Scientific Reports

https://doi.org/10.1038/s41598-026-41069-2

Article in Press

Efficient multi-party private set union resistant to maximum collusion attacks

Received: 26 September 2025

Accepted: 17 February 2026

Published online: 12 March 2026

Cite this article as: Liu Q. & Lee J. Efficient multi-party private set union resistant to maximum collusion attacks. *Sci Rep* (2026). https://doi.org/10.1038/s41598-026-41069-2

Qiang Liu & Joon-Woo Lee

We are providing an unedited version of this manuscript to give early access to its findings. Before final publication, the manuscript will undergo further editing. Please note there may be errors present which affect the content, and all legal disclaimers apply.

If this paper is publishing under a Transparent Peer Review model then Peer Review reports will publish with the final article.

ARTICLE IN PRESS

© The Author(s) 2026. **Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

ARTICLE IN PRESS

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

Qiang Liu[1] **and Joon-Woo Lee**[1,*]

[1]School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea.
[*]jwlee2815@cau.ac.kr

ABSTRACT

Multi-party Private Set Union (MPSU) enables multiple parties to compute the union of their private sets while revealing nothing beyond the resulting union. Gao et al. (PETS 2024) proposed an efficient MPSU protocol for large-scale sets that resists maximum semi-honest collusion, where only a designated leader learns the union. Their protocol nevertheless incurs high communication overhead due to transmitting large volumes of threshold re-randomizable public-key encryption (TR-PKE) ciphertexts during interaction.

In this work, we design an optimized protocol, $\Pi_{\text{MPSU}}^{\text{one-leader}}$, that reduces communication by using symmetric-key encryption (SKE) for most message payloads and restricting TR-PKE to a few indispensable operations. Security is preserved by separating roles. TR-PKE is used only for set items that require re-randomization and joint threshold decryption, while SKE protects all other messages carrying masked or randomized intermediate payloads. Building on $\Pi_{\text{MPSU}}^{\text{one-leader}}$, we further propose a leaderless protocol, $\Pi_{\text{MPSU}}^{\text{leaderless}}$, which allows all parties to obtain the union output and mitigates the risk of a single leader dominating in decentralized deployments.

We implement our schemes and conduct a comparison with state-of-the-art. The results demonstrate that $\Pi_{\text{MPSU}}^{\text{one-leader}}$ achieves a 4–5$\times$ shrinking in communication and a 1.3–1.8$\times$ speedup, depending on the set size and the number of parties.

Keywords Multi-party private set union, Maximum collusion, Symmetric key encryption, Threshold re-randomizable public key encryption

ARTICLE IN PRESS

Introduction

Private Set Union (PSU) is a fundamental cryptographic protocol that enables two or more parties to securely compute the union of their sets without disclosing any private information. As the demand for privacy protection continues to grow, PSU has become increasingly important for joint data analysis. It has found applications in areas such as network risk assessment[1,2] and data mining[3,4], where computing set unions over sensitive data is often essential. For example, financial institutions (e.g., banks) may need to share blacklists of suspicious transactions or fraudulent accounts to better prevent financial fraud. These blacklists are sensitive and are typically derived from proprietary risk-detection strategies, making direct sharing impractical. PSU allows banks to compute the union of their blacklists without exposing customer information or internal detection rules, enabling collaboration while limiting privacy and strategy leakage.

During the past decade, Private Set Intersection (PSI) has garnered extensive research, with two-party PSI making particularly notable progress[5–13]. Benefiting from advancements in two-party PSI, 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. After Kissner and Song initially proposed the two-party PSU protocol[19], PSU designs were dominated by expensive additive homomorphic encryption (AHE)[19,23–25]. As a result, performance remained limited, hindering practical large-scale deployments. 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 sets[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 scheme proposed by Zhang et al. achieved linear complexity for PSU in 2023[22], marking a significant breakthrough in this field.

Similarly to two-party PSU, existing work on MPSU has largely relied on AHE and general secure multiparty computation (SMPC) frameworks[26]. However, due to expensive computation associated with these techniques, their applications remain largely theoretical and are challenging to implement for large-scale sets in practice. In 2023, Liu et al. introduced the first MPSU protocol fully based on SKE, capable of handling large sets[27]. This efficiency, however, comes with weaker security guarantees, since it is limited to the setting where the designated leader who learns the union is non-colluding. More recently, Gao et al. proposed the first efficient MPSU protocol that supports large-scale sets while resisting maximum semi-honest collusion[28]. Nevertheless, their protocol remains communication-heavy, as the interaction phase exchanges a large number of

ARTICLE IN PRESS

TR-PKE ciphertexts.

In this work, under the assumption of security against the maximum semi-honest collusion, we propose an improved MPSU protocol that enables the leader to obtain the union result. Furthermore, we extend it to a protocol where all parties can receive the union result. Specifically, our contribution can be summarized as follows:

- We introduce a one-leader MPSU protocol, $\Pi_{\text{MPSU}}^{\text{one-leader}}$, that is secure against maximum semi-honest collusion. Our protocol reduces the communication cost by avoiding the pervasive transmission of TR-PKE ciphertexts. In particular, TR-PKE is used only for the chained transfer of set items, which must remain confidential and later be recovered via joint threshold decryption. All other interactions carry only intermediate payloads and are protected using SKE. $\Pi_{\text{MPSU}}^{\text{one-leader}}$ consists of two phases: preparation and interaction. In the preparation phase, parties set up system parameters and preprocess their input sets before any actual interaction. In the interaction phase, parties perform communication to compute the union, after which only the designated leader learns the final result.

- To mitigate the dominance risk of a single output holder in decentralized settings, we design a leaderless variant, $\Pi_{\text{MPSU}}^{\text{leaderless}}$, building on $\Pi_{\text{MPSU}}^{\text{one-leader}}$. It enables all parties to learn the union. $\Pi_{\text{MPSU}}^{\text{leaderless}}$ reuses the same preparation phase as $\Pi_{\text{MPSU}}^{\text{one-leader}}$, and its interaction phase is realized via multiple invocations of the one-leader interaction, with each party taking a turn as the output recipient. While the leader in $\Pi_{\text{MPSU}}^{\text{one-leader}}$ could broadcast the union after completing the protocol under semi-honest behavior, such a broadcast is external to the formally verified computation and provides no fairness if the leader crashes or aborts. By allowing every party to obtain the union directly, $\Pi_{\text{MPSU}}^{\text{leaderless}}$ improves robustness and fairness. This comes at a higher communication cost than $\Pi_{\text{MPSU}}^{\text{one-leader}}$, which is the price of eliminating reliance on a single party for output delivery.

