1$, that is $r_{\pi(ij)} = r'_{\pi(ij)}$, $\mathrm{Sim}_{\mathcal{R}}$ chooses random group elements v_{ij} and sets $u_{ij} = v_{ij}^a$, $i \in [\alpha]$, $j \in [m]$ instead of $H(r'_{\pi(ij)})^b$ and $H(r_{\pi(ij)})^{ab}$. The hybrid is the view output by $\mathrm{Sim}_{\mathcal{R}}$.

We argue that the view in Hyb_0 and Hyb_1 are computationally indistinguishable based on the DDH assumption. Given the DDH challenge $g^x, g^{y_{ij}}, g^{y'_{ij}}, g^{z_{ij}}, g^{z'_{ij}}$, where $x, y_{ij}, y'_{ij} \leftarrow \mathbb{Z}_q$, $\mathcal{A}$ is asked to distinguish if $z_{ij} = x \cdot y_{ij}$, $z'_{ij} = x \cdot y'_{ij}$ or random values. $\mathcal{A}$ implicitly sets $b = x$, and simulates (with the knowledge of $\mathbf{R}'$ and π) the view as below:

- RO queries: $\mathrm{Sim}_{\mathcal{R}}$ honestly emulates random oracle H. For queries r_{ij} and r'_{ij}, if $r_{ij} \notin \mathbf{R}$, $r'_{ij} \notin \mathbf{R}'$, it assigns $H(r_{\pi(ij)})$, $H(r'_{\pi(ij)})$ with random group elements. If $r_{ij} \in \mathbf{R}$, $r'_{ij} \in \mathbf{R}'$, it assigns $H(r_{ij}) = g^{a^{-1} y_{ij}}$, $H(r'_{ij}) = g^{y'_{ij}}$.
- Outputs $g^{z_{ij}}$, $g^{z'_{ij}}$.

Clearly, if $z_{ij} = x \cdot y_{ij}$, $z'_{ij} = x \cdot y'_{ij}$, $\mathcal{A}$ simulates Hyb_0. Else, it simulates Hyb_1. In the Hyb_1, $\mathrm{Sim}_{\mathcal{R}}$ needs not to know the $\mathbf{R}'$ and π in these positions with $b_{ij} = 0$, because in these positions, it responds to all random oracle queries with random group elements.

We argue that the view in Hyb_1 and Hyb_2 are computationally indistinguishable based on the DDH assumption. Given the DDH challenge $g^x, g^{y_{ij}}, g^{z_{ij}}$ where $x, y_{ij} \leftarrow \mathbb{Z}_q$, $\mathcal{A}$ is asked to distinguish if $z_{ij} = x \cdot y_{ij}$ or random values. $\mathcal{A}$ implicitly sets $b = x$, and simulates (with the knowledge of $\mathbf{R}'$ and π for all positions with $b_{ij} = 1$) the view as below:

– RO queries: $\mathrm{Sim}_{\mathcal{R}}$ honestly emulates random oracle H. For queries r_{ij}, r'_{ij}, if $r_{ij} \notin \mathbf{R}$, $r'_{ij} \notin \mathbf{R}'$, it assigns $H(r_{ij})$, $H(r'_{ij})$ with random group elements. If $r_{ij} = r'_{ij} \in \mathbf{R}$, it assigns $H(r_{ij}) = H(r'_{ij}) = g^{y_{ij}}$.
– Outputs $g^{a \cdot z_{ij}}$, $g^{z_{ij}}$.

Clearly, if $z_{ij} = x \cdot y_{ij}$, $\mathcal{A}$ simulates Hyb_1. Else, it simulates Hyb_2. In the Hyb_2, $\mathrm{Sim}_{\mathcal{R}}$ needs not to know the $\mathbf{R}'$ and π in these positions with $b_{ij} = 1$, because in these positions, it responds to all random oracle queries with random group elements. Therefore, the outputs of $\mathrm{Sim}_{\mathcal{R}}$ are computationally indistinguishable from the real view based on the DDH assumption.

