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@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 except the union and it has found numerous applications in practice. Recently, some computationally efficient PSU protocols have been designed in the balanced case, but a potential limitation with these approaches is the communication complexity, which scales 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 (cellphone) holding a small set, and another is a large service provider.
In this work, we propose a generic framework of using the leveled fully homomorphic encryption and a newly introduced protocol called permute matrix Private EQuality Test (pm-PEQT) to construct the *unbalanced* PSU that is secure against semi-honest adversaries. By instantiating the pm-PEQT, we obtain fast unbalanced PSU protocols with a small communication overhead. Our protocol has communication complexity *linear in the size of the smaller set, and logarithmic in the larger set.* More precisely, if the set sizes are $|X| \ll |Y|$, our protocol achieves a communication overhead of $O(|X| \log |Y|)$.
Finally, we implement our protocols that can compare with the state-of-the-art PSU. Experiments show that our protocols are more efficient than all previous protocols in the unbalanced case, especially, the larger the difference of two set sizes, the better our protocols perform. Our running-time-optimized benchmarks show that it takes 18.782 seconds of computation and 2.179 MB of communication to compute the union between 2^{10} strings and 2^{19} strings. Compared to prior secure PSU proposed by Jia et al. (Usenix Security 2022), this is roughly a 300× reduction in communication and 20× reduction in computational overhead with a single thread in WAN/LAN settings.

1 Introduction

PSU is a cryptographic technique that allows two parties holding sets X and Y respectively, to compute the union $X \cup Y$, without revealing anything else. Recently, some works have been made on PSU, which have become considerably efficient and been deployed in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data [23,18,22], privacy-preserving data aggregation [4], private ID [14] etc. However, most of the works on PSU 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 some certain applications, the sender's set may be much smaller than the receiver's. The sender may be a mobile device with limited battery, computing power and storage, whereas the receiver 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 above unbalanced case.

The unbalanced PSU (uPSU) can be seen as a special case of PSU, where (1) the set size of one side is significantly smaller than another's, and (2) the side (with the smaller set) has a low-power device. Chen et al. [9,7] first consider unbalanced case and design an efficient unbalanced private set intersection (uPSI) based on the leveled fully homomorphic encryption (FHE). Their fast uPSI breaks the bound of communication complexity linear with the size of the larger set and achieves the communication complexity linear in the size of the smaller set, and logarithmic in the larger set. However, they only realize uPSI without considering the construction of the uPSU. Recently, Jia et al. [21] give a construction of uPSU*[4] with shuffling

[4] In this paper, we use PSU* to denote a PSU protocol with information leakage.

technique, but their uPSU* protocol is not entirely satisfactory. On one hand, their uPSU* cannot achieve full security of PSU, it *leaks the information of the size of set intersection* to the sender. This is a critical information leakage for uPSU. When the set size of one side is very small, the probability of guessing the intersection elements is higher, in particular, the protocol leaks the intersection element when the sender inputs one-element set. On the other hand, the communication overhead of their uPSU* is not ideal and still requires at least *linearly with the size of the larger set*. Furthermore, their uPSU* does not support for arbitrary length items, because of using Permute + Share instead of OT protocol. Therefore, how to design a secure and fast uPSU is an open problem. Based on the above discussions, we ask the following question:

Is it possible to design a ***secure*** *and* ***fast*** *unbalanced PSU protocol which breaks the bound of communication complexity linear with the size of the larger set, meanwhile, it supports for* ***arbitrary length items****?*

1.1 Contributions and Roadmap

In this paper, we give an affirmative answer to above question. We construct *secure* and *fast* unbalanced PSU protocols which have communication complexity linear in the size of the smaller set, and logarithmic in the larger set, meanwhile, they can support for arbitrary length items. Our protocols are particularly powerful when the receiver's set is much larger than the sender's set. In detail, our contributions and roadmap can be summarized as follows:

1. We first give a **basic uPSU** protocol based on fully homomorphic encryption with communication linear in the smaller set, achieving optimal communication that is on par with the naive solution. However, the basic protocol requires a high computational cost and a deep homomorphic circuit.
2. We use an array of optimizations following [9,7,10] to get an **uPSU* with optimizations** which significantly reduces computational cost and the depth of the homomorphic circuit. However, we observe that using these optimizations directly in uPSU could lead to information leakage.
3. We introduce a new cryptographic protocol named permute matrix private equality test **(pm-PEQT)**. In the pm-PEQT, the sender with a matrix $\mathbf{R}'$ and a matrix permutation π interacts with a receiver holding a matrix $\mathbf{R}$. As a result, the receiver learns (only) a bit matrix $\mathbf{B}$ indicating that some elements in permuted positions are equal or not, while the sender learns nothing about the matrix $\mathbf{R}$. Compare with the private equality test (PEQT), pm-PEQT can provide private equality test of all elements in the matrix with **positions permutation**. Then, we give a **generic framework** of uPSU protocol based on our uPSU* with optimizations, pm-PEQT and OT protocol.
4. We instantiate our pm-PEQT efficiently in two methods. The first is based on Permute + Share [14,21] and multi-point oblivious pesudorandom function (mp-OPRF) [28,6]. Another is based on the decisional Diffie-Hellman (DDH) assumption.
   Then, we obtain secure and fast uPSU protocols with communication complexity linear in the size of the smaller set and logarithmic in the larger set. We show the communication and security comparison of PSU in the table 1.
5. Finally, we implement our uPSU protocols and compare with the state-of-the-art PSU [21], uPSU* [21] and PSU* [22] in terms of runtime and communication in the same environment. Experiments show that our protocols are more efficient than all previous protocols in the unbalanced case, especially, the larger the difference of two set sizes, the better our protocols perform. Our running-time-optimized benchmarks show that it takes 18.782 seconds of computation and 2.179 MB of communication to compute the union between 2^{10} strings and 2^{19} strings. Compared to prior secure PSU [21], this is roughly a 300× reduction in communication and 20× reduction in computational overhead with a single thread in WAN/LAN settings.

1.2 Related Works

We revisit some recent private set operation protocols including uPSU [21], PSU [22,14,21,32] and uPSI protocols [9], with particular emphasis on the semi-honest model.

Table 1: Communication and security comparison of PSU

Protocols	Communication	Security
PSU* [22]	$O(n \log n)$	$\not\checkmark$
PSU [14]	$O(n \log n)$	✓
PSU [32]	$O(n)$	✓
PSU [21]	$O(n \log n)$	✓
uPSU* [21]	$O(n + m \log m)$	$\not\neq$
Our uPSU	$O(m \log n)$	✓

‡ n denotes the size of the large set, and m denotes the size of the small set. PSU* denotes it is not full secure. $\not\checkmark$ denotes the PSU leaks the information of some subsets have the items in the set intersection. $\not\neq$ denotes the PSU leaks the information of the size of the set intersection.

uPSU protocol. Jia et al. [21] propose an uPSU* with the shuffling technique. The sender $\mathcal{S}$ inputs a set X and the receiver $\mathcal{R}$ inputs a set Y, where $|X| \ll |Y|$. As a result, $\mathcal{S}$ gets the size of set intersection $X \cap Y$ and $\mathcal{R}$ gets the set union $X \cup Y$. Informally, $\mathcal{S}$ inserts X into the Cuckoo hash table and the item in i-th bin denotes as $X_c[i]$, $i \in [m_c]$. $\mathcal{R}$ inserts Y into the simple hash table and the set of items in the i-th bin denotes as $Y_b[i]$, $i \in [m_c]$, and the bin size is ρ. And then, they use shuffling technique to permute and share X_c by a permutation π chosen by $\mathcal{R}$. $\mathcal{S}$ gets shuffled shares $\{a_{\pi(1)}, a_{\pi(2)}, \cdots, a_{\pi(m_c)}\}$, and $\mathcal{R}$ gets shuffled shares $\{a'_{\pi(1)}, a'_{\pi(2)}, \cdots, a'_{\pi(m_c)}\}$, where $X_c[\pi(i)] = a_{\pi(i)} \oplus a'_{\pi(i)}$. $\mathcal{R}$ permutes $Y_b[i]$ to $Y_b[\pi(i)]$, $i \in [m_c]$, and for all $y_{\pi(i)} \in Y_b[\pi(i)]$, computes $y_{\pi(i)} \oplus a'_{\pi(i)}$. As we can see, if $X_c[\pi(i)] \in Y_b[\pi(i)]$, there exists an item $y_{\pi(i)} \in Y_b[\pi(i)]$, s.t $y_{\pi(i)} \oplus a'_{\pi(i)} = a_{\pi(i)}$. Then, both parties run mp-OPRF to let $\mathcal{S}$ get all PRF values $F_k(a_{\pi(i)})$ of $a_{\pi(i)}$ and let $\mathcal{R}$ get the key k. For each bin, $\mathcal{R}$ can compute $F_k(y_{\pi(i)} \oplus a'_{\pi(i)})$ and sends them to $\mathcal{S}$. $\mathcal{S}$ tests whether the item in each bin belongs to the union by checking $F_k(a_{\pi(i)}) \stackrel{?}{=} F_k(y_{\pi(i)} \oplus a'_{\pi(i)})$, for all $y_{\pi(i)} \in Y_b[\pi(i)]$. $\mathcal{S}$ obtains a bit vector $\mathbf{b} = (b_1, \cdots, b_{m_c})$, if $b_i = 1$, $X_c[\pi(i)] \notin Y_b$, else, $X_c[\pi(i)] \in Y_b$. Then, two parties run shuffling technique to permute and share $\{a'_{\pi(1)}, a'_{\pi(2)}, \cdots, a'_{\pi(m_c)}\}$ by a permutation π' chosen by $\mathcal{S}$. $\mathcal{S}$ obtains $\{s_{\pi'(1)}, s_{\pi'(2)}, \cdots, s_{\pi'(m_c)}\}$, and $\mathcal{R}$ gets $\{s'_{\pi'(1)}, s'_{\pi'(2)}, \cdots, s'_{\pi'(m_c)}\}$, where $s_{\pi'(i)} \oplus s'_{\pi'(i)} = a'_{\pi'(i)}$, $\{a'_{\pi'(1)}, \cdots, a'_{\pi'(m_c)}\} = \pi'(\{a'_{\pi(1)}, \cdots, a'_{\pi(m_c)}\})$. Finally, $\mathcal{S}$ computes $\{b_{\pi'(1)}, \cdots, b_{\pi'(m_c)}\} = \pi'(\{b_1, \cdots, b_{m_c}\})$ and sets $z_{\pi'(i)} = \perp$ if $b_{\pi'(i)} = 0$, else, $z_{\pi'(i)} = a_{\pi'(i)} \oplus s_{\pi'(i)}$ and sends $(z_{\pi'(1)}, \cdots, z_{\pi'(m_c)})$ to $\mathcal{R}$. For all $i \in [m_c]$, $\mathcal{R}$ checks $z_{\pi'(i)} \neq \perp$ and $z_{\pi'(i)} \oplus s'_{\pi'(i)}$ is not dummy item, and outputs $Y \cup \{z_{\pi'(i)} \oplus s'_{\pi'(i)}\}$.