- We implement our protocols using threshold finite field Elgamal encryption as the instantiation of our TR-PKE. We further benchmark $\Pi_{\text{MPSU}}^{\text{one-leader}}$ against Gao et al.'s protocol[28]. To ensure a fair comparison, we (i) run their implementation as reported and (ii) re-evaluate our protocol under the same threshold Curve25519-based ElGamal instantiation used by Gao et al. Across a range of set sizes and numbers of parties, our protocol reduces communication by 4–5× and improves runtime by 1.3–1.8×.

Related Work

In this section, we review previous MPSU protocols that are relevant to our work as below.

Kissner and Song proposed the earliest MPSU protcol based on polynomial representation and AHE[19]. Each party P_i ($i \in [t]$) represents their input set $X_i = \{x_{i1}, x_{i2}, \cdots, x_{in}\}$ 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 $O(t^3n^2)$. Subsequently, Frikken proposed an improved method, 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 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 bloom filter (BF) in 2022[25]. The protocol constructs a BF to store the union and exploits the no-collusion 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 cost.

In additional to using expensive AHE, employing the general SMPC to address the MPSU is also an interesting approach. In 2012, Blanton and Aguiar proposed an MPSU protocol based on oblivious sorting and general SMPC, which operates in an honest majority setting[26]. The core idea is to merge all parties' 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 parties.

Liu and Gao in 2023 proposed an MPSU protocol for efficiently computing large-scale sets, based on SKE[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 parties. However, the security of their protocol depends on the assumption that the leader does not collude with other parties, which results in weaker security guarantees.

Recently, in 2024, Gao et al. proposed a MPSU protocol capable of resisting maximum collusion attacks[28]. In their scheme, a sub-protocol called "membership oblivious transfer (mOT)" was designed by combining OT with secret-shared

ARTICLE IN PRESS

Protocol	Year	Comm.	Comp.	Threshold	Ope.
Blanton et al.[26]	2012	$O(t^2 n \log^2(nt))$	$O(t^2 n \log^2(nt))$	$< \lceil t/2 \rceil$	Oblivious sorting
Seo et al.[24]	2012	$O(t^3 n^2)$	$O(t^4 n^2)$	$< \lceil t/2 \rceil$	SKE
Liu et al.[27]	2023	$O(t^2 n \log(tn))$	$O(tn)$	$< t$ (leader no colludes with others)	
Kissner et al.[19]	2005	$O(t^2 n)$	$O(t^3 n^2)$	$< t$	PKE
Frikken et al.[23]	2007	$O(t^2 n)$	$O(t^2 n \log n)$	$< t$	
Gong et al.[25]	2022	$O(tn)$	$O(tn)$	$< t$	
Gao et al.[28]	2024	$O(t^2 n \log(n))$	$O(t^2 n \log(n))$	$< t$	SKE + PKE
Our $\Pi_{\text{MPSU}}^{\text{one-leader}}$	-	$O(t^2 n \log(n))$	$O(t^2 n \log(n)))$	$< t$	SKE (+partially PKE)

Table 1. Complexity comparisons of semi-honest MPSU protocols. Note: Comm./Comp.: Communication/Computational complexity; Ope.: Operation technique; PKE: Public-key encryption operations; SKE: Symmetric key encryption operations; n: Size of sets; t: Number of parties.

private membership test to perform oblivious item membership detection. The protocol encrypts data using TR-PKE and uses the encrypted data as the input for mOT to enable interactive computation. The leader then obtains the encrypted union data, and the final union is derived through joint decryption. However, since nearly all messages exchanged during the protocol are TR-PKE encrypted, the communication volume grows substantially, resulting in high overall bandwidth cost.

Following, we summarize the computational and communication complexities, resistance to collusion attacks, as well as the encryption operations used in MPSU protocols, in Table 1.

Preliminaries

Notation

We organize and summarize the notation used in this paper in Table 2.

Notation	Meaning
$\mathscr{P}_i$	Party i, where $i \in [t]$
t	Number of parties
X_i	Private set held by $\mathscr{P}_i$
n	Set size (number of elements in each X_i)
x_i^j	The j-th element of X_i, where $j \in [n]$
$[n]$	Index set $\{1, 2, \ldots, n\}$
κ	Computational security parameter
λ	Statistical security parameter
$r \xleftarrow{\$} \{0,1\}^\ell$	Uniform sampling of an ℓ-bit string r
ℓ	Bit-length of a string
$\|\|$	Concatenation of two strings
PRF	Pseudorandom function
$\overset{c}{\equiv}$	Computational indistinguishability
PPT	Probabilistic polynomial-time

Table 2. List of notation.

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"[31] to prove the semi-honest security, with the specific definition given below.

Definition 1. *Let $f : (\{0,1\}^*)^t \to (\{0,1\}^*)^t$ be an t-ary functionality, where $f_i(x_1, \cdots, x_t)$ denotes the i-th item of $f(x_1, \cdots, x_t)$. For $I = (i_1, \cdots, i_k) \subseteq [t]$, we let $f_I(x_1, \cdots, x_t)$ denote the subsequence $f_{i_1}(x_1, \cdots, x_t), \cdots, f_{i_k}(x_1, \cdots, x_t)$. Let Π be an t-party protocol for computing f. The view of the i-th party during an execution of Π on $\bar{x} = (x_1, \cdots, x_t)$ is denoted by $\mathsf{View}_i^\Pi(\bar{x})$. For $I = (i_1, \cdots, i_k)$, we let $\mathsf{View}_I^\Pi(\bar{x}) \overset{def}{=} (I, \mathsf{View}_{i_1}^\Pi(\bar{x}), \cdots, \mathsf{View}_{i_k}^\Pi(\bar{x}))$. 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*

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

ARTICLE IN PRESS

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

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 $\mathscr{P} = \{(x_j, y_j)\}_{j\in[n]}$, 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 hint. Importantly, the receiver cannot distinguish whether an output value $F(k, \mathsf{hint}, q_i)$ corresponds to a programmed point. Specifically, 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 $\mathscr{F}_{\text{OPPRF}}$ is given in Functionality 1.

