Efficient Multi-party Private Set Union Resistant to Maximum Collusion Attacks

Qiang Liu[1] and Joon-Woo Lee[1(✉)]

[1] Department of Computer Science and Engineering,
Chung-Ang University, Seoul, Republic of Korea
liuqiang0321@gmail.com,
jwlee2815@cau.ac.kr

Abstract. Multi-party Private Set Union (MPSU) enables multiple participants to jointly compute the union of their private sets without leaking any additional information beyond the resulting union. Liu et al. (ASIACRYPT 2023) presented the first MPSU protocol that scales to large data sets, designating one participant as the "leader" responsible for obtaining the final union. However, this approach assumes that the leader does not collude with any other participant, which can be impractical due to the inherent lack of mutual trust among participants, thereby limiting its applicability.

On the other hand, the state-of-the-art protocol that allows all participants to learn the computed union was proposed by Seo et al. (PKC 2012). While their construction achieves $O(1)$ round complexity, it remains secure only if fewer than half of the participants collude, leaving open the problem of designing stronger collusion tolerance and multi-party output.

In this work, we address these limitations by first proposing $\Pi_{\text{MPSU}}^{\text{one-leader}}$ that designates one participant as leader to obtain the union result. Building upon this construction, we extend this design to $\Pi_{\text{MPSU}}^{\text{leaderless}}$, which enables every participant to receive the union result simultaneously. Both protocols operate under the semi-honest model, tolerate maximal collusion among participants, and efficiently handle large-scale set computation. We implement these schemes and conducted a comprehensive comparison against state-of-the-art solutions. The result shows that, for input sizes of 2^{12} at a comparable security level, $\Pi_{\text{MPSU}}^{\text{one-leader}}$ achieves a 663 times speedup in online runtime compared to the state-of-the-art. Furthermore, it also remains 22 times faster than half-collusion-tolerant protocol.

1 Introduction

Private Set Union (PSU) is a fundamental cryptographic protocol that enables two or more participants to securely compute the union of their datasets without disclosing any private information. With the growing demand for privacy protection, PSU protocols have become increasingly vital in scenarios involving joint data analysis, finding applications in areas such as network risk assessment [1], [2] and data mining [3], [4], where securely computing the union of sets is crucial. For example, financial institutions like banks may need to share blacklists of suspicious transactions or fraudulent accounts to better prevent financial fraud. These blacklists contain sensitive information and are generated based on each bank's risk detection strategy, making direct sharing impractical due to privacy and security concerns. Using PSU, banks can securely compute the union of their blacklists without revealing customer privacy or detection strategies, thereby achieving comprehensive risk assessment and efficient collaboration while mitigating privacy and strategy leakage inherent in traditional sharing methods.

During the past decade, Private Set Intersection (PSI) has garnered extensive research, with two-party PSI making particularly notable progress [5–13]. The most efficient two-party PSI protocol [12] now achieve performance comparable to that of insecure naive hashing PSI. Furthermore, benefiting from advancements in two-party PSI, efficient multi-party PSI (MPSI) for large sets containing millions of items have also seen significant development [14–18]. In contrast, PSU has received relatively less attention. For more than a decade after Kisser and Song initially proposed the two-party PSU protocol [19], PSU remained limited to relying on expensive additive homomorphic encryption (AHE) and complex circuits, resulting in low performance that made practical large-scale set operations nearly impossible. It wasn't until 2019, when Kolesnikov et al. introduced the first two-party PSU protcol based on symmetric key operations suitable for large-scale datasets [3], that significant progress was made. In the following years, several higher-performance two-party PSU protocols were developed [20, 21], building upon Kolesnikov et al.'s work. Notably, the proposed by Zhang et al. achieved linear communication and computation complexity for PSU in 2023 [22], marking a significant breakthrough in this field.

Similar to the situation with two-party PSU, research on MPSU has primarily relied on expensive AHE [19, 23–25] and generic MPC settings [26], keeping it largely theoretical and inefficient and no scalable for practical use. In 2023, Liu et al. introduced the first MPSU protocol capable of handling large sets [27], which allows a leader to obtain the computed union. Compared to previous state-of-the art, Liu et al. achieved over a hundredfold performance improvement. However, despite its efficiency in handling large datasets, it also has significant limitations:

- Firstly, the protocol relies on a leader, and its security is guaranteed only when the leader does not collude with other participants, which is impractical in real-world applications since we cannot ensure that the leader will not collude with other participants.

- Secondly, the protocol only allows a leader to obtain the final result. Although the leader can theoretically distribute the result to other participants honestly, in time-sensitive scenarios such as auctions, bidding, or decision-making processes, gaining the result first may confer an unfair advantage. Consequently, many organizations would rather incur additional computational and communication overhead than tolerate any participant acquiring the result before the others. Thus, the protocol may prove inadequate when faced with the aforementioned scenarios.

Currently, the scheme [24] proposed by Seo et al. in 2012 can ensure all participants obtain the union result simultaneously. However, it can only resist collusion attacks involving up to half of the participants. Considering these limitations in [27] and [24], we pose the following question:

Under the assumption of maximal semi-honest collusion, is it possible to design two scalable and efficient protocols: (1) a one-leader protocol in which a single participant obtains the final union result, and (2) a leaderless protocol in which all participants receive the union result simultaneously?

1.1 Contribution

In this work, we provide a definitive answer to the aforementioned question. Specifically, our contribution can be summarized as follows:

- We analyze and summarize the major two-party PSU and MPSU protocols to date, and conduct a comprehensive review and classification of their communication and computation complexity from both two-party and multi-party perspectives.
- We propose a MPSU protocol, $\Pi_{\text{MPSU}}^{\text{one-leader}}$, which tolerates maximal semi-honest collusion. The protocol is divided into an offline phase and an online phase, with the core part of the online phase labeled as $\Pi_{\text{MPSU}}^{\text{Core}}$. This design enables efficient and scalable computation for large input sets. Upon completion of the protocol, the leader obtains the union result.
- We then extend $\Pi_{\text{MPSU}}^{\text{one-leader}}$ to a leaderless variant, $\Pi_{\text{MPSU}}^{\text{leaderless}}$, which also tolerates maximal semi-honest collusion and allows all participants to simultaneously obtain the union result. Its offline phase remains the same as in $\Pi_{\text{MPSU}}^{\text{one-leader}}$, while the online cost grows proportionally with the number of participants.
- We implement the proposed protocols and compare them with [25] and [26] under a scenario involving four participants. The results show that our protocols significantly outperform the state-of-the-art schemes. Specifically, for the one-leader setting with input size 2^{12}, our protocol reduces the online runtime by approximately 663 times and 22 times compared to [25] and [26], respectively, while cutting down communication overhead by about 4 times and 471 times. Moreover, for input size 2^{16}, our protocol achieves the 568 times reduction in communication overhead relative to [26]. The implementation is released on Github: `https://github.com/QIANG-crypto-230608/LL2024.git`.

1.2 Related Work

We conduct a theoretical comparative analysis of the current major two-party PSU and MPSU protocols under the semi-honest security setting.

Two-party PSU. Initially, in 2005, Kissner and Song proposed a pioneering PSU protocol based on polynomial representations and AHE [19]. The union of two sets was determined by computing the roots of the polynomial $f \times g$, but its quadratic complexity limited its practical utility. Later, in 2007, Frikken introduced an improved PSU protocol with linear communication complexity [23]. The receiver encrypts a polunomial f and sends it to the sender, who processes each $y \in Y$ and returns the results. The receiver then decrypts these to determine the union. Despite communication efficiency has been improved, the computational complexity remained $O(n^2)$. In 2007, Davidson and Cid proposed an enhanced PSU protocol using Bloom Filters (BF) along with AHE [28]. The receiver flips bits in the BF, encrypts it to create an Encrypted Inverse BF (EIBF), and sends it to the sender. The sender processes the EIBF and returns ciphertexts to the receiver, who decrypts them to obtain the union. While the computational complexity is reduced to $O(\lambda n)$, communication complexity increases to $O(\kappa\lambda n)$ due to IBF encryption and decryption.

Kolesnikov et al. (2019) introduced a PSU protocol based on symmetric key encryption [3], utilizing the RPMT protocol to verify if an item from the sender is in the receiver's set. This protocol achieved a magnitude performance improvement with communication and computational complexity of $O(\kappa\lambda \log n)$ and $O(n \log n \log\log n)$, respectively. Later, in 2021, Garimella et al. proposed a PSU protocol using Permuted Characteristic (PC) [10], relying on the Oblivious Switching Network (OSN) [39], which reduced the computational complexity to $O(n \log n)$, resulting in a performance improvement of $2 - 2.5$ times over [3]. In 2022, Jia et al. further enhanced the protocol using OSN to address input set imbalances by shuffling both the receiver's and sender's sets [21], achieving a $4 - 5$ times speedup compared to [3], while maintaining similar complexity to [10]. Recently, in 2023, Zhang et al. designed a novel multi-query RPMT (mq-RPMT) functionality for constructing their PSU protocol. They proposed two approaches: one based on symmetric key encryption (SKE) and the other based on re-randomized public key encryption (PKE). Their proposed achieved linear complexity with the lowest communication cost among current protocols.

The core idea of the current two-party PSU protocols suitable for large-scale sets is that receiver first determines whether each item in the sender's set X belongs to the receiver's set Y. Subsequently, the difference $Y \backslash X$ is obtained using OT protocol. However, this approach is challenging to directly compute the union $\bigcup_{i=1}^{n} X_i$ using the current two-party technologies, such as by computing the union $X_1 \cup (X_2 \backslash X_1) \cup \cdots \cup (X_n \backslash (X_1 \cup X_2 \cup \cdots \cup X_{n-1}))$, it would not only reveal the cardinality of difference $|X_i \backslash (X_1 \cup X_2 \cup \cdots \cup X_{i-1})|$ but also fail to guarantuee the privacy of the items in $X_i \backslash (X_1 \cup X_2 \cup \cdots \cup X_{i-1})$ during each union computation. Therefore, the approach of two-party PSU cannot meet the privacy and security requirements of the MPSU.

MPSU. Kissner and Song proposed the earliest MPSU protcol based on polynomial representation and AHE [19]. In their protocol, each party P_i ($i \in [t]$) represents their input set $X_i = \{x_{i1}, x_{i2}, ..., x_{in}\}$ as a polynomial $f_i(x) = \prod_{j=1}^{n}(x - x_j)$, whose roots are the set items. The product of these polynomials represents the union of all input sets. Their approach requires a large number of AHE operations and high-degree polynomial evaluations with a computational complexity of $O(t^3 n^2)$. Subsequently, Frikken proposed an improved method based on the ideas in [19], but it still requires $O(t^2 n \log n)$ multiplication operations [23]. In their protocol, party P_1 encrypts f_1 using AHE and sends it to P_2, who then evaluates it for each item in set X_2. The difference set $X_2 \backslash X_1$ is then computed based on these evaluations, which can be repeated for additional sets to compute the final union. In 2012, Seo et al. proposed a new approach based on rational polynomial functions and reversed Laurent series, associating each party's input set with a rational function and providing constant-round complexity [24]. Although it improves efficiency compared to [19], [23], its security is limited to scenarios where at most $t/2$ parties collude. Gong et al. proposed a constant-round MPSU protocol based on AHE and BF in 2022 [25]. The protocol first constructs a BF to store the union and exploits the no-collision property of BF to determine if each position is mapped by only one item, thereby identifying the items in the set. With the length of the BF depends on the statistical security parameter and the union size, resulting in a large number of AHE operations and computational overhead, making it impractical.

In additional to using expensive AHE, employing generic Secure Multi-party Computation (SMPC) to address the MPSU is also an interesting approach. In 2012, Blanton and Aguiar proposed an MPSU protocol based on Oblivious Sorting and generic SMPC, which operates in an honest majority setting [26]. The core idea is to merge all participants' sets into a large set, perform an oblivious sort, and then remove duplicate items by comparing adjacent items to obtain the union. Nevertheless, due to its reliance on SMPC techniques, specifically using Batcher's network [30] to sort the union with $O(n \log^2 n)$ comparisons, the protocol is inefficient when handling larger set sizes or a greater number of participants.

Recently, Liu and Gao in 2023 proposed an MPSU protocol for efficiently computing large-scale sets, based on symmetric key operations [27]. The protocol relies on a multi-party secret-shared shuffle and a multi-query secret-shared private membership test as its core components. Under the presence of a leader, the protocol allows the leader to obtain the union result by interacting with other participants. However, the security of their protocol depends on the assumption that the leader does not collude with other participants, which results in weaker security guarantees.

Following, we summarize the communication and computational complexity, as well as the encryption operations used in two-party PSU schemes, in Table 1. Additionally, Table 2 presents a comparison of the communication and computational complexity, resistance to collusion attacks, and encryption operations for MPSU schemes.

Protocol	**Year**	**Comm.**	**Comp.**	**Enc. ope.**
[19]	2005	$O(n^2)$	$O(n^2)$	PKE
[23]	2007	$O(n)$	$O(n \log \log n)$	
[28]	2017	$O(n)$	$O(n)$	
PKE-based [22]	2023	$O(n)$	$O(n)$	
[3]	2019	$O(n \log n)$	$O(n \log n \log \log n)$	SKE
[10]	2021	$O(n \log n)$	$O(n \log n)$	
[21]	2022			
SKE-based [22]	2023	$O(n)$	$O(n)$	

Table 1: Asymptotic communication and computation complexities of semi-honest secure Two-party PSU protocols. Note: Comm./Comp.: Communication/Computational complexity; Enc. ope.: Encryption operations; PKE: Public-key encryotion operations; SKE: Symmetric key encryotion operations; n: Size of sets.

2 Preliminaries

2.1 Notation

Assume $\mathcal{P}_i$ ($i \in [t]$) denotes a participant, and $X_i = \{x_i^1, x_i^2, ..., x_i^n\}$ represents the set held by each participant $\mathcal{P}_i$, where x_i^j ($j \in [n]$) denotes the j-th item in set X_i. $[n]$ denotes the set

Protocol	Year	Comm.	Comp.	Threshold	Ope.
[26]	2012	$O(t^2 n \log^2(nt))$	$O(t^2 n \log^2(nt))$	$< \lceil t/2 \rceil$	Oblivious sorting
[24]	2012	$O(t^3 n^2)$	$O(t^4 n^2)$	$< \lceil t/2 \rceil$	SKE
[27]	2023	$O(t^2 n \log(tn))$	$O(tn)$	$< t$ (leader no colludes with others)	
[19]	2005	$O(t^2 n)$	$O(t^3 n^2)$	$< t$	PKE
[23]	2007	$O(t^2 n)$	$O(t^2 n \log n)$	$< t$	
[25]	2022	$O(tn)$	$O(tn)$	$< t$	
Our one-leader	-	$O(t^2 n)$	$O(t^2 n)$	$< t$	
Our leaderless	-	$O(t^3 n)$	$O(t^3 n)$	$< t$	

Table 2: Asymptotic communication and computation complexities of semi-honest secure MPSU protocols. Note: Comm./Comp.: Communication complexity/Computational complexity; Ope.: Operation technique; PKE: Public-key encryotion operations; SKE: Symmetric key encryotion operations; t: Number of parties; n: Size of sets.