Remark. We note that our pm-PEQT can be generalized to multi-query private equality test with permutation, which in turn can be built from permuted OPRF [10] in a general manner.

6 Full PSU Protocol

In this section, we detail our full PSU protocol in Figure 13. It is easy to see that the protocol correctly computes the union conditioned on the hashing succeeding, which happens with overwhelming probability $1-2^{-\lambda}$. Note that in step 1-6, the communication requires $O(|X| \log |Y|)$; the communication of the pm-PEQT requires $O(|X| \log |X|)$ (based on $\mathsf{Permute} + \mathsf{Share}$ and mp-OPRF) or $O(|X|)$ (based on DDH), since we omit the parameter α which is used to make trade-off between the computation and communication of our PSU; the communication of OT requires $O(|X|)$. In summary, the communication of our PSU is $O(|X| \log |Y|)$.

Independent of items length (except for OT). We note that the communication of our PSU protocol is independent of the item length except for OT phase, because in the hashing phase, both parties hash their items down to a smaller domain, and set $X' = \{H(x)|x \in X\}$ and $Y' = \{H(y)|y \in Y\}$ for a hash function H. Then they perform the rest of the protocol with (X', Y') instead of (X, Y), except that in the OT phase, the sender inputs the original $x \in X$ instead of hash value $H(x)$.

Offline/online. Following [9,8,12], the pre-processing of the receiver in our PSU can be done entirely offline without involving the sender. Specifically, given an upper bound on the sender's set size, the receiver can locally choose parameters and perform the pre-processing. Upon learning the sender's actual set size, the receiver can send the parameters to the sender, and the sender can pad the same dummy items like $\perp$ which are known two both parties.

Theorem 4. *The protocol in Figure 13, is a secure protocol for $\mathcal{F}_{PSU}$ in the ($\mathcal{F}_{pm\text{-}PEQT}$, $\mathcal{F}_{OT}$)-hybrid model, in the presence of semi-honest adversaries.*

Proof. For ease of exposition, we will assume that all parameters are fixed and public. 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.

Corrupt Sender. $\mathrm{Sim}_{\mathcal{S}}(X)$ simulates the view of corrupt $\mathcal{S}$ as follows. $\mathsf{Sim}_{\mathcal{S}}$ hashes X into X_c as the real protocol, and encrypts random values in place of the ciphertexts in step 5. Then it decrypts the ciphertexts as $\mathbf{R}'$ and chooses randomly permutation $\pi = (\pi_c, \pi_r)$. It invokes $\mathrm{Sim}^{\mathcal{S}}_{\text{pm-PEQT}}(\mathbf{R}', \pi)$ and $\mathrm{Sim}^{\mathcal{S}}_{\text{OT}}(\perp, X_c[\pi_c(j)])$, $j \in [m_c]$ appends the output to the view. Now we argue that the view output by $\mathrm{Sim}_{\mathcal{S}}$ is indistinguishable from the real one. The plaintexts are randomized in the real view which is indistinguishable from the random values in the simulated view. The FHE satisfies the circuit privacy which hides the computational circuit. The views of the underlying pm-PEQT and OT simulator are indistinguishable. Thus, the simulation is indistinguishable from the real view.

Corrupt Receiver. $\mathsf{Sim}_{\mathcal{R}}$ $(Y, X \cup Y)$ simulates the view of corrupt receiver $\mathcal{R}$ as follows: $\mathrm{Sim}_{\mathcal{R}}$ encrypts random value in place of the ciphertexts in step 4. It chooses the random matrix $\mathbf{R}$ in step 3. $\mathsf{Sim}_{\mathcal{R}}$ computes $\hat{X} = (X \cup Y) \backslash Y$ and pads $\hat{X}$ with $\perp$ to m_c items and permutes these items randomly. For all items in $\hat{X}$, if $\hat{x}_i \neq \perp$, it sets $b_i = 1$, else $b_i = 0$. And then it generates $\mathbf{B}_{\alpha \times m_c}$, for all columns $\mathbf{b}_i$, if $b_i = 1$, it sets all

Input: The receiver $\mathcal{R}$ inputs set $Y \subset \{0,1\}^*$ of size $|Y|$. The sender $\mathcal{S}$ inputs set $X \subset \{0,1\}^*$ of size $|X|$.
Output: The receiver outputs $X \cup Y$. The sender outputs $\perp$.