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

Parameters:

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

Functionality 1. Ideal Functionality for Oblivous Programmble PRF.

ARTICLE IN PRESS

Permute + Share.

The Permute + Share (PS) functionality $\mathscr{F}_{\text{PS}}$ was proposed by Chase et al. in 2020[33]. In this functionality, there are two parties, namely a sender and a receiver. The sender holds a set $\{x_1, \cdots, x_n\}$ 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 $\mathscr{F}_{\text{PS}}$ is for the sender to learn the share values $\{s_{\pi(1)}, \cdots, s_{\pi(n)}\}$, while the receiver obtains the other shared values $\{s_{\pi(1)} \oplus x_{\pi(1)}, \cdots, s_{\pi(n)} \oplus x_{\pi(n)}\}$. The concrete ideal functionality $\mathscr{F}_{\text{PS}}$ is shown in Functionality 2.

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

Parameters:

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

Functionality:

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

Functionality 2. Ideal Functionality for Permute + Share.

Private Equality Test.

Private Equality Test (PEqT) allows two parties to test whether their input strings are equal. The sender holds x and a receiver holds y. Upon invoking PEqT, the sender obtains an output bit b and the receiver obtains an output bit b^*, such that $b \oplus b^* = 0$ if $x = y$, and $b \oplus b^* = 1$ otherwise. The ideal functionality $\mathscr{F}_{\text{PEqT}}$ is illustrated in Functionality 3.

ARTICLE IN PRESS

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

Parameters:

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

Functionality:

1. Wait for input x from the sender $\mathscr{S}$;
2. Wait for input y from the receiver $\mathscr{R}$;
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 $\mathscr{S}$ and output b^* to the receiver $\mathscr{R}$.

Functionality 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 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 vector b. Finally, the receiver obtains the output result $x_{b_i}^i$, but the sender obtains nothing. The ideal functionality $\mathscr{F}_{\binom{2}{1}\text{-OTe}}$ is shown in Functionality 4.

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

Parameters:

- Two parties: Sender $\mathscr{S}$ and Receiver $\mathscr{R}$;
- 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 $\mathscr{S}$;
2. Wait for input choice vector $b \in \{0,1\}^n$ from the receiver $\mathscr{R}$;
3. Give output $(x_{b_1}^1, \cdots, x_{b_n}^n)$ to the receiver $\mathscr{R}$.

ARTICLE IN PRESS

Functionality 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. The following inequality shows that 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]:

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

Cuckoo Hashing.

Cuckoo Hashing was introduced by Pagh and Rodler in 2001[36]. It uses multiple hash functions $h_1, h_2, \ldots, h_\beta$ to map items to bins in a hash table of size $b = \varepsilon n$. The distinctive feature of Cuckoo Hashing is its ability to ensure that each bin contains only one item, thereby maintaining structural simplicity and minimizing collusions. The core lies in a deterministic eviction process. When an item x is inserted, it is initially hashed into one of the bins $B_{h_1(x)}, B_{h_2(x)}, \ldots, B_{h_\beta(x)}$. If all designated bins are occupied, then 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. Mathematically, by appropriately tuning the parameters β 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].

ARTICLE IN PRESS

Threshold Re-randomizable Public-key Encryption.

A re-randomizable public-key encryption (PKE) scheme is a tuple of five 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 $\mathscr{M}, \mathscr{C}$.
- $\mathsf{KeyGen}(\mathsf{pp})$: 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 $m \in \mathscr{M}$, outputs a ciphertext $c \in \mathscr{C}$.
- $\mathsf{Dec}(\mathsf{sk}, c)$: On input a secret key sk and a ciphertext $c \in \mathscr{C}$, outputs a message $m \in \mathscr{M}$ or a symbol $\perp$.
- $\mathsf{ReRand}(pk, c)$: On input a public key pk and a ciphertext $c \in \mathscr{C}$, outputs another ciphertext $c^* \in \mathscr{C}$.

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

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

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

$$\mathrm{Adv}_{\mathscr{A}}(1^\kappa) = \Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa); \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \\ (m_0, m_1) \leftarrow \mathscr{A}_1(\mathsf{pp}, \mathsf{pk}); \\ b \xleftarrow{\$} \{0,1\}; \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b); \\ b' \leftarrow \mathscr{A}_2(\mathsf{pp}, c) \end{array} \right] - \tfrac{1}{2}$$

is negligible in κ.

Threshold PKE (TPKE) scheme is an advanced variant of PKE scheme, where the secret key is split among several parties 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 parties, each holding a secret share sk_i. The system ensures that decryption is only possible if a subset of at least k parties collaboratively decrypt the message, combining their shares. TPKE key generation can be carried out either via a distributed key-generation algorithm KeyGen, which produces secret shares sk_i without ever reconstructing the full secret key at any single party. Alternatively, a trusted dealer may generate the key material and distribute the shares sk_i to the parties. The encryption algorithm Enc works similarly to the regular PKE, using the public key pk generated in the KeyGen. For the decryption algorithm $PartDec$, $\mathscr{P}_i$ uses its private key sk_i to partially decrypt the ciphertext and then forwards the result to the next party $\mathscr{P}_{i+1}$. This process continues sequentially until the k-th party performs the final decryption step and recovers the plaintext message.

If a TPKE scheme is re-randomizable, we say that the TPKE is threshold re-randomizable PKE (TR-PKE). In addition, we present a remark on sequential re-randomization based on TR-PKE as below, which will be employed later in our protocol.

