Fast *Unbalanced* Private Set Union from Fully Homomorphic Encryption

Binbin Tu
Shandong University

Yu Chen
Shandong University

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. However, most PSU protocols have been designed for the balanced case, where the two sets are rough of equal size, and their communication cost scales at least linearly with the size of the larger set. Thus, they are low efficient in dealing with unbalanced case or frequent data updates. Even, Jia et al. (Usenix Security 2022) consider unbalanced PSU setting, but their protocol is not entirely satisfactory. On one hand, their protocol *leaks the information of the size of set intersection*. On the other hand, their communication cost still requires at least *linearly with the size of the larger set*. Therefore, how to construct a secure and efficient unbalanced PSU which breaks the bound of communication complexity linear with the size of the larger set is an open problem.

In this work, we resolve this open problem by proposing a generic framework of using the leveled fully homomorphic encryption and a newly introduced protocol called matrix private equality test with permutation (mPEQT-wP) to construct *unbalanced* PSU that is secure against semi-honest adversaries. By instantiating mPEQT-wP, we obtain a fast uPSU with a small communication overhead. Our protocol satisfies full security of PSU, where one party obtains the union and another gets nothing, and 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 two PSU protocols and compare with the state-of-the-art PSU. Experiments show that our protocol has the lowest communication of all schemes in the unbalanced case.

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. Recently, some work has been made on PSU, which has become considerably efficient and been deployed in practice, such as cyber risk assessment and management via joint IP blacklists and joint vulnerability data [19, 23, 24], privacy-preserving data aggregation [5], private ID [15] 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. Therefore, the existing PSU protocol is not efficient in dealing with unbalanced case or data updates within a short period.

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. [8, 10] 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. [22] give a construction of uPSU* with shuffling technique, but their uPSU* protocol *leaks the information of the size of set intersection* to the sender. This is a critical information leakage for uPSU, in particular, the protocol leaks the set intersection when the sender inputs one-element set. Furthermore, the communication overhead of their uPSU* is not ideal and still requires at least *linearly with the size of the larger set*. 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?*

1.1 Contributions

In this paper, we give an affirmative answer to above question. We construct a *secure* and *fast* unbalanced PSU protocol which has communication complexity linear in the size of the smaller set, and logarithmic in the larger set. In detail, our contribution 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 [8, 10, 11], such as cuckoo hashing bucketing, batching, windowing, partitioning, modulus switching, etc, to get an **uPSU* with optimizations** which significantly reduces computational cost and the depth of the homomorphic circuit, while only adding a logarithmic overhead to the communication. We observed that above optimized techniques can be used in uPSI [8, 10, 11], because of computing a large (n,m)-uPSI can be split into computing m small $(n/m,1)$-PSI securely, where n denotes the size of the large set and m denotes the size of the small set. However, using them directly in uPSU could lead to information leakage. More precisely, the receiver not only gets the set union, but also knows that some of its subsets size of n/m have the item in the set intersection. This is because for each $(n/m,1)$-PSU, if the one item of the sender belongs to the set intersection, the receiver believe that the item belongs to n/m items.

3. We introduce a new cryptographic protocol named matrix private equality test with permutation **(mPEQT-wP)**. In the mPEQT-wP, the sender with a matrix $\mathbf{R}'$ and a permutation $\pi = (\pi_c, \pi_r)$ interacts with a receiver holding a matrix $\mathbf{R}$. As a result, the receiver learns (only) the bit matrix $\mathbf{B}$ indicating that if $r_{\pi(ij)} = r'_{\pi(ij)}$, $b_{\pi(ij)} = 1$, else, $b_{\pi(ij)} = 0$, $i \in [\alpha], j \in [m]$, while the sender learns nothing about the vector $\mathbf{R}$. Compare with the private equality test (PEQT), mPEQT-wP can provide matrix private equality test with ***positions permutation***. That is, the receiver only knows that some elements are equal at certain positions which is permuted by the sender. Based on our uPSU* with optimizations, mPEQT-wP and OT protocol, we give a generic framework of uPSU protocol.

4. We **instantiate our mPEQT-wP** efficiently and obtain secure and fast uPSU protocols with communication complexity linear in the size of the smaller set and logarithmic in the larger set.

   – We construct mPEQT-wP based on permute and share protocol [15, 22] and multi-point oblivious pesudorandom function (mp-OPRF), and this protocol requires the communication cast $O(m \log m)$.

   – We construct mPEQT-wP based on decisional Diffie-Hellman (DDH) assumption and this protocol requires the communication cast $O(m)$.

5. Finally, we realize our fast uPSU protocols. Our implementation is released on Github:.... Experiments show that our protocol...

1.2 Related Works

We revisit some recent private set operation protocols including uPSU [22], PSU [15, 22, 23, 33] and uPSI protocols [10], and show the communication cost and security comparison of PSU in the semi-honest setting.

Table 1: Communication Comparison of PSU in the semi-honest setting

Protocols	Communication	Security
PSU* [23]	$O(n \log n)$	✓
PSU [15]	$O(n \log n)$	✓
PSU [33]	$O(n)$	✓
PSU [22]	$O(n \log n)$	✓
uPSU* [22]	$O(n + m \log m)$	✓
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. ✓ denotes the PSU leaks information of some subsets have the items in the set intersection. ✓ denotes the PSU leaks information of the size of the set intersection.

uPSU protocol. Jia et al. [22] 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 $m = |X| \ll n = |Y|$. As a result, $\mathcal{S}$ gets the size of set intersection and $\mathcal{R}$ gets the set union. Informally, $\mathcal{S}$ inserts set X into the Cuckoo hash table and the item in i-th bin as $X_C[i]$, $i \in [m]$. $\mathcal{R}$ inserts set Y into the simple hash table and $Y_B[i]$, $i \in [m]$ denotes the set of items in the i-th bin, and 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 $\{a_{\pi(1)}, a_{\pi(2)}, \cdots, a_{\pi(m)}\}$, and $\mathcal{R}$ gets shuffled shares $\{a'_{\pi(1)}, a'_{\pi(2)}, \cdots, a'_{\pi(m)}\}$, where $X_C[\pi(i)] = a_{\pi(i)} \oplus a'_{\pi(i)}$. $\mathcal{R}$ permutes $Y_B[i]$ to $Y_B[\pi(i)]$, $i \in [m]$, 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 $\mathbf{B} = (b_{\pi(1)}, \cdots, b_{\pi(m)})$, if $b_{\pi(i)} = 1$, $X_C[\pi(i)] \notin Y_B$,