1. [**Setup**] $\mathcal{R}$ and $\mathcal{S}$ agree on the hashing, FHE scheme, mp-PEQT and OT parameters.
2. [**Hashing**] $\mathcal{S}$ hashes the set X into $X_c[i], i \in [m_c]$ by Cuckoo hash and $\mathcal{R}$ hashes the set Y into $\mathbf{Y}_{B \times m_c}$ by simple hash.
3. [**Pre-process Y**]
   (a) [Partitioning] $\mathcal{R}$ partitions $\mathbf{Y}_{B \times m_c}$ by rows into α subtables $\mathbf{Y}_1, \mathbf{Y}_2, \cdots, \mathbf{Y}_\alpha$. Each subtable has $B' = B/\alpha$ rows and m columns. Let i-th subtable be $\mathbf{Y}_i = [\mathbf{y}_{i,1}, \cdots, \mathbf{y}_{i,B'}]^T, i \in [\alpha]$, where $\mathbf{y}_{i,k}$, $k \in [B']$ indicates k-th row of $\mathbf{Y}_i$.
   (b) [Computing coefficients] For j-th columns of i-th subtable $\mathbf{y}'_{i,j} = [y_{i,j,1}, y_{i,j,2}, \cdots, y_{i,j,B'}]^T$, $i \in [\alpha], j \in [m_c]$, where $y_{i,j,k}$, $k \in [B']$ indicates k-th item of $\mathbf{y}'_{i,j}$, $\mathcal{R}$ computes the coefficients of the polynomial $f_{i,j}(y) = \Pi_{k=1}^{B'}(y - y_{i,j,k}) = a'_{i,j,0} + a_{i,j,1}y + \cdots + a_{i,j,B'-1}y^{B'-1} + a_{i,j,B'}y^{B'}$. $\mathcal{R}$ computes the coefficient matrix $\mathbf{A}$ as follows: $\mathcal{R}$ chooses a random matrix $\mathbf{R}_{\alpha \times m_c}$. $\mathcal{R}$ sets j-th columns of i-th subtable $\mathbf{A}_{i,j} = [a_{i,j,0}, a_{i,j,1}, \cdots, a_{i,j,B'}]^T$, $i \in [\alpha], j \in [m]$, where $a_{i,j,0} = a'_{i,j,0} + r_{i,j}$, $r_{i,j} \in \mathbf{R}_{\alpha \times m_c}$.
   (c) [Batching] For each subtable obtained from the previous step, $\mathcal{R}$ interprets each of its row as a vector of length m with elements in $\mathbb{Z}_t$. Then $\mathcal{R}$ batches each vector into $\beta = m/n$ plaintext polynomials. As a result, each row of i-th subtable $\mathbf{A}_i$ is transformed into β polynomials denoted $\hat{\mathbf{A}}_{i,j}$, $i \in [\alpha]$, $j \in [\beta]$.
4. [**Encrypt X**]
   (a) [Batching] $\mathcal{S}$ interprets X_c as a vector of length m_c with items in $\mathbb{Z}_t$. It batches this vector into $\beta = m/n$ plaintext polynomials $\hat{X}_1, \cdots, \hat{X}_\beta$.
   (b) [Windowing] For each batched plaintext polynomial $\hat{X}$, $\mathcal{S}$ computes the component-wise $i \cdot 2^j$-th powers $\hat{X}^{i \cdot 2^{lj}}$, for $1 \le i \le 2^l - 1$ and $0 \le j \le \lceil \log_2(B')/l \rceil$.
   (c) [Encrypt] $\mathcal{S}$ uses FHE scheme to encrypt each such power, obtaining β collections of ciphertexts $\mathbf{C}_j, j \in [\beta]$, and each collection consists of the ciphertexts $[c_{i,j}]$, $1 \le i \le 2^l - 1$ and $0 \le j \le \lceil \log_2(B')/l \rceil$. $\mathcal{S}$ sends these ciphertexts to $\mathcal{R}$.
5. [**Computation**]
   (a) [Homomorphically compute encryptions of all powers] For each collection $\mathbf{C}_j, j \in [\beta]$, $\mathcal{R}$ homomorphically computes encryptions of all powers $\mathbf{C}_j = [\mathbf{c}_{j,0}, \cdots, \mathbf{c}_{j,B'}]$, where $\mathbf{c}_{j,k}, 0 \le k \le B'$ is a homomorphic ciphertext encrypting $\hat{X}_j^k$.
   (b) [Homomorphically evaluate the dot product] $\mathcal{R}$ homomorphically evaluates $\mathbf{C}'_{i,j} = \mathbf{C}_j \hat{\mathbf{A}}_{i,j}$, $i \in [\alpha]$, $j \in [\beta]$, performs modulus switching on $\mathbf{C}'_{i,j}, i \in [\alpha], j \in [\beta]$ to reduce sizes, and sends them to $\mathcal{S}$.
6. [**Decrypt**] $\mathcal{S}$ decrypts all ciphertexts and concatenates the resulting matrixes into one matrix $\mathbf{R}'_{\alpha \times m_c}$.
7. [**pm-PEQT**] $\mathcal{R}$ inputs the matrix $\mathbf{R}_{\alpha \times m_c}$, and $\mathcal{S}$ inputs the permutation $\pi = (\pi_c, \pi_r)$ and the matrix $\mathbf{R}'_{\alpha \times m_c}$. Both parties invoke the pm-PEQT functionality. As a result, $\mathcal{R}$ gets a bit matrix $\mathbf{B}_{\alpha \times m_c}$, where if $b_{ij} = 1$, $r_{\pi(ij)} = r'_{\pi(ij)}$, else $r_{\pi(ij)} \neq r'_{\pi(ij)}$, $i \in [\alpha], j \in [m_c]$.
8. [**Output**] $\mathcal{R}$ sets a bit vector $\mathbf{b} = [b_j]$, $j \in [m_c]$, where if for all $i \in [\alpha]$, $b_{ij} = 0$, it sets $b_j = 1$, else $b_j = 0$. Then, $\mathcal{R}$ and $\mathcal{S}$ invoke the OT functionality, in which $\mathcal{R}$ inputs b_j, $j \in [m_c]$ and $\mathcal{S}$ inputs $(\perp, X_c[\pi_c(j)])$. If $b_j = 1$, $\mathcal{R}$ gets $X_c[\pi_c(j)]$, else, it gets $\perp$. Finally, $\mathcal{R}$ outputs the set union $Y \cup \{X_c[\pi_c(j)]\}$, for all $j \in [m_c]$.