$\{1, 2, .., n\}$. κ, λ are the computational security parameter and the statistical security parameter respectively. $r \xleftarrow{\$} \{0,1\}^\ell$ denotes a randomly generated binary number r with bit length ℓ. $||$ represents the string concatenation operator, such taht $a||b$ means concatenating strings a and b. Pseudorandom function is denoted by PRF, and probabilistic polynomial-time is abbreviated as PPT.

2.2 Security Model

In this work, we consider the adversaries are semi-honest, or honest-but-curious, meaning that adversaries strictly follow the protocol specifications but attempt to learn additional information by examining the protocol transcripts. We use the "simulation paradigm" method from [31] to prove the semi-honest security, with the specific definition given below.

Definition 2.1. *Let $f : (\{0,1\}^*)^t \to (\{0,1\}^*)^t$ be an t-ary functionality, where $f_i(x_1, ..., x_t)$ denotes the i-th item of $f(x_1, ..., x_t)$. For $I = (i_1, ..., i_k) \subseteq [t]$, we let $f_I(x_1, ..., x_t)$ denote the subsequence $f_{i_1}(x_1, ..., x_t)$,...,$f_{i_k}(x_1, ..., x_t)$. Let Π be an t-party protocol for computing f. The view of the i-th party during an execution of Π on $\bar{x} = (x_1, ..., x_t)$ is denoted by $\mathsf{View}_i^\Pi(\bar{x})$. For $I = (i_1, ..., i_k)$, we let $\mathsf{View}_I^\Pi(\bar{x}) \overset{def}{=} (I, \mathsf{View}_{i_1}^\Pi(\bar{x}), ..., \mathsf{View}_{i_k}^\Pi(\bar{x}))$. We say that Π privately computes f if there exists a PPT algorithm, denoted Sim, such that for every $I \subseteq [t]$, it holds that*

$$\{(\mathsf{Sim}(I, (x_{i_1}, ..., x_{i_k}), f_I(\bar{x})), f(\bar{x}))\}_{\bar{x} \in (\{0,1\}^*)^t} \overset{c}{\equiv} \{\mathsf{View}_I^\Pi(\bar{x}), \mathsf{Output}^\Pi(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^t},$$

where $\mathsf{Output}^\Pi(\bar{x})$ denotes the output sequence of all parties during the execution represented in $\mathsf{View}_I^\Pi(\bar{x})$.

2.3 Building Blocks

Oblivious Programmable PRF An Oblivious PRF (ORRF) [32] allows the sender to learn the PRF key k, while the receiver learns $F(k, x)$, where F is a PRF and x is the receiver's input. In an Oblivious Programmable PRF (OPPRF) [14], the PRF further allows to "program" the output of F on a limited number of inputs, namely $\mathcal{P} = \{(x_j, y_j)\}_{j \in [n]}$, which means that the PRF value for x_j is encoded as y_j. The receiver's input is $\{q_i\}_{i \in [t]}$. After the execution, the sender obtains (k, hint), while the receiver learns the PRF output $\{F(k, \mathsf{hint}, q_i)\}_{i \in [t]}$ and the hint. Importantly, the receiver does not know whether the obtained output items $F(k, \mathsf{hint}, q_i)$ match the sender's programmed PRF input items y_j, i.e., if $q_i = x_j$, then $F(k, \mathsf{hint}, q_i) = y_j$; otherwise, $F(k, \mathsf{hint}, q_i)$ is pseudorandom. The ideal functionality $\mathcal{F}_{\text{OPPRF}}$ is given in Fig. 1.

Functionality $\mathcal{F}_{\text{OPPRF}}$

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$;
- A Programmable PRF F with the key generation algorithm $\mathsf{KeyGen}(\cdot)$;
- Upper bound n on the number of points to be programmed;
- Upper bound t on the number of queries.

Functionality:

1. Wait for input $\mathcal{P} = \{(x_1, y_1), ..., (x_n, y_n)\}$ from the sender $\mathcal{S}$;
2. Wait for input $\{q_1, ..., q_t\}$ from the receiver $\mathcal{R}$;
3. Run $(k, \mathsf{hint}) \leftarrow \mathsf{KeyGen}(\mathcal{P})$ and compute $F(k, \mathsf{hint}, q_1), ..., F(k, \mathsf{hint}, q_t)$;
4. Give output (k, hint) to the sender $\mathcal{S}$ and output $(\mathsf{hint},\ \{F(k, \mathsf{hint}, q_i)\}_{i \in [t]})$ to the receiver $\mathcal{R}$.

Fig. 1. Ideal Functionality for Oblivous Programmble PRF.

Permute + Share The Permute + Share (PS) functionality $\mathcal{F}_{\text{PS}}$ was proposed by Chase et al. in 2020 [33]. In this functionality, there are two participants, namely a sender and a receiver. The sender holds a set $\{x_1, ..., x_n\}$ of size n, where each item has a bit length of ℓ, while the receiver chooses a permutation function π to permute the n items. The goal of $\mathcal{F}_{\text{PS}}$ is for the sender to learn the share values $\{s_{\pi(1)}, ..., s_{\pi(n)}\}$, while the receiver obtains the other shared values $\{s_{\pi(1)} \oplus x_{\pi(1)}, ..., s_{\pi(n)} \oplus x_{\pi(n)}\}$. The concrete ideal functionality $\mathcal{F}_{\text{PS}}$ is shown in Fig. 2.

Functionality $\mathcal{F}_{\text{PS}}$

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$.

Functionality:

1. Wait for input $\{x_1, ..., x_n\} \in (\{0,1\}^{\ell})^n$ from the sender $\mathcal{S}$;
2. Wait for input a permutation $\pi : [n] \rightarrow [n]$ from the receiver $\mathcal{R}$;
3. Generate the shares $\{s_{\pi(1)}, ..., s_{\pi(n)}\} \in (\{0,1\}^{\ell})^n$;
4. Give output $\{s_{\pi(1)}, ..., s_{\pi(n)}\}$ to the sender $\mathcal{S}$ and output $\{s_{\pi(1)} \oplus x_{\pi(1)}, ..., s_{\pi(n)} \oplus x_{\pi(n)}\}$ to the receiver $\mathcal{R}$.

Fig. 2. Ideal Functionality for Permute + Share.

Private Equality Test Private Equality Test (PEqT) is used to determine whether two strings are equal. More specifically, two participants, namely a sender and a receiver, each hold a string x and y respectively. By invoking the PEqT functionality, both participants input their respective strings x and y, and eventually the sender obtains an output bit b, while the receiver obtains an output bit b^*, such that if $x = y$, then $b \oplus b^* = 0$; otherwise, $b \oplus b^* = 1$. The ideal functionality $\mathcal{F}_{\text{PEqT}}$ is illustrated in Fig. 3.

Functionality $\mathcal{F}_{\text{PEqT}}$

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$.

Functionality:

1. Wait for input x from the sender $\mathcal{S}$;
2. Wait for input y from the receiver $\mathcal{R}$;
3. Generate b and b^* satisfying that if $x = y$, $b \oplus b^* = 0$, otherwise, $b \oplus b^* = 1$;
4. Give output b to the sender $\mathcal{S}$ and output b^* to the receiver $\mathcal{R}$.

Fig. 3. Ideal Functionality for Private Equality Test

1-out-of-2 OT Extension 1-out-of-2 OT Extension ($\binom{2}{1}$-OTe) is a major cryptology building block [34]. It allows the sender input n message pairs (x_0^i, x_1^i) with each message is ℓ-bit, while the receiver holds n-bit choice vector b. At the end, the receiver obtains the output result $x_{b[i]}^i$, but the sender obtains nothing. The ideal functionality $\mathcal{F}_{\binom{2}{1}\text{-OTe}}$ is shown in Fig. 4

Functionality $\mathcal{F}_{\binom{2}{1}\text{-OTe}}$

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$;
- Computational security parameter κ;
- Number of based-OTs $\beta = \kappa$.

Functionality:

1. Wait for input $(x_0^i, x_1^i) \in (\{0,1\}^\ell)^{2n}$ from the sender $\mathcal{S}$;
2. Wait for input choice vector $b \in \{0,1\}^n$ from the receiver $\mathcal{R}$;
3. Give output $(x_{b[1]}^1, ..., x_{b[n]}^n)$ to the receiver $\mathcal{R}$.

Fig. 4. Ideal Functionality for 1-out-of-2 OT Extension

Simple Hashing In the simple hashing scheme, β hash functions $h_1, \cdots, h_\beta : \{0,1\}^* \to [b]$ are used to map n items into b bins $B_1, \cdots, B_b$. Each item x_i $(i \in [n])$ is added to bins $B_{h_1(x_i)}, B_{h_2(x_i)}, \cdots, B_{h_\beta(x_i)}$, regardless of whether those bins are already occupied. In other words, an item can be assigned to multiple bins, which introduces redundancy in data distribution, thereby improving the flexibility and reliability of data retrieval. According to the inequality, when hashing n items into b bins, the maximum bin size ρ can be set to ensure that except with a probability of $2^{-\lambda}$, no bin will contain more than ρ items [35]. Specifically, ρ satisfies the condition

$$Pr[\exists \text{bin with} \geq \rho \text{ items}] \leq b[\sum_{i=\rho}^{n} \binom{n}{i} \cdot (\tfrac{1}{b})^i \cdot (1-\tfrac{1}{b})^{n-i}].$$

Cuckoo Hashing Cuckoo Hashing was introduced by Pagh and Rodler in 2001 [36]. It uses multiple hash functions $h_1, h_2, \ldots, h_\gamma$ to map items to bins in a hash table of size $b = \epsilon n$. The distinctive feature of Cuckoo Hashing is its ability to ensure that each bin contains only one item, thereby maintaining structural simplicity and minimizing collisions. The core of this technique lies in a deterministic eviction process. When an item x is inserted, it is initially hashed into one of the bins $B_{h_1(x)}, B_{h_2(x)}, \ldots, B_{h_\gamma(x)}$. If all designated bins are occupied, the algorithm randomly selects one of the occupied bins, evicts the item currently in that bin, and re-inserts it into another bin determined by a different hash function. This process is repeated iteratively until either a vacant bin is found, or the number of evictions reaches a predefined threshold, indicating that insertion into the regular hash table is infeasible. In such cases, the item is placed into a small auxiliary space called the "stash," reserved for overflow items. Mathematical analysis of Cuckoo Hashing shows that, by appropriately tuning the parameters γ and ϵ, the stash size can be minimized to zero in most practical scenarios while maintaining a failure probability as low as $2^{-\lambda}$ [6].

Encryption Scheme In our constructed protocols, the AHR-TPKE (Additively Homomorphic Re-randomizable TPKE) scheme is used. We provide the formal definition of AHR-TPKE as given in Appendix A following the description in [37].

3 Technique Overview

We present the ideal functionality of the MPSU we aim to achieve, as shown in Fig. 5. Assume there are t participants $\mathcal{P}_1, ..., \mathcal{P}_t$ each holding a set $X_i = \{x_1^i, ..., x_n^i\}$ of size n, where the maximum bit length of items in the set is ℓ. This ideal functionality encompasses both the one-leader and leaderless settings. In both scenarios, t participants $\mathcal{P}_i$ input their respective sets X_i to compute the union of all input sets. However, under the one-leader mode, only the leader receives the computed union $\bigcup_{i=1}^{t} X_i$, whereas in the leaderless mode, all participants obtain the union simultaneously. It is worth noting that the ideal functionality can easily compute the union of all input sets, and ensure that each participant obtains the result. However, during the execution, no additional information beyond the final output should be disclosed.

Based on the ideal functionality presented in Fig. 5, we design our MPSU protocols under the semi-honest security model that can resist collusion by up to $t-1$ adversarial participants among t participants. Before diving into the technical overview of our MPSUs, we briefly describe a interesting module that plays a fundamental role in our designed MPSU protocols.

Batched Equality-tested Oblivious Random Generation (BEtORG) is constructed in this work as a interesting block of our MPSU protocol, and the flow is illustrated in Fig. 6. As shown in Fig. 6, the BEtORG protocol begins by invoking the PEqT functionality n times to determine the relationship between x_i and y_i for each pair. Specifically, in each PEqT invocation, a bit value is sent to the sender and the receiver, denoted e_i and $\tilde{e}_i$, respectively. It holds that

Functionality $\mathcal{F}_{\text{MPSU}}$

Parameters:

- t parties $\mathcal{P}_1, ..., \mathcal{P}_t$ and the set size n;
- The maximum bit length of all items is ℓ

Functionality:

1. Wait for input set $X_i = \{x_1^i, ..., x_n^i\} \in (\{0,1\}^*)^n$ from $\mathcal{P}_i$;
2. (a) One-leader: Give output $\bigcup_{i=1}^t X_i$ to $\mathcal{P}_t$.
   (b) Leaderless: Give output $\bigcup_{i=1}^t X_i$ to $\mathcal{P}_1, ..., \mathcal{P}_t$ simultaneously.