else, $X_C[\pi(i)] \in Y_B$. Then, two parties run shuffling technique to permute and share $\{a'_{\pi(1)}, a'_{\pi(2)}, \cdots, a'_{\pi(m)}\}$ by a permutation π' chosen by $\mathcal{S}$. $\mathcal{S}$ obtains $\{s_{\pi'(1)}, s_{\pi'(2)}, \cdots, s_{\pi'(m)}\}$, and $\mathcal{R}$ gets $\{s'_{\pi'(1)}, s'_{\pi'(2)}, \cdots, s'_{\pi'(i)}\}$, where $s_{\pi'(i)} \oplus s'_{\pi'(i)} = a'_{\pi'(i)}$, $\{a'_{\pi'(1)}, \cdots, a'_{\pi'(m)}\} = \pi'(\{a'_{\pi(1)}, \cdots, a'_{\pi(m)}\})$. Finally, $\mathcal{S}$ computes $\{b_{\pi'(1)}, \cdots, b_{\pi'(m)}\} = \pi'(\{b_{\pi(1)}, \cdots, b_{\pi(m)}\})$ and sets $z_{\pi'(i)} = \perp$ if $b_{\pi'(i)} = 0$, else, $z_{\pi'(i)} = a_{\pi'(i)} \oplus s_{\pi'(i)}$ and sends $(z_{\pi'(1)}, \cdots, z_{\pi'(m)})$ to $\mathcal{R}$. For all $i \in [m]$, $\mathcal{R}$ 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 [22], compared with their balanced PSU [22], their uPSU* described above is efficient in unbalanced case, because of using shuffling technique in the small set to obtain the items of the set union instead of using OT protocol in the large set. The communication overhead in this phase drops from $O(n)$ to $O(m \log m)$. However, using shuffling technique instead of OT protocol leaks the set intersection size to the sender. That is because that the sender gets the bit vector $\mathbf{B} = (b_{\pi(1)}, \cdots, b_{\pi(m)})$, and it can compute the number of $b_{\pi(i)} = 0$, $i \in [m]$. On the other hand, the receiver needs to send all PRF values of large set to the sender, so the communication overhead in this phase is $O(n)$, and the communication overhead of their uPSU* is $O(n + 2m \log m)$ that requires at least linearly with the size of the large set X.

Other balanced PSU protocol. Kolesnikov et al. [23] 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 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 can know that the subsets with size ρ have items in $X \cap Y$.

Garimella et al. [15] 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. [22] propose a PSU with the shuffling technique and oblivious transfer. They use cuckoo hash technique to hash receiver's set Y into m bins and each bin consists of one item, and hash sender's set X into m 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 $\{s_{\pi(1)}, \cdots, s_{\pi(m)}\}$, and the receiver inputs its bins $\{a_1, \cdots, a_m\}$ and gets another shuffled shares $\{s_{\pi(1)} \oplus a_{\pi(1)}, \cdots, s_{\pi(m)} \oplus s_{\pi(m)}\}$. 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. [33] 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. [10] propose the first unbalanced PSI based on FHE. In their unbalanced PSI [10], the sender $\mathcal{S}$ inputs a set X and the receiver $\mathcal{R}$ inputs a set Y, where $|Y| \ll |X|$. They first give a strawman protocol where $|X| = n, |Y| = 1$ as follows: the receiver $\mathcal{R}$ encrypts one item y and send the ciphertexts $c \leftarrow \mathsf{FHE.Enc}(y)$ to the sender $\mathcal{S}$; $\mathcal{S}$ chooses a random non-zero plaintext r and homomorphically computes $r\Pi_{x_i \in X}(c - x_i)$ and gets a new ciphertext $c' \leftarrow \mathsf{FHE.Enc}(r \cdot f(y))$, where the polynomial $f(x) = \Pi_{x_i \in X}(x - x_i)$, and then returns c' to $\mathcal{R}$; $\mathcal{R}$ can decrypt the ciphertext: if $rf(y) = 0$, it knows $y \in X \cap Y$, 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 where $m = |Y| \ll n = |X|$. 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, $i \in m$ 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 partition its set into α subsets. For i-th bin, each subset consists of $B' = B/\alpha$ items. Therefore, the large (n,m)-PSI is divided into many small $(B',1)$-PSI, and each small PSI has low homomorphic circuit and low computational cost. We review the uPSI protocol and optizimations in Figure 1. Furthermore, Chen et al. [8] and Cong et al. [11] based on the above framework and give fast labeled unbalanced PSI.

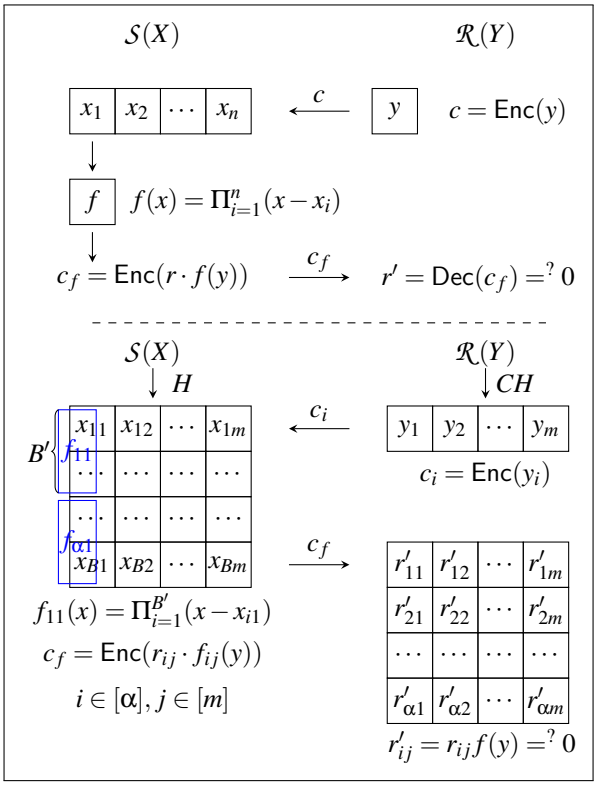

Figure 1: uPSI protocol and its optimizations [10]

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 matrix private equality test with permutation (mPEQT-wP). Finally, by instantiating our mPEQT-wP 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 2.

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 Y will receive 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 the ciphertext $c = \mathsf{FHE.Enc}(pk_{\mathcal{S}}, x)$ to the receiver $\mathcal{R}$; $\mathcal{R}$

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 $\mathcal{R}$.

Figure 2: Ideal functionality $\mathcal{F}_{\text{PSU}}^{m,n}$ for private set union with one-sided output

chooses random non-zero plaintext r, and homomorphically computes $r + \Pi_{y \in Y}(c - y)$, and returns the new ciphertext c_f to $\mathcal{S}$; $\mathcal{S}$ can decrypt c_f and get $r' = r + \Pi_{y \in Y}(c - y)$, then it returns r' to $\mathcal{R}$; $\mathcal{R}$ can 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 run OT protocol to let the receiver obtain and output the union $\{x\} \cup Y$.

Compare with the basic uPSI [10]. The key different step between our uPSU and the basic unbalanced PSI [10] is using the different randomization methods. In the PSI [10], they compute the product of randomness r and the polynomial value $f(x)$, where $f(x) = \Pi_{y \in Y}(x - y)$. If $f(x) = 0$, $rf(x) = 0$ denotes $x \in Y$, else $x \notin Y$, and the receiver only gets a randomness $rf(x) \neq 0$ which hides the information of f and Y. 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 f and 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. [10] 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, use 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 can know 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 there exists a subset with size B', such that $f(x) = 0$, then the receiver can get other subset in same bin, such that $f'(x) \neq 0$ which leaks the information of $X \cap Y$.