As mentioned in [21], compared with their balanced PSU [21], their uPSU* described above is efficient in unbalanced case, because of replacing OT related to the large set with Permute + Share related to the small set to obtain the items of the set union. The communication overhead in this phase drops from $O(n)$ to $O(m \log m)$, where $m = |X| \ll n = |Y|$. However, this part leaks the set intersection size to the sender, since the sender gets the bit vector $\mathbf{b}$, and knows the number of zero in the $\mathbf{b}$. Meanwhile, by using Permute + Share instead of OT, the sender needs to send its shuffled shares which are the shares of the union items, such that the receiver can combine the union items. So both parties need to inserts their items into hash tables rather than hash values. This leads to their uPSU* does not support for arbitrary length items. Moreover, the receiver needs to send all PRF values of the large set to the sender, so the communication overhead in this phase is $O(n)$ that requires at least linearly with the size of the large set Y.

Recent balanced PSU protocol. Kolesnikov et al. [22] propose a PSU protocol based on the reverse private membership test (RPMT) and OT. In RPMT, the sender with input x interacts with a receiver holding a set Y, and the receiver can learn a bit indicating whether $x \in Y$, while the sender learns nothing. After that, both parties run OT protocol to let the receiver obtain $\{x\} \cup Y$. RPMT requires $O(n \log^2 n)$ computation, and $O(n)$ communication, where $|Y| = n$. If the size of the sender's set is $|X| = n$, for computing the set union, the protocol runs RPMT n times independently, which requires $O(n^2)$ communication and $O(n^2 \log^2 n)$ computation. By using the bucketing technique, two parties can hash each set X or Y in m

bins, each bin consists of ρ items. Computing a large (n, n)-PSU[5] can be divided into computing m small (ρ, ρ)-PSU. The complexity can be reduced to $O(n \log n)$ communication and $O(n \log n \log \log n)$ computation. However, the bucketing technique leads to information leakage about the items in the set intersection. In the ideal (n, n)-PSU, from the view of receiver, any item in Y could be an item in $X \cap Y$. But in each (ρ, ρ)-PSU, the receiver knows that some subsets with size ρ have items in $X \cap Y$.

Garimella et al. [14] give a PSU protocol based on oblivious switching and OT. They first propose the permuted characteristic functionality and give a construction based on oblivious switching, in which the sender inputs the set $X = \{x_1, \cdots, x_n\}$ and get a random permutation π, the receiver inputs the set $Y = \{y_1, \cdots, y_n\}$ and gets a vector $\mathbf{e} \in \{0, 1\}^n$, where $e_i = 1$ if $x_{\pi(i)} \in Y$, else, $e_i = 0$. Then two parties run OT protocol to let the receiver obtain the set union. Their protocol requires $O(n \log n)$ communication and $O(n \log n)$ computation.

Jia et al. [21] propose a PSU with the shuffling technique and oblivious transfer. They use Cuckoo hash technique to hash receiver's set Y into m_c bins and each bin consists of one item, and hash sender's set X into m_c bins and each bin consists of ρ items. And then, they use shuffling technique to permute and share receiver's bins, in which the sender inputs a permutation π and get the shuffled shares $\{s_{\pi(1)}, \cdots, s_{\pi(m_c)}\}$, and the receiver inputs its bins $\{a_1, \cdots, a_{m_c}\}$ and gets another shuffled shares $\{s_{\pi(1)} \oplus a_{\pi(1)}, \cdots, s_{\pi(m_c)} \oplus a_{\pi(m_c)}\}$. That is, for same bin i, if $x_{\pi(i)} \oplus s_{\pi(i)} = s_{\pi(i)} \oplus a_{\pi(i)}$, $x_{\pi(i)}$ belongs to Y. Then, the sender and receiver run mp-OPRF to compute PRF values of $x_{\pi(i)} \oplus s_{\pi(i)}$ and $s_{\pi(i)} \oplus a_{\pi(i)}$. For each bin, the sender sends its PRF values to the receiver. And the receiver can test whether the item belongs to the union. Finally, two parties runs OT protocol to let the receiver the set union. Their protocol requires $O(n \log n)$ communication and $O(n \log n)$ computation, where $|X| = |Y| = n$.

Zhang et al. [32] recently give a generic framework of PSU based on multi-query reverse private membership test (mq-RPMT) and OT. In the mq-RPMT, the sender inputs $X = \{x_1, \cdots, x_n\}$ and get nothing, and the receiver inputs $Y = \{y_1, \cdots, y_n\}$ and gets a bit vector $\mathbf{b} \in \{0, 1\}^n$, satisfying $b_i = 1$ if and only if $x_i \in Y, i \in [n]$. And then two parties runs OT protocol to let the receiver the set union. They give two concrete PSU protocols based on symmetric-key encryption and general 2PC techniques, and re-randomizable public-key encryption techniques respectively. Both constructions lead to PSU with linear computation $O(n)$ and communication $O(n)$ complexity.

uPSI protocol. To our knowledge, Chen et al. [9] propose the first unbalanced PSI based on FHE. In their uPSI [9], the sender $\mathcal{S}$ inputs a set X and the receiver $\mathcal{R}$ inputs a set Y, where $m = |Y| \ll n = |X|$. They first give a strawman protocol as follows: the receiver $\mathcal{R}$ encrypts its item y_i, $i \in [|Y|]$ and send all ciphertexts $c_i \leftarrow \text{FHE.}\mathsf{Enc}(y_i)$ to the sender $\mathcal{S}$; $\mathcal{S}$ chooses random non-zero plaintexts r_i and homomorphically computes $r_i \cdot \Pi_{x \in X}(c - x)$ and gets new ciphertexts $c_i' \leftarrow \text{FHE.}\mathsf{Enc}(r_i \cdot f(y_i))$, $i \in [|Y|]$, where the polynomial $f(y) = \Pi_{x \in X}(y - x)$, and then returns c_i' to $\mathcal{R}$; $\mathcal{R}$ decrypts all ciphertexts c_i': if $r_i \cdot f(y_i) = 0$, 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 a high computational cost and a deep homomorphic circuit.

Then they use cuckoo hashing, batching, windowing, partitioning, modulus switching, etc to optimize the strawman protocol and give a fast uPSI. More precisely, $\mathcal{R}$ inserts set Y into the Cuckoo hash table and denotes the filled Cuckoo hash table as Y_c and the item in i-th bin as y_i, $i \in [m_c]$ and each bin consists of one item. $\mathcal{S}$ inserts set X into the simple hash table and denotes the set of items in the i-th bin as $X_b[i]$ and each bin consists of B items. For same bin i, if $y_i \in X_b[i]$, $y_i \in X \cap Y$, else, $y_i \notin X \cap Y$. Then the sender partitions each bin $X_b[i]$ into α subsets. For i-th bin, each subset consists of $B' = B/\alpha$ items. Therefore, the large (n, m)-PSI can be divided into many small $(B', 1)$-PSI, and each small PSI has low homomorphic circuit and low computational cost. Moreover, both parties can compress their strings with an agreed-upon hash function to a fixed length, so this uPSI supports for arbitrary length items. Chen et al. [7] and Cong et al. [10] based on the above framework and give fast labeled unbalanced PSI.

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

2 Overview of Our Techniques

We start with a special case and give our basic uPSU protocol based on leveled FHE. Then, we develop a new cryptographic protocol named permute matrix private equality test (pm-PEQT). Finally, by instantiating our pm-PEQT and using some optimization techniques on the basic uPSU protocol, we give a secure and fast uPSU protocol that satisfies the ideal functionality of PSU in Figure 1.

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 sender $\mathcal{S}$, and an input $Y = \{y_1, y_2, \cdots, y_n\} \subset \{0,1\}^*$ from receiver $\mathcal{R}$.
2. Give output $X \cup Y$ to the receiver $\mathcal{R}$.