Fig. 13: Full PSU protocol

Parameters		Protocols	Comm. (MB)			Runtime (s), $T = 1$				Runtime (s), $T = 4$			
						Sender	Receiver		Total	Sender	Receiver		Total
$\lvert X\rvert$	$\lvert Y\rvert$		$\mathcal{S} \rightarrow \mathcal{R}$	$\mathcal{R} \rightarrow \mathcal{S}$	Total		Offline	Online			Offline	Online	
2^{10}	2^{18}	PSU_{DDH}	2.026	0.216	2.242	1.35	2.01	1.35	4.71	0.68	1.16	0.68	2.52
		PSU_{PS}	2.14	0.93	3.07	1.39	2.8	1.39	5.58	0.92	1.42	0.92	3.26
	2^{20}	PSU_{DDH}	2.223	0.428	2.651	2.27	17.88	2.27	22.42	1.09	7.15	1.08	9.32
		PSU_{PS}	2.45	1.848	4.298	2.03	18.22	2.03	22.28	1.19	7.9	1.19	10.28
	2^{22}	PSU_{DDH}	3.24	1.33	4.57	5.74	88.41	5.75	99.9	2.42	32.66	2.41	37.49
		PSU_{PS}	4.11	6.82	10.93	4.32	84.65	4.32	93.29	1.84	34.71	1.87	38.42
2^{11}	2^{18}	PSU_{DDH}	3.063	0.436	3.499	2.35	2.01	2.32	6.68	0.89	1.16	0.91	2.96
		PSU_{PS}	3.56	1.85	5.41	2.1	2.02	2.07	6.19	1.06	1.2	1.05	3.31
	2^{20}	PSU_{DDH}	3.343	0.436	3.779	2.95	17.91	2.94	23.8	1.19	7.15	1.19	9.53
		PSU_{PS}	3.56	1.85	5.41	2.71	17.74	2.69	23.14	1.36	7.18	1.35	9.89
	2^{22}	PSU_{DDH}	4.51	1.55	6.06	6.8	84.39	6.79	97.98	2.37	31.85	2.36	36.58
		PSU_{PS}	5.69	7.64	13.33	5.15	83.96	5.14	94.25	2.12	31.88	2.11	36.11