Remark 1. Assume there exists a set $\{x_1, \cdots, x_n\}$ and a TR-PKE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, PartDec, \mathsf{ReRand})$. There are t parties who jointly execute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(\mathsf{pp}, k, t)$, and each $\mathscr{P}_i$ obtains their own keypair $(\mathsf{pk}, \mathsf{sk}_i)$, where $k \le t$ is threshold. For $i \in [t], j \in [n]$, the inductive definition based on mathematical induction can be obtained:

- For $i = 1$, A_j^i is defined as $A_j^1 = \mathsf{Enc}(\mathsf{pk}, x_j) \oplus a_j^1$;
- For $2 \le i \le t-1$, A_j^{i-1} is defined, then A_j^i is defined as $A_j^i = \left(\mathsf{ReRand}(A_j^{i-1} \oplus b_j^{i-1})\right) \oplus a_j^i$;
- 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 $(a_j^i \oplus b_j^i) = 0$, $A_j^t = c = \mathsf{ReRand}(...\mathsf{ReRand}(\mathsf{Enc}(\mathsf{pk}, x_j)))$, 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 parties compute $c_1 = PartDec(\mathsf{sk}_1^*, c), c_2 = PartDec(\mathsf{sk}_2^*, c_1) \cdots, c_k = PartDec(\mathsf{sk}_k^*, c)$ sequentially, where $\{\mathsf{sk}_1^*, \cdots, \mathsf{sk}_k^*\} \subseteq \{\mathsf{sk}_1, \cdots, \mathsf{sk}_n\}$ and c_k is the recovered message. 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 TR-PKE with IND-CPA security.

ARTICLE IN PRESS

Technique Overview

We present the ideal functionality of the MPSU, as shown in Functionality 5. Assume there are t parties $\mathscr{P}_1, \cdots, \mathscr{P}_t$, where each $\mathscr{P}_i$ holds a set $X_i = \{x_1^i, \cdots, x_n^i\}$ of size n. This ideal functionality captures both the one-leader and leaderless variants. In both cases, the parties provide their respective input sets $\{X_i\}_{i\in[t]}$ and the functionality outputs only the union $\bigcup_{i=1}^t X_i$, without revealing anything beyond this output. In the one-leader setting, only the designated leader receives $\bigcup_{i=1}^t X_i$; in the leaderless setting, every party receives it.

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

Parameters:

- t parties $\mathscr{P}_1, \cdots, \mathscr{P}_t$ and the set size n;

Functionality:

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

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

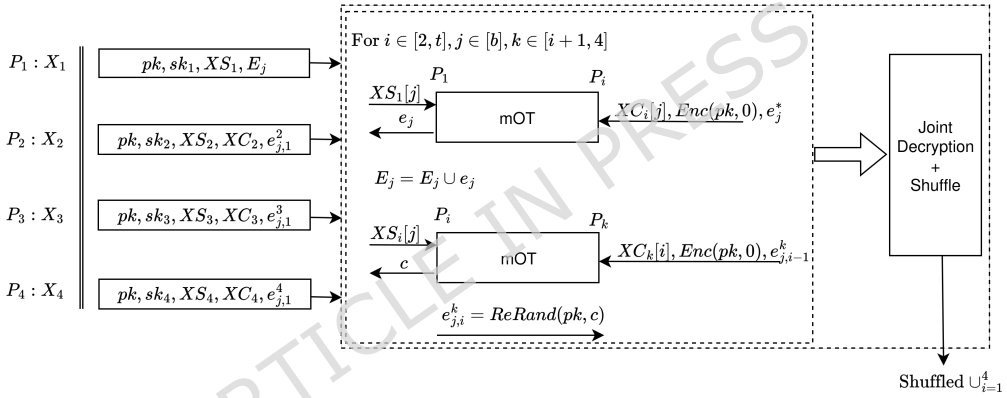


Figure 1. Execution Flowchart of Gao et al.'s One-leader Multi-party Set Union Protocol for $t = 4$ Parties

Overview of Gao et al.'s One-leader MPSU Protocol

The protocol of Gao et al.[28] realizes the one-leader MPSU functionality in Functionality 5. Figure 1 sketches Gao et al.'s one-leader MPSU protocol for $t = 4$ parties. Following Figure 1, we review their construction and highlight the overhead bottlenecks that motivate our optimizations.

- **Construction review.**
  - First, The parties perform the following operations to generate the TR-PKE keys and hash-based data structures:
    - Each P_i ($i \in [t]$) holds a shared public key pk, generated by a TR-PKE scheme, along with its secret key sk_i.
    - Each P_i hashes its input set X_i into a simple hash table XS_i with b bins.
    - For each $i \in \{2, \ldots, t\}$, P_i additionally builds a cuckoo hash table XC_i of size b (padding empty bins with a dummy value) and encrypts each entry as $e^i_{j,1} \leftarrow \mathsf{Enc}(pk, XC_i[j])$ for all $j \in [b]$.
    - The leader P_1 initializes b empty sets $\{E_j\}_{j\in[b]}$.
  - Then, the protocol uses repeated invocations of mOT to aggregate an encrypted version of $(\bigcup_{i=1}^t X_i) \setminus X_1$, padded with extra $\mathsf{Enc}(pk, 0)$ ciphertexts. In mOT, the receiver inputs a set $\tilde{S} = \{\tilde{s}_1, \ldots, \tilde{s}_m\}$, while the sender inputs a

ARTICLE IN PRESS

keyword s and a pair (v_0, v_1); the receiver obtains v_0 if $s \in \tilde{S}$ and v_1 otherwise. Specifically, for $i \in [2,t], j \in [b], k \in [i+1,t]$:

* P_1 and P_j invoke mOT, where P_1 inputs $XS_1[j]$, and P_i inputs keyword $XC_i[j]$ and the pair $(\mathsf{Enc}(pk,0), e^*_j)$. As a result, P_1 obtains

$$e_j = \begin{cases} \mathsf{Enc}(pk,0), & \text{if } XC_i[j] \in XS_1[j], \\ e^*_j, & \text{otherwise,} \end{cases}$$

where $e^*_j = \mathsf{Enc}(pk,0)$ if $XC_i[j]$ is a dummy value and $e^*_j = e^i_{j,i-1}$ otherwise. The leader P_1 then updates $E_j \leftarrow E_j \cup \{e_j\}$.