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

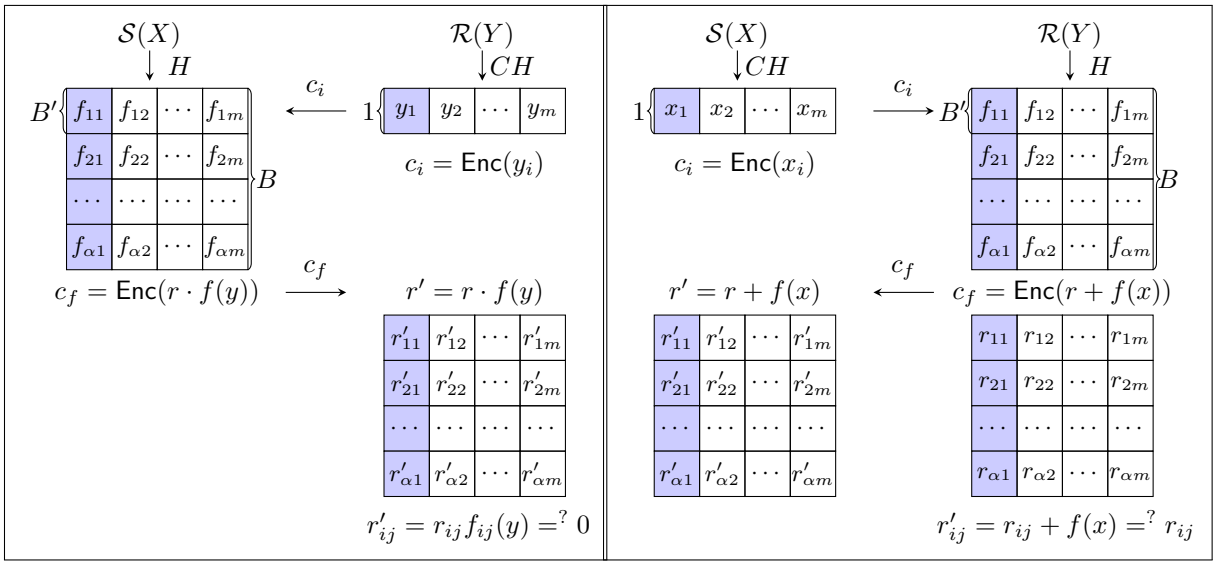

Fig. 2: Comparison of uPSI [9] (left) and our basic uPSU with optimizations (right)

2.1 Our basic uPSU based on FHE

Suppose that the sender has only one item x in its set X and the receiver holding the set Ygets the resulting union $\{x\} \cup Y$. We show our basic uPSU based on FHE as follows: The sender $\mathcal{S}$ uses its FHE public key to encrypt x and sends $c = \text{FHE.}\mathsf{Enc}(x)$ to the receiver $\mathcal{R}$; $\mathcal{R}$ chooses random non-zero number r, and homomorphically computes $r + \Pi_{y \in Y}(c - y)$, and returns the new ciphertext $c' = \text{FHE.}\mathsf{Enc}(r + \Pi_{y \in Y}(x - y))$ to $\mathcal{S}$; $\mathcal{S}$ decrypts c' and get $r' = r + \Pi_{y \in Y}(x - y)$, then it returns r' back to $\mathcal{R}$; $\mathcal{R}$ checks $r' =^? r$, if $r' = r$, it sets $b = 0$ denoted $x \in X \cap Y$, else $b = 1$ denoted $x \notin X \cap Y$. Finally, the receiver and the sender invoke the OT protocol to let the receiver obtain and output the union $\{x\} \cup Y$.

Compare with the basic uPSI [9]. The key different step between our basic uPSU and the basic uPSI [9] is using the different randomization methods. In the uPSI [9], they compute the product of randomness r and the polynomial value $f(y)$, where $f(y) = \Pi_{x \in X}(y - x)$. If $f(y) = 0$, $rf(y) = 0$ denotes $y \in X$, else $y \notin X$, and the receiver only gets a randomness $rf(y) \neq 0$ which hides the information of X. In our uPSU, we compute the sum of randomness r and the polynomial value $f(x)$, the sender decrypts the ciphertext and gets the plaintext $r + f(x)$ which hides the information of Y. Then the sender sends the plaintext $r + f(x)$ to the receiver, if $f(x) = 0$, $r + f(x) = r$ denotes $x \in Y$, else $x \notin Y$, and the receiver can get $f(x)$. This method 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.

In the uPSI, Chen et al. [9] uses some optimization techniques to divide a large (n, m)-PSI into many small $(B', 1)$-PSI to reduce the depth of homomorphic circuit. Intuitively, we can use same optimization techniques in our basic uPSU to develop a efficient full uPSU protocol. We emphasize that, unlike PSI, using above optimization techniques is not very natural for PSU. This is because a large PSI can be divided into many small PSI, and the receiver can combine all small set intersection into the output securely. However, in the PSU, a large (n, m)-PSU is divided into many small $(B', 1)$-PSU directly, this leads to information leakage about the items in the set intersection. The receiver knows that some subsets with size B' have items in $X \cap Y$. In the ideal (n, m)-PSU, from the view of receiver, any item in the set Y could be an item in $X \cap Y$. Furthermore, if the sender returns its decrypted results back directly for the receiver checks which items of X belong to the union like the basic uPSU, this will leaks the information of $X \cap Y$, because there are many subset with size B' in one bin, if corresponding polynomial $f(x) = 0$ of one subset, the receiver can get other subset in same bin, such that $f'(x) \neq 0$ which leaks the information of $X \cap Y$, even the receiver can compute the intersection items, if the number of polynomial values is sufficient. Therefore, the sender cannot sends its decrypted results to the receiver directly, meanwhile, the receiver checks the items are equal without knowing the position information. We show the comparison of uPSI [9] and our basic uPSU with optimizations in Figure 2.

2.2 Permute Matrix Private Equality Test

We develop a new cryptographic protocol named permute matrix private equality test (pm-PEQT) and give corresponding tailored efficient construction, which we believe to be of independent interest. The pm-PEQT is related to the traditional private equality test (PEQT), which is a two-party protocol in which a receiver who has an input string x interacts with a sender holding an input string y. The result is that the receiver learns a bit indicating whether $x = y$ and nothing else, whereas the sender learns nothing. In the pm-PEQT, the sender with a matrix $\mathbf{R}'_{\alpha \times m}$ and a 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 the vector $\mathbf{R}$. Compare with the private equality test (PEQT), pm-PEQT can provide matrix private equality test with **positions permutation**. We show the ideal functionality of pm-PEQT in Figure 3.

This seemingly simple functionality adjustment (PEQT $\rightarrow$ pm-PEQT) doesn't seem to be fixable by a small tweak of parallel many PEQT with permutation. This is because it is difficult to permute the receiver's item with the permutation of the sender.

Instantiation of pm-PEQT. We give two constructions of pm-PEQT. The first construction is based on Permute + Share functionality [14,21] and mp-OPRF [28,6]. We review the Permute + Share in Figure 5. $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal Permute + Share functionality $\mathcal{F}_{\text{PS}}$ twice: first, both parties permute and share the columns of $\mathbf{R}$, where each column of $\mathbf{R}$ can be seen as an item. $\mathcal{R}$ inputs each column $\mathbf{r_j}$, $j \in [m]$ 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 $s_{\pi_r(ij)} \oplus s'_{\pi_r(ij)} = s_{\pi_c(ij)}$. $\mathcal{R}$ gets the shuffled matrix shares $\mathbf{S}_{\pi} = \mathbf{S}_{\pi_r}$

Parameters: Two parties: P_1 with a matrix R, P_0 with a matrix R' 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,

$$\mathbf{R} = \begin{bmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{\alpha 1} & \cdots & r_{\alpha m} \end{bmatrix}, \mathbf{R}' = \begin{bmatrix} r'_{11} & \cdots & r'_{1m} \\ \vdots & \ddots & \vdots \\ r'_{\alpha 1} & \cdots & r'_{\alpha m} \end{bmatrix}$$

Functionality:

1. Wait for an input $\mathbf{R}'$ and a permutation π from P_0, and an input $\mathbf{R}$ from P_1.
2. Give the bit matrix $\mathbf{B}$ to P_1, where

$$\mathbf{B} = \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{\alpha 1} & \cdots & b_{\alpha m} \end{bmatrix},$$

if $r_{\pi(ij)} = r'_{\pi(ij)}$, $b_{ij} = 1$, else, $b_{ij} = 0$, for $i \in [\alpha], j \in [m]$.

Fig. 3: permute matrix private equality test $\mathcal{F}_{\text{pm-PEQT}}$

and $\mathcal{S}$ gets the shuffled matrix shares $\mathbf{S}'_\pi = \pi_r(\mathbf{S}'_{\pi_c}) \oplus \mathbf{S}'_{\pi_r}$, where $s_{\pi(ij)} \oplus s'_{\pi(ij)} = \pi(r_{ij})$, $i \in [\alpha], j \in [m]$. Then, $\mathcal{R}$ acts as P_0 with shuffled shares $\mathbf{S}$, and obtains the outputs $F_k(s_{\pi(ij)})$, $i \in [\alpha], j \in [m]$, and $\mathcal{S}$ obtain the key k. Furthermore, $\mathcal{S}$ permute 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, $b_{ij} = 0$, and gains a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha], j \in [m]$. We note that Permute + Share [14,21] and mp-OPRF [28,6] are fast cryptographic tools, and the communication overhead of our pm-PEQT based on Permute + Share and mp-OPRF is equal to $O(m \log m)$.