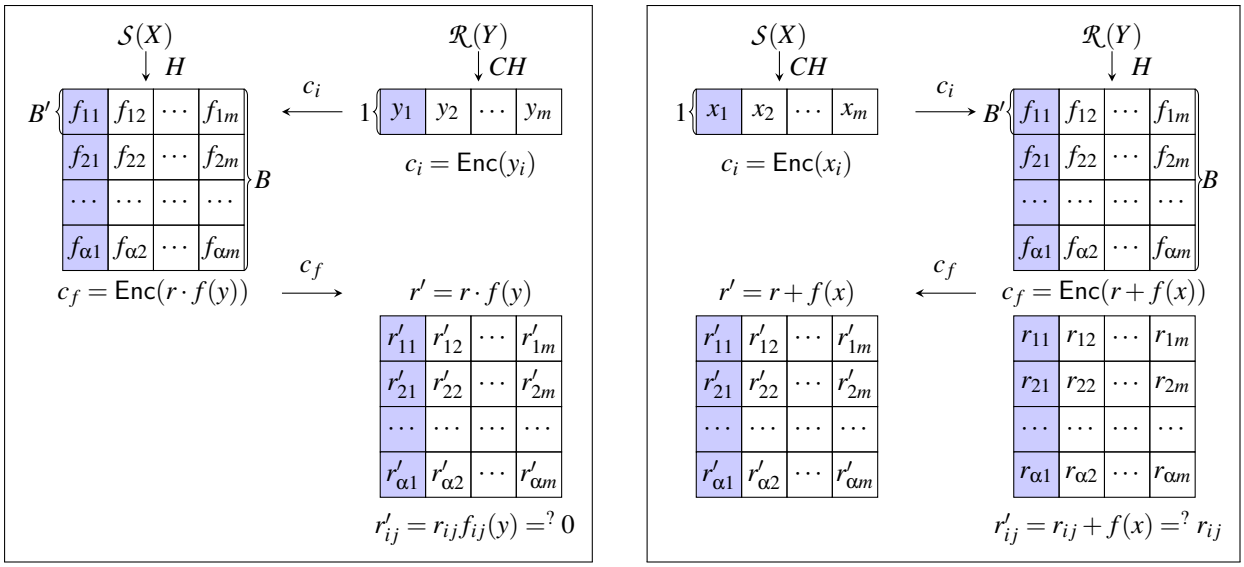

Figure 3: Comparison of uPSI [10] (left) and our basic uPSU with optimizations (right)

2.2 Matrix Private Equality Test with Permutation (mPEQT-wP)

We develop a new cryptographic protocol named matrix private equality test with permutation (mPEQT-wP) and give corresponding tailored efficient construction, which we believe to be of independent interest. The mPEQT-wP is related to the traditional private equality test (PEQT), which is a 2-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 mPEQT-wP, the sender with a matrix $\mathbf{R}'$ and a permutation $\pi = (\pi_c, \pi_r)$ interacts with a receiver holding a matrix $\mathbf{R}$. As a result, the receiver learns (only) the bit matrix $\mathbf{B}$ indicating that if $r_{\pi(ij)} = r'_{\pi(ij)}$, $b_{\pi(ij)} = 1$, else, $b_{\pi(ij)} = 0$, $i \in [\alpha]$, $j \in [m]$, while the sender learns nothing about the vector $\mathbf{R}$. Compare with the private equality test (PEQT), mPEQT-wP can provide matrix private equality test with ***positions permutation***. We show the ideal functionality of mPEQT-wP in Figure 4.

This seemingly simple functionality adjustment (PEQT $\rightarrow$ mPEQT-wP) 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 mPEQT-wP. We give two constructions of mPEQT-wP. The first construction is based on permute and share (PS) functionality [15, 22] and mp-OPRF. We review the permute and share functionality in figure 7. $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal permute and 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

Parameters: Two parties: P_1 with a matrix R, P_0 with a matrix R' and a permutation $\pi = (\pi_c, \pi_r)$, where π_c (over $\{1, 2, \cdots, m\}$) and π_r (over $\{1, 2, \cdots, \alpha\}$),

$$\mathbf{R} = \begin{bmatrix} r_{11} & \cdots & r_{1m} \\ r_{21} & \cdots & r_{2m} \\ \vdots & \ddots & \vdots \\ r_{\alpha 1} & \cdots & r_{\alpha m} \end{bmatrix}, \mathbf{R}' = \begin{bmatrix} r'_{11} & \cdots & r'_{1m} \\ r'_{21} & \cdots & r'_{2m} \\ \vdots & \ddots & \vdots \\ r'_{\alpha 1} & \cdots & r'_{\alpha m} \end{bmatrix}$$

Functionality:

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

$$\mathbf{B} = \begin{bmatrix} b_{\pi(11)} & \cdots & b_{\pi(1m)} \\ b_{\pi(21)} & \cdots & b_{\pi(2m)} \\ \vdots & \ddots & \vdots \\ b_{\pi(\alpha 1)} & \cdots & b_{\pi(\alpha m)} \end{bmatrix},$$

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

Figure 4: Matrix private equality test with permutation $\mathcal{F}_{\text{mPEQT-wP}}$

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 $s_{\pi(ij)} \oplus s'_{\pi(ij)} = \pi(r_{ij})$, $i \in [\alpha], j \in [m]$. 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 $b_{\pi(ij)} = 1$, if $F_k(s_{\pi(ij)}) = F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, else, $b_{\pi(i)} = 0$, and gains a bit matrix $\mathbf{B} = [b_{\pi(ij)}]$, $i \in [\alpha], j \in [m]$. We note that PS and OPRF are fast cryptographic tools, and the communication overhead of our mPEQT-wP based on PS and 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 $b_{\pi(ij)} = 1$, if $H''_{\pi(ij)} = H'^a_{\pi(ij)}$, else $b_{\pi(ij)} = 0$, and gains a bit matrix $\mathbf{B} = [b_{\pi(ij)}]$, $i \in [\alpha]$, $j \in [m]$. We note that the communication overhead of our mPEQT-wP based DDH is equal to $O(m)$.

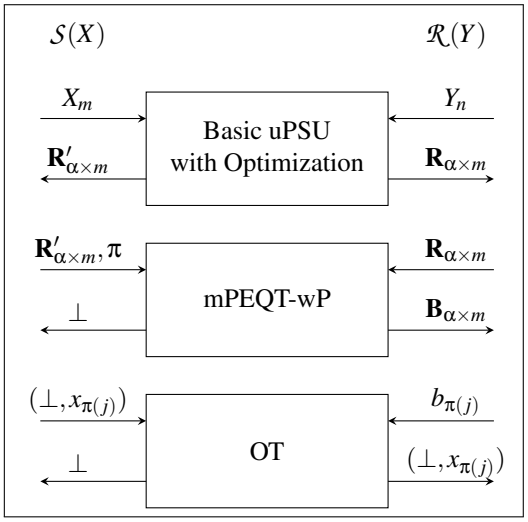

Figure 5: Core design idea of Our full uPSU protocol

2.3 Our Full uPSU protocol

We start with our basic uPSU protocol based on FHE and using optimization techniques [8, 10, 11] to divide a large PSU into many small PSU to reduce the depth of homomorphic circuit. Then, by using mPEQT-wP 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 mPEQT-wP is $O(|Y|\log|Y|)$ (based on PS and mp-OPRF) 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 5 and the details are as follows.