Fig. 5. Ideal Functionality for Multi-party Private Set Union

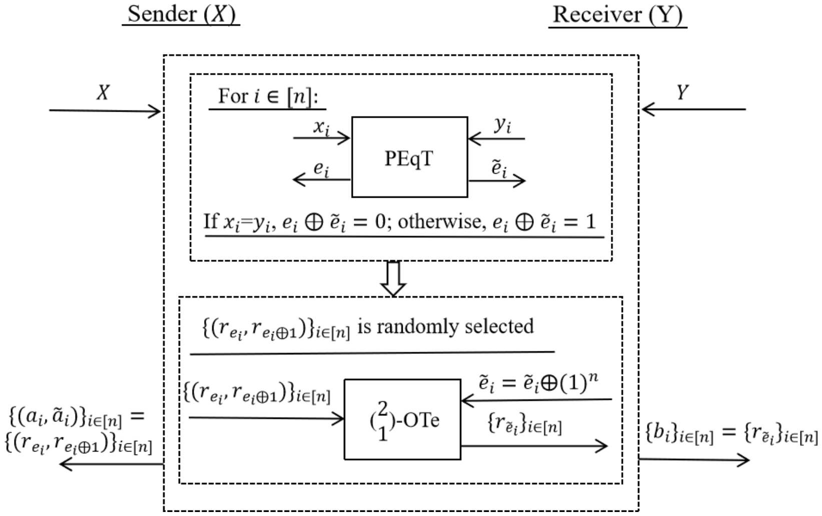
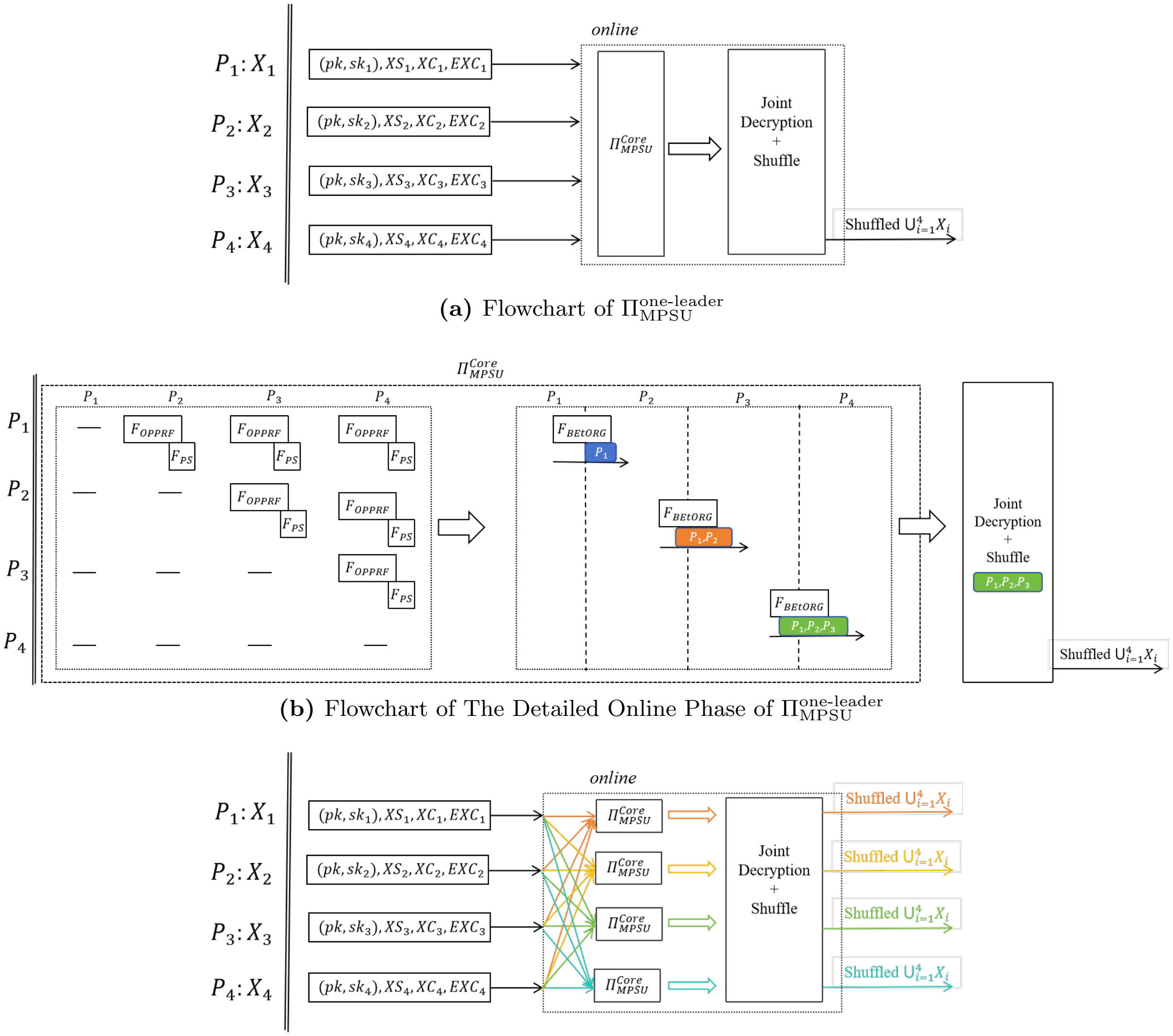

Sender (X)
Receiver (Y)
X
Y
For $i \in [n]$:
x_i
y_i
PEqT
e_i
$\tilde{e}_i$
If x_i=y_i, $e_i \oplus \tilde{e}_i = 0$; otherwise, $e_i \oplus \tilde{e}_i = 1$
$\{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$ is randomly selected
$\{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$
$\binom{2}{1}$-OTe
$\tilde{e}_i = \tilde{e}_i \oplus (1)^n$
$\{r_{\tilde{e}_i}\}_{i\in[n]}$
$\{(a_i, \tilde{a}_i)\}_{i\in[n]} =$
$\{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$
$\{b_i\}_{i\in[n]} = \{r_{\tilde{e}_i}\}_{i\in[n]}$

Fig. 6. Illustration of The BEtORG Protocol

if $x_i = y_i$, then $e_i \oplus \tilde{e}_i = 0$; otherwise, $e_i \oplus \tilde{e}_i = 1$. Next, the sender randomly generates n pairs of strings $\{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$ as inputs for the 1-out-of-2 OTe, while the receiver inputs $\tilde{e}_i = \tilde{e}_i \oplus \{1\}^n$. As a result, the receiver obtains the n obliviously transferred string $\{r_{\tilde{e}_i}\}_{i\in[n]}$. More specifically, if BEtORG is treated as a block box, where the sender and the receiver input their sets X and Y, the final output will provide each party with a respective result: the sender will obtain $\{(r_{e_i}, r_{e_i\oplus 1}\}_{i\in[n]}$, denoted as $\{(a_i, \tilde{a}_i\}_{i\in[n]}$, while the receiver will obtain $\{r_{\tilde{e}_i}\}_{i\in[n]}$, denoted as $\{b_i\}_{i\in[n]}$. In other words, apart from the final output obtained by themselves, the participants cannot obtain any other private information.

3.1 Our One-leader MPSU Protocol

Fig. 7 (a) and (b) respectively illustrate the overall workflow of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ and the detailed flowchart of its online phase. As shown in Fig. 7 (a), for simplicity, we take four participants as an example to illustrate the joint computation. The four participants $\mathcal{P}_i$ jointly execute the threshold re-randomizable PKE scheme to generate a shared public key pk and individual secret key sk_i. Simultaneously, based on their respective sets X_i, each participant also constructs their


online
$P_1: X_1$
$P_2: X_2$
$P_3: X_3$
$P_4: X_4$
$(pk, sk_1), XS_1, XC_1, EXC_1$
$(pk, sk_2), XS_2, XC_2, EXC_2$
$(pk, sk_3), XS_3, XC_3, EXC_3$
$(pk, sk_4), XS_4, XC_4, EXC_4$
Π^{Core}_{MPSU}
Joint Decryption + Shuffle
Shuffled $\cup^4_{i=1} X_i$
P_1
P_2
P_3
P_4
F_{OPPRF}
F_{PS}
F_{BEtORG}
P_1
P_1, P_2
P_1, P_2, P_3

(a) Flowchart of $\Pi^{\text{one-leader}}_{\text{MPSU}}$

(b) Flowchart of The Detailed Online Phase of $\Pi^{\text{one-leader}}_{\text{MPSU}}$

(c) Flowchart of The Protocol $\Pi^{\text{leaderless}}_{\text{MPSU}}$

Fig. 7. Illustration of Our New MPSU Protocols for 4 Participants.

own simple hashing table XS_i, cuckoo hashing table XC_i, and encrypted cuckoo hashing table EXC_i, all with a maximum bin size b. Notably, for $j \in [b]$, $\mathsf{EXC}_i[j] = \mathsf{Enc}(\mathsf{pk}, H(\mathsf{XC}_i[j])||\mathsf{XC}_i[j])$.

After the preparation phase is completed, each of the four participants $\mathcal{P}_i$ provides the constructed inputs $((\mathsf{pk}, \mathsf{sk}_i), \mathsf{XS}_i, \mathsf{XC}_i, \mathsf{EXC}_i)$ to the online phase. As illustrated in Fig. 7 (a), the online phase consists of two parts. First, the intermediate output is computed through the $\Pi^{\text{Core}}_{\text{MPSU}}$. Then, this output undergoes partial decryption and shuffling sequentially through each participant, until the final participant, $\mathcal{P}_4$, completes the process and computes the final union set, that is shuffled $\bigcup_{i=1}^4 X_i$.

The above provided an overview of our $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol execution process. Next, we provide a detailed breakdown of the online phase in the $\Pi^{\text{one-leader}}_{\text{MPSU}}$, as depicted in Fig. 7 (b). First, the operation flow of $\Pi^{\text{Core}}_{\text{MPSU}}$ is as follows. The first stage within $\Pi^{\text{Core}}_{\text{MPSU}}$ is the $\mathcal{F}_{\text{OPPRF}}+\mathcal{F}_{\text{PS}}$ execution phase. (1) $\mathcal{P}_i$ $(i \in [4])$ runs $\mathcal{F}_{\text{OPPRF}}$ with other $\mathcal{P}_j$ $(j \in \{i+1, .., 4\})$, where $\mathcal{P}_i$ acts as the sender and $\mathcal{P}_j$ acts as the receiver. (2) Using the results from $\mathcal{F}_{\text{OPPRF}}$, $\mathcal{P}_i$ then runs $\mathcal{F}_{\text{PS}}$ with other $\mathcal{P}_j$, where $\mathcal{P}_i$ acts as the sender and $\mathcal{P}_j$ acts as the receiver. Next is the second stage of execution within $\Pi^{\text{Core}}_{\text{MPSU}}$. Initially, $\mathcal{P}_1$ and $\mathcal{P}_2$ execute $\mathcal{F}_{\text{BEtORG}}$, where $\mathcal{P}_1$ acts as the sender and $\mathcal{P}_2$ acts as the receiver. Then, $\mathcal{P}_1$ uses the result from $\mathcal{F}_{\text{BEtORG}}$ to mask the items in EXC_1, and sends the masked data to $\mathcal{P}_2$. For simplicity, we denote the masked and transmitted data as $\mathsf{packaged}[\mathcal{P}_1]$. Subsequently, $\mathcal{P}_2$ combines its own data with $\mathsf{packaged}[\mathcal{P}_1]$, and repeats the above process with $\mathcal{P}_3$. This process continues sequentially until $\mathcal{P}_3$ and $\mathcal{P}_4$ complete the operation, with $\mathcal{P}_4$ finally obtaining the data, denoted as $\mathsf{packaged}[\mathcal{P}_1\mathcal{P}_2\mathcal{P}_3]$ from $\mathcal{P}_3$.

Following, the second phase of the online phase, namely the joint decryption and shuffle phase, proceeds as follows. In this phase, the four participants sequentially perform partially decryption operations and shuffle the partially decrypted results. Once $\mathcal{P}_4$ completes this process, it receives the computed data, which corresponds to the shuffled set $(X_1 \cup X_2 \cup X_3) \setminus X_4$. Then the $\mathcal{P}_4$ performs the operation $((X_1 \cup X_2 \cup X_3) \setminus X_4) \cup X_4 = \bigcup_{i=1}^{4} X_i$. This marks the completion of $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol. It is worth noting that during the joint decryption + shuffle in the online phase, the decrypted data is randomly shuffled, meaning that the order of final output union set is different each time.

3.2 Our Leaderless MPSU Protocol

Our $\Pi^{\text{leaderless}}_{\text{MPSU}}$ protocol is an extension of the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol, as illustrated in Fig. 7 (c). From the flowchart, it is clear that the offline phase follows the same steps as the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol. This includes a series of set processing and computation steps, as well as the collaborative execution of the key generation algorithm for the encryption scheme by all participants. These steps produce the required inputs for the online phase, namely $((\mathsf{pk}, \mathsf{sk}_i), \mathsf{XS}_i, \mathsf{XC}_i, \mathsf{EXC}_i)$.

In the online phase, the $\Pi^{\text{leaderless}}_{\text{MPSU}}$ protocol extends the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol. Specifically, as described in Fig. 7 (b), running $\Pi^{\text{Core}}_{\text{MPSU}}$ enables the last participant in the protocol to obtain valid output information. Thus, in $\Pi^{\text{leaderless}}_{\text{MPSU}}$, we run $\Pi^{\text{Core}}_{\text{MPSU}}$ four times, with each participant taking on different roles during these runs. This ensures that, by the end of the protocol execution, each participant has obtained valid output information.

During the subsequent joint decryption and shuffle phase, all participants execute the following operations in parallel to ensure that each participant simultaneously obtains the union output. Specifically, for $i \in [4]$, the joint decryption and shuffle operations are sequentially performed in the order $\mathcal{P}_i \to \mathcal{P}_{i+1} \to \mathcal{P}_{i+2} \to \mathcal{P}_{i+3}$, where the indices are computed modulo 5 with an offset of +1. At the end of this process, $\mathcal{P}_{i+3}$ obtains the shuffled union set. It is important to note that, since the participants corresponding to each position vary with different values of i, the above process can be executed in parallel even in a single-threaded setting. This ensures that all participants simultaneously obtain the computed union set upon completion of the protocol. However, due to the random shuffling at each step, the union sets obtained by each participant, while identical in terms of elements, will have random orderings.

Summary, we have provided a detailed overview of the overall flow of our proposed protocols. In the next section, we will present a complete description and analysis of our protocols.

4 New MPSU protocol

In this section, we provide a detailed construction and analysis of the aforementioned $\Pi^{\text{one-leader}}_{\text{MPSU}}$ and $\Pi^{\text{leaderless}}_{\text{MPSU}}$ protocols. A key building used in the protocols is Π_{BEtORG}, which is newly developed in this work. Below, we first describe this new building block in detail, followed by a through introduction to $\Pi_{\text{MPSU}}^{\text{one-leader}}$ and $\Pi_{\text{MPSU}}^{\text{leaderless}}$.