The second construction is based on DDH. $\mathcal{R}$ and $\mathcal{S}$ choose random number a, b and compute $H_{ij} = H(r_{ij})^a, H'_{ij} = H(r'_{ij})^b$ for $i \in [\alpha]$, $j \in [m]$, where $H = H(\cdot)$ are (multiplicative) group elements output by hash functions H. $\mathcal{R}$ sends $H_{ij} = H(r_{ij})^a$ to $\mathcal{S}$. Then $\mathcal{S}$ computes $H''_{ij} = (H(r_{ij})^a)^b$ and uses the permutation $\pi = (\pi_c, \pi_r)$ and computes $H''_{\pi(ij)} = \pi(H''_{ij})$, $H'_{\pi(ij)} = \pi(H'_{ij})$ and sends them to $\mathcal{R}$. Finally, $\mathcal{R}$ set $b_{ij} = 1$, if $H''_{\pi(ij)} = H'^a_{\pi(ij)}$, else $b_{ij} = 0$, and gains a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha]$, $j \in [m]$. We note that the communication overhead of our pm-PEQT based DDH is equal to $O(m)$.

2.3 Our Full uPSU protocol

We start with our basic uPSU protocol based on FHE and using optimization techniques [9,7,10] to divide a large PSU into many small PSU to reduce the depth of homomorphic circuit. Then, by using pm-PEQT and OT protocol, we can obtain secure and fast uPSU protocols that is secure against semi-honest adversaries. We note that the communication cost of our basic uPSU protocol with optimization is $O(|Y| \log |X|)$, the communication cost of pm-PEQT is $O(|Y| \log |Y|)$ (based on Permute + Share [14,21] and mp-OPRF [28,6]) or $O(|Y|)$ (based on DDH), and the communication cost of OT is $O(|Y|)$. Therefore, our full uPSU requires the communication cost $O(|Y| \log |X|)$. We provide the high-level technical overview for our framework of uPSU in Figure 4 and the details are as follows.

First, we use basic uPSU protocol with optimizations to divide the large PSU, where the sender inputs a small set X and the receiver inputs a large set Y. As a result, the receiver outputs a matrix $\mathbf{R}_{\alpha \times m}$, and the sender outputs a matrix $\mathbf{R}'_{\alpha \times m}$, where α denotes the number of partitions, m denotes the number of bins, r_{ij} denotes the random number used to hiding each small subset and r'_{ij} denotes the plaintext. We note that if for all $i \in [\alpha]$, $r'_{ij} \neq r_{ij}$, $x_j \notin Y$, else, $x_j \in Y$.

Then, by using pm-PEQT, the sender inputs $\mathbf{R}'$ and a permutation $\pi = (\pi_c, \pi_r)$ and the receiver inputs $\mathbf{R}$. As a result, the receiver 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

$r_{\pi(ij)} \neq r'_{\pi(ij)}$. The receiver set a bit vector $\mathbf{b} = [b_j]$, $j \in [m]$, if for all $i \in [\alpha]$, $b_{ij} = 0$, $b_{\pi(j)} = 1$, else, $b_j = 0$. The sender permutes the set X by π_c and gets $\pi_c(\mathbf{X}) = [x_{\pi_c(1)}, x_{\pi_c(2)}, \cdots, x_{\pi_c(m)}]$. We note that if $b_j = 1$, $x_{\pi_c(j)} \notin Y$, else $x_{\pi(i)} \in Y$, $j \in [m]$.

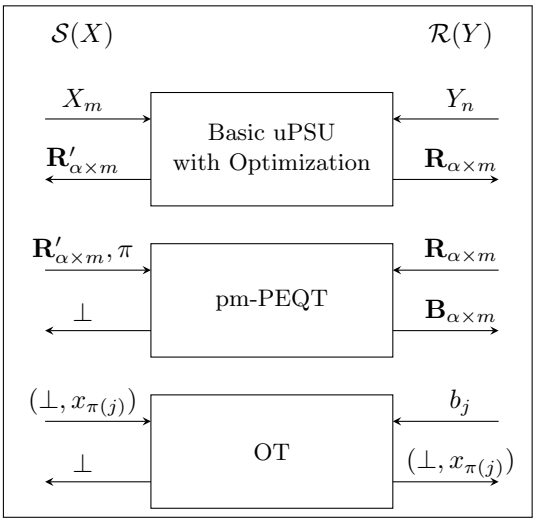

Fig. 4: Core design idea of Our full uPSU protocol

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

3 Preliminaries

3.1 Notation

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 denote the computational and statistical security parameters, respectively. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. We denote a probabilistic polynomial-time algorithm by PPT. If S is a set then $s \leftarrow S$ denotes the operation of sampling an item s of S at random, and S^T denotes its transpose. For any permutations π on n items, we set $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\} = \pi(\{s_1, s_2, \cdots, s_n\})$, where $s_{\pi(i)}$ denotes the i-th element after the permutation. For any column permutations π_c on a matrix $\mathbf{S} = [s_{ij}]$, we set $\mathbf{S}_{\pi_c} = \pi_c(\mathbf{S}) = [s_{\pi_c(ij)}]$, where $s_{\pi_c(ij)}$ denotes the i-th row and j-th column element after the permutation. For any row permutations π_r on a matrix $\mathbf{S} = [s_{ij}]$, we set $\mathbf{S}_{\pi_r} = \pi_r(\mathbf{S}) = [s_{\pi_r(ij)}]$, where $s_{\pi_r(ij)}$ denotes the i-th row and j-th column element after the permutation. We denote the parties as the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, and their respective input sets as X and Y, set sizes $|X|$ and $|Y|$. In the unbalanced setting, we assume that $|X| \ll |Y|$.

3.2 Hashing

As mentioned in [9], two parties hash the items in their sets into two hash tables using some agreed-upon deterministic hash function, and they only perform a PSI for each bin, since items in different bins are necessarily different. We also use some hash techniques in our PSU and review them here.

Simple Hashing. There are some hash functions $H_1, \cdots, H_h : \{0,1\}^* \to [m]$ used to map n items into m bins $\mathbf{B}_1, \cdots, \mathbf{B}_m$. Following [26], 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.

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

By using simple hash, both parties can compress their items with an agree-upon hash function to a fixed length, and execute the PSU protocol on these hashed strings.

Cuckoo hashing. Cuckoo hashing [27,11,13] can be used to build dense hash tables by many hash functions. There are h hash functions $H_1, \cdots, H_h$ used to map n items into $m = \epsilon n$ bins and a stash, where each bin at most one item. For an item x, we choose a random index i from $[h]$, and insert the tuple (x, i) at location $H_i(x)$ in the table. If this location was already occupied by a tuple (y, j), we replace (y, j) with (x, i), choose a random j' from $[h]\backslash\{j\}$, and recursively re-insert (y, j') into the table. The above procedure is repeated until no more evictions are necessary, or until the number of evictions has reached a threshold. In the latter case, the last item will be put in the stash. According to the analysis in [29], we can adjust the values of m and ϵ to reduce the stash size to 0 while achieving a hashing failure probability of $2^{-\lambda}$.

Note that following [9], we also let the sender perform cuckoo hashing with $m \geq |X|$ bins. The receiver inserts each of its items into a two-dimensional hash table using three hash functions.

3.3 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) [15] is a form of encryption schemes that allow arbitrary operations to be performed on encrypted data without requiring access to the decryption key. For improved performance, the encryption parameters are typically chosen to support only circuits of a certain bounded depth (leveled fully homomorphic encryption), and we use this in our implementation following [9,7,10]. There are several FHE implementations that are publicly available. We use the homomorphic encryption library SEAL, which implements the variant of [1] of the Brakerski/FanVercauteren (BFV) scheme [12]. We also need some optimization techniques of FHE following [9,7,10], such as batching, windowing, partitioning, modulus switching, etc, and review them here.

Batching. Batching is a well-known and powerful technique in fully homomorphic encryption to enable SIMD (Single Instruction, Multiple Data) operations on ciphertexts [16,2,31,8,17]. The batching technique allows the sender to operate on n items from the receiver simultaneously, resulting in n-fold improvement in both the computation and communication. Since in typical cases n has size of several thousand, this results in a significant improvement over the basic protocol.

Windowing. We use a standard windowing technique to lower the depth of the arithmetic circuit that the sender needs to evaluate on the receiver's homomorphically encrypted data, resulting in a valuable computation-communication trade-off.

If the sender only has an encryption of y, it samples a random element r in $\mathbb{Z}_t\backslash\{0\}$ and homomorphically computes $r + \Pi_{x \in X}(y - x)$. The sender needs to compute at worst the product $y^{|X|}$, which requires a circuit of depth $\lceil \log_2(|X| + 1) \rceil$. If the receiver sends encryptions of extra powers of y, the sender can use these powers to evaluate the same computation with a much lower depth circuit. More precisely, for a window size of l bits, the receiver computes and sends $c(i, j) = \text{FHE.}\mathsf{Enc}(y^{i \cdot 2^{lj}})$ to the sender for all $1 \leq i \leq 2^l - 1$, and all $0 \leq j \leq \lfloor \log_2(|X|)/l \rfloor$. For example, when $l = 1$, the receiver sends encryptions of $y, y^2, y^4, \cdots, y^{2^{\lfloor \log_2 |X| \rfloor}}$. This technique results in a significant reduction in the circuit depth. To see this, we write

$$r + \Pi_{x \in X}(y - x) = r + a_0 + a_1 y + \cdots + a_{|X|-1} y^{|X|-1} + y^{|X|}.$$

The cost of windowing is in increased communication. The communication from the receiver to the sender is increased by a factor of $(2^l - 1)(\lfloor \log_2(|X|)/l \rfloor + 1)$, and the communication back from the sender to the receiver does not change.