Table 2: Communication (in MB) and runtime (in seconds) of PSU_{PS} and PSU_{DDH} for unbalanced sets size ($|X| \in \{2^{10}, 2^{11}\}$ and $|Y| \in \{2^{18}, 2^{20}, 2^{22}\}$) with threads $T \in \{1, 4\}$, and 10 Gbps network bandwidth, 0.2 ms RTT.

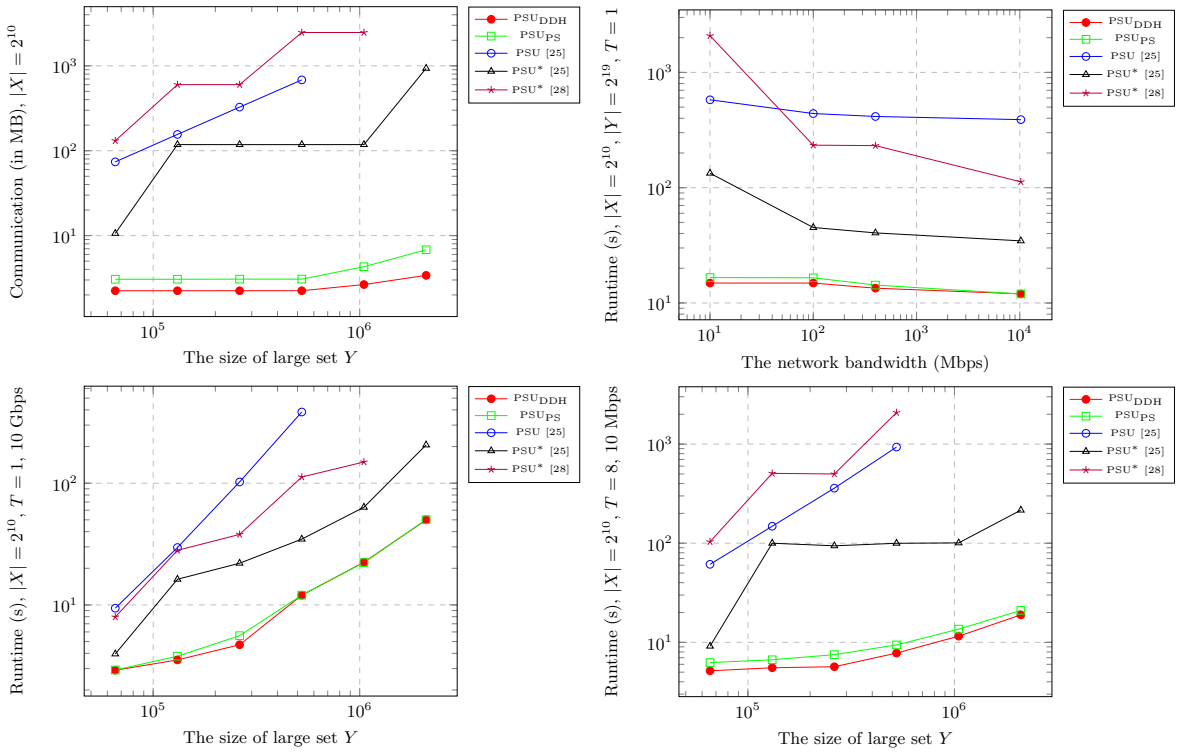

Fig. 14: Comparisons of communication (in MB) and runtime (in seconds) between PSU [25], PSU* [25], PSU* [28], PSU_{DDH} and PSU_{PS}