4.1 New Building Block

As mentioned above, we construct a new building block called "Batched Equality-tested Oblivious Random Generation"(BEtORG), whose ideal functionality is shown in Fig. 8. In this construction, the sender and receiver hold sets $\{x_i\}_{i\in[n]} \in (\{0,1\}^\ell)^n$ and $\{y_i\}_{i\in[n]} \in (\{0,1\}^\ell)^n$, respectively, and use them as inputs to the functionality $\mathcal{F}_{\text{BEtORG}}$. Finally, the sender obtains the set $\{(a_i, \tilde{a}_i)\}_{i\in[n]}$ and the receiver obtains the set $\{b_i\}_{i\in[n]}$, satisfying that if $x_i = y_i$, $b_i = \tilde{a}_i$; otherwise, $b_i = a_i$. During the functionality execution, the string pairs of set $\{(a^i, \tilde{a}_i)\}_{i\in[n]}$ is generated randomly, satisfying that $(a_i, \tilde{a}_i) \overset{\$}{\leftarrow} (\{0,1\}^\ell)^2$. Subsequently, a conditional set $\{b_i\}_{i\in[n]}$ is generated, where $b_i \in \{0,1\}^\ell$. The term "conditional" means that if $x_i \neq y_i$, $b_i = a_i$; otherwise, $b_i = \tilde{a}_i$. Finally, set $\{(a_i, \tilde{a}_i)\}_{i\in[n]}$ is output to the sender, while $\{b_i\}_{i\in[n]}$ is output to the receiver. It is worth noting that upon completion of the functionality $\mathcal{F}_{\text{BEtORG}}$, neither the sender nor the receiver can distinguish whether $b_i = a_i$ or $b_i = \tilde{a}_i$.

Functionality $\mathcal{F}_{\text{BEtORG}}$

Parameters:

- Two parties: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$.

Functionality:

1. Wait for input set $\{x_1, ..., x_n\} \in (\{0,1\}^\ell)^n$ from the sender $\mathcal{S}$;
2. Wait for input set $\{y_1, ..., y_n\} \in (\{0,1\}^\ell)^n$ from the receiver $\mathcal{R}$;
3. Generate n string pairs $\{(a_i, \tilde{a}_i)\}_{i\in[n]}$ randomly. Construct a conditional set $\{b_i\}_{i\in[n]}$, s.t. if $x_i \neq y_i$, then $b_i = a_i$; otherwise set $b_i = \tilde{a}_i$;
4. Give output $\{(a_i, \tilde{a}_i)\}_{i\in[n]}$ to the sender $\mathcal{S}$, and output $\{b_i\}_{i\in[n]}$ to the receiver $\mathcal{R}$.

Fig. 8. Ideal Functionality for Batched Equality-tested Oblivious Randomness Generation

According to the description of the functionality $\mathcal{F}_{\text{BEtORG}}$ above, we constructed a concrete protocol Π_{BEtORG}, as shown in Fig. 9. The protocol is designed with $\mathcal{F}_{\text{PEqT}}$, $\mathcal{F}_{\binom{2}{1}\text{-OT}}$ as the main building blocks. For a more detailed description of the protocol, the sender and receiver first invoke $\mathcal{F}_{\text{PEqT}}$ n times, where each time two parties input an item, x_i and y_i, respectively. In the end of $\mathcal{F}_{\text{PEqT}}$, the sender and receiver each obtain a bit value, denoted as e_i and $\tilde{e}_i$ (i.e, if $x_i = y_i$, then $e_i \oplus \tilde{e}_i = 0$; otherwise, $e_i \oplus \tilde{e}_i = 1$). Next, based on the vector e of length n bits, the sender randomly generates n pairs of strings $(r_{e_i}, r_{e_i \oplus 1})$, each with a bit length of ℓ, while ensuring that the two strings in each generated pair are distinct. After this, the sender and receiver call $\mathcal{F}_{\binom{2}{1}\text{-OT}}$, where the sender inputs the constructed random string pairs $\{(r_{e_i}, r_{e_i \oplus 1})\}_{i\in[n]}$, and the receiver inputs the vector $\tilde{e} = \tilde{e} \oplus \{1\}^n$. After the completion of $\mathcal{F}_{\binom{2}{1}\text{-OT}}$, the receiver obtains the set $\{r_{\tilde{e}_i}\}_{i\in[n]}$, while the sender obtains nothing. Upon completion of the protocol Π_{BEtORG}, the sender holds $\{(a_i, \tilde{a}_i)\}_{i\in[n]} = \{(x^i_{b_i}, x^i_{b_i \oplus 1})\}_{i\in[n]}$, while the receiver holds $\{b_i\}_{i\in[n]} = \{r_{\tilde{e}_i}\}_{i\in[n]}$. We show the security of Π_{BEtORG} in Theorem 4.1 as below.

Protocol Π_{BEtORG}

Inputs:

- Sender $\mathcal{S}$: set $\{x_1, ..., x_n\} \in (\{0,1\}^\ell)^n$;
- Receiver $\mathcal{R}$: set $\{y_1, ..., y_n\} \in (\{0,1\}^\ell)^n$.

Protocol:

1. For $i \in [n]$: $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\text{PEqT}}$, where $\mathcal{S}$ acts as sender with input x_i and $\mathcal{R}$ acts as receiver with input y_i. $\mathcal{S}$ obtains the output e_i and $\mathcal{R}$ obtains the output $\tilde{e}_i$;
2. $\mathcal{S}$ randomly selects n pairs of strings $\{(r_{e_i}, r_{e_i \oplus 1})\}_{i\in[n]} \xleftarrow{\$} (\{0,1\}^\ell)^{2n}$, ensuring that the two strings in each pair are distinct;
3. $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathcal{F}_{\binom{2}{1}\text{-OTe}}$, where $\mathcal{S}$ acts as sender with input $\{(r_{e_i}, r_{e_i \oplus 1})\}_{i\in[n]}$ and $\mathcal{R}$ acts as receiver with input $\tilde{e} = \tilde{e} \oplus \{1\}^n$. $\mathcal{R}$ obtains the output $\{r_{\tilde{e}_i}\}_{i\in[n]}$, where if $\tilde{e}_i = e_i$, $r_{\tilde{e}_i} = r_{e_i}$, otherwise, $r_{\tilde{e}_i} = r_{e_i \oplus 1}$;
4. $\mathcal{S}$ obtains $\{(a_i, \tilde{a}_i)\}_{i\in[n]} = \{(r_{e_i}, r_{e_i \oplus 1})\}_{i\in[n]}$, $\mathcal{R}$ obtains $\{b_i\}_{i\in[n]} = \{r_{\tilde{e}_i}\}_{i\in[n]}$.

Fig. 9. Batched Equality-tested Oblivious Randomness Generation Protocol

Theorem 4.1. *The protocol in Fig. 9 securely computes $\mathcal{F}_{BEtORG}$ against semi-honest adversaries in the ($\mathcal{F}_{PEqT}$ and $\mathcal{F}_{\binom{2}{1}\text{-}OT}$)-hybrid model.*

Proof. We will demonstrate that for any adversary $\mathcal{A}$, we can construct a simulator Sim that simulates the view of the corrupted sender $\mathcal{S}$ and the corrupted receiver $\mathcal{R}$, such that the produced transcript in the ideal-world is distinguishable from that in the real-world in any PPT environment.

Corrupted $\mathcal{S}$: The simulator Sim simulates a real execution in which $\mathcal{S}$ is corrupted. Since $\mathcal{A}$ is semi-honest, Sim can directly obtain the input set $\{x_1, ..., x_n\}$ of $\mathcal{S}$ and externally send it to $\mathcal{F}_{\text{BEtORG}}$. Upon receiving x_i from $\mathcal{A}$, Sim randomly selects $e_i \xleftarrow{\$} \{0,1\}$, and simulates the execution of Π_{PEqT}. Upon receiving the input $\{(r_{e_i}, r_{\tilde{e}_i \oplus 1})\}_{i\in[n]}$ for $\Pi_{\binom{2}{1}\text{-OT}}$ from $\mathcal{A}$, Sim simulates the execution of $\Pi_{\binom{2}{1}\text{-OT}}$.

Next, we demonstrate that the outputs generated by Sim are indistinguishable from the real view of $\mathcal{S}$ through the use of the following hybrids:

Hybrid **0** $\mathcal{S}$'s view in the real protocol.

Hybrid **1** This hybrid is identical to Hybrid 0, except that the output of Π_{PEqT} is replaced by e_i, which is chosen by Sim, and Sim invokes the $\mathcal{F}_{\text{PEqT}}$ simulator to generate the simulated view for $\mathcal{S}$. The computational indistinguishability between the view in simulation and that in the real protocol is ensured by the security of the protocol Π_{PEqT}.

Hybrid **2** This hybrid is identical to Hybrid 1, except that Sim runs the simulator $\mathcal{F}_{\binom{2}{1}\text{-OTe}}$ to generate the simulated view for $\mathcal{S}$. The computational indistinguishability between the view in simulation and that in the real protocol is ensured by the security of the protocol $\Pi_{\binom{2}{1}\text{-OTe}}$. The bybrid is the view output by Sim.

Corrupted $\mathcal{R}$: The simulator Sim simulates a real execution in which $\mathcal{R}$ is corrupted. Since $\mathcal{A}$ is semi-honest, Sim can directly obtain the input set $\{y_1, ..., y_n\}$ of $\mathcal{R}$ and externally send it to $\mathcal{F}_{\text{BEtORG}}$. Upon receiving y_i from $\mathcal{A}$, Sim randomly selects $\tilde{e}_i \overset{\$}{\leftarrow} \{0,1\}$, and simulates the execution of Π_{PEqT}. Upon receiving the input $\tilde{e}$ for $\Pi_{\binom{2}{1}\text{-OTe}}$ from $\mathcal{A}$, and obtaining the output $\{r_{\tilde{e}_1}, ..., r_{\tilde{e}_n}\}$ from $\mathcal{F}_{\text{BEtORG}}$, Sim simulates the execution of $\Pi_{\binom{2}{1}\text{-OTe}}$ with $\{r_{\tilde{e}_1}, ..., r_{\tilde{e}_n}\}$ as output.

Following, we demonstrate that the outputs generated by Sim are indistinguishable from the real view of $\mathcal{R}$ through the use of the following hybrids:

Hybrid **0** $\mathcal{R}$'s view in the real protocol.

Hybrid **1** This hybrid is identical to Hybrid 0, except that the output of Π_{PEqT} is replaced by $\tilde{e}_i$, which is chosen by Sim, and Sim invokes the $\mathcal{F}_{\text{PEqT}}$ simulator to generate the simulated view for $\mathcal{R}$. The computational indistinguishability between the view in simulation and that in the real protocol is ensured by the security of the protocol Π_{PEqT}.

Hybrid **2** This hybrid is identical to Hybrid 1, except that the output of $\Pi_{\binom{2}{1}\text{-OTe}}$ is replaced by $\{r_{\tilde{e}_1}, ..., r_{\tilde{e}_n}\}$ output by $\mathcal{F}_{\text{BEtORG}}$, and Sim invokes the $\mathcal{F}_{\binom{2}{1}\text{-OTe}}$ simulator to generate the simulated view for $\mathcal{R}$. Regardless of whether $r_{\tilde{e}_i}$ is generated by $\Pi_{\binom{2}{1}\text{-OTe}}$ or $\mathcal{F}_{\text{BEtORG}}$, it holds that if $x_i \neq y_i$, then $r_{\tilde{e}_i} = r_{e_i}$; otherwise $r_{\tilde{e}_i} = r_{e_i \oplus 1}$. The computational indistinguishability between the view in simulation and that in the real protocol is ensured by the security of the protocol $\Pi_{\binom{2}{1}\text{-OTe}}$.

□

4.2 New MPSU Protocol $\Pi_{\text{MPSU}}^{\text{one-leader}}$

In this section, we provide detailed information about our constructed MPSU protocol ($\Pi_{\text{MPSU}}^{\text{one-leader}}$), which is used to realize the one-leader scenario of the $\mathcal{F}_{\text{MPSU}}$ described in Fig. 5. The complete $\Pi_{\text{MPSU}}^{\text{one-leader}}$ is shown in Fig. 10, with the core part $\Pi_{\text{MPSU}}^{\text{Core}}$, shown in Fig. 11.

In the $\Pi_{\text{MPSU}}^{\text{one-leader}}$, each of the t participants $\mathcal{P}_i$ ($i \in [t]$) holds a set $X_i = \{x_1^i, ..., x_n^i\}$. During the preparation phase, the t participants jointly invoke the AHR-TPKE scheme, allowing each participant to obtain a joint public key pk and their respective private key sk_i. Next, each participant $\mathcal{P}_i$ constructs a simple hashing table XS_i and a Cuckoo hashing table XC_i using hashing functions $h_1, ..., h_\beta$, where the simple hashing tables contain b bins with a maximum capacity of m items per bin, and cuckoo hashing tables contain b bins with each bin can hold at most one item. Subsequently, an encrypted cuckoo hashing table EXC_i is constructed for each $\mathcal{P}_i$ such that each item in the j-th bin of EXC_i is represented as $\mathsf{EXC}_i[j] = \mathsf{Enc}(\mathsf{pk}, H(\mathsf{XC}_i[j])||\mathsf{XC}_i[j])$.

After completing the above process, the first step involves all participants $\mathcal{P}_1, ..., \mathcal{P}_t$ jointly executing $\Pi_{\text{MPSU}}^{\text{Core}}$, where only $\mathcal{P}_t$ obtains the valid output. The detailed process is as follows. We use $\mathcal{F}_{\text{OPPRF}}$ to anonymize the items, followed by $\mathcal{F}_{\text{PS}}$ to shuffle the anonymized items. Next, we utilize $\mathcal{F}_{\text{BEtORG}}$ to securely determine whether the shuffled and anonymized items are equal. Afterward, the results are encrypted using the AHR-TPKE scheme, then packaged and transmitted to the next participant. The use of AHR-TPKE ensures protection against privacy leakage due to possible collusion between participants. After conducting $\mathcal{F}_{\text{BEtORG}}$ comparisons t times and sequentially encrypting and packaging items for transmission, the leader P_t eventually

Protocol $\Pi_{\text{MPSU}}^{\text{one-leader}}$

Parameters:

- Hashing functions $h_1, h_2, ..., h_\beta$: $\{0,1\}^{\ell_1} \rightarrow [b]$;
- A simple hashing table based on $h_1, h_2, ..., h_\beta$, with $b = \epsilon \cdot n$ bins and bin size $m = O(\log(\beta n))$;
- A cuckoo hashing table based on $h_1, h_2, ..., h_\beta$, with $b = \epsilon \cdot n$ bins and no stash;
- A collision-against hash function $H(x) : \{0,1\}^* \rightarrow \{0,1\}^\ell$;
- A AHR-TPKE schem $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Combine})$.