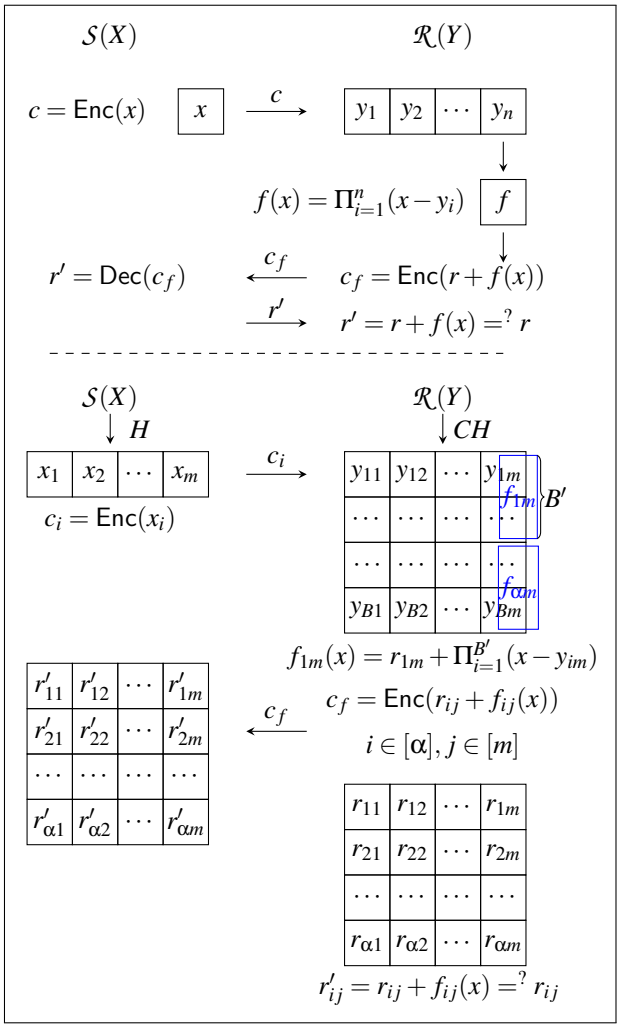

Figure 6: uPSU protocol and its optimizations

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 mPEQT-wP, 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 $\mathbf{B} = [b_{\pi(ij)}]$, where if $b_{\pi(ij)} = 1$, $i \in [\alpha], j \in [m]$, $r_{\pi(ij)} = r'_{\pi(ij)}$, else $r_{\pi(ij)} \neq r'_{\pi(ij)}$. The receiver set a bit vector $\mathbf{b} = [b_{\pi(j)}]$, $j \in [m]$, if for all $i \in [\alpha]$, $r_{\pi(ij)} \neq r'_{\pi(ij)}$, $b_{\pi(j)} = 1$, else $b_{\pi(j)} = 0$. 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 $b_{\pi(j)} = 1$, $x_{\pi_c(j)} \notin Y$, else $x_{\pi(i)} \in Y$, $j \in [m]$.

Finally, by using OT protocol, the sender inputs $(x_{\pi(j)}, \perp)$, $j \in [m]$, and the receiver inputs $b_{\pi(j)}$, $j \in [m]$. If $b_{\pi(j)} = 1$, 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. For any permutations π on n items, we set $\{s_{\pi(1)}, s_{\pi(2)}, \cdots, s_{\pi(n)}\} = \pi(\{s_1, s_2, \cdots, s_n\})$. 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 [10], 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 $H_1, \cdots, H_h : \{0,1\}^* \rightarrow [m]$ used to map n items into m bins $\mathbf{B}_1, \cdots, \mathbf{B}_m$. Following [27], 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.

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

Cuckoo hashing. Cuckoo hashing [12, 14, 28] 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 $m = \varepsilon n$ 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 [30], we can adjust the values of m and ε to reduce the stash size to 0 while achieving a hashing failure probability of 2^{-40}.

Note that following [10], we also let the receiver perform cuckoo hashing with $m \geq |Y|$ bins. The sender inserts each of its items into a two-dimensional hash table using all h hash functions $H_1, \cdots, H_h$, because there is no way for it to know which one of the hash functions the receiver eventually ended up using for the items.

Permutation-based hashing. The permutation-based hashing [1] is an optimization to reduce the length of the items stored in the hash tables by encoding a part of an item into the bin index. For simplicity, we assume m is a power of two. To insert a bit string x into the hash table, we parse it as $x_L || x_R$, where the length of x_R is equal to $\log_2 m$. The hash functions $H_1, \cdots, H_h$ are used to construct location functions as

$$\mathsf{Loc}_i(x) = H_i(x_L) \oplus x_R, 1 \leq i \leq h$$

Instead of inserting the entire tuple (x, i) into the hash table, we only insert (x_L, i) at the location specified by $\mathsf{Loc}_i(x)$. If $(x_L, i) = (y_L, j)$ for two items x and y, then $i = j$ and $x_L = y_L$. If in addition these are found in the same location, then $H_i(x_L) \oplus x_R = H_j(y_L) \oplus x_R = H_j(y_L) \oplus y_R$, so $x_R = y_R$, and hence $x = y$. The lengths of the strings stored in the hash table are thus reduced by $\log_2 m - \lceil \log_2 h \rceil$ bits.

3.3 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) [16] 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 [8, 10, 11]. There are several FHE implementations that are publicly available. We use the homomorphic encryption library SEAL, which implements the variant of [2] of the Brakerski/FanVercauteren (BFV) scheme [13]. The core parameters of the BFV scheme are three integers: n, q, and t.

We also need some optimization techniques of FHE following [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 [3, 9, 17, 18, 32]. 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 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 $c(i,j) = \mathrm{FHE.Enc}(y^{i \cdot 2^{lj}})$ to the sender for all $1 \le i \le 2^l - 1$, and all $0 \le j \le \lfloor \log_2(|X|)/l \rfloor$. 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 [4], 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. This trick allows the sender to "compress" the return ciphertexts before sending them to the receiver. 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 and Share. We recall the permute and share functionality $\mathcal{F}_{\mathrm{PS}}$ defined by Chase et al. [6] in Figure 7. Roughly speaking, in the permute and 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 [6], some earlier works [20, 26] can also be used to realize $\mathcal{F}_{\mathrm{PS}}$.

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.

Figure 7: Permute and share functionality $\mathcal{F}_{\mathrm{PS}}$

Multi-Point Oblivious Pesudorandom Function (mp-OPRF). 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. [29] proposes multi-point OPRF (mp-OPRF) and realizes efficient PSI protocols. Recently, Chase et al. [7] 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 8.

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.

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

Oblivious Transfer. Oblivious Transfer (OT), introduced by Rabin [31] 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. [21] 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 9.

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.

Figure 9: 1-out-of-2 oblivious transfer functionality $\mathcal{F}_{\text{OT}}$

4 The Basic Protocol

We describe our basic uPSU protocol in Figure 10 as a strawman protocol. The receiver encrypts each of its items y, and sends them to the sender. For each y, the sender then evaluates homomorphically the product of differences of y with all of the sender's items x (computing a function $f = (x-x_1)\cdots(x-x_{|X|})$, s.t. $f(x) = 0$ for each $x \in X$), randomizes the product by adding it with differences uniformly random non-zero plaintext r, and sends the ciphertext c back to the receiver. The receiver decrypts c to $r + f(y)$ and sends $r + f(y)$ to the sender. If $r + f(y) = r$, $y \in X$, otherwise, $y \notin X$.