Partitioning. Another way to reduce circuit depth is to let the sender partition its set into α subsets. In the basic protocol, this reduces sender's circuit depth from $\lceil \log_2(|X|+1) \rceil$ to $\lceil \log_2(|X|/\alpha+1) \rceil$, at the cost of increasing the return communication from sender to receiver by a factor of α. In the PSU, the sender needs to compute encryptions of all powers $y, \cdots, y^{|X|}$ for each of the receiver's items y. With partitioning, the sender only needs to compute encryptions of $y, \cdots, y^{|X|/\alpha}$, which it can reuse for each of the α partitions. Thus, with both partitioning and windowing, the sender's computational cost reduces by a factor of α.

Modulus switching. We can employ modulus switching [3], which effectively reduces the size of the response ciphertexts. Modulus switching is a well-known operation in lattice-based fully homomorphic encryption schemes. It is a public operation, which transforms a ciphertext with encryption parameter q into a ciphertext encrypting the same plaintext, but with a smaller parameter $q' < q$. As long as q' is not too small, correctness of the encryption scheme is preserved. Note that the security of the protocol is trivially preserved as long as the smaller modulus q' is determined at setup.

3.4 Building Blocks

Permute + Share. We recall the Permute + Share (PS) functionality $\mathcal{F}_{\text{PS}}$ defined by Chase et al. [5] in Figure 5. Roughly speaking, in the Permute + 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 learn the shares $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\}$ and P_1 learn nothing but the other shares $\{x_{\pi(1)} \oplus s_{\pi(1)}, x_{\pi(2)} \oplus s_{\pi(2)}, \cdots, x_{\pi(n)} \oplus s_{\pi(n)}\}$. As mentioned in [5], some works [19,25] can also be used to realize $\mathcal{F}_{\text{PS}}$.

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 $\{x_{\pi(1)} \oplus s_{\pi(1)}, x_{\pi(2)} \oplus s_{\pi(2)}, \cdots, x_{\pi(n)} \oplus s_{\pi(n)}\}$ to P_1.

Fig. 5: Permute + Share functionality $\mathcal{F}_{\text{PS}}$

Multi-Point Oblivious Pesudorandom Function. An oblivious pesudorandom 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. [28] proposes multi-point OPRF (mp-OPRF) and realizes efficient PSI protocols. Recently, Chase et al. [6] develop a more efficient mp-OPRF based on oblivious transfer (OT) extension. In the mp-OPRF, P_0 inputs $\{x_1, x_2, \cdots, x_n\}_{n>1}$ and learns all PRF values $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_n)\}$, and P_1 gets the key k. We recall the mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$ in Figure 6.

Oblivious Transfer. Oblivious Transfer (OT), introduced by Rabin [30] 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. [20] introduced 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: A PRF F, and two parties: P_0 and P_1;
Functionality:

1. Wait for input $\{x_1, \cdots, x_n\}$ from P_0
2. Sample a random PRF seed k and give it to P_1. Give $\{F_k(x_1), F_k(x_2), \cdots, F_k(x_n)\}$ to P_1.

Fig. 6: mp-OPRF functionality $\mathcal{F}_{\text{mp-OPRF}}$

Parameters: Two parties: P_0 and P_1.
Functionality:

1. Wait for input $\{x_0, x_1\}$ from P_0; Wait for input $b \in \{0, 1\}$ from P_1;
2. Give x_b to P_1.

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

4 The Basic Protocol

We describe our basic uPSU protocol in Figure 8 as a strawman protocol. The sender encrypts each of its items x, and sends the ciphertexts $c = \text{FHE.}\mathsf{Enc}_{pk}(x)$ to the receiver. For each y_i, the receiver then evaluates homomorphically the product of differences of x with all of the receiver's items y_i (computing a function $f = (x - y_1) \cdots (x - y_{|Y|})$, s.t. $f(y_i) = 0$ for all $y_i \in Y$), randomizes the product by adding it with differences uniformly random non-zero plaintext r, and sends the ciphertext $c' = \text{FHE.}\mathsf{Enc}_{pk}(f(x) + r)$ back to the sender. The sender decrypts c' to $r + f(x)$ and sends $r + f(x)$ to the receiver. The receiver checks the results, if $r + f(x) = r$, $x \in Y$, otherwise, $x \notin Y$. Finally, The receiver and the sender invoke OT protocol to let the receiver get the union. This method 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.

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 $X \cup Y$ and the sender outputs $\perp$.

1. [**Setup**] Both parties agree on a fully homomorphic encryption scheme: $\mathcal{S}$ generates a public-secret key pair for the scheme and keeps the secret key itself.
2. [**Set encryption**] $\mathcal{S}$ encrypts each item $x_i \in X$, $c_i = \text{FHE.}\mathsf{Enc}(x_i), i \in [|X|]$ 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 $c'_i = \text{FHE.}\mathsf{Enc}(f(x_i) + r_i), i \in [|X|]$ where for all $y \in Y$, s.t. $f(y) = 0$.
   (c) sends $c'_i, i \in [|X|]$ to $\mathcal{S}$.
4. [**Decryption**] $\mathcal{S}$ decrypts $c'_i, i \in [|X|]$ to $m_i = f(x_i) + r_i$ and sends them to $\mathcal{R}$.
5. [**Output**] $\mathcal{R}$ checks all plaintexts and sets a bit vector $B = [b_i]$, $i \in [|X|]$. If $m_i = r_i$, $i \in [|X|]$, $x_i \in Y$ and sets $b_i = 0$, otherwise, $x_i \notin Y$ and sets $b_i = 1$. $\mathcal{R}$ inputs the bit vector B and $\mathcal{S}$ inputs $(x_i, \perp)$, $i \in [|X|]$, then both parties invoke the ideal functionality $\mathcal{F}_{OT}$. For $i \in [|X|]$, $\mathcal{R}$ gets x_i, if $b_i = 1$, else gets $\perp$. Finally, $\mathcal{R}$ outputs $X \cup Y$.

Fig. 8: Basic uPSU protocol

We proceed to show the semi-honest security of the basic uPSU protocol in Figure 8 in the $\mathcal{F}_{\text{OT}}$-hybrid model.

Theorem 1. *The uPSU protocol described in Figure 8 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.

Corrupt Sender. $\text{Sim}_{\mathcal{S}}(X)$ simulates the view of corrupt $\mathcal{S}$ by encrypting $|X|$ randomness to simulate $|X|$ ciphertexts. As for the $\mathcal{F}_{\text{OT}}$, $\mathcal{A}$ does not need to obtain outputs from it, thus $\text{Sim}_{\mathcal{S}}$ does nothing. Since the fully homomorphic encryption scheme satisfies the circuit privacy, above 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: $\text{Sim}_{\mathcal{R}}$ computes $\hat{X} = X \cup Y \backslash X$ and randomly chooses $z = |X| - |\hat{X}|$ items $y_{i_1}, \cdots, y_{i_z} \in Y$ to pad $\hat{X}$ to $|X|$ items and permutes these items randomly. Then $\text{Sim}_{\mathcal{R}}$ inputs $\hat{X}$ to run the real protocol. $\text{Sim}_{\mathcal{R}}$ encrypts all items in $\hat{X}$ and sends the ciphertexts to $\mathcal{A}$. It waits for new ciphertexts from $\mathcal{A}$ and decrypts them and sends the plaintexts back. When receiving the input b_i of $\mathcal{F}_{\text{OT}}$ from $\mathcal{A}$, if $b_i = 0$, $\text{Sim}_{\mathcal{R}}$ sends $\hat{x}_i$ to $\mathcal{A}$, otherwise, it sends $\perp$.

We argue that the outputs of $\text{Sim}_{\mathcal{R}}$ 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 first round are replaced by by $\hat{c}_i = \text{FHE.}\mathsf{Enc}(\hat{x}_i), i \in [|X|]$, generated by $\text{Sim}_{\mathcal{R}}$. Since the fully homomorphic encryption scheme is IND-CPA secure, above simulation is indistinguishable from the real view.
Hyb_2: Same as Hyb_1 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 $\text{Sim}_{\mathcal{R}}$.

5 Permute Matrix Private Equality Test

Our pm-PEQT protocol is described in Figure 3. The formal protocol follows the intuition presented in the first part of Section 2.2. We describe two efficient instantiations of pm-PEQT, which is a semi-honest secure protocol for the functionality specified in Figure 3. And then, we prove the security properties of the protocol.

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

We give the first construction of pm-PEQT from the $\mathsf{Permute} + \mathsf{Share}$ [14,21] and mp-OPRF [6] in Figure 9, and its communication overhead is equal to $O(m \log m)$.

Theorem 2. *The construction of Figure 9 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 $\text{Sim}_{\mathcal{R}}$ and $\text{Sim}_{\mathcal{S}}$ for simulating corrupt $\mathcal{R}$ and $\mathcal{S}$ respectively, and argue the indistinguishability of the produced transcript from the real execution.

Corrupt Sender. $\text{Sim}_{\mathcal{S}}(\mathbf{R}', \pi = (\pi_c, \pi_r))$ simulates the view of corrupt $\mathcal{S}$ as follows. When receiving a permutation π_c from $\mathcal{A}$, $\text{Sim}_{\mathcal{S}}$ randomly chooses $\mathbf{S}'_{\pi_c}$ as shuffled shares, and simulates $\mathcal{F}_{\text{PS}}$ sending them to $\mathcal{A}$. When receiving a permutation π_r from $\mathcal{A}$, $\text{Sim}_{\mathcal{S}}$ randomly chooses $\mathbf{S}'_{\pi_r}$ as shuffled shares, and simulates $\mathcal{F}_{\text{PS}}$ sending them to $\mathcal{A}$. Then, $\text{Sim}_{\mathcal{S}}$ randomly selects a key k of PRF and sends it to $\mathcal{A}$ to simulate $\mathcal{F}_{\text{pm-PEQT}}$.

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

Hyb_0: $\mathcal{S}$'s view in the real protocol.