Parameters		Protocols	Comm. (MB)	Total running time (s)								
				10 Gbps			100 Mbps			10 Mbps		
$\|X\|$	$\|Y\|$			$T=1$	$T=4$	$T=8$	$T=1$	$T=4$	$T=8$	$T=1$	$T=4$	$T=8$
2^{10}	2^{18}	PSU [25]	326.313	102.36	96.12	95.05	121.49	115.78	115.84	359.02	359.54	359.24
		PSU* [25]	117.99	22.01	5.8	3.76	28.55	12.67	10.86	113.15	96.84	94.17
		PSU* [28]	600.62	37.95	10.94	7.01	71.94	49.3	47.69	498.37	505.89	499.34
		Our PSU_{PS}	3.07	5.59	3.26	3.01	10.1	7.69	7.81	10.18	7.73	7.51
		Our PSU_{DDH}	2.242	4.71	2.52	2.72	8.53	5.89	5.95	8.48	5.86	5.68
	2^{19}	PSU [25]	683.001	384.55	371.21	371.45	431.45	417.35	417.36	931.05	959.75	933.67
		PSU* [25]	117.991	34.72	9.08	5.98	42.01	16.02	12.22	125.86	100.05	99.84
		PSU* [28]	2470.1	112.41	31.8	20.33	233.79	208.04	207.86	2080.19	2077.88	2079.4
		Our PSU_{PS}	3.07	12.03	5.97	5.13	16.64	10.42	9.58	16.55	10.54	9.42
		Our PSU_{DDH}	2.242	12.01	5.49	4.73	15.06	8.67	7.68	14.77	8.52	7.79
2^{11}	2^{18}	PSU [25]	326.386	102.31	95.99	95.24	121.63	115.73	116.02	359.11	361.03	359.75
		PSU* [25]	235.966	30.88	8.13	5.22	47.55	24.99	21.96	220.02	200.07	199.64
		PSU* [28]	600.62	37.99	10.67	7.23	71.84	49.23	49.37	505.8	505.72	499.41
		Our PSU_{PS}	5.41	6.9	3.31	3.09	11.46	8.85	8.45	10.64	8.7	8.64
		Our PSU_{DDH}	3.499	6.67	2.96	2.97	10.18	7.05	6.64	10.19	6.79	6.75
	2^{19}	PSU [25]	683.001	385.66	372.01	370.67	429.82	416.11	417.49	932.46	937.55	932.15
		PSU* [25]	237.864	43.33	11.79	7.38	59.98	28.34	23.38	232.97	200.176	199.65
		PSU* [28]	2470.1	112.35	31.7	22.17	232.72	207.87	207.99	2081.05	2078.27	2080.11
		Our PSU_{PS}	5.41	10.58	5.18	4.44	15.54	10.59	10.25	15.33	10.34	9.99
		Our PSU_{DDH}	3.499	10.51	4.86	4.32	14.16	8.88	8.18	14.09	8.57	8.17

Table 3: Comparisons of communication (in MB) and runtime (in seconds) between PSU [25], PSU* [25], PSU* [28], PSU_{PS} and PSU_{DDH} for unbalanced sets size ($|X| \in \{2^{10}, 2^{11}\}$, $|Y| \in \{2^{18}, 2^{19}\}$) with threads $T \in \{1, 4, 8\}$, and 10 Gbps bandwidth, 0.2 ms RTT; 100Mbps and 10 Mbps bandwidth, 80 ms RTT. The best results are marked in blue.

items in $\mathbf{b}_i$ be 0, else, it sets one random position in $\mathbf{b}_i$ be 1 and all other positions are 0. $\text{Sim}_{\mathcal{R}}$ invokes $\text{Sim}^{\mathcal{R}}_{\text{pm-PEQT}}(\mathbf{R}, \mathbf{B})$ appends the output to the view. Then, for all $i \in [m_c]$, it invokes $\text{Sim}^{\mathcal{R}}_{\text{OT}}(b_i, \hat{x}_i)$ and appends the output to the view.