We give the following informal theorem with regards to the security and correctness of above basic protocol.

Theorem 1. *(informal). The protocol described in Figure 10 securely and correctly computes the private set union of X and Y in the semi-honest security model, provided that the fully homomorphic encryption scheme is IND-CPA secure with circuit privacy and the oblivious transfer is secure.*

Proof. (Proof sketch). For each index i, the sender gets the ciphertext $\mathsf{FHE}_{pk_R}(y_i)$, and then compute and randomize it to gain $\mathsf{FHE}_{pk_R}(r_i + f(y_i))$. The receiver decrypt it and send $m_i = r_i + f(y_i)$ to the sender. The sender can remove randomization by $m_i - r_i = f(y_i)$, and check that if $f(y_i) = 0$, $y_i \in X \cap Y$, else, $y_i \in X \cup Y \backslash X$. Then, the sender runs oblivious transfer protocol with the receiver to gain the item $y_i \in X \cup Y$ (correctness).

Receiver's security is straightforward: the receiver sends an array of ciphertexts, which looks pseudorandom to the sender since the fully homomorphic encryption scheme is IND-CPA secure. For sender's security, we note that the receiver can decrypt ciphertexts, but only get an array of randomness, since the plaintext is randomized by the sender. Moreover, the oblivious transfer can help the sender get the item in $X \cup Y \backslash X$ and protect the security of $X \cap Y$.

Therefore, the sender learns no additional information beyond the union $X \cup Y$ and the receiver learns nothing. □

Input: The sender inputs set X of size $|X|$ and the receiver inputs set Y of size $|Y|$. Both sets consist of bit strings of length σ.
Output: The sender outputs $X \cup Y$ and the receiver outputs $\perp$.

1. [**Setup**] The sender and the receiver jointly agree on a fully homomorphic encryption scheme: The receiver generates a public-secret key pair for the scheme, and keeps the secret key itself.
2. [**Set encryption**] The receiver encrypts each item y_i in its set Y using the fully homomorphic encryption scheme, and sends the $|Y|$ ciphertexts $(c_1, \cdots, c_{|Y|})$ to sender.
3. [**Computation**] For each c_i, the sender
   (a) samples a random non-zero plaintext item r_i;
   (b) homomorphically computes $c'_i = \text{FHE.Enc}(f(y_i) + r_i), i \in [|Y|]$ where for all $x \in X$, s.t. $f(x) = 0$.
   (c) sends $c'_i, i \in [|Y|]$ to receiver.
4. [**Decryption**] The receiver decrypt $c'_i, i \in [|Y|]$ to $m_i = f(y_i) + r_i$ and sends them to sender.
5. [**Output**] The sender checks all plaintext. If $m_i = r_i, i \in [|Y|]$, $y_i \in X$, otherwise, $y_i \notin X$. The sender and the receiver invoke the ideal functionality $\mathcal{F}_{OT}$. For $i \in [|Y|]$, the sender gets y_i, if $y_i \notin X$. And then sender outputs $X \cup Y$.

Figure 10: Basic uPSU protocol

5 Matrix Private Equality Test with Permutation (mPEQT-wP)

Our mPEQT-wP protocol is described in Figure 4. The formal protocol follows the intuition presented in the first part of Section 2.2. We describe two efficient instantiations of mPEQT-wP, which is a semi-honest secure protocol for the functionality specified in Figure 4. And then, we prove the security properties of the protocol.

5.1 mPEQT-wP from PS and mp-OPRF

We give the first construction of mPEQT-wP based on the permute and share (PS) [15, 22] and mp-OPRF [7] in Figure 11. We note that PS and OPRF are fast cryptographic tools, and the communication overhead of our mPEQT-wP based on PS and OPRF is equal to $O(m \log m)$.

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 $\{1, 2, \cdots, m\}$) and π_r (over $\{1, 2, \cdots, \alpha\}$).
Output: The receiver outputs a bit matrix $\mathbf{B}$; the sender outputs $\perp$.

1. [PS functionality] $\mathcal{S}$ and $\mathcal{R}$ invoke the ideal permute and 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 $\mathbf{S}$, 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 $b_{\pi(ij)} = 1$, if $F_k(s_{\pi(ij)}) = F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, else, $b_{\pi(i)} = 0$, and gains a bit matrix $\mathbf{B} = [b_{\pi(ij)}]$, $i \in [\alpha], j \in [m]$.

Figure 11: mPEQT-wP from PS and mp-OPRF

Theorem 2. *The construction of Figure 11 securely implements functionality $\mathcal{F}_{mPEQT\text{-}wP}$ in the semi-honest model, given the secure PS and mp-OPRF defined in Figure 7, and Figure 8, respectively.*

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 $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output and received messages. $\text{Sim}_\mathcal{S}$ proceeds as follows. It invokes PS simulator $\text{Sim}^\mathcal{S}_{\text{PS}}(\pi_c)$ and appends the output to the view, where the outputs consist of $\mathbf{S}'_{\pi_c}$. And then, it invokes PS simulator $\text{Sim}^\mathcal{S}_{\text{PS}}(\pi_r)$ and appends the output to the view, where the outputs consist of $\mathbf{S}'_{\pi_r}$. $\text{Sim}_\mathcal{S}$ computes $\mathbf{S}'_\pi = \pi_r(\mathbf{S}'_{\pi_c}) \oplus \mathbf{S}'_{\pi_r}$. Then, it invokes mp-OPRF simulator $\text{Sim}^\mathcal{S}_{\text{mp-OPRF}}(\cdot)$ and appends the output to the view, where the outputs consist of the key k of PRF. Finally, it uses the key k to compute all PRF values $F_k(r'_{\pi(ij)} \oplus s'_{\pi(ij)})$, $i \in [\alpha]$ and $j \in [m]$ and appends them to the view.

The view generated by $\text{Sim}_\mathcal{S}$ in indistinguishable from a real view because of the indistinguishability of the transcripts of the underlying simulators.

Corrupt Receiver. $\text{Sim}_\mathcal{R}(\mathbf{R}, \mathbf{B} = [b_{\pi(ij)}]$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output and received messages. $\text{Sim}_\mathcal{R}$ proceeds as follows. It invokes PS simulator $\text{Sim}^\mathcal{R}_{\text{PS}}(\mathbf{R})$ and appends the output to the view, where the outputs consist of $\mathbf{S}_{\pi_c}$. And then, it invokes PS simulator $\text{Sim}^\mathcal{R}_{\text{PS}}(\mathbf{S}_{\pi_c})$ and appends the output to the view, where the outputs consist of $\mathbf{S}_{\pi_r}$. $\text{Sim}_\mathcal{R}$ sets $\mathbf{S}_\pi = \mathbf{S}_{\pi_r}$. And then it invokes mp-OPRF simulator $\text{Sim}^\mathcal{R}_{\text{mp-OPRF}}(\mathbf{S}_\pi)$ and appends the output to the view, where the outputs consist of $F_k(s_{\pi(ij)}), i \in [\alpha], j \in [m]$. Finally, $\text{Sim}_\mathcal{R}$ sets $v_{\pi(ij)} = F_k(s_{\pi(ij)})$ if $b_{\pi(i)} = 1$, else, it chooses $v_{\pi(ij)}$ randomly and it appends all $v_{\pi(ij)}$, $i \in [\alpha]$, $j \in [m]$ to the view.