Inputs:

- t parties $\mathcal{P}_i$ $(i \in [t])$: $X_i = \{x_i^1, ..., x_i^n\} \in (\{0,1\}^*)^n$.

Preparation:

- Each $\mathcal{P}_i$ $(i \in [t])$ jointly run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(\mathsf{pp}, t, t)$ to obtain their respective keypair $(\mathsf{pk}, \mathsf{sk}_i)$;
- Each $\mathcal{P}_i$ $(i \in [t])$ respectively inserts the set X_i into the simple hashing table XS_i and the cuckoo hashing table XC_i, and fills the empty bins with the dummy d in XC_i. Let denote $\mathsf{XS}_i[j]$ and $\mathsf{XC}_i[j]$ as the items in i-th bins, where $j \in [b]$. Each $\mathcal{P}_i$ $(i \in [t])$ respectively constructs EXC_i, s.t. $\mathsf{EXC}_i[j] = \mathsf{Enc}(\mathsf{pk}, H(\mathsf{XC}_i[j])||\mathsf{XC}_i[j])$.

Protocol:

1. $\mathcal{P}_1, ..., \mathcal{P}_t$ jointly execute the $\Pi_{\text{MPSU}}^{\text{Core}}$, and then $\mathcal{P}_t$ obtains the ciphertexts $\{c_1^t, ..., c_{(t-1)b}^t\}$. $\mathcal{P}_t$ sends $\{c_1^t, ..., c_{(t-1)b}^t\}$ to $\mathcal{P}_1$;
2. For $k \in [t-1]$: $\mathcal{P}_k$ decrypts and shuffles $\{c_1^{k-1}, ..., c_{(t-1)b}^{k-1}\}$ obtaining $\{c_1^k, ..., c_{(t-1)b}^k\}$, where $c^0 = c^t$; $\mathcal{P}_k$ sends $\{c_1^k, ..., c_{(t-1)b}^k\}$ to $\mathcal{P}_{k+1}$;
3. $\mathcal{P}_t$ constructs a empty set Z_t and decrypts $\{c_1^{t-1}, ..., c_{(t-1)b}^{t-1}\}$ obtaining $\{m_1, ..., m_{(t-1)b}\}$. For $k \in [(t-1)b]$: $\mathcal{P}_t$ checks whether $m_k = H(s_k)||s_k$ and $s_k \neq d$, if yes, setting $Z_t = Z_t \cup \{s_k\}$; Finally, $\mathcal{P}_t$ obtains $Z_t \cup X_t$.

Fig. 10. $\Pi_{\text{MPSU}}^{\text{one-leader}}$ Protocol Achieving One-leader $\mathcal{F}_{\text{MPSU}}$

receives the encrypted set, which consists of encrypted $\{\mathsf{EXC}_1 \cup ... \cup \mathsf{EXC}_t\} \setminus \mathsf{EXC}_t$ and random values, where each EXC_j $(j \in [t])$ is shuffled by $\mathcal{F}_{\text{PS}}$.

In the second step, a joint decryption and shuffle process is executed sequentially in the order $\mathcal{P}_1 \rightarrow \mathcal{P}_2 \rightarrow ... \rightarrow \mathcal{P}_t$ to recover the information. Since $\mathcal{P}_t$ performs the final decryption, the recovered information is ultimately held by $\mathcal{P}_t$. Then $\mathcal{P}_t$ performs the final verification step to filter out random values, retaining only the items of $\{X_1 \cup ... \cup X_t\} \setminus X_t$ and storing them in the set Z_t, as described in Step 3 of the $\Pi_{\text{MPSU}}^{\text{one-leader}}$ protocol. Finally, $\mathcal{P}_t$ computes $Z_t \cup X_t$ to obtain the final shuffled union of items.

Correctness The first potential point of collusion occurs during the execution of $\mathcal{F}_{\text{OPPRF}}$ in Step 1 of $\Pi_{\text{MPSU}}^{\text{CORE}}$, i.e., if $\mathsf{XC}_j[k] \notin \mathsf{XS}_q[k]$ (where $j \in [t]$, $q \in \{j+1, ..., t\}$, an $k \in [b]$) but $w_k^{q,j} = F(\mathcal{K}_k^{q,j}, \mathsf{hint}_k^{q,j}, \mathsf{XC}_j[k])$. Therefore, to reduce the probability of such a collision to a negligible

$\Pi_{\text{MPSU}}^{\text{Core}}$

1. Each $\mathcal{P}_j$ ($j \in [t]$) randomly selects j-1 string group $\{r_k^{j,1}\}, ..., \{r_k^{j,j-1}\} \xleftarrow{\$} (\{0,1\}^{\ell_2})^b$, where $k \in [b]$; Then each $\mathcal{P}_j$ ($j \in [t]$) severally invokes $\mathcal{F}_{\text{OPPRF}}$ with all parties in $\mathcal{I} = \{\mathcal{P}_{j+1}, ..., \mathcal{P}_t\}$ in parallel:
   a) $\mathcal{P}_q$ acts as sender with input $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), ..., (\mathsf{XS}_q[k][m], r_k^{q,j})\}$, and $\mathcal{P}_j$ acts as receiver with input $\{\mathsf{XC}_j[k]\}$, where $k \in [b]$ and $q \in \{j+1, ..., t\}$;
   b) $\mathcal{P}_j$ obtains $(\mathsf{hint}^{q,j}, \{w_1^{q,j}, ..., w_b^{q,j}\})$ and $\mathcal{P}_q$ obtains $(\mathsf{hint}^{q,j}, \{\mathcal{K}_1^{q,j}, ..., \mathcal{K}_b^{q,j}\})$, where $w_k^{q,j} = F(\mathcal{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k])$. Specifically, if $\mathsf{XC}_j[k] \in \mathsf{XS}_q[k]$, then $F(\mathcal{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k]) = r_k^{q,j}$; otherwise, $F(\mathcal{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k]) \neq r_k^{q,j}$;
2. Each $\mathcal{P}_j$ ($j \in [t]$) randomly selects a permutation function $\pi_j : [b] \to [b]$; Then each $\mathcal{P}_j$ ($j \in [t]$) severally invokes $\mathcal{F}_{\text{PS}}$ with all parties in $\mathcal{I} = \{\mathcal{P}_{j+1}, ..., \mathcal{P}_t\}$ in parallel:
   a) $\mathcal{P}_j$ acts as sender with input π_j, and $\mathcal{P}_q$ acts as receiver with input $\{r_1^{q,j}, ..., r_b^{q,j}\}$, where $q \in \{j+1, ..., t\}$;
   b) $\mathcal{P}_j$ obtains $\{s_1^{q,j}, ..., s_b^{q,j}\}$ and $\mathcal{P}_q$ obtains $\{\tilde{s}_1^{q,j}, ..., \tilde{s}_b^{q,j}\}$, where $s_k^{q,j} \oplus \tilde{s}_k^{q,j} = r_{\pi_j(k)}^{q,j}$ and $k \in [b]$;
3. For $j \in [t-1]$, t parties sequentially execute the following processes:
   a) $\mathcal{P}_j$ invokes $\mathcal{F}_{\text{BEtORG}}$ with $\mathcal{P}_{j+1}$:
      - $\mathcal{P}_j$ acts as sender with input $\{\{sw_k^{j+1,q}\}_{k\in[b]}\}_{q\in[j]}$, where $sw_k^{j+1,q} = s_k^{j+1,q} \oplus w_{\pi_q(k)}^{j+1,q} \oplus_{p=q+1}^{j\geq p+1} (a_k^{p,q} \oplus b_k^{p,q})$, and $\mathcal{P}_{j+1}$ acts as receiver with input $\{\{\tilde{s}_k^{j+1,q}\}_{k\in[b]}\}_{q\in[j]}$;
      - $\mathcal{P}_j$ obtains $\{\{(a_k^{j+1,q}, \tilde{a}_k^{j+1,q})\}_{k\in[b]}\}_{q\in[j]}$ and $\mathcal{P}_{j+1}$ obtains $\{\{b_k^{j+1,q}\}_{k\in[b]}\}_{q\in[j]}$, where if $sw_k^{j+1,q} = \tilde{s}_k^{j+1,q}$, then $\tilde{x}_k^{j+1,q} = b_k^{j+1,q}$; otherwise $b_k^{j+1,q} = x_k^{j+1,q}$;

   b) $\mathcal{P}_j$ sends $\{\{A_k^{j,q}\}_{k\in[b]}\}_{q\in[j]}$ and $\{\{\{s\tilde{w}_k^{l,q}\}_{k\in[b]}\}_{q\in[j]}\}_{l\in\{j+2,...,t\}}$ to $\mathcal{P}_{j+1}$, where

$$A_k^{q,q} = \{\mathsf{EXC}_q[\pi_q(k)] \oplus a_k^{q+1,q}\}$$

$$A_k^{j,q} = \{((A_k^{j-1,q} \oplus b_k^{j,q}) +_H \mathsf{Enc}(\mathsf{pk}, 0)) \oplus a_k^{j+1,q}\}$$

$$s\tilde{w}_k^{l,q} = s_k^{l,q} \oplus w_{\pi_q(k)}^{l,q} \oplus_{p=q+1}^{j\geq q+1} (a_k^{p,q} \oplus b_k^{p,q}) \oplus a_k^{j+1,p}.$$

   Then $\mathcal{P}_{j+1}$ computes each $s\tilde{w}_k^{l,q} \oplus b_k^{j+1,q}$; Specifically, when $j = t-1$, $\mathcal{P}_{t-1}$ only sends $\{\{A_k^{t-1,q}\}_{k\in[b]}\}_{q\in[t-1]}$ to $\mathcal{P}_t$. Then $\mathcal{P}_t$ only computes $\{(A_k^{t-1,q} \oplus b_k^{t,t-1}) +_H \mathsf{Enc}(\mathsf{pk}, 0))\}$, denote as $\{c_1, ..., c_{(t-1)b}\}$;

Fig. 11. The core protocol of $\Pi_{\text{MPSU}}^{\text{one-leader}}$

level, specifically $2^{-\lambda}$, the output bit length ℓ_2 of PRF function F is typically set to $\lambda + \log(\epsilon \cdot n)$. Additionally, in 3) of $\Pi_{\text{MPSU}}^{\text{CORE}}$, items in $\mathcal{P}_j$ are compared sequentially with those in $\mathcal{P}_{j+1}$ to check for duplicates, where $j \in [t]$. If duplicates are found, the duplicate items in $\mathcal{P}_j$ are randomized, while retaining the items in $\mathcal{P}_{j+1}$, as described in Appendix B. This process continues until $\mathcal{P}_t$ obtains the $\{\{X_1 \cup ... \cup X_t\} \setminus X_t +$ random values$\}$. Eventually after joint decryption and shuffling, $\mathcal{P}_t$ performs a verification to remove the random values and retain $\{X_1 \cup ... \cup X_t\} \setminus X_t$, thereby obtaining the final shuffled union set $\{\{X_1 \cup ... \cup X_t\} \setminus X_t\} \cup X_t = \bigcup_{j=1}^{t} X_j$. In the above process, potential errors may arise, such as decrypted item being random value that still pass $\mathcal{P}_t$'s union check (i.e., $z_k = H(s_k)||s_k$ when s_k is a random value), or item belonging to the union failing the check. However, the properties of the AHR-TPKE and the collision-resistant hash function ensures that the probability of such errors is negligible.

Security The security of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ is proved as follows.

Theorem 4.2. *The protocol in Fig. 10 securely computes $\mathcal{F}_{MPSU}$ of one-leader scenario against any number of corrupt, colluding, semi-honest parties in the (AHR-TPKE, $\mathcal{F}_{OPPRF}$, $\mathcal{F}_{PS}$, and $\mathcal{F}_{BEtORG}$)-hybrid model.*

Proof. All interactions between participants in $\Pi_{\text{MPSU}}^{\text{one-leader}}$ occur within online phase, that is $\Pi_{\text{MPSU}}^{\text{Core}}$ and joint decryption + shuffling operation. Assuming $\mathcal{P}_i$ (where $i \in [t]$) is honest and $I = \{(\mathcal{P}_1, ..., \mathcal{P}_t) \backslash \mathcal{P}_i\}$ is a colluding coalition, to satisfy the correctness of Theorem 4.2, we need to prove that $\mathcal{P}_i$ can resist coalition attacks when it respectively assumes the roles $\mathcal{P}_1, ..., \mathcal{P}_t$. That is, for any adversary $\mathcal{A}$, we can construct a simulator Sim that simulates the view of the corrupted coalition I, such that the produced transcript in the idel-world is distinguishable from that in the real-world in any PPT environment. Since $\mathcal{P}_1, ..., \mathcal{P}_{t-1}$ perform the same role in the protocol, we will prove in two different cases: when $\mathcal{P}_i$ acts as the leader $\mathcal{P}_t$, when $\mathcal{P}_i$ acts as the other participant roles.

(1) $\mathcal{P}_i$ acts as the leader $\mathcal{P}_t$ and I acts as the $\{\mathcal{P}_1, ..., \mathcal{P}_{t-1}\}$. The simulator Sim simulates a real execution. Since $\mathcal{A}$ is semi-honest, Sim can obtain the input $I_X = \{X_1, ..., X_{t-1}\}$ of I directly, and externally send I_X to $\mathcal{F}_{\text{MPSU}}$. When receiving $\{\mathsf{XC}_j[1], ..., \mathsf{XC}_j[b]\}$ ($j \in [t-1]$) from $\mathcal{A}$, Sim randomly selects $(\mathsf{hint}^{t,j}, \{w_k^{t,j}\})$ (where $k \in [b]$), and simulates the execution of Π_{OPPRF}, where $\mathcal{P}_j$ acts as the receiver and $\mathcal{P}_t$ acts as the sender. Once receiving the permutation function π_j from $\mathcal{A}$, Sim checks if it is a permutation of b items. If yes, Sim randomly selects $\{s_1^{t,j}, ..., s_b^{t,j}\}$, where $s_k^{t,j} \in \{0,1\}^{\ell_2}$, and simulates the execution of Π_{PS}, where $\mathcal{P}_j$ acts as the sender and $\mathcal{P}_t$ acts as the receiver. After obtaining the $\{\{sw_k^{t,q}\}_{k\in[b]}\}_{q\in[t-1]}$ from $\mathcal{A}$, Sim randomly selects $\{\{(a_k^{t,q}, \tilde{a}_k^{t,q})\}_{k\in[b]}\}_{q\in[t-1]}$ and simulates the execution Π_{BEtORG}, where $\mathcal{P}_{t-1}$ acts as the sender and $\mathcal{P}_t$ acts as the receiver. Once obtaining $\{\{A_k^{t-1,q}\}_{k\in[b]}\}_{q\in[t-1]}$ from $\mathcal{A}$, and also obtaining $\{c_1^t, ..., c_{(t-1)b}^t\}$ from $\mathcal{A}$, Sim simulates the set $\{c_1^{t-1}, ..., c_{(t-1)b}^{t-1}\}$ that joint decryption after $t-1$ rounds.