The view generated by $\text{Sim}_{\mathcal{R}}$ 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 the ciphertexts are replaced by encrypting random values generated by $\text{Sim}_{\mathcal{R}}$. Since the fully homomorphic encryption scheme is IND-CPA secure, the simulation is indistinguishable from the real view.

Hyb_2: Same as Hyb_1 except that the output of $\mathcal{F}_{\text{pm-PEQT}}$ is replaced by $\mathbf{B}$ generated by $\text{Sim}_{\mathcal{R}}$, and $\text{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{pm-PEQT}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of the pm-PEQT protocol guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.

Hyb_3: Same as Hyb_2 except that $\text{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{OT}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of OT protocol guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. The hybrid is the view output by $\text{Sim}_{\mathcal{R}}$.

7 Implementation and Performance

In this section, we experimentally evaluate our two PSU protocols:

– PSU_{PS}: PSU protocol based on FHE, pm-PEQT and OT, where pm-PEQT is constructed with Permute + Share and mp-OPRF.
– PSU_{DDH}: PSU protocol based on FHE, DDH-based pm-PEQT and OT.

We first give our experimental environment. Then we compare our protocols with the state-of-the-art works in terms of communication and runtime on different networks. Our complete implementation is available upon request.

7.1 Experimental Setup

We run our experiments on an Intel Core with 3.00GHz and 8GB RAM. We perform all tests using this single machine, and simulate network latency and bandwidth using the Linux tc command. Specifically, we consider the following LAN setting, where the two parties are connected via a local host with 10Gbps throughput, and a 0.2ms round-trip time (RTT). We also consider two WAN settings with 100Mbps, and 10Mbps bandwidth, each with an 80ms RTT.

7.2 Implementation Details

We implement Permute + Share with the design in [30] and OT extension [24] using libOTe library. For the FHE scheme, we use the source code from SEAL and APSI libraries. For mp-OPRF, we use the source code from [7]. For concrete analysis we set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. Our protocols are written in C++, and we use the following libraries in our implementation.

– FHE: SEAL https://github.com/microsoft/SEAL and APSI https://github.com/microsoft/APSI
– Permute + Share: https://github.com/dujiajun/PSU
– mp-OPRF: https://github.com/peihanmiao/OPRF-PSI and https://github.com/yuchen1024/Kunlun
– OT: https://github.com/osu-crypto/libOTe

7.3 Performance Comparisons

In this section, we list the experimental results of our protocols in Table 2. Then, we compare our PSU protocols with PSU [25], PSU* [25] and PSU* [28] in terms of runtime and communication, and the results are reported in Table 3 and Figure 14.

We stress that all reported costs are computed in the same environment. For comparisons of other works [25,28], we use the parameters recommended in their open-source code.

– PSU and PSU* [25]: https://github.com/dujiajun/PSU
– PSU* [28]: https://github.com/osu-crypto/PSU

Communication comparison. Our PSU_{DDH} has the lowest communication among all protocols [25,28] in unbalanced case. For set sizes ($|X| = 2^{10}, |Y| = 2^{19}$), 128-bit length item, the communication of our PSU_{DDH} requires 2.242 MB, which is about 300× lower than PSU [25] requiring 683 MB, about 50× lower than PSU* [25] requiring 117.9MB and 1100× lower than PSU* [28] requiring 2470 MB. As shown in Figure 14, the larger difference between two set sizes, the better our protocols perform. For small set size $|X| = 2^{10}$ and large set size in the order of millions $|Y| = 2^{22}$ with 128-bit length items, the communication of our PSU_{DDH} is only 4.57 MB.