The view generated by $\text{Sim}_\mathcal{R}$ in indistinguishable from a real view because of the indistinguishability of the transcripts of the underlying simulators, and the security of PRF. □

5.2 mPEQT-wP based on DDH

We describe the second construction of mPEQT-wP is based on DDH in Figure 12. We note that the communication overhead of our mPEQT-wP based DDH is equal to $O(m)$.

Theorem 3. *The construction of Figure 12 securely implements functionality $\mathcal{F}_{mPEQT\text{-}wP}$ based on DDH in the semi-honest model.*

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 $\mathcal{S}$, which consists of $\mathcal{S}$'s randomness, input, output and received messages. It chooses b randomly and computes $H'_{ij} = H(r'_{ij})^b$, $i \in [\alpha]$, $j \in [m]$ as the real protocol, and then it chooses $v_{\pi(ij)}$, $i \in [\alpha]$, $j \in [m]$ randomly and appends them to the view. The view generated by $\text{Sim}_\mathcal{S}$ in indistinguishable from a real view based on the security of DDH.

Corrupt Receiver. $\text{Sim}_\mathcal{R}(\mathbf{R}', \mathbf{B} = [b_{\pi(ij)}]$ simulates the view of corrupt $\mathcal{R}$, which consists of $\mathcal{R}$'s randomness, input, output and received messages. $\text{Sim}_\mathcal{R}$ proceeds as follows. It chooses a randomly and computes $H_{ij} = H(r_{ij})^a$ as the real

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 $\{1, 2, \cdots, m\}$) and π_r (over $\{1, 2, \cdots, \alpha\}$).
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 $H = H(\cdot)$ are (multiplicative) group elements output by hash functions H. $\mathcal{R}$ sends $H_{ij} = H(r_{ij})^a$ to $\mathcal{S}$.
2. $\mathcal{S}$ 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 $\mathcal{R}$.
3. $\mathcal{R}$ set $b_{\pi(ij)} = 1$, if $H''_{\pi(ij)} = H'^a_{\pi(ij)}$, else $b_{\pi(ij)} = 0$, and gains a bit matrix $\mathbf{B} = [b_{\pi(ij)}]$, $i \in [\alpha]$, $j \in [m]$.

Figure 12: Instantiation of mPEQT-wP based on DDH

protocol. And then, it chooses a random vector $[u_{\pi(ij)}]$, $i \in [\alpha]$, $j \in [m]$ and sets $v_{\pi(ij)} = u^a_{\pi(ij)}$, if $b_{\pi(ij)} = 1$, else, it chooses $v_{\pi(ij)}$ randomly, and $\mathsf{Sim}_{\mathcal{R}}$ appends them to the view. The view generated by $\mathsf{Sim}_{\mathcal{R}}$ in indistinguishable from a real view based on the security of DDH. □

6 Full Unbalanced PSU and Security Proof

In this section, We start from our basic uPSU protocol described in Figure 10 and use some optimization techniques following [8, 10, 11] to reduce the homomorphic circuits, and then we give a full uPSU based on mPEQT-wP 8 and OT protocol 9.

6.1 Full uPSU Protocol

We detail our full uPSU protocol in Figure 13, 14, 15, given a secure fully homomorphic encryption scheme with circuit privacy and secure mPEQT-wP and OT protocols.

In the setup phase 13, the sender and the receiver agree on the hashing parameters and the FHE scheme parameters. After the setup phase, the sender and the receiver take advantage of the optimization techniques [10], such as Hashing, Batching, Windowing, Partitioning, Modulus Switching, etc, to pre-process the set X and Y offline 14, respectively. After offline pre-processing phase, the sender and the receiver begin the efficient online phase 15: first, the sender sends the ciphertexts to the receiver, and the receiver homomorphically computes ciphertexts and returns them back. Then, the sender decrypts the new ciphertexts and run mPEQT-wP 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 OT protocol together to let the receiver obtains the union $X \cup Y$ and outputs it.

Input: Receiver inputs set $Y \subset \{0,1\}^\sigma$ of size $|Y|$; sender inputs set $X \subset \{0,1\}^\sigma$ of size $|X|$. κ and λ denote the computational and statistical security parameters, respectively.
Output: The receiver outputs $Y \cup X$; the sender outputs $\perp$.

1. [**Setup**] The sender and the receiver agree on the hashing parameters and the FHE scheme parameters.
   (a) [Hashing parameters] They agree on hashing parameters h, m, B, such that simple hashing $h|X|$ balls into m bins with max load B, and cuckoo hashing $|Y|$ balls into m bins succeed with probability $\geq 1 - 2^{-\lambda}$, and publicly choose hash functions $H_1, \cdots, H_h : \{0,1\}^\sigma \to [m]$.
   (b) [FHE parameters] They agree on parameters (n, q, t) for an IND-CPA secure FHE scheme with circuit privacy. The parties agree on the windowing parameter $l \in [1, \log_2 B]$ and partitioning parameter $\alpha \in [1, B]$ as to minimize the overall cost.
   (c) [(PS+mpOPRF or DDH) + OT parameters] ...

Figure 13: Full uPSU protocol (setup phase)

6.2 Security Proof

We prove security in the standard semi-honest simulation-based paradigm. Loosely put, we say that the protocol of Figure 13, 14, 15 securely realizes the functionality $\mathcal{F}_{PSU}$, if it is correct, and there exist two simulators (PPT algorithms) $\mathsf{Sim}_{\mathcal{S}}$, $\mathsf{Sim}_{\mathcal{R}}$ with the following properties. The simulator $\mathsf{Sim}_{\mathcal{S}}$ takes the sender's set and the union as input, and needs to generate a transcript for the protocol execution that is indistinguishable from the sender's view of the real interaction. $\mathsf{Sim}_{\mathcal{R}}$ is similarly defined, with the exception of not taking the union as input. For a formal definition of simulation-based security in the semi-honest setting, we refer the reader to [25].

Theorem 4. *The protocol in Figure 13, 14, 15 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 routine succeeding, which happens with overwhelming probability $1 - 2^{-\lambda}$.

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.

2. [**Hashing**] Two parties take the parameters h, m, B and hash functions $H_1, \cdots, H_h : \{0,1\}^{\sigma - \log_2 m} \to \{0,1\}^{\log_2 m}$ as input. The receiver hashs the set Y into $\mathbf{B}_y$ and the sender hash the set X into $\mathbf{B}_x$.

3. [**Pre-process Y**]

   (a) [Partitioning] The receiver partitions its table $\mathbf{B}_y$ vertically (i.e. by columns) into α subtables $\mathbf{B}_{y,1}, \mathbf{B}_{y,2}, \cdots, \mathbf{B}_{y,\alpha}$. Each subtable has $B' = B/\alpha$ columns and m rows. Let $\mathbf{B}_y = [\mathbf{B}_{y,i,j}], \mathbf{B}_{y,i,j} = [y_1^{i,j}, y_2^{i,j}, \cdots, y_{B'}^{i,j}], i \in [\alpha], j \in [m]$.