Input: The receiver $\mathcal{R}$ inputs a matrix $\mathbf{R} = [r_{ij}]$, $i \in [\alpha]$, $j \in [m]$; the sender $\mathcal{S}$ inputs a matrix $\mathbf{R}' = [r'_{ij}]$, $i \in [\alpha]$, $j \in [m]$ and a permutation $\pi = (\pi_c, \pi_r)$ where π_c (over $[m]$) and π_r (over $[\alpha]$).
Output: The receiver outputs a bit matrix $\mathbf{B}$; the sender outputs $\perp$.

1. [Permute + Share functionality] $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal Permute + Share functionality $\mathcal{F}_{\text{PS}}$ twice: first, both parties permute and share the columns of $\mathbf{R}$, where each column of $\mathbf{R}$ can be seen as an item. $\mathcal{R}$ inputs each column $\mathbf{r}_j$, $j \in [m]$ 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 $s_{\pi(ij)} \oplus s'_{\pi(ij)} = \pi(r_{ij})$, $i \in [\alpha], j \in [m]$.
2. [mp-OPRF functionality] $\mathcal{R}$ acts as P_0 with shuffled shares $\mathbf{S}_\pi$, and obtains the outputs $F_k(s_{\pi(ij)})$, $i \in [\alpha], j \in [m]$, and $\mathcal{S}$ 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 gains a bit matrix $\mathbf{B} = [b_{ij}]$, $i \in [\alpha], j \in [m]$.

Fig. 9: pm-PEQT from Permute + Share and mp-OPRF

Hyb_1: Same as Hyb_0 except that the output of $\mathcal{F}_{\text{PS}}$ is replaced by $\mathbf{S}'_{\pi_c}$, $\mathbf{S}'_{\pi_r}$ chosen by $\text{Sim}_{\mathcal{S}}$, and $\text{Sim}_{\mathcal{S}}$ runs the $\mathcal{F}_{\text{PS}}$ simulator to produce the simulated view for $\mathcal{S}$. The security of Permute + 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 $\text{Sim}_{\mathcal{S}}$, and $\text{Sim}_{\mathcal{S}}$ 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 $\text{Sim}_{\mathcal{S}}$.

Corrupt Receiver. $\text{Sim}_{\mathcal{R}}(\mathbf{R}, \mathbf{B} = [b_{ij}])$ simulates the view of corrupt $\mathcal{R}$ as follows. When receiving all columns of $\mathbf{R}$ from $\mathcal{A}$, $\text{Sim}_{\mathcal{R}}$ randomly chooses $\mathbf{S}_{\pi_c}$ as shuffled shares, and simulates $\mathcal{F}_{\text{PS}}$ sending them to $\mathcal{A}$. When receiving all rows of $\mathbf{S}_{\pi_c}$ from $\mathcal{A}$, $\text{Sim}_{\mathcal{R}}$ randomly chooses $\mathbf{S}_{\pi_r}$ as shuffled shares, and simulates $\mathcal{F}_{\text{PS}}$ sending them to $\mathcal{A}$. When receiving $\mathbf{S}_{\pi_r}$, $\text{Sim}_{\mathcal{R}}$ randomly selects a key k of PRF and sends $F_k(s_{\pi(ij)})$, $i \in [\alpha], j \in [m]$ to $\mathcal{A}$. Finally, $\text{Sim}_{\mathcal{R}}$ sets $v_{\pi(ij)} = F_k(s_{\pi(ij)})$ if $B_{ij} = 1$, else, it chooses $v_{\pi(ij)}$ randomly and it sends all $v_{\pi(ij)}$, $i \in [\alpha]$, $j \in [m]$ to $\mathcal{A}$.

The view generated by $\text{Sim}_{\mathcal{R}}$ in 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 $\mathcal{F}_{\text{PS}}$ is replaced by $\mathbf{S}_{\pi_c}$, $\mathbf{S}_{\pi_r}$ chosen by $\text{Sim}_{\mathcal{R}}$, and $\text{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{PS}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of Permute + 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 $\text{Sim}_{\mathcal{R}}$, and $\text{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{pm-PEQT}}$ simulator to produce the simulated view for $\mathcal{R}$. The security of mp-OPRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol.
Hyb_3: Same as Hyb_2 except that the PRF values is replaced by the v_{ij} chosen by $\text{Sim}_{\mathcal{R}}$. The security of PRF guarantees the view in simulation is computationally indistinguishable from the view in the real protocol. The hybrid is the view output by $\text{Sim}_{\mathcal{R}}$.

5.2 pm-PEQT based on DDH

We give the second construction of pm-PEQT based on DDH described in Figure 10. We note that the communication overhead of our DDH-based pm-PEQT is equal to $O(m)$.

Theorem 3. *The construction of Figure 10 securely implements functionality $\mathcal{F}_{pm\text{-}PEQT}$ based on DDH in the random oracle model, in the presence of semi-honest security adversaries.*

Input: The receiver inputs a matrix $\mathbf{R} = [r_{ij}]$, $i \in [\alpha]$, $j \in [m]$; the sender inputs a matrix $\mathbf{R}' = [r'_{ij}]$, $i \in [\alpha]$, $j \in [m]$ and a permutation $\pi = (\pi_c, \pi_r)$ where π_c (over $[m]$) and π_r (over $[\alpha]$).
Output: The receiver outputs a bit matrix $\mathbf{B}$; the sender outputs $\perp$.

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

Fig. 10: Instantiation of pm-PEQT based on DDH

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 view in Hyb_0 and Hyb_1 are computationally indistinguishable. Let $\mathcal{A}$ be a 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 randomness $a = x$, and simulates (with the knowledge of $\mathbf{R}$) the view as below:

- RO queries: $\mathrm{Sim}_{\mathcal{S}}$ honestly emulates random oracle H. For every query r_{ij}, if $r_{ij} \notin \mathbf{R}$, it picks a random group element t_{ij} and assigns $H(r_{ij}) = t_{ij}$. If $r_{ij} \in \mathbf{R}$, 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 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. $\mathrm{Sim}_{\mathcal{R}}(\mathbf{R}, \mathbf{B})$ 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 $u_{ij} = v^a_{ij}$ 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 $H(r'_{\pi(ij)})^b$, $H(r_{\pi(ij)})^{ab}$, $i \in [\alpha]$, $j \in [m]$, 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 $H(r'_{\pi(ij)})^b$, $H(r_{\pi(ij)})^{ab}$.
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 $u_{ij} = v^a_{ij}$, $i \in [\alpha]$, $j \in [m]$ instead of $H(r'_{\pi(ij)})^b$, $H(r_{\pi(ij)})^{ab}$. 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 $g^x, g^{y_{ij}}, g^{z_{ij}}$, and $g^{y'_{ij}}, g^{z'_{ij}}$ 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 randomness $b = x$, 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 every query $r_{\pi(ij)}$ and $r'_{\pi(ij)}$, if $r_{\pi(ij)} \notin \mathbf{R}$, $r'_{\pi(ij)} \notin \mathbf{R}'$, it picks a random group element $t_{\pi(ij)}$, $t'_{\pi(ij)}$, and assigns $H(r_{\pi(ij)}) = t_{\pi(ij)}$, $H(r'_{\pi(ij)}) = t'_{\pi(ij)}$. If $r_{\pi(ij)} \in \mathbf{R}$, $r'_{\pi(ij)} \in \mathbf{R}'$, it assigns $H(r_{\pi(ij)}) = g^{y_{\pi(ij)}}$, $H(r'_{\pi(ij)}) = g^{y'_{\pi(ij)}}$.
– Outputs $g^{a \cdot z_{\pi(ij)}}$, $g^{z'_{\pi(ij)}}$.

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 all RO 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 randomness $b = x$, 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 every query $r_{\pi(ij)}$, $r'_{\pi(ij)}$, if $r_{\pi(ij)} \notin \mathbf{R}$, $r'_{\pi(ij)} \notin \mathbf{R}'$, it picks a random group element $t_{\pi(ij)}$, $t'_{\pi(ij)}$, and assigns $H(r_{\pi(ij)}) = t_{\pi(ij)}$, $H(r'_{\pi(ij)}) = t'_{\pi(ij)}$. If $r_{\pi(ij)} = r'_{\pi(ij)} \in \mathbf{R}$, it assigns $H(r_{\pi(ij)}) = H(r'_{\pi(ij)}) = g^{y_{\pi(ij)}}$.
– Outputs $g^{a \cdot z_{\pi(ij)}}$, $g^{z_{\pi(ij)}}$.

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 all RO 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.

6 Full Unbalanced PSU and Security Proof

In this section, We start from our basic uPSU protocol described in Figure 8 and use some optimization techniques following [9,7,10] to reduce the homomorphic circuits, and then we give a full uPSU based on pm-PEQT 6 and OT protocol 7.

6.1 Full uPSU Protocol

We detail our full uPSU protocol in Figure 11, given a secure fully homomorphic encryption scheme with circuit privacy and secure pm-PEQT and OT protocols.

In the setup phase, the sender and the receiver agree on the hashing parameters and the FHE scheme parameters. Then, the sender and the receiver take advantage of the optimization techniques [7,10]to pre-process the set X and Y offline, respectively. After offline pre-processing phase, the sender and the receiver begin the efficient online phase: First, the sender sends the ciphertexts to the receiver, and the receiver homomorphically computes ciphertexts and returns them back, and then the sender decrypts the new ciphertexts and run pm-PEQT with the receiver and let the receiver obtains a bit vector which denotes the elements at the corresponding positions (with permutation) belong to the union. Last, the sender and the receiver run the OT protocol together to let the receiver obtains the union $X \cup Y$ and outputs it.