* P_i and P_k invoke mOT, where P_i inputs $XS_i[j]$, and P_k inputs keyword $XC_k[j]$ and the pair $(\mathsf{Enc}(pk,0), e^k_{j,i-1})$. As a result, P_i obtains c, where $c = Enc(pk,0)$ if $XC_k[j] \in XS_i[j]$, and $c = e^k_{j,i-1}$ otherwise. P_i then sends $e^k_{j,i}$ to P_k.

– Finally, P_1 forms $E = \bigcup_{j=1}^{b} E_j$, which contains encryptions of $(\bigcup_{i=1}^{t} X_i) \setminus X_1$ along with padding ciphertexts $\mathsf{Enc}(pk,0)$. All parties then jointly decrypt and randomly shuffle E. After removing zeros, P_1 unions the recovered items with its local set X_1 to obtain the final output $\bigcup_{i=1}^{t} X_i$.

- **Overhead bottlenecks and optimization motivation**

– *Overhead bottlenecks.*

* Their mOT is implemented by first using a garbled circuit to test membership ($s \in \tilde{S}$) and then using OT to deliver either v_0 or v_1 to the receiver. In their protocol, the keyword s is the entry of a cuckoo-hash bin, the pair (v_0, v_1) consists of TR-PKE ciphertexts, and $\tilde{S}$ is the set of items stored in the corresponding bin of the simple hash table (of size $O(\log n)$). Consequently, each mOT invocation incurs a garbled circuit of size $O(\log n)$, together with OT over TR-PKE-encrypted ciphertext pairs.
* Overall, their protocol performs $b(t-1)(t-2)/2$ such mOT calls, leading to substantial overhead from both the large number of garbled circuits and the repeated OT on bulky TR-PKE ciphertexts.

Optimization motivations. This motivates two complementary optimizations: (i) batching comparisons to avoid per-bin invocations, and (ii) avoiding TR-PKE ciphertexts as inputs to sub-protocols whenever possible. Our design pursues both directions. First, it batches comparisons to avoid per-bin invocations. Second, it uses SKE for transporting payloads and intermediate values, reserving TR-PKE only for the few steps that require re-randomization and joint threshold decryption.

ARTICLE IN PRESS

Overview of Our One-leader MPSU

In this section, we provide a technical overview of our $\Pi^{\text{one-leader}}_{\text{MPSU}}$ protocol. We also introduce a new module, called Batched Equality-tested Oblivious Random Generation (BEtORG), which plays a key role in the protocol design.

Batched Equality-tested Oblivious Random Generation. BEtORG is a two-party functionality for generating equality-dependent correlated randomness. In this functionality, the sender inputs $X = \{x_i\}_{i\in[n]}$ and the receiver inputs $Y = \{y_i\}_{i\in[n]}$. As a result, BEtORG outputs $\{(a_i, \tilde{a}_i)\}_{i\in[n]}$ to the sender and $\{b_i\}_{i\in[n]}$ to the receiver. It guarantees that for every $i \in [n]$, if $x_i = y_i$ then $b_i = \tilde{a}_i$; otherwise $b_i = a_i$. We realize this functionality using $\mathscr{F}_{\text{PEqT}}$ and $\mathscr{F}_{\binom{2}{1}\text{-OT}}$. Figure 2 depicts the protocol flow, which we describe in detail below.

- For each $i \in [n]$, the parties invoke $\mathscr{F}_{\text{PEqT}}$ with inputs x_i and y_i. The sender receives bits $\{e_i\}_{i\in[n]}$ and the receiver receives bits $\{\tilde{e}_i\}_{i\in[n]}$, such that $e_i \oplus \tilde{e}_i = 0$ if $x_i = y_i$, and $e_i \oplus \tilde{e}_i = 1$ otherwise.
- The sender samples, for each $i \in [n]$, a random pair of strings $\{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$ and uses them as input to $\mathscr{F}_{\binom{2}{1}\text{-OT}}$. The receiver uses the choice bits $\{\tilde{e}_i \oplus 1\}_{i\in[n]}$ and obtains $\{b_i\}_{i\in[n]}$.
- The sender outputs the pairs $\{(a_i, \tilde{a}_i)\}_{i\in[n]} = \{(r_{e_i}, r_{e_i\oplus 1})\}_{i\in[n]}$, while the receiver outputs $\{b_i\}_{i\in[n]}$. By construction, for each $i \in [n]$, $b_i = \tilde{a}_i$ if $x_i = y_i$; otherwise, $b_i = a_i$.

ARTICLE IN PRESS

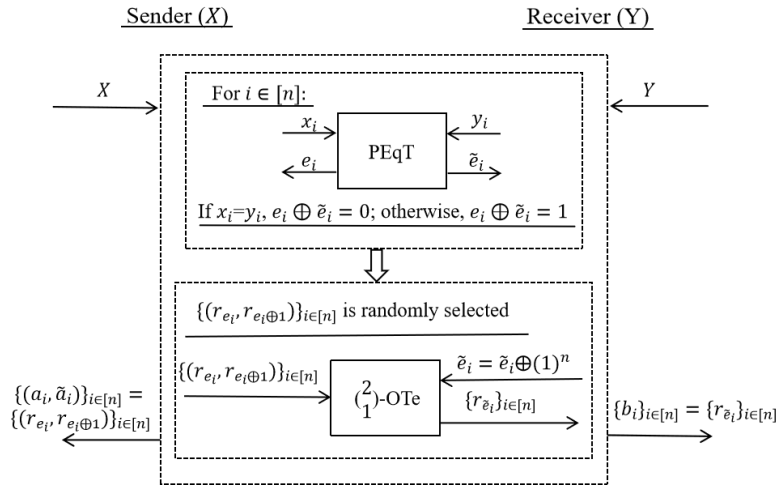


Figure 2. Execution Flowchart of Our Batched Equality-tested Oblivious Random Generation Protocol

Our One-leader MPSU. Figure 3(a) depicts the overall framework of our $\Pi_{\text{MPSU}}^{\text{one-leader}}$, and Figure 3(c) details its interaction phase for $t = 4$ parties. Briefly, our protocol has two phases: a preparation phase that sets up system parameters and preprocesses their input sets before any actual interaction, and an interaction phase that produces the encrypted union material and releases the final union to the designated leader $\mathscr{P}_t$.