   (b) [Computing coefficients] For each rows of a subtable $\mathbf{B}_{y,i,j} = [y_1^{i,j}, y_2^{i,j}, \cdots, y_{B'}^{i,j}]$, $i \in [\alpha], j \in [m]$, the receiver computes the coefficients of the polynomial $f^{i,j}(y) = \Pi_{k=1}^{B'}(y - y_k^{i,j}) = a_0^{i,j} + a_1^{i,j} y + \cdots + a_{B'-1}^{i,j} y^{B'-1} + a_{B'}^{i,j} y^{B'}$, and then replaces each row $\mathbf{B}_{y,i,j}$, with coefficients of the polynomial $f^{i,j}(y)$, $\mathbf{A}_{i,j} = [a_0^{i,j}, a_1^{i,j}, \cdots, a_{B'}^{i,j}]$, $i \in [\alpha], j \in [m]$.

   (c) [Batching] For each subtable, the receiver interprets each of its column as a vector of length m with items in $\mathbb{Z}_t$. Then the receiver batches each vector into $\beta = m/n$ plaintext polynomials, and the coefficients are $\hat{\mathbf{A}}_{i,j} = [\hat{\mathbf{a}}_0^{i,j}, \hat{\mathbf{a}}_1^{i,j}, \cdots, \hat{\mathbf{a}}_{B'}^{i,j}], i \in [\alpha], j \in [\beta]$, where $\hat{\mathbf{a}}_k^{i,j} = [\hat{a}_{k,1}^{i,j}, \cdots, \hat{a}_{k,n}^{i,j}], k \in [0, B']$

4. [**Pre-process and Encrypt X**]

   (a) [Batching] The sender interprets $\mathbf{B}_x$ as a vector of length m with items in $\mathbb{Z}_t$. It batches this vector into $\beta = m/n$ plaintext polynomials $\bar{X}_1, \cdots, \bar{X}_\beta$.

   (b) [Windowing] For each batched plaintext polynomial $\bar{X}$, the sender computes the component-wise $i \cdot 2^j$-th powers $\bar{X}^{i \cdot 2^{lj}}$, for $1 \le i \le 2^l - 1$ and $0 \le j \le \lceil \log_2(B')/l \rceil$.

   (c) [Encrypt] The sender uses FHE to encrypt each such power, obtaining β collections of ciphertexts $\mathbf{C}_j, j \in [\beta]$. The sender sends these ciphertexts to the sender.

Figure 14: Full uPSU protocol (offline pre-processing)

5. [**Computation**]

   (a) [Homomorphically compute encryptions of all powers] For each collection of ciphertexts $\mathbf{C}_j, j \in [\beta]$, the receiver homomorphically compute encryptions of all powers $\hat{\mathbf{C}}_j = [\hat{\mathbf{c}}_0^j, \hat{\mathbf{c}}_1^j, \cdots, \hat{\mathbf{c}}_{B'}^j], j \in [\beta]$, where $\hat{\mathbf{c}}_k^j = [\hat{c}_{k,1}^j, \hat{c}_{k,2}^j, \cdots, \hat{c}_{k,n}^j], k \in [B']$.

   (b) [Homomorphically evaluate the dot product] The receiver homomorphically evaluates

   $$\mathbf{C}'_{i,j} = \hat{\mathbf{A}}_{i,j} \hat{\mathbf{C}}_j = \sum_{k=0}^{B'} \hat{a}_k^{i,j} \hat{c}_k^j, i \in [\alpha], j \in [\beta].$$

   optionally performs modulus switching on the ciphertexts $\mathbf{C}'_{i,j}, i \in [\alpha], j \in [\beta]$ to reduce their sizes, and sends them back to the receiver.

6. [**Decrypt**] For each $1 \le i \le \alpha, 1 \le j \le \beta$, the sender decrypts all ciphertexts, and it receives and concatenates the resulting β matrixes into one matrix $\mathbf{R}'_{m,\alpha}$.

7. [**mPEQT-wP**] The receiver inputs the matrix $\mathbf{R} = [r_{\pi(ij)}]$, and the sender inputs a permutation $\pi = (\pi_c, \pi_r)$ where π_c (over $\{1, 2, \cdots, m\}$) and π_r (over $\{1, 2, \cdots, \alpha\}$) and the matrix $\mathbf{R}' = [r'_{\pi(ij)}]$. As a result, the receiver gains a bit matrix $\mathbf{B} = [b_{\pi(ij)}]$, where if $b_{\pi(ij)} = 1$, $r_{\pi(ij)} = r'_{\pi(ij)}$, else $r_{\pi(i)} \ne r'_{\pi(i)}$.

8. [**Output**] The receiver sets a bit vector $\mathbf{b} = [b_j]$, $j \in [m]$, where if for all $i \in [\alpha]$, $b_{\pi(ij)} = 0$, it sets $b_j = 1$, else $b_j = 0$. Then, the receiver and the sender run the OT protocol, in which the receiver inputs $\mathbf{B} = [b_1, \cdots, b_m]$ and the receiver inputs $\{y_{\pi_c(1)}, y_{\pi_c(2)}, \cdots, y_{\pi_c(m)}\}$. If $b_i = 1$, the receiver gets $y_{\pi(i)}$, else, it gets $\perp$. Thus, the receiver can get the set $\hat{Y}^* = X \cup Y \backslash X$. Finally, the receiver outputs the set union

   $$X \cup Y = \hat{Y}^* \cup X$$

Figure 15: Full uPSU protocol (online phase)

We start with a corrupt receiver, and show the existence of $\mathsf{Sim}_{\mathcal{R}}$. 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. $\mathsf{Sim}_{\mathcal{R}}$ $(Y = \{y_1, \cdots, y_{|Y|}\}, X \cup Y)$ simulates the view of corrupt semi-honest receiver. It executes as follows: $\mathsf{Sim}_{\mathcal{R}}$ computes the set $\hat{X}^* = X \cup Y \backslash Y$, and uses $|X| - |\hat{X}^*|$ items $\perp$ to pad $\hat{X}$ to $|X|$ items and permutes these items randomly. Let $\hat{X} = \{\hat{x}_1, \cdots, \hat{x}_{|\hat{X}|}\}$. Next it runs step 1-4 as real protocol, and encrypts $r'_{ij} = r_{ij} + f(\hat{x}_i)$, for $\hat{x}_i \in \hat{X}^*$, $i \in [m]$, $j \in [\alpha]$, and encrypts $r'_{ij} = r_{ij} + 0$ for $\hat{x}_i = \perp, i \in [m], j \in [\alpha]$. $\mathsf{Sim}_{\mathcal{R}}$ runs mPEQT-wP simulator $\mathsf{Sim}_{\mathcal{R},\text{mPEQT-wP}}(\mathbf{R})$ and gets $\mathbf{B} = [b_{\pi(ij)}]$. It sets a bit vector $\mathbf{b} = [b_{\pi(i)}]$, $i \in [m]$, where if for all $j \in [\alpha]$, $b_{\pi(ij)} = 0$, it sets $b_{\pi(i)} = 1$, else, $b_{\pi(i)} = 0$. And Then $\mathsf{Sim}_{\mathcal{R}}$ invokes OT simulator $\mathsf{Sim}_{\mathcal{R},\text{OT}}(b_{\pi(i)}, \hat{x}_{\pi(i)})$ and appends the output to the view.