6.2 Security Proof

We prove security in the standard semi-honest simulation-based paradigm [24].

Theorem 4. *The protocol in Figure 11, is a secure protocol for $\mathcal{F}_{PSU}$ in the semi-honest setting.*

Proof. It is easy to see that the protocol correctly computes the union conditioned on the hashing succeeding, which happens with overwhelming probability $1 - 2^{-\lambda}$. For easy of exposition, we will assume that the simulator/protocol is parameterized by $(h, m, B, n, q, t, \alpha, l, \{H_i\}_{1 \le i \le h})$, which are fixed and public.

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, where each row denotes $\mathbf{y}_i = [y_{i,1}, \cdots, y_{i,m_c}], i \in [B]$.
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 $\mathbf{y}_{i,k}$, $k \in [B']$ denotes k-th row of $\mathbf{Y}_i$.
   (b) [Computing coefficients] $\mathcal{R}$ chooses a random matrix $\mathbf{R}_{\alpha \times m_c}$. 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 $y_{i,j,k}$, $k \in [B']$ denotes k-th item of $\mathbf{Y}_{i,j}$, $\mathcal{R}$ computes the coefficients of the polynomial $f_{i,j}(y) = \Pi_{k=1}^{B'}(y - y_{i,j,k}) = a'_{i,j,0} + a_{i,j,1}y + \cdots + a_{i,j,B'-1}y^{B'-1} + a_{i,j,B'}y^{B'}$, and then replaces each column $\mathbf{Y}_{i,j}$, with coefficients of the polynomial $f_{i,j}(y)$, where the leading coefficient adds a randomness in $\mathbf{R}_{\alpha \times m_c}$, to get the coefficient matrix $\mathbf{A}_{i,j} = [a_{i,j,0}, a_{i,j,1}, \cdots, a_{i,j,B'}]^T$, $i \in [\alpha], j \in [m]$, where $a_{i,j,0} = a'_{i,j,0} + r_{i,j}$.
   (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 $\hat{\mathbf{a}}_{i,j}$, $i \in [\alpha]$, $j \in [\beta]$, $\hat{\mathbf{a}}_{i,j} = [\hat{a}_{i,j,k}]$, $k \in [0, B']$.
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 $\mathbf{C}_j = [\mathbf{c}_{j,0}, \cdots, \mathbf{c}_{j,B'}]^T$, 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 $\mathbf{C}'_{i,j} = \hat{\mathbf{A}}_{i,j}\mathbf{C}_j = \sum_{k=0}^{B'} \hat{a}_{i,j,k}c_{j,k}, i \in [\alpha], j \in [\beta]$, 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**] For each $i \in [\alpha], j \in [\beta]$, $\mathcal{S}$ decrypts ciphertexts and concatenates the resulting β 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 a permutation $\pi = (\pi_c, \pi_r)$ where π_c (over $[m_c]$) and π_r (over $[\alpha]$) and the matrix $\mathbf{R}'_{\alpha \times m_c}$. As a result, $\mathcal{R}$ gains 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 protocol, in which $\mathcal{R}$ inputs b_j, $j \in [m_c]$ and $\mathcal{S}$ inputs $(X_c[\pi_c(j)], \perp)$. If $b_j = 1$, $\mathcal{R}$ gets $X_c[\pi_c(j)]$, else, it gets $\perp$. Finally, $\mathcal{R}$ outputs the set union $Y \cup X = Y \cup \{X_c[\pi_c(j)]\}$, $j \in [m_c]$.

Fig. 11: Full uPSU protocol

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. The case of a corrupt sender is straightforward. $\mathrm{Sim}_{\mathcal{S}}(X)$ simulates the view of corrupt $\mathcal{S}$ as follows. $\mathsf{Sim}_{\mathcal{S}}$ generates new encryptions of randomness in place of the encryptions in step 5. As for the $\mathcal{F}_{\text{pm-PEQT}}$ and $\mathcal{F}_{\text{OT}}$, $\mathcal{A}$ does not need to obtain outputs from them, and thus $\mathrm{Sim}_{\mathcal{S}}$ does nothing. Since the fully homomorphic encryption scheme satisfies the circuit privacy, above simulation is indistinguishable from the real view.

Corrupt Receiver. $\mathsf{Sim}_{\mathcal{R}}$ $(Y, X \cup Y)$ simulates the view of corrupt semi-honest receiver. It executes as follows: $\mathrm{Sim}_{\mathcal{R}}$ computes $X^* = X \cup Y \backslash X$ and randomly chooses $|X| - |X^*|$ items $y_{i_1}, \cdots, y_{i_z} \in Y$ to pad $\hat{X}$ to $|X|$ items and permutes these items randomly. Let $\hat{X} = \{\hat{x}_1, \cdots, \hat{x}_{|X|}\}$. Next it runs step 1-5 as real protocol. When receiving the ciphertexts (X) in step 6, it can decrypt the ciphertexts and get the plaintexts $\mathbf{R}'$. When receiving an input $\mathbf{R}$ of $\mathcal{F}_{\text{pm-PEQT}}$ from $\mathcal{A}$, $\mathsf{Sim}_{\mathcal{R}}$ chooses a random matrix permutation $\pi = (\pi_c, \pi_r)$ to permutes $\mathbf{R}$, $\mathbf{R}'$ and gets $\pi(\mathbf{R})$, $\pi(\mathbf{R}')$. It sets $b_{ij} = 1$, if $r'_{\pi(ij)} = r_{\pi(ij)}$, otherwise, sets $b_{ij} = 0$. Then, $\mathsf{Sim}_{\mathcal{R}}$ sends the bit matrix $\mathbf{B} = [b_{ij}]$ to $\mathcal{A}$. When receiving a bit b_i of $\mathcal{F}_{\text{OT}}$ from $\mathcal{A}$, $\mathsf{Sim}_{\mathcal{R}}$ permutes the X_c by π_c, if $b_i = 1$, it sends $X_c[\pi_c(i)]$, otherwise, it sends $\perp$.

The view generated by $\mathrm{Sim}_{\mathcal{R}}$ in 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 in the first round are replaced by by $\hat{c}_i = \text{FHE.}\mathsf{Enc}(\hat{x}_i), i \in [|X|]$, generated by $\mathrm{Sim}_{\mathcal{R}}$. 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 $\mathrm{Sim}_{\mathcal{R}}$, and $\mathrm{Sim}_{\mathcal{R}}$ runs the $\mathcal{F}_{\text{pm-PEQT}}$ simulator to produce the simulated view for $_{\mathcal{R}}$. The security of 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 $\mathrm{Sim}_{\mathcal{R}}$ 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 $\mathrm{Sim}_{\mathcal{R}}$.

When sender holds the larger set. We have made the assumption that the receiver's set size is much smaller than the sender's set size. Follows [9], our uPSU protocol can also handle the opposite case, where the sender holds the larger set, by switching their roles. This protocol is still secure in the semi-honest setting, and the communication remains linear in the smaller set and logarithmic in the larger set.

7 Implementation and Performance

In this section, we experimentally evaluate our two uPSU protocols $\text{uPSU}^{\text{FHE}}_{\text{PS}}$ and $\text{uPSU}^{\text{FHE}}_{\text{DDH}}$, we will refer to them as:

- $\text{uPSU}^{\text{FHE}}_{\text{PS}}$: uPSU protocol with FHE, pm-PEQT and OT where pm-PEQT are instantiated with $\mathsf{Permute} + \mathsf{Share}$ and mp-OPRF.
- $\text{uPSU}^{\text{FHE}}_{\text{DDH}}$: uPSU protocol with 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.

7.1 Experimental Setup

We run our experiments on an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz RAM 8G Ubuntu 22.04. We perform all tests using this single machine, and simulate network latency and bandwidth using the Linux tc command. Specifically, we consider a LAN setting, where the two parties are connected via 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.

Table 2: Communication (in MB) and runtime (in seconds) of $\text{uPSU}_{\text{PS}}^{\text{FHE}}$ and $\text{uPSU}_{\text{PS}}^{\text{FHE}}$ for small set X and large set Y with unrestricted bandwidth

Parameters		Protocols	Comm. size (MB)			Runtime (seconds), $T=1$				Runtime (seconds), $T=4$			
						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}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	1.963	0.216	2.179	1.347	4.600	1.347	7.294	0.682	1.971	0.682	3.336
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	2.14	0.793	2.933	1.392	3.571	1.392	6.355	0.916	1.987	0.917	3.821
	2^{20}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	2.223	0.428	2.651	2.274	2.275	33.652	38.201	1.086	1.087	12.275	14.449
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	2.45	1.8482	4.2982	2.031	2.030	34.202	38.263	1.311	1.312	11.888	14.512
	2^{22}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	3.24	1.33	4.57	5.746	5.746	159.275	170.768	2.071	2.071	56.919	61.062
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	4.11	6.82	10.93	4.315	4.333	159.466	168.114	2.140	2.142	55.377	59.659
2^{11}	2^{18}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	3.063	0.436	3.499	2.325	3.029	2.315	7.670	0.889	1.500	0.880	3.270
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	3.56	1.85	5.41	2.079	3.021	2.071	7.171	1.057	1.472	1.048	3.578
	2^{20}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	3.3435	0.4363	3.7798	3	2.984	32.585	38.569	1.206	1.196	11.141	13.543
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	3.56	1.85	5.41	2.668	2.660	32.168	37.497	1.321	1.465	11.083	13.870
	2^{22}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	4.51	1.55	6.06	6.883	6.873	157.499	171.255	2.644	2.649	51.156	56.450
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	5.69	7.64	13.33	5.363	5.357	152.236	162.957	2.411	2.416	52.404	57.231