- **Preparation.** The t parties $\mathscr{P}_i$ ($i \in [t]$) jointly execute the TR-PKE to generate a shared public key pk and individual secret key sk_i. Then, each party builds three hash-based structures over their set X_i:
  - A simple hashing table XS_i with b bins, where each bin stores up to m items;
  - A cuckoo hashing table XC_i with b bins, where each bin stores at most one item;
  - An encrypted cuckoo hashing table EXC_i with b bins, where $\mathsf{EXC}_i[j] = \mathsf{Enc}(\mathsf{pk}, H(\mathsf{XC}_i[j])||\mathsf{XC}_i[j])$ for all $j \in [b]$.
- **Interaction.** After preparation, $\mathscr{P}_i$ enters the interaction phase with $\big((\mathsf{pk}, \mathsf{sk}_i), \mathsf{XS}_i, \mathsf{XC}_i, \mathsf{EXC}_i\big)$. As shown in Figure 3(a), the interaction phase consists of two modules: (i) a core computation module $\Pi_{\text{MPSU}}^{\text{core}}$, and (ii) a Joint Decryption + Shuffle module.
  - $\Pi_{\text{MPSU}}^{\text{core}}$.
    - Π_{MPSU}^{core} *(what it produces).* The goal of $\Pi_{\text{MPSU}}^{\text{core}}$ is to let the leader $\mathscr{P}_t$ obtain an encrypted multiset

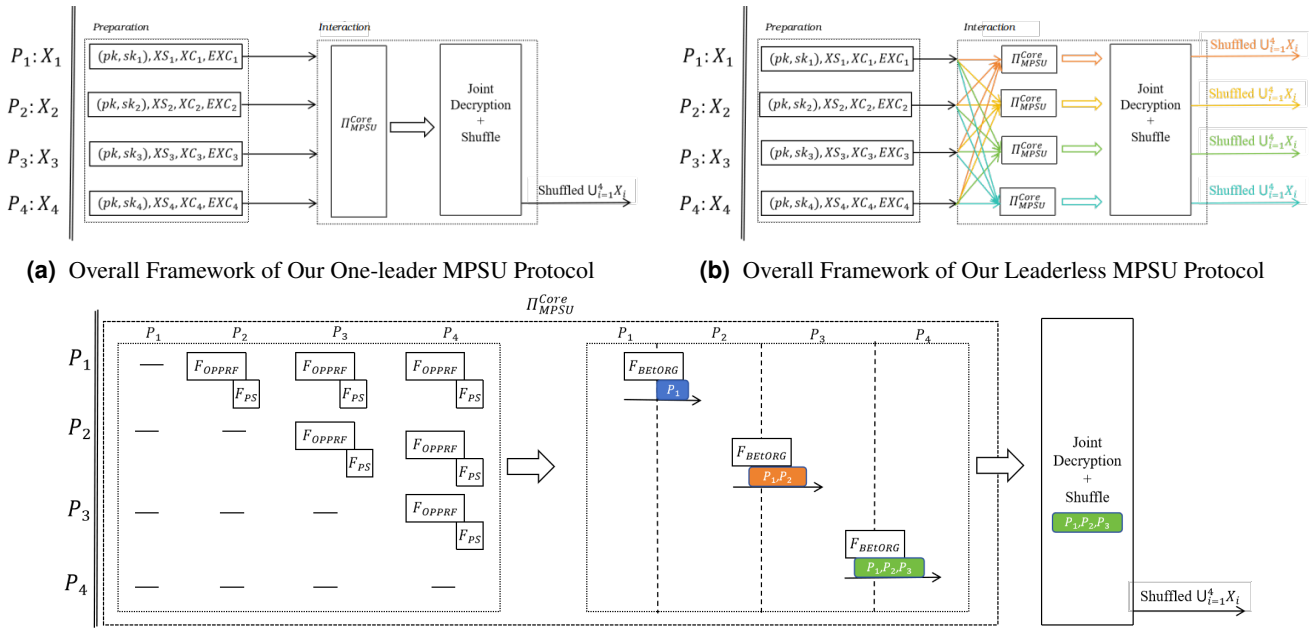
      $$\mathsf{EncSet} \;=\; \mathsf{ReRand}\big((\mathsf{EXC}_1 \cup \cdots \cup \mathsf{EXC}_{t-1}) \setminus \mathsf{EXC}_t\big) \;\cup\; \mathsf{Pad},$$

      where Pad contains $(t-1)n - \big|(\mathsf{EXC}_1 \cup \cdots \cup \mathsf{EXC}_{t-1}) \setminus \mathsf{EXC}_t\big|$ encryptions of random values. Intuitively, EncSet hides the true cardinality and masks duplicates, while keeping all items encrypted under TR-PKE.
    - Π_{MPSU}^{core} *(how it works).* It is organized as four subroutines executed in sequence (as shown in Figure 3(c)):

      $$\mathscr{F}_{\text{OPPRF}} \;\rightarrow\; \mathscr{F}_{\text{PS}} \;\rightarrow\; \mathscr{F}_{\text{BEtORG}} \;\rightarrow\; \mathsf{package\&send}.$$

      Below we describe the purpose of each subroutine at a high level.
      - $\mathscr{F}_{\textbf{OPPRF}}$ **(bin-aligned labels).** For each $j \in [t]$, $\mathscr{P}_j$ randomly selects $j-1$ strings group $\{r_1^{j,q}, \ldots, r_b^{j,q}\}_{q\in[j-1]}$. By interacting, each $\mathscr{P}_q$ ($q \in \{j+1, \ldots, t\}$) derives a label vector $\{w_k^{q,j}\}_{k\in[b]}$ such that $w_k^{q,j} = r_k^{q,j}$ if $\mathsf{XC}_j[k] \in \mathsf{XS}_q[k]$, and is pseudorandom otherwise. This ensures that equal items hashed to the same bin index obtain consistent labels across parties, without revealing the items themselves.