Now we argue that the view output by $\mathsf{Sim}_{\mathcal{R}}$ is indistinguishable from the real one. In the simulation, the way $\mathcal{R}$ obtains the items in $\hat{Y}^* = X \cup Y \backslash X$ is identical to the real execution. By the IND-CPA security of the fully homomorphic encryption scheme and the security of the mPEQT-wP protocol and the OT protocol, this result is indistinguishable from the sender's view in the real protocol.

The case of a corrupt sender is straightforward. The simulator $\mathsf{Sim}_{\mathcal{S}}(X = \{x_1, \cdots, x_{|X|}\})$ can generate new encryptions of randomness in place of the encryptions in step 5. And Then $\mathsf{Sim}_{\mathcal{S}}$ invokes mPEQT-wP simulator $\mathsf{Sim}_{\mathcal{S},\text{mPEQT-wP}}$ and OT simulator $\mathsf{Sim}_{\mathcal{S},\text{OT}}$ and appends the outputs to the view. This result is indistinguishable from the sender's view in the real protocol, because of the indistinguishability of the transcripts of the underlying simulators. □

7 Implementation

References

[1] ARBITMAN, Y., NAOR, M., AND SEGEV, G. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA* (2010), IEEE Computer Society, pp. 787–796.

[2] BAJARD, J., EYNARD, J., HASAN, M. A., AND ZUCCA, V. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers* (2016), R. Avanzi and H. M. Heys, Eds., vol. 10532 of *Lecture Notes in Computer Science*, Springer, pp. 423–442.

[3] BRAKERSKI, Z., GENTRY, C., AND HALEVI, S. Packed ciphertexts in lwe-based homomorphic encryption. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings* (2013), K. Kurosawa and G. Hanaoka, Eds., vol. 7778 of *Lecture Notes in Computer Science*, Springer, pp. 1–13.

[4] BRAKERSKI, Z., GENTRY, C., AND VAIKUNTANATHAN, V. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012* (2012), S. Goldwasser, Ed., ACM, pp. 309–325.

[5] BURKHART, M., STRASSER, M., MANY, D., AND 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* (2010), USENIX Association, pp. 223–240.

[6] CHASE, M., GHOSH, E., AND POBURINNAYA, O. Secret-shared shuffle. In *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* (2020), S. Moriai and H. Wang, Eds., vol. 12493 of *Lecture Notes in Computer Science*, Springer, pp. 342–372.

[7] CHASE, M., AND MIAO, P. Private set intersection in the internet setting from lightweight oblivious PRF. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III* (2020), D. Micciancio and T. Ristenpart, Eds., vol. 12172 of *Lecture Notes in Computer Science*, Springer, pp. 34–63.

[8] CHEN, H., HUANG, Z., LAINE, K., AND RINDAL, P. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018* (2018), D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM, pp. 1223–1237.

[9] CHEN, H., LAINE, K., AND PLAYER, R. Simple encrypted arithmetic library - SEAL v2.1. In *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers* (2017), M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds., vol. 10323 of *Lecture Notes in Computer Science*, Springer, pp. 3–18.

[10] CHEN, H., LAINE, K., AND RINDAL, P. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017* (2017), B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM, pp. 1243–1255.

[11] CONG, K., MORENO, R. C., DA GAMA, M. B., DAI, W., ILIASHENKO, I., LAINE, K., AND ROSENBERG, M. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021* (2021), Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., ACM, pp. 1135–1150.

[12] DEVROYE, L., AND MORIN, P. Cuckoo hashing: Further analysis. *Inf. Process. Lett. 86*, 4 (2003), 215–219.

[13] FAN, J., AND VERCAUTEREN, F. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.* (2012), 144.

[14] FOTAKIS, D., PAGH, R., SANDERS, P., AND SPIRAKIS, P. G. Space efficient hash tables with worst case constant access time. In *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings* (2003), H. Alt and M. Habib, Eds., vol. 2607 of *Lecture Notes in Computer Science*, Springer, pp. 271–282.

[15] GARIMELLA, G., MOHASSEL, P., ROSULEK, M., SADEGHIAN, S., AND SINGH, J. Private set operations from oblivious switching. In *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* (2021), J. A. Garay, Ed., vol. 12711 of *Lecture Notes in Computer Science*, Springer, pp. 591–617.

[16] GENTRY, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009* (2009), M. Mitzenmacher, Ed., ACM, pp. 169–178.

[17] GENTRY, C., HALEVI, S., AND SMART, N. P. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings* (2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*, Springer, pp. 850–867.

[18] GILAD-BACHRACH, R., DOWLIN, N., LAINE, K., LAUTER, K. E., NAEHRIG, M., AND WERNSING, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016* (2016), M. Balcan and K. Q. Weinberger, Eds., vol. 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 201–210.

[19] HOGAN, K., LUTHER, N., SCHEAR, N., SHEN, E., STOTT, D., YAKOUBOV, S., AND YERUKHIMOVICH, A. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE Cybersecurity Development, SecDev 2016, Boston, MA, USA, November 3-4, 2016* (2016), IEEE Computer Society, pp. 75–76.

[20] HUANG, Y., EVANS, D., AND 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* (2012), The Internet Society.

[21] ISHAI, Y., KILIAN, J., NISSIM, K., AND PETRANK, E. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings* (2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 145–161.

[22] JIA, Y., SUN, S., ZHOU, H., DU, J., AND GU, D. Shuffle-based private set union: Faster and more secure. *IACR Cryptol. ePrint Arch.* (2022), 157.

[23] KOLESNIKOV, V., ROSULEK, M., TRIEU, N., AND WANG, X. Scalable private set union from symmetric-key techniques. In *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* (2019), S. D. Galbraith and S. Moriai, Eds., vol. 11922 of *Lecture Notes in Computer Science*, Springer, pp. 636–666.

[24] LENSTRA, A. K., AND VOSS, T. Information security risk assessment, aggregation, and mitigation. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings* (2004), H. Wang, J. Pieprzyk, and V. Varadharajan, Eds., vol. 3108 of *Lecture Notes in Computer Science*, Springer, pp. 391–401.

[25] LINDELL, Y. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, Y. Lindell, Ed. Springer International Publishing, 2017, pp. 277–346.

[26] MOHASSEL, P., AND SADEGHIAN, S. S. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings* (2013), T. Johansson and P. Q. Nguyen, Eds., vol. 7881 of *Lecture Notes in Computer Science*, Springer, pp. 557–574.

[27] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[28] PAGH, R., AND RODLER, F. F. Cuckoo hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings* (2001), F. M. auf der Heide, Ed., vol. 2161 of *Lecture Notes in Computer Science*, Springer, pp. 121–133.

[29] PINKAS, B., ROSULEK, M., TRIEU, N., AND YANAI, A. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III* (2019), A. Boldyreva and D. Micciancio, Eds., vol. 11694 of *Lecture Notes in Computer Science*, Springer, pp. 401–431.

[30] PINKAS, B., SCHNEIDER, T., AND ZOHNER, M. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur. 21*, 2 (2018), 7:1–7:35.

[31] RABIN, M. O. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.* (2005), 187.

[32] SMART, N. P., AND VERCAUTEREN, F. Fully homomorphic SIMD operations. *Des. Codes Cryptogr. 71*, 1 (2014), 57–81.

[33] ZHANG, C., CHEN, Y., LIU, W., ZHANG, M., AND 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.