Next, we demonstrate that the outputs generated by Sim are indistinguishable from the real view of I through the use of the following hybrids:

Hybrid **0** I's view in the real protocol.

Hybrid **1** This hybrid is identical to Hybrid 0, except that the output of Π_{OPPRF} is replaced by $(\mathsf{hint}^{t,j}, \{w_1^{t,j}, ..., w_b^{t,j}\})$, chosen by Sim, when $\mathcal{P}_j$ (where $j \in [t-1]$) acts as the receiver and jointly invokes with $\mathcal{P}_t$. Afterwords, Sim invokes the $\mathcal{F}_{\text{OPPRF}}$ simulator to simulate the $t-1$ calls and generate the simulated view for I. Since each invocation is independent of the others, i.e., the invocation between $\mathcal{P}_1, ..., \mathcal{P}_{t-1}$ and $\mathcal{P}_t$ are independent, the computational indistinguishability between the simulated view and the real protocol view is ensured by the security of Π_{OPPRF}.

Hybrid **2** This hybrid is identical to Hybrid 1, except that the output of Π_{PS} is replaced by $\{s_1^{t,j}, ..., s_b^{t,j}\}$, chosen by Sim, when $\mathcal{P}_j$ (where $j \in [t-1]$) acts as the sender and jointly calls with $\mathcal{P}_t$. Afterwords, Sim invokes the $\mathcal{F}_{\text{PS}}$ simulator to simulate the $t-1$ calls and generate the simulated view for I. Since each invocation is independent of the others, i.e., the invocation between $\mathcal{P}_1, ..., \mathcal{P}_{t-1}$ and $\mathcal{P}_t$ are independent, the computational indistinguishability between the simulated view and the real protocol view is ensured by the security of Π_{PS}.

Hybrid **3** This hybrid is identical to Hybrid 2, except that the output of Π_{BEtORG} is replaced by $\{(a_1^{t,q}, \tilde{a}_1^{t,q}), ..., (a_b^{t,q}, \tilde{a}_b^{t,q})\}$, chosen by Sim, when $\mathcal{P}_{t-1}$ acts as the sender and jointly calls with $\mathcal{P}_t$. Afterwords, Sim invokes the $\mathcal{F}_{\text{BEtORG}}$ simulator to simulate the invocation and generate the simulated view for I. The computational indistinguishability between the simulated view and the real protocol view is ensured by the security of Π_{BEtORG}.

Hybrid **4** This hybrid is identical to Hybrid 3, except that the joint decryption result of the $t-1$ participants in I is replaced by $\{c_1^{t-1}, ..., c_{(t-1)b}^{t-1}\}$, chosen by Sim. Afterwords, Sim invokes the joint decryption simulator to simulate the invocation and generate the simulated view for I. The computational indistinguishability between the simulated view and the real protocol view is ensured by the security of AHR-TPKE and randomly shuffling. This hybrid is the view output by the simulator Sim.

(2) $\mathcal{P}_i$ acts as the $\mathcal{P}_j$ (where $j \in [t-1]$) and I acts as the $\{\mathcal{P}_1, ..., \mathcal{P}_t\} \setminus \mathcal{P}_j$. The simulator Sim simulates a real execution. Since $\mathcal{A}$ is semi-honest, Sim can obtain the input $I_X = \{X_1, ..., X_t\} \setminus X_j$ of I directly, and externally send I_X to $\mathcal{F}_{\text{MPSU}}$. When obtaining $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), ..., (\mathsf{XS}_q[k][m], r_k^{q,j})\}$ (where $q \in \{j+1, ..., t\}$ and $k \in [b]$) and $\{\mathsf{XC}_p[1], ..., \mathsf{XC}_p[b]\}$ (where $p \in [j-1]$), Sim randomly selects $(\mathsf{hint}^{q,j}, \{\mathcal{K}_1^{q,j}, ..., \mathcal{K}_b^{q,j}\}$ and $(\mathsf{hint}^{j,p}, \{w_1^{j,p}, ..., w_b^{j,p}\})$, respectively. Then, Sim simulates the execution of Π_{OPPRF}. Specifically, for $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), ..., (\mathsf{XS}_q[k][m], r_k^{q,j})\}$, Sim simulates the scenario where $\mathcal{P}_q$ acts as the sender and $\mathcal{P}_j$ acts as the receiver. For $\{\mathsf{XC}_p[1], ..., \mathsf{XC}_p[b]\}$, Sim simulates the scenario where $\mathcal{P}_p$ acts as the receiver and $\mathcal{P}_j$ acts as the sender. Once obtaining π_q (where $q \in \{j+1, ..., t\}$) and $\{r_1^{j,p}, ..., r_b^{j,p}\}$ (where $p \in [j-1]$), Sim randomly selects $\{s_1^{q,j}, ..., s_b^{q,j}\}$ and $\{\tilde{s}_1^{j,p}, ..., \tilde{s}_b^{j,p}\}$, respectively. Then, Sim simulates the execution of Π_{PS}. Specifically, for π_q, Sim simulates the scenario where $\mathcal{P}_q$ acts as the sender and $\mathcal{P}_j$ acts as the receiver. For $\{r_1^{j,p}, ..., r_b^{j,p}\}$, Sim simulates the scenario where $\mathcal{P}_p$ acts as the receiver and $\mathcal{P}_j$ acts as the sender. Upon obtaining $\{\{sw_k^{j,q}\}_{k\in[b]}\}_{q\in[j-1]}$ from $\mathcal{A}$, Sim randomly selects $\{(a_1^{j,q}, \tilde{a}_1^{j,q}), ..., (a_b^{j,q}, \tilde{a}_b^{j,q})\}$, and simulates the Π_{BEtORG}, where $\mathcal{P}_{j-1}$ acts as the sender and $\mathcal{P}_j$ acts as the receiver. Upon obtaining $\{\tilde{s}_1^{j+1,j}, ..., \tilde{s}_b^{j+1,j}\}$ from $\mathcal{A}$, Sim randomly selects $\{a_1^{j+1,j}, ..., a_b^{j+1,j}\}$, and simulates the Π_{BEtORG}, where $\mathcal{P}_j$ acts as the sender and $\mathcal{P}_{j+1}$ acts as the receiver. Once obtaining $\{\{A_k^{j,1}\}, ..., \{A_k^{j,j}\}\}$ from $\mathcal{A}$, and also obtaining $\{c_1^j, ..., c_{(t-1)b}^j\}$ from $\mathcal{A}$, Sim simulates the set $\{m_1, ..., m_{(t-1)b}\}$ representing decrypted message after joint decryption of t rounds. After obtaining the shuffled $Z_t = \bigcup_{i=1}^t X_i$ from $\mathcal{F}_{\text{MPSU}}$, Sim computes $\tilde{X}_j = Z_t \backslash I_X$ and sets $m_i = H(\tilde{X}_j[i]) || \tilde{X}_j[i]$ for $i \in [|\tilde{X}_j|]$. Then, Sim randomly selects $m_i \xleftarrow{\$} \{0,1\}^{\tilde{\ell}_i}$ for $i \in \{|\tilde{X}_j| + 1, ..., (t-1)b\}$, where $\tilde{\ell}_i$ represents the bit length of the sequential items in the set I_X plus ℓ. Finally, Sim sends the randomly shuffled $\{m_1, ..., m_{(t-1)b}\}$ to $\mathcal{A}$.

Next, we demonstrate that the outputs generated by Sim are indistinguishability from the view of I through the use of the following hybrids:

Hybrid **0** I's view in the real protocol.

Hybrid **1** This hybrid is identical to Hybrid 0, except that when $\mathcal{P}_{j+1}, ..., \mathcal{P}_t$ act as the sender and jointly invoke Π_{OPPRF} with $\mathcal{P}_j$, the output is replaced by $(\mathsf{hint}^{q,j}, \{\mathcal{K}_1^{q,j}, \ldots, \mathcal{K}_b^{q,j}\})$ (where $q \in \{j+1, ..., t\}$), chosen by Sim. Similarly, when $\mathcal{P}_1, \ldots, \mathcal{P}_{j-1}$ act as the receiver and jointly invoke Π_{OPPRF} with $\mathcal{P}_j$, the output is replaced by $(\mathsf{hint}^{t,j}, \{w_1^{p,j}, \ldots, w_b^{p,j}\})$ (where $p \in [j-1]$), also chosen by Sim. Then, Sim invokes the $\mathcal{F}_{\text{OPPRF}}$ simulator to simulate the $t-$

1 invocations and generate a simulated view for I. Since each invocation is independent of the others, the security of Π_{OPPRF} ensures the computational indistinguishability between the simulated view and the real protocol view.

Hybrid **2** This hybrid is identical to Hybrid 1, except that when $\mathcal{P}_{j+1}, ..., \mathcal{P}_t$ act as the receiver and jointly invoke Π_{PS} with $\mathcal{P}_j$, the output is replaced by $\{\tilde{s}_1^{q,j}, ..., \tilde{s}_b^{q,j}\}$ (where $q \in \{j+1, ..., t\}$), chosen by Sim. Similarly, when $\mathcal{P}_1, \ldots, \mathcal{P}_{j-1}$ act as the sender and jointly invoke Π_{PS} with $\mathcal{P}_j$, the output is replaced by $\{s_1^{j,p}, ..., s_b^{j,p}\}$ (where $p \in [j-1]$), also chosen by Sim. Then, Sim invokes the $\mathcal{F}_{\text{PS}}$ simulator to simulate the $t-1$ invocations and generate a simulated view for I. Since each invocation is independent of the others, the security of Π_{PS} and the random permutation functions ensures the computational indistinguishability between the simulated view and the real protocol view.

Hybrid **3** This hybrid is identical to Hybrid 2, except that when $\mathcal{P}_{j-1}$ acts as the sender and jointly invokes Π_{BEtORG} with $\mathcal{P}_j$, the output is replaced by $\{(a_1^{j,q}, \tilde{a}_1^{j,q}), \ldots, (a_b^{j,q}, \tilde{a}_b^{j,q})\}$ (where $q \in [j-1]$), chosen by Sim. Afterwards, Sim invokes the $\mathcal{F}_{\text{BEtORG}}$ simulator to simulate the invocation and generate a simulated view for I. Additionally, when $\mathcal{P}_{j+1}$ acts as the receiver and jointly invokes Π_{BEtORG} with $\mathcal{P}_j$, the output is replaced by $\{b_1^{j+1,q}, \ldots, b_b^{j+1,q}\}$ (where $q \in [j]$), chosen by Sim. Subsequently, Sim calls the $\mathcal{F}_{\text{BEtORG}}$ simulator again to simulate the invocation and generate a simulated view for I. The security of Π_{BEtORG} ensures that the simulated views in both cases are computationally indistinguishable from the views in the real protocol.

Hybrid **4** This hybrid is identical to Hybrid 3, except that the $\{c_1^{t-1}, \cdots, c_{(t-1)b}^{t-1}\}$ obtained by $\mathcal{P}_t$ is randomized due to $\{c_1^j, \cdots, c_{(t-1)b}^j\}$ is randomly chosen by Sim at $\mathcal{P}_j$. Subsequently, Sim invokes the joint decryption simulator to simulate the invocation and generate a simulated view for I. Consequently, the final decrypted information $\{m_1, ..., m_{(t-1)b}\}$ is also changed. The security of AHR-TPKE and random shuffling ensure that the simulated view is computationally indistinguishable from the real protocol view. This hybrid is the view output by the simulator Sim.

□

4.3 New MPSU Protocol $\Pi_{\text{MPSU}}^{\text{leaderless}}$

The $\Pi_{\text{MPSU}}^{\text{leaderless}}$ protocol is an extension of the $\Pi_{\text{MPSU}}^{\text{one-leader}}$ protocol, with the same offline operations. The distinction lies in the extended online phase, as detailed in Fig. 12. Specifically, by executing $\Pi_{\text{MPSU}}^{\text{Core}}$ t times, each participant assumes a different role within $\Pi_{\text{MPSU}}^{\text{Core}}$ during each execution. This ensures that after t executions, all t participants have played the role of internal $\mathcal{P}_t$ and obtained valid output results. Subsequently, all participants perform the linear decryption and shuffle operations in parallel, following the order $\mathcal{P}_{i+1} \to \cdots \to \mathcal{P}_{i+t-1} \to \mathcal{P}_{i+t}$, where $i \in [t]$, and the index calculation omits the $(\mod (t+1))+1$ modulus rule. It is worth noting that during the parallel execution of linear operations, since the participants in each linear stage are distinct, even in a single-threaded setting, all participants can still complete the linear computation simultaneously. Finally, each participant sequentially verifies the decrypted information, removes the random values, and obtains the final union set. Due to the shuffle operations performed during the decryption phase, while all participants obtain identical items in their union set, the order of items in each participant's set is randomized.