ARTICLE IN PRESS



(a) Overall Framework of Our One-leader MPSU Protocol

(b) Overall Framework of Our Leaderless MPSU Protocol

(c) Execution Flowchart of Interaction Phase of Our One-leader MPSU Protocol

Figure 3. Execution Flowchart of Our New Multi-party Set Union Protocols for $t = 4$ Parties.

- $\mathcal{F}_{\mathbf{PS}}$ **(secret permutation of labels).** $\mathscr{P}_j$ samples a permutation $\pi_j : [b] \to [b]$ and interacts with each $\mathscr{P}_q$ $(q > j)$ to obtain permuted label shares. Concretely, $\mathscr{P}_j$ and $\mathscr{P}_q$ receive $\{s_k^{q,j}\}_{k\in[b]}$ and $\{\tilde{s}_k^{q,j}\}_{k\in[b]}$ such that $s_k^{q,j} \oplus \tilde{s}_k^{q,j} = r^{q,j}_{\pi_j^{-1}(k)}$. This step hides the correspondence between bins and labels, which is later used to support private counting and filtering.
- $\mathcal{F}_{\mathbf{BEtORG}}$ **+ package&send (chained filtering).** Using the labels produced above, parties perform a chained interaction from $\mathscr{P}_1 \to \mathscr{P}_2 \to \cdots \to \mathscr{P}_t$. In each hop, items that are determined to be already present are replaced by random paddings, while the remaining items are kept (in encrypted form) and forwarded. At the end of the chain, $\mathscr{P}_t$ obtains EncSet, which consists of encrypted union candidates plus padding ciphertexts. We emphasize that set items remain protected under TR-PKE (possibly re-randomized), and only masked/intermediate payloads are carried during interaction.

– *Joint Decryption + Shuffle.* After $\mathscr{P}_t$ obtains EncSet, it re-randomizes the ciphertexts and forwards the set to $\mathscr{P}_1$. For $i \in [t-1]$, $\mathscr{P}_i$ partially decrypts and shuffles the set using sk_i, and forwards the result to $\mathscr{P}_{i+1}$. Finally, $\mathscr{P}_t$ applies sk_t to complete decryption and recovers a shuffled plaintext multiset containing

$$(\mathsf{XC}_1 \cup \cdots \cup \mathsf{XC}_{t-1}) \setminus \mathsf{XC}_t \quad \text{and} \quad (t-1)n - \left|(\mathsf{XC}_1 \cup \cdots \cup \mathsf{XC}_{t-1}) \setminus \mathsf{XC}_t\right| \text{ random values.}$$

The leader filters plaintexts by checking the format $H(x)||x$, discards random values, and unions the remaining items with X_t to output $\bigcup_{i=1}^{t} X_i$.

Overview of Our Leaderless MPSU

Figure 3(b) illustrates the overall framework of our $\Pi_{\text{MPSU}}^{\text{leaderless}}$ protocol, which extends $\Pi_{\text{MPSU}}^{\text{one-leader}}$ so that every party learns the union. The preparation phase is identical to that of $\Pi_{\text{MPSU}}^{\text{one-leader}}$. The interaction phase repeats the one-leader interaction t times, rotating the role of the output recipient. We provide a technical overview of our $\Pi_{\text{MPSU}}^{\text{leaderless}}$ below.

- **Recap: one-leader interaction.** In $\Pi_{\text{MPSU}}^{\text{one-leader}}$, the core module $\Pi_{\text{MPSU}}^{\text{core}}$ produces for the designated leader $\mathscr{P}_t$ an encrypted padded set consisting of $\mathsf{ReRand}\big((\mathsf{EXC}_1 \cup \cdots \cup \mathsf{EXC}_{t-1}) \setminus \mathsf{EXC}_t\big)$ and $(t-1)n - \big|(\mathsf{EXC}_1 \cup \cdots \cup \mathsf{EXC}_{t-1}) \setminus \mathsf{EXC}_t\big|$ encryptions of random values. The parties then run Joint Decryption + Shuffle, after which $\mathscr{P}_t$ filters padding (via the format $H(x)||x$) and outputs the union.
- **Leader rotation in the leaderless setting.** To ensure that every party obtains the union, $\Pi_{\text{MPSU}}^{\text{leaderless}}$ executes the core module $\Pi_{\text{MPSU}}^{\text{core}}$ for each output recipient $\mathscr{P}_j$ with $j \in [t]$. In the j-th invocation, $\mathscr{P}_j$ plays the role of the leader (i.e., it

ARTICLE IN PRESS

takes the place of $\mathscr{P}_t$ in the one-leader core), while the remaining parties take the other roles in cyclic order. At the end of this invocation, $\mathscr{P}_j$ obtains an encrypted multiset

$$\mathsf{EncSet}_j = \mathsf{ReRand}\left(\left(\bigcup_{i\in[t]\setminus\{j\}} \mathsf{EXC}_i\right)\setminus \mathsf{EXC}_j\right) \cup \mathsf{Pad}_j,$$

where Pad_j contains $(t-1)n - \left|\left(\bigcup_{i\in[t]\setminus\{j\}} \mathsf{EXC}_i\right)\setminus \mathsf{EXC}_j\right|$ encryptions of random values. The parties then execute Joint Decryption + Shuffle on EncSet_j, and $\mathscr{P}_j$ removes padding and unions the remaining items with X_j to obtain $\bigcup_{i=1}^{t} X_i$.

- **Joint Decryption + Shuffle across all invocations.** The t decryption-and-shuffle procedures (one for each EncSet_j) follow the same cyclic pattern. For each $j \in [t]$, the ciphertext set is passed along $\mathscr{P}_{j+1} \to \mathscr{P}_{j+2} \to \cdots \to \mathscr{P}_t \to \mathscr{P}_1 \to \cdots \mathscr{P}_{j-1} \to \mathscr{P}_j$, where each party partially decrypts and applies a fresh random shuffle before forwarding. These t chains can be interleaved (or pipelined) during execution, so all parties obtain the union upon completion. The resulting outputs contain the same items but may appear in different random orders due to shuffling.