Table 3: Communication (in MB) and runtime (in seconds) comparing our $\text{uPSU}_{\text{DDH}}^{\text{FHE}}$ and $\text{uPSU}_{\text{PS}}^{\text{FHE}}$ to PSU [21], uPSU* [21] and PSU* [22]

Parameters		Protocols	Comm. Size (MB)	Runtime (seconds)								
				10 Gbps			100 Mbps			10 Mbps		
$\lvert X\rvert$	$\lvert Y\rvert$			T=1	T=4	T=8	T=1	T=4	T=8	T=1	T=4	T=8
2^{10}	2^{18}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	2.179	7.294	3.336	3.097	10.394	6.407	6.136	10.303	6.415	5.994
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	2.933	6.355	3.821	3.310	11.992	8.242	8.132	11.997	8.258	8.015
		PSU [21]	326.313	113.184	100.600	98.621	151.469	139.903	139.930	626.365	627.132	626.731
		uPSU* [21]	117.931	43.959	11.548	7.463	57.105	25.330	21.709	226.297	193.665	188.333
		PSU* [22]	600.62	37.953	10.940	14.024	71.938	49.301	47.698	498.366	505.892	499.343
	2^{19}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	2.179	18.772	7.296	3.238	21.901	10.464	7.865	21.766	10.438	8.910
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	2.933	18.782	7.838	6.159	23.421	12.276	10.616	23.153	12.332	10.684
		PSU [21]	683.001	406.630	380.477	378.820	498.873	471.702	471.697	1499.081	1531.532	1502.059
		uPSU* [21]	117.931	69.344	18.065	11.859	83.974	31.985	24.388	251.667	200.054	199.629
		PSU* [22]	2470.1	112.409	31.804	40.653	233.787	207.859	207.862	2080.189	2077.877	2079.402
2^{11}	2^{18}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	3.499	7.670	3.270	3.238	11.254	7.175	6.898	11.163	7.121	7.007
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	5.41	7.171	3.578	3.338	12.485	9.036	8.573	11.693	9.069	8.941
		PSU [21]	326.386	113.130	100.478	98.818	151.606	139.888	140.112	626.555	628.775	627.690
		uPSU* [21]	235.966	61.717	16.215	10.383	95.090	49.959	43.907	440.023	400.135	399.277
		PSU* [22]	600.62	37.994	10.665	14.458	71.842	49.229	49.370	505.802	505.717	499.411
	2^{19}	$\text{uPSU}_{\text{DDH}}^{\text{FHE}}$	3.499	14.297	5.745	5.045	17.819	9.670	8.590	17.585	9.625	8.987
		$\text{uPSU}_{\text{PS}}^{\text{FHE}}$	5.41	13.782	6.054	5.164	15.255	10.584	10.658	17.795	11.401	10.736
		PSU [21]	683.001	407.864	381.286	378.067	496.140	470.493	471.833	1501.041	1511.784	1500.811
		uPSU* [21]	117.931	86.566	22.913	14.662	119.895	56.619	46.706	465.878	400.293	399.243
		PSU* [22]	2470.1	37.861	31.704	44.334	232.716	207.870	207.996	2081.052	2078.273	2080.110

7.2 Implementation details

Our protocols are built on FHE, Permute + Share, mp-OPRF, and OT extension. We implement Permute + Share with the design in [25] and OT extension [20] using libOTe library. For FHE, we use the source code from SEAL and APSI library. For mp-OPRF, we use the source code from [6]. 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.git and APSI https://github.com/microsoft/APSI.git
- Permute + Share: https://github.com/dujiajun/PSU.git
- mp-OPRF: https://github.com/peihanmiao/OPRF-PSI and https://github.com/yuchen1024/Kunlun.git
- OT: https://github.com/osu-crypto/libOTe.git

7.3 Performance Comparisons

In this section, we implement our $\text{uPSU}^{\text{FHE}}_{\text{PS}}$ and $\text{uPSU}^{\text{FHE}}_{\text{DDH}}$ and show the results in Table 2. Then, We compare our uPSU protocols with PSU [21], uPSU* [21] and PSU* [22] in terms of runtime and communication, and the results are reported in Table 3. We stress that we used the same environment to compute all the reported costs in this section. Experiments show that our protocols are more efficient than all previous protocols in the unbalanced case, especially, the larger the difference of two set sizes, the better our protocols perform. Our running-time-optimized benchmarks show that it takes 18.782 seconds of computation and 2.179 MB of communication to compute the union between 2^{10} strings and 2^{19} strings. Compared to prior secure PSU [21], this is roughly a 300× reduction in communication and 20× reduction in computational overhead with a single thread in WAN/LAN settings.

References

1. Bajard, J., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: Avanzi, R., Heys, H.M. (eds.) Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10532, pp. 423–442. Springer (2016). https://doi.org/10.1007/978-3-319-69453-5_23, https://doi.org/10.1007/978-3-319-69453-5_23
2. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in lwe-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7778, pp. 1–13. Springer (2013). https://doi.org/10.1007/978-3-642-36362-7_1, https://doi.org/10.1007/978-3-642-36362-7_1
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 309–325. ACM (2012). https://doi.org/10.1145/2090236.2090262, https://doi.org/10.1145/2090236.2090262
4. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.A.: SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In: 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings. pp. 223–240. USENIX Association (2010), http://www.usenix.org/events/sec10/tech/full_papers/Burkhart.pdf
5. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12493, pp. 342–372. Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_12, https://doi.org/10.1007/978-3-030-64840-4_12
6. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 34–63. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_2, https://doi.org/10.1007/978-3-030-56877-1_2

7. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1223–1237. ACM (2018). https://doi.org/10.1145/3243734.3243836, https://doi.org/10.1145/3243734.3243836
8. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library - SEAL v2.1. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10323, pp. 3–18. Springer (2017). https://doi.org/10.1007/978-3-319-70278-0_1, https://doi.org/10.1007/978-3-319-70278-0_1
9. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1243–1255. ACM (2017). https://doi.org/10.1145/3133956.3134061, https://doi.org/10.1145/3133956.3134061
10. 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: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 1135–1150. ACM (2021). https://doi.org/10.1145/3460120.3484760, https://doi.org/10.1145/3460120.3484760
11. Devroye, L., Morin, P.: Cuckoo hashing: Further analysis. Inf. Process. Lett. **86**(4), 215–219 (2003). https://doi.org/10.1016/S0020-0190(02)00500-8, https://doi.org/10.1016/S0020-0190(02)00500-8
12. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
13. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. In: Alt, H., Habib, M. (eds.) STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2607, pp. 271–282. Springer (2003). https://doi.org/10.1007/3-540-36494-3_25, https://doi.org/10.1007/3-540-36494-3_25
14. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 591–617. Springer (2021). https://doi.org/10.1007/978-3-030-75248-4_21, https://doi.org/10.1007/978-3-030-75248-4_21
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
16. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_49, https://doi.org/10.1007/978-3-642-32009-5_49
17. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016), http://proceedings.mlr.press/v48/gilad-bachrach16.html
18. Hogan, K., Luther, N., Schear, N., Shen, E., Stott, D., Yakoubov, S., Yerukhimovich, A.: Secure multiparty computation for cooperative cyber risk assessment. In: IEEE Cybersecurity Development, SecDev 2016, Boston, MA, USA, November 3-4, 2016. pp. 75–76. IEEE Computer Society (2016). https://doi.org/10.1109/SecDev.2016.028, https://doi.org/10.1109/SecDev.2016.028
19. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012. The Internet Society (2012), https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols
20. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 145–161. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_9, https://doi.org/10.1007/978-3-540-45146-4_9
21. Jia, Y., Sun, S., Zhou, H., Du, J., Gu, D.: Shuffle-based private set union: Faster and more secure. IACR Cryptol. ePrint Arch. p. 157 (2022), https://eprint.iacr.org/2022/157

22. Kolesnikov, V., Rosulek, M., Trieu, N., Wang, X.: Scalable private set union from symmetric-key techniques. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11922, pp. 636–666. Springer (2019). https://doi.org/10.1007/978-3-030-34621-8_23, https://doi.org/10.1007/978-3-030-34621-8_23
23. Lenstra, A.K., Voss, T.: Information security risk assessment, aggregation, and mitigation. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3108, pp. 391–401. Springer (2004). https://doi.org/10.1007/978-3-540-27800-9_34, https://doi.org/10.1007/978-3-540-27800-9_34
24. Lindell, Y.: How to simulate it - A tutorial on the simulation proof technique. In: Lindell, Y. (ed.) Tutorials on the Foundations of Cryptography, pp. 277–346. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-57048-8_6, https://doi.org/10.1007/978-3-319-57048-8_6
25. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 557–574. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_33, https://doi.org/10.1007/978-3-642-38348-9_33
26. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995). https://doi.org/10.1017/cbo9780511814075, https://doi.org/10.1017/cbo9780511814075
27. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2161, pp. 121–133. Springer (2001). https://doi.org/10.1007/3-540-44676-1_10, https://doi.org/10.1007/3-540-44676-1_10
28. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 401–431. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_13, https://doi.org/10.1007/978-3-030-26954-8_13
29. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. **21**(2), 7:1–7:35 (2018). https://doi.org/10.1145/3154794, https://doi.org/10.1145/3154794
30. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. p. 187 (2005), http://eprint.iacr.org/2005/187
31. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptogr. **71**(1), 57–81 (2014). https://doi.org/10.1007/s10623-012-9720-4, https://doi.org/10.1007/s10623-012-9720-4
32. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Optimal private set union from multi-query reverse private membership test. Cryptology ePrint Archive, Report 2022/358 (2022), https://ia.cr/2022/358