Protocol $\Pi^{\text{leaderless}}_{\text{MPSU}}$

Protocol:

1. All participants execute $\Pi^{\text{Core}}_{\text{MPSU}}$ for t rounds. In the i-th round (where $i \in [t]$), $\mathcal{P}_i, \mathcal{P}_{i+1}, ..., \mathcal{P}_{i+t-1}$ sequentially take on the roles of $\mathcal{P}_t, \mathcal{P}_1, \ldots, \mathcal{P}_{t-1}$ within $\Pi^{\text{Core}}_{\text{MPSU}}$, where indices omit $(\mod (t+1)) + 1$. At the end of each round of $\Pi^{\text{Core}}_{\text{MPSU}}$, $\mathcal{P}_i$ obtains the output set $\{c^i_1, ..., c^i_{(t-1)b}\}$ and sends it to $\mathcal{P}_{i+1}$.
2. Execute the following operations in parallel: For $i \in [t]$, $\mathcal{P}_j$ performs decryption and shuffle operations on $\{c^{j-1}_1, ..., c^{j-1}_{(t-1)b}\}$, generating a new set $\{c^j_1, ..., c^j_{(t-1)b}\}$, and sends it to P_{j+1}, where $j \in \{i+1, \ldots, i+t-1\}$. Note that indices omit the rule $(\mod (t+1)) + 1$ and $c^0 = c^t$.
3. Each $\mathcal{P}_i$ (where $i \in [n]$) constructs a empty set Z_i and decrypts $\{c^{i-1}_1, ..., c^{i-1}_{(t-1)b}\}$ obtaining $\{m^i_1, ..., m^i_{(t-1)b}\}$. For $k \in [(t-1)b]$: $\mathcal{P}_i$ checks whether $m^i_k = H(s^i_k)||s^i_k$ and $s^i_k \neq d$, if yes, setting $Z_i = Z_i \cup \{s^i_k\}$; Finally, $\mathcal{P}_i$ obtains $Z_i \cup X_i$.

Fig. 12. $\Pi^{\text{leaderless}}_{\text{MPSU}}$ Protocol Achieving Leaderless $\mathcal{F}_{\text{MPSU}}$

Since the offline phase of the $\Pi^{\text{leaderless}}_{\text{MPSU}}$ protocol is identical to that of the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol, and its online phase can be regarded as t independent executions of the $\Pi^{\text{one-leader}}_{\text{MPSU}}$'s online phase, proving the correctness and security of the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ is sufficient to establish the correctness and security of the $\Pi^{\text{leaderless}}_{\text{MPSU}}$. The correctness and security of the $\Pi^{\text{one-leader}}_{\text{MPSU}}$ have been detailed in Section 4.2. Therefore, we can directly derive the following theorem.

Theorem 4.3. *The protocol in Fig. 12 securely computes $\mathcal{F}_{MPSU}$ of leaderless scenario against any number of corrupt, colluding, semi-honest parties in the (AHR-TPKE, $\mathcal{F}_{OPPRF}$, $\mathcal{F}_{PS}$, and $\mathcal{F}_{BEtORG}$)-hybrid model.*

4.4 Efficiency Optimization Strategy

Recall that in our protocols, during 3) of $\Pi^{\text{Core}}_{\text{MPSU}}$, each internal participant $\mathcal{P}_j$ (where $j \in [t]$) should send relevant data to $\mathcal{P}_{j+1}$ after running the $\mathcal{F}_{\text{BEtORG}}$. To ensure secure transmission and resist collusion among semi-honest participants, we use the AHR-TPKE scheme to continuously re-randomize the data to be transmitted. This process involves two steps: first, computing $\mathsf{Enc}(\mathsf{pk}, 0)$, and second, performing homomorphic addition $+_H$. It is worth noting that the majority of the runtime in this part is consumed by $\mathsf{Enc}(\mathsf{pk}, 0)$, while the time spent on $+_H$ is significantly lower, approximately $T_{+_H} \approx 5\% \times T_{\mathsf{Enc}(\mathsf{pk},0)}$. Therefore, generating a sufficient number of $\mathsf{Enc}(\mathsf{pk}, 0)$ during the preparation phase, and only performing $+_H$ during the online phase can significantly improve the protocol's efficiency.

4.5 Cost Analysis

During the interaction phase, our $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol is executed through $\Pi^{\text{Core}}_{\text{MPSU}}$ and the joint decryption + shuffle operations. The theoretical computational and communication complexities are analyzed in detail in Table 3.

We assume there are t participants, each holding a set of size n. Recalling our $\Pi_{\text{MPSU}}^{\text{one-leader}}$, each participant first jointly calls the AHR-TPKE scheme to obtain a joint public key and an individual private key, and inserts the items into a simple hash table and a cuckoo hash table respectively using γ hash functions (with a encrypted Cuckoo hash table also constructed). The cuckoo hash table contains ϵn items, while the simple hash table contains γn items. Then, the t participants proceed with executing $\Pi_{\text{MPSU}}^{\text{Core}}$ t times, involving the execution of Π_{OPPRF}, Π_{PS}, Π_{BEtORG}, as well as encryption, decryption operations of the AHE-TPKE scheme.

For the implementation of Π_{OPPRF}, we used the batched OPPRF from [38] to hide the number of items in each bin. Specifically, the complexity for computing hints and communication is linear, i.e., $O(\gamma n)$, and the sender additionally needs $O(\epsilon n \log(\epsilon n))$ to compute the PRF values. Thus, in one round execution of $\Pi_{\text{MPSU}}^{\text{Core}}$, Π_{OPPRF} results in a computational complexity of $O(A(\gamma + \epsilon \log(\epsilon n))n)$ and a communication complexity of $O(A\gamma n)$, where $A = \sum_{i=1}^{t-1} i$.

We use the construction from [39] to implement Π_{PS}, resulting in a computational complexity of $O(\epsilon n \log(\epsilon n))$. Thus, in one round execution of $\Pi_{\text{MPSU}}^{\text{Core}}$, Π_{PS} results in $O(A\epsilon n \log(\epsilon n))$ for both computation and communication complexity.

Our Π_{BEtORG} consists of Π_{PEqT} and $\Pi_{\binom{2}{1}\text{-OTe}}$, both of which exhibit linear complexity. Thus, the computational and communication complexity of Π_{BEtORG} are $O(\epsilon n)$. Therefore, in one round execution of $\Pi_{\text{MPSU}}^{\text{Core}}$, $\Pi_{\binom{2}{1}\text{-OTe}}$ results in $O(A\epsilon n)$ for both computation and communication complexity. Additionally, in the encryption phase, there are $O(A\epsilon n)$ encryption operations, while the decryption phase involves $O(t^2\epsilon n)$ decryption operations.

	$\mathcal{F}_{\text{OPPRF}}$	$\mathcal{F}_{\text{PS}}$	$\mathcal{F}_{\text{BEtORG}}$	Encryption	Decryption
Comp.	$O(A(\gamma + \epsilon \log(\epsilon n))n)$	$O(A\epsilon n \log(\epsilon n))$	$O(A\epsilon n)$	$O(A\epsilon n)$	$O(t^2\epsilon n)$
Comm.	$O(Arn)$	$O(A\epsilon n \log(\epsilon n))$	$O(A\epsilon n)$	$O(A\epsilon n)$	$O(t^2\epsilon n)$

Table 3: The theoretical complexities of $\Pi_{\text{MPSU}}^{\text{Core}}$. Note: Comm./Comp.: Communication /Computational complexity; t: Number of parties; n: Size of sets; $A = \sum_{i=1}^{t-1} i$; γ: Number of hashing functions (in Simple hashing table and cuckoo hashing table); ϵn: Number of bins of Simple hashing table and cuckoo hashing table.

For our $\Pi_{\text{MPSU}}^{\text{leaderless}}$ protocol, it can be considered as t invocations of the $\Pi_{\text{MPSU}}^{\text{one-leader}}$ protocol. Thus, it is clear and intuitive that its computational complexity and communication complexity are t times that of the $\Pi_{\text{MPSU}}^{\text{one-leader}}$. Additionally, in $\Pi_{\text{MPSU}}^{\text{leaderless}}$, each participant sequentially assumes all roles within the t rounds of $\Pi_{\text{MPSU}}^{\text{Core}}$. Consequently, after t rounds, the computational and communication overhead for each participant equals the overhead incurred by a single round of $\Pi_{\text{MPSU}}^{\text{Core}}$. Similarly, after t parallel rounds of joint decryption and shuffle operations, the computational and communication overhead for each participant equals the overhead of a single round of joint decryption and shuffle. Therefore, in $\Pi_{\text{MPSU}}^{\text{leaderless}}$, the total computational and communication overhead for each participant is identical and equals the overhead incurred by a single round of $\Pi_{\text{MPSU}}^{\text{one-leader}}$.

5 Performance Evaluation

In this section, we experimentally evaluate the performance of our MPSU protocols and compare with the previous work.

Benchmarking Environment. We conduct experiments using CLion 2024.2.2 on a MacBook Pro running a macOS 14.0, equipped with an Apple M3 Pro chip and 36GB RAM. The WAN environment is configured with a network latency of 10ms and a bandwidth of 100Mbps.

Implementation Details. We use threshold Elgamal encryption as our AHR-TPKE scheme. Specifically, we employ a key generation algorithm to generate moduli N of 1024 and 2058 bits, approximately corresponding to computational security parameters κ of 80 and 116 bits, respectively. Additionally, the statistical security parameter is set to $\lambda = 40$. Furthermore, $\gamma = 3$ hashing functions are used to respectively insert sets X_i (where $i \in [t]$) into the simple hashing table and cuckoo hashing table, and the encrypted cuckoo hashing table, respectively. The bin size ϵn of each hashing table is set to $1.27n$. Our protocol is written in C/C++, and utilizes the following libraries for its implementation.

- **OPPRF.** We use the BTOA19 batched OPPRF structure [38] to implement the OPPRF functionality. The implementation can be found at `https://github.com/encryptogroup/OPPRF-PSI`.
- **PS functionality.** We use the PS13 Oblivious Switching Network (OSN) structure [39] to implement the PS functionality. The implementation can be found at `https://github.com/dujiajun/PSU`.
- **PEqT functionality.** We use the NDA22 Private Set Membership (PSM) [40] to implement the PEqT functionality. The implementation can be found at `https://github.com/shahakash28/2PC-Circuit-PSI`.
- **1-out-of-2 OTe functionality.** We use the library in `https://github.com/osu-crypto/libOTe` to implement the 1-out-of-2 OTe functionality.
- Additionally, we also **OpenSSL** library to construct the threshold Elgamal ecnryption and employ the **cryptoTools** library as the general framework to compute hash functions, PRNG calls, creating channels, sending 128-bit blocks and so on. The implementation can be found at `https://github.com/ladnir/cryptoTools`.

5.1 Performance of Our Proposed

In our experiments, we set the set sizes to $\{2^{12}, 2^{16}, 2^{20}\}$ and the number of participants to $\{3, 4, 6, 8\}$ to examine the runtime and communication overhead of our proposed. Table 4 consolidates all results, presenting the runtime for both the offline phase and the execution of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ during the online phase, as well as the online execution time and communication overhead for $\Pi_{\text{MPSU}}^{\text{leaderless}}$. These results are evaluated under threshold Elgamal of 1024 and 2048.

Based on the results presented in the table, for both of our protocols, when the set size and encryption modulus are fixed, the offline runtime increases approximately 1.5 times linearly with the number of participants grows, while the online runtime increases by approximately 2 times linearly. Additionally, when the number of participants and set size are fixed, an increase in the encryption modulus leads to higher runtime and communication costs due to increased encryption/decryption computation complexity and ciphertext size. These impacts are primarily observed during the preparation phase, including generating encrypted cuckoo hashing table, and in $\Pi_{\text{MPSU}}^{\text{Core}}$ during step 3.b, where set A is transmitted and re-randomized, as well as step 4,

Party Number	Set Size	Modulus (bits)	Offline Runtime (s)	Online Runtime ($\Pi_{\text{MPSU}}^{\text{one-leader}}$) (s)	Comm. Cost ($\Pi_{\text{MPSU}}^{\text{one-leader}}$) (MB)	Online Runtime ($\Pi_{\text{MPSU}}^{\text{leaderless}}$) (s)	Comm. Cost ($\Pi_{\text{MPSU}}^{\text{leaderless}}$) (MB)
3	2^{12}	1024	7.3	18.79	20	31.81	60
		2048	69.76	81.15	32	94.17	96
	2^{16}	1024	108.48	294.27	355	498.89	1065
		2048	826.91	1392.78	538	1597.41	1614
	2^{20}	1024	1943.73	6035.23	2083	10032.71	6249
		2048	13274.1	28691.48	9005	32688.96	27015
4	2^{12}	1024	9.52	37.74	41	76.52	164
		2048	77.52	171.42	64	210.18	256
	2^{16}	1024	139.07	582.77	719	1174.61	2876
		2048	1034.15	2764.91	1085	3356.75	4340
	2^{20}	1024	2495.75	12686.05	12279	24925.09	49116
		2048	16380.24	57784.64	18131	70023.67	72524
6	2^{12}	1024	12.61	93.42	107	252.12	642
		2048	92.57	405.06	164	563.76	984
	2^{16}	1024	192.33	1424.94	1836	3708.49	11016
		2048	1379.63	6937.53	2751	9221.08	16506
	2^{20}	1024	3279.41	24395.28	31306	65855.63	187836
		2048	19689.05	106843.80	45936	148304.15	275617
8	2^{12}	1024	15.2	167.65	204	589.82	1632
		2048	123.17	797.37	311	1219.54	2488
	2^{16}	1024	246.73	2649.95	3501	8906.13	28008
		2048	1864.94	12974	5208	19230.18	8896
	2^{20}	1024	4200.93	45838.73	59575	155990.17	476600
		2048	23469.34	210589.13	86885	—	—

Table 4: The complete offline and online runtime (seconds) and communication cost of $\Pi_{\text{MPSU}}^{\text{one-leader}}$, as well as the online runtime and communication overhead of $\Pi_{\text{MPSU}}^{\text{leaderless}}$ ($N = 1024$ and $N = 2048$). Each party's set contains of 64-bit elements, while the collusion-against hash function H produces outputs of 128-bit length. The decimal portion of the communication overhead is disregarded, retained only the integer values.

where joint decryption is performed. In other parts of the protocol, the size of the encryption modulus has minimal impact on performance.

Additionally, the computational efficiency of the $\Pi_{\text{MPSU}}^{\text{leaderless}}$ is determined by the runtime of t executions of $\Pi_{\text{MPSU}}^{\text{Core}}$ and t joint decryption + shuffle operations. Since the t joint decryption + shuffle operations are executed in parallel, their total time equals the time for one operation. Thus, the computational efficiency of the $\Pi_{\text{MPSU}}^{\text{leaderless}}$ is:

$$t \times \text{Runtime of } \Pi_{\text{MPSU}}^{\text{Core}} + \text{Runtime of 1 Joint Decryption + Shuffle}.$$

5.2 Comparison with Previous Work

To compare the performance of our proposed, we benchmark it against the state-of-the-art protocols operating under the semi-honest security model. Both protocols proposed in [25] and [26] ensure that, at the end of the execution, only one party obtains the union set. However, the protocol in [25] achieves the same security guarantees as ours - namely, it can withstand collusion by up to $t - 1$ participants - while the protocol [26] only tolerates collusion by up to $t/2$ participants. To clearly illustrate the performance of our $\Pi_{\text{MPSU}}^{\text{one-leader}}$, we consider a four-party scenario under a 2048-bit modulus encryption scheme. As shown in Table 5, we compare the online runtime and communication overhead for set sizes $\{2^{12}, 2^{16}\}$ among the protocols from [25], [26], and our $\Pi_{\text{MPSU}}^{\text{one-leader}}$.