Runtime comparison. Our PSU_{PS} and PSU_{DDH} are faster than PSU [25,28] in unbalanced case. As shown in Figure 14, the larger difference between two set sizes, the better our protocols perform. For two set sizes ($|X| = 2^{10}$, $|Y| = 2^{19}$), all item length 128-bit with $T = 1$ thread in LAN setting, the runtime of our PSU_{DDH} requires 12 seconds, while PSU [25] requires 384.55 seconds, about 30× improvement, PSU* [25] requires 34.72 seconds, about 2.8× improvement, PSU* [28] requires 112.41 seconds, about 9× improvement. The

performance of our protocols improves significantly in the case of low bandwidth. For two set sizes ($|X| = 2^{10}$, $|Y| = 2^{19}$), all item length 128-bit with $T = 8$ thread in 10 Mbps, our PSU_{DDH} requires 7.79 seconds and PSU_{PS} requires 9.42 seconds, while PSU [25] requires 933.67 seconds, about 120× improvement, PSU* [25] requires 99.84 seconds, about 12× improvement, PSU* [28] requires 2079.4 seconds, about 260× improvement.

References

1. Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS*, pages 40–41, 2012.
2. Dan Boneh. The decision diffie-hellman problem. In *Algorithmic Number Theory*, pages 48–63, 1998.
3. Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In *Advances in Cryptology - CRYPTO 2016*, pages 62–89, 2016.
4. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *19th USENIX Security Symposium*, pages 223–240, 2010.
5. Andrea Cerulli, Emiliano De Cristofaro, and Claudio Soriente. Nothing refreshes like a repsi: Reactive private set intersection. In *Applied Cryptography and Network Security ACNS 2018*, pages 280–300.
6. Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *Advances in Cryptology - ASIACRYPT 2020*, pages 342–372.
7. Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *Advances in Cryptology - CRYPTO 2020*, pages 34–63.
8. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 1223–1237.
9. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1243–1255.
10. Yu Chen, Min Zhang, Cong Zhang, and Minglang Dong. Private set operations from multi-query reverse private membership test. Cryptology ePrint Archive, Paper 2022/652, 2022. `https://eprint.iacr.org/2022/652`.
11. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018*, pages 464–482, 2018.
12. Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150, 2021.
13. Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017*, pages 261–278, 2017.
14. Luc Devroye and Pat Morin. Cuckoo hashing: Further analysis. *Inf. Process. Lett.*, 86(4):215–219, 2003.
15. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 789–800, 2013.
16. Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 14–25.
17. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.
18. Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. In *STACS 2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 271–282.
19. Keith B. Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security, ACNS 2007*, pages 237–252, 2007.
20. Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *Public-Key Cryptography - PKC 2021*, pages 591–617.
21. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *Public-Key Cryptography - PKC 2017*, volume 10174 of *Lecture Notes in Computer Science*, pages 175–203.
22. Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE Cybersecurity Development, SecDev 2016*, pages 75–76.

23. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, 2012*.
24. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, 2003.
25. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. *IACR Cryptol. ePrint Arch.*, page 157, 2022.
26. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005*, pages 241–257, 2005.
27. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1257–1272.
28. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *Advances in Cryptology - ASIACRYPT 2019*, pages 636–666.
29. Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In *Information Security and Privacy: 9th Australasian Conference, ACISP*, pages 391–401, 2004.
30. Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology - EUROCRYPT 2013*, pages 557–574.
31. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA 2001*, pages 121–133.
32. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019*, pages 401–431.
33. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019*, pages 401–431.
34. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *Advances in Cryptology - EUROCRYPT 2019*, pages 122–153.
35. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *Advances in Cryptology - EUROCRYPT 2018*, pages 125–157, 2018.
36. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium*, pages 797–812, 2014.
37. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, (2):7:1–7:35, 2018.
38. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, pages 7:1–7:35, 2018.
39. Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 187, 2005.
40. Amanda Cristina Davi Resende and Diego de Freitas Aranha. Faster unbalanced private set intersection in the semi-honest setting. *J. Cryptogr. Eng.*, (1):21–38, 2021.
41. Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2017*, pages 235–259.
42. Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Optimal private set union from multi-query reverse private membership test. Cryptology ePrint Archive, Report 2022/358, 2022. `https://ia.cr/2022/358`.