As shown in Table 5, in the scenario where only one party obtains the union, when the set size is 2^{12}, the runtime of the protocol from [25] is approximately 663 times longer than ours, and its communication cost is about 4 times ours. In comparison, under the same conditions, the protocol from [26] takes about 22 times longer than ours and incurs about 471 times more communication. When the set size increases to 2^{16}, [26]'s communication cost is roughly 568 times ours.

	Runtime (s)		**Comm. Cost (MB)**	
	2^{12}	2^{16}	2^{12}	2^{16}
$\Pi_{\mathbf{MPSU}}^{\textbf{one-leader}}$	171.42	2764.91	64	1085
[25]	113596.02	—	252	—
[26]	3645.39	63845.08	30117	617047

Table 5: Performance comparison of the online runtime and communication overhead of ours with those of [25] and [26] in a 4-party scenario.

It is also worth noting that although the protocol in [24] achieves the same goal as our $\Pi_{\text{MPSU}}^{\text{leaderless}}$, which allows all parties to obtain the union, it can only defend against collusions involving up to $t/2$ participants. Moreover, since no publicly available implementation of that protocol currently exists, we cannot conduct a direct comparison. In addition, while the protocol in [27] achieves a one-leader scenario and does so with very high performance, its security guarantee holds only if the leader does not collude with any other participant. This weaker security assumption is the reason we do not provide a direct comparison here.

6 Acknowledgement

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00399491, Development of Privacy-Preserving Multiparty Computation Techniques for Secure Multiparty Data Integration)

References

[1] Hogan K, Luther N, Schear N, et al., "Secure multiparty computation for cooperative cyber risk assessment," in *2016 IEEE Cybersecurity Development (SecDev)*, IEEE, 2016: 75-76.

[2] Ramanathan, Sivaramakrishnan, Jelena Mirkovic, and Minlan Yu, "Blag: Improving the accuracy of blacklists," NDSS, 2020.

[3] Kolesnikov V, Rosulek M, Trieu N, et al., "Scalable private set union from symmetric-key techniques," in *International Conference on the Theory and Application of Cryptology and Information Security*, Cham: Springer International Publishing, 2019: 636-666.

[4] Nomura K, Shiraishi Y, Mohri M, et al., "Secure association rule mining on vertically partitioned data using private-set intersection," in *IEEE Access*, 2020, 8: 144458-144467.

[5] Kolesnikov V, Kumaresan R, Rosulek M, et al., "Efficient batched oblivious PRF with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 818-829.

[6] Pinkas B, Schneider T, Zohner M, "Scalable private set intersection based on OT extension," in *ACM Transactions on Privacy and Security (TOPS)*, 2018, 21(2): 1-35.

[7] Pinkas B, Rosulek M, Trieu N, et al., "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 39. Springer International Publishing, 2019: 401-431.

[8] Chase M, 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 40. Springer International Publishing, 2020: 34-63.

[9] Pinkas B, Rosulek M, Trieu N, et al., "PSI from PaXoS: fast, malicious private set intersection," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Cham: Springer International Publishing, 2020: 739-767.

[10] Garimella G, Pinkas B, Rosulek M, et al., "Oblivious key-value stores and amplification for private set intersection," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference*, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41. Springer International Publishing, 2021: 395-425.

[11] Aranha D F, Lin C, Orlandi C, et al., "Laconic private set-intersection from pairings," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022: 111-124.

[12] Raghuraman S, Rindal P, "Blazing fast PSI from improved OKVS and subfield VOLE," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022: 2505-2517.

[13] Bui D, Couteau G, "Improved private set intersection for sets with small entries," in *IACR International Conference on Public-Key Cryptography*, Cham: Springer Nature Switzerland, 2023: 190-220.

[14] Kolesnikov V, Matania N, Pinkas B, et al., "Practical multi-party private set intersection from symmetric-key techniques," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 1257-1272.

[15] Inbar R, Omri E, Pinkas B, "Efficient scalable multiparty private set-intersection via garbled bloom filters," in *International conference on security and cryptography for networks*, Cham: Springer International Publishing, 2018: 235-252.

[16] Kavousi A, Mohajeri J, Salmasizadeh M, "Efficient scalable multi-party private set intersection using oblivious PRF," in *Security and Trust Management: 17th International Workshop*, STM 2021, Darmstadt, Germany, October 8, 2021, Proceedings 17. Springer International Publishing, 2021: 81-99.

[17] Gordon S D, Hazay C, Le P H, "Fully Secure PSI via MPC-in-the-Head," in *Proceedings on Privacy Enhancing Technologies*, 2022.

[18] Ben-Efraim A, Nissenbaum O, Omri E, et al, ”Psimple: Practical multiparty maliciously-secure private set intersection,” in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022: 1098-1112.

[19] Kissner L, Song D, ”Privacy-preserving set operations,” in *Annual International Cryptology Conference*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005: 241-257.

[20] Garimella G, Mohassel P, Rosulek M, et al., ”Private set operations from oblivious switching,” in *IACR International Conference on Public-Key Cryptography*, Cham: Springer International Publishing, 2021: 591-617.

[21] Jia Y, Sun S F, Zhou H S, et al., ”Shuffle-based private set union: Faster and more secure,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022: 2947-2964.

[22] Zhang C, Chen Y, Liu W, et al., ”Linear private set union from {Multi-Query} reverse private membership test,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023: 337-354.

[23] Frikken K, ”Privacy-preserving set union,” in *Applied Cryptography and Network Security: 5th International Conference*, ACNS 2007, Zhuhai, China, June 5-8, 2007. Proceedings 5. Springer Berlin Heidelberg, 2007: 237-252.

[24] Seo J H, Cheon J H, Katz J, ”Constant-round multi-party private set union using reversed laurent series,” in *Public Key Cryptography–PKC 2012: 15th International Conference on Practice and Theory in Public Key Cryptography*, Darmstadt, Germany, May 21-23, 2012. Proceedings 15. Springer Berlin Heidelberg, 2012: 398-412.

[25] Gong X, Hua Q S, Jin H, ”Nearly optimal protocols for computing multi-party private set union,” in *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, IEEE, 2022: 1-10.

[26] Blanton M, Aguiar E, ”Private and oblivious set and multiset operations,” in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012: 40-41.

[27] Liu X, Gao Y, ”Scalable multi-party private set union from multi-query secret-shared private membership test,” in *International Conference on the Theory and Application of Cryptology and Information Security*, Singapore: Springer Nature Singapore, 2023: 237-271.

[28] Davidson A, Cid C, ”An efficient toolkit for computing private set operations,” in *Information Security and Privacy: 22nd Australasian Conference*, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22. Springer International Publishing, 2017: 261-278.

[29] Mohassel P, Sadeghian S, ”How to hide circuits in MPC an efficient framework for private function evaluation,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 557-574.

[30] Batcher K E, ”Sorting networks and their applications,” in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 1968: 307-314.

[31] Oded G, ”Foundations of cryptography: volume 2, basic applications,” 2009.

[32] Freedman M J, Ishai Y, Pinkas B, et al., ”Keyword search and oblivious pseudorandom functions,” in *Theory of Cryptography: Second Theory of Cryptography Conference*, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2. Springer Berlin Heidelberg, 2005: 303-324.

[33] Chase M, Ghosh E, 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 26. Springer International Publishing, 2020: 342-372.

[34] Kolesnikov V, Kumaresan R, ”Improved OT extension for transferring short secrets,” in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference*, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Springer Berlin Heidelberg, 2013: 54-70.

[35] Motwani R, ”Randomized Algor ithms,” in *Cambridge University Press*, 1995.

[36] Pagh R, Rodler F F, ”Cuckoo hashing,” in *European Symposium on Algorithms,* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001: 121-133.

[37] Bay A, Erkin Z, Hoepman J H, et al., ”Practical multi-party private set intersection protocols,” in *IEEE Transactions on Information Forensics and Security*, 2021, 17: 1-15.

[38] Pinkas B, Schneider T, Tkachenko O, et al., ”Efficient circuit-based PSI with linear communication,” in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38. Springer International Publishing, 2019: 122-153.

[39] Mohassel P, Sadeghian S, ”How to hide circuits in MPC an efficient framework for private function evaluation,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques,* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 557-574.

[40] Chandran N, Gupta D, Shah A, ”Circuit-PSI with linear complexity via relaxed batch OPPRF,” in *Proceedings on Privacy Enhancing Technologies*, 2022.

A AH Re-randomizable Threshold PKE

A.1 Public-key Encryption.

A Public-key Encryption (PKE) scheme is a tuple of four probabilistic polynomial-time algorithms:

- $\mathsf{Setup}(1^\kappa)$: On input the security parameter κ outputs public parameters pp, which include the description of the message and ciphertext space $\mathcal{M}, \mathcal{C}$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameter pp, outputs a keypair $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Enc}(\mathsf{pk}, m)$: On input a public key pk and a message $m \in \mathcal{M}$, outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec}(\mathsf{sk}, c)$: On input a secret key sk and a ciphertext $c \in \mathcal{C}$, outputs a message $m \in \mathcal{M}$ or a symbol $\perp$.

Correctness. For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any $m \in \mathcal{M}$, any $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, and any $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, it holds that $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, c^*) = m$.

Indistinguishablility. For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and any $(m_0, m_1) \in \mathcal{M}$, the distribution $c_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0)$ and the distribution $c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1)$ are indentical.

Security. Formally, a PKE scheme is considered indistinguishability under chosen plaintext attack if for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\mathrm{Adv}_{\mathcal{A}}(1^\kappa) = \Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa); \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \\ (m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk}); \\ b \xleftarrow{\$} \{0,1\}; \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b); \\ b' \leftarrow A_2(\mathsf{pp}, c) \end{array} \right] - \frac{1}{2}$$

is negligible in κ.

Informally, a PKE scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is additively homomorphic if for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, all $m_0, m_1 \in \mathcal{M}$ and arbitrary constant a, such that

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m_0) +_H \mathsf{Enc}(\mathsf{pk}, m_1)) = m_0 + m_1$
- $\mathsf{Dec}(\mathsf{sk}, a\mathsf{Enc}(\mathsf{pk}, m_0)) = am_0$

Notably, an additively homomorphic PKE scheme satisfies that re-randomizable algorithm ReRand, s.t. $\mathsf{ReRand}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1)) = \mathsf{Enc}(\mathsf{pk}, m_1) +_H \mathsf{Enc}(\mathsf{pk}, 0)$.

A.2 Threshold PKE

Threshold PKE (TPKE) scheme is an advanced variant of PKE scheme, where the secret key is split among several participants instead of being held by a single party. Specifically, in a threshold system denoted as (k, t)-TPKE, the secret key sk of a PKE scheme is distributed among t participants, each holding a secret share sk_i. The system ensures that decryption is only possible if a subset of at least k participants collaboratively decrypt the message, combining their shares. The key generation in TPKE can be done in two ways: either through a distributed key generation algorithm KeyGen, which securely generates sk_i among the participants without revealing the entire secret key to any single entity, or by relying on a trusted third party to generate and distribute sk_i. The encryption algorithm Enc works similarly to the regular PKE, using the public key pk generated in the KeyGen.

For decryption algorithm Dec, TPKE involves two main algorithms:

1. **Share Decryption** ($\mathsf{ShareDec}$): Each of the k participants uses their sk_i to produce a partial decryption of the ciphertext, resulting in decryption shares c_i.

2. **Combining Algorithm** ($\mathsf{Combine}$): The decryption shares $c_1, ..., c_k$ are combined with the public key pk to produce the final plaintext m (or output $\perp$ to indicate an invalid decryption).

Similarly, if a TPKE scheme satisfies both the additive homomorphism property and the re-randomization condition of a PKE scheme, we say that the TPKE is **Additively Homomorphic Re-randomizable TPKE** (AHR-TPKE).

B Remark on Sequential Re-randomization Encryption Based on AHR-TPKE

Remark 1. Assume there exists a set $\{x_1, ..., x_n\}$ and a AHR-TPKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ShareDec}, \mathsf{Combine})$. There are t participants who jointly execute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(\mathsf{pp}, k, n)$, and each participant $\mathcal{P}_i$ obtains their own keypair $(\mathsf{pk}, \mathsf{sk}_i)$, where $k \leq t$ is threshold. For $i \in [t]$ and $j \in [n]$, we have the following inductive definition based on mathematical induction:

- For $i = 1$, A_j^i is defined as:
$$A_j^1 = \mathsf{Enc}(\mathsf{pk}, x_j) \oplus a_j^1$$
- For $2 \leq i \leq t-1$, A_j^{i-1} is defined, then A_j^i is defined as:
$$A_j^i = \left((A_j^{i-1} \oplus b_j^{i-1}\right) +_H \mathsf{Enc}(\mathsf{pk}, 0)) \oplus a_j^i$$
- For $i = t$, A_j^{t-1} is defined, then A_j^i is defined as:
$$A_j^t = A_j^{t-1} \oplus b_j^{t-1}$$

, where a_j^i and b_j^i are randomly selected strings. For each a_j^i and b_j^i, if $(a_j^i \oplus b_j^i) = 0$, $A_j^t = c = \mathsf{Enc}(\mathsf{pk}, x_j) +_H \mathsf{Enc}(\mathsf{pk}, 0) +_H ... +_H \mathsf{Enc}(\mathsf{pk}, 0)$, that is, x_j has undergone initial encryption followed by $t-2$ rounds of re-randomization. At this point, x_j can be recovered by having k out of the t participants compute jointly $\mathsf{Combine}(c, \{\mathsf{ShareDec}(\mathsf{sk}_1^*, c), ..., \mathsf{ShareDec}(\mathsf{sk}_k^*, c)\})$, where $\{\mathsf{sk}_1^*, \cdots, \mathsf{sk}_k^*\} \subseteq \{\mathsf{sk}_1, \cdots,_n\}$. Conversely, if for each a_j^i and b_j^i, there exists at least one $a_j^i \oplus b_j^i \neq 0$, then x_j cannot be recovered. The correctness and security are ensured by the encryption/decryption and re-randomization properties of the AHR-TPKE.