Our New MPSU protocol

In this section, we first formalize the ideal functionality of BEtORG and then present a concrete construction. We subsequently provide the full protocol descriptions of our $\Pi_{\text{MPSU}}^{\text{one-leader}}$ and $\Pi_{\text{MPSU}}^{\text{leaderless}}$ protocols.

New Building Block

BEtORG is defined by the ideal functionality $\mathscr{F}_{\text{BEtORG}}$ in Functionality 6. The sender inputs $X = \{x_i\}_{i\in[n]} \in (\{0,1\}^\ell)^n$ and the receiver inputs $Y = \{y_i\}_{i\in[n]} \in (\{0,1\}^\ell)^n$. The functionality samples independent random pairs $(a_i, \tilde{a}_i) \xleftarrow{\$} (\{0,1\}^\ell)^2$ for all $i \in [n]$, outputs $\{(a_i,\tilde{a}_i)\}_{i\in[n]}$ to the sender, and outputs $\{b_i\}_{i\in[n]}$ to the receiver where

$$b_i = \begin{cases} \tilde{a}_i, & \text{if } x_i = y_i, \\ a_i, & \text{otherwise.} \end{cases}$$

ARTICLE IN PRESS

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

Parameters:

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

Functionality:

1. Wait for input set $\{x_1,\cdots,x_n\} \in (\{0,1\}^\ell)^n$ from the sender $\mathscr{S}$;
2. Wait for input set $\{y_1,\cdots,y_n\} \in (\{0,1\}^\ell)^n$ from the receiver $\mathscr{R}$;
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 $\mathscr{S}$, and output $\{b_i\}_{i\in[n]}$ to the receiver $\mathscr{R}$.

Functionality 6. Ideal Functionality for Batched Equality-tested Oblivious Randomness Generation

We present a concrete construction Π_{BEtORG} in Protocol 1, built from $\mathscr{F}_{\text{PEqT}}$ and $\mathscr{F}_{\binom{2}{1}\text{-OT}}$. A detailed step-by-step description was already provided in our technical overview; we therefore omit it here. We show the security of Π_{BEtORG} in Theorem 1 as below.

Theorem 1. *The protocol in Protocol 1 securely computes $\mathscr{F}_{BEtORG}$ in Functionality 6 against semi-honest adversaries in the ($\mathscr{F}_{PEqT}$ and $\mathscr{F}_{\binom{2}{1}\text{-}OT}$)-hybrid model.*

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

ARTICLE IN PRESS

Protocol Π_{BEtORG}

Inputs:

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

Protocol:

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

Protocol 1. Our Batched Equality-tested Oblivious Randomness Generation Protocol

Corrupted $\mathscr{S}$: The simulator Sim simulates a real execution in which $\mathscr{S}$ is corrupted. Since $\mathscr{A}$ is semi-honest, Sim can directly obtain the input set $\{x_1, \cdots, x_n\}$ of $\mathscr{S}$ and externally send it to $\mathscr{F}_{\text{BEtORG}}$. Upon receiving x_i from $\mathscr{A}$, 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 $\mathscr{A}$, 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 $\mathscr{S}$ through the use of the following hybrids:

Hybrid **0** $\mathscr{S}$'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 Π_{PEqT} simulator to generate the simulated view for $\mathscr{S}$. 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 $\Pi_{\binom{2}{1}\text{-OTe}}$ to generate the simulated view for $\mathscr{S}$. 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 hybrid is the view output by Sim.

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

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

Hybrid **0** $\mathscr{R}$'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 Π_{PEqT} simulator to generate the simulated view for $\mathscr{R}$. 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 $\{r_{\tilde{e}_1}, \cdots, r_{\tilde{e}_n}\}$ output by $\mathscr{F}_{\text{BEtORG}}$, and Sim invokes the $\Pi_{\binom{2}{1}\text{-OTe}}$ simulator to generate the simulated view for $\mathscr{R}$. Regardless of whether $r_{\tilde{e}_i}$ is generated by $\Pi_{\binom{2}{1}\text{-OTe}}$ or $\mathscr{F}_{\text{BEtORG}}$, 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 $\Pi_{\binom{2}{1}\text{-OTe}}$.

□

ARTICLE IN PRESS

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

Parameters:

- Hashing functions $h_1, h_2, \cdots, h_\beta$: $\{0,1\}^{\ell_1} \to [b]$;
- A simple hashing table based on $h_1, h_2, \cdots, h_b$, with $b = \varepsilon \cdot n$ bins and bin size $m = O(\log(\beta n))$;
- A cuckoo hashing table based on $h_1, h_2, \cdots, h_\beta$, with $b = \varepsilon \cdot n$ bins and no stash;
- A collusion-against hash function $H(x) : \{0,1\}^* \to \{0,1\}^\ell$, where $\ell = \lambda + \log(tn)$;
- A TR-PKE schem $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathit{PartDec}, \mathsf{ReRand})$.

Inputs:

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

Preparation:

- Each $\mathscr{P}_i$ $(i \in [t])$ jointly run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(\mathsf{pp})$ to obtain their respective keypair $(\mathsf{pk}, \mathsf{sk}_i)$;
- Each $\mathscr{P}_i$ $(i \in [t])$ 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 $\mathscr{P}_i$ $(i \in [t])$ respectively constructs EXC_i, s.t. $\mathsf{EXC}_i[j] = \mathsf{Enc}(\mathsf{pk}, H(\mathsf{XC}_i[j]) || \mathsf{XC}_i[j])$.

Interaction:

1. $\mathscr{P}_1, \cdots, \mathscr{P}_t$ jointly execute the $\Pi_{\text{MPSU}}^{\text{core}}$, and then $\mathscr{P}_t$ obtains the ciphertexts $\{c_1^t, \cdots, c_{(t-1)b}^t\}$. $\mathscr{P}_t$ sends $\{c_1^t, \cdots, c_{(t-1)b}^t\}$ to $\mathscr{P}_1$;
2. For $k \in [t-1]$: $\mathscr{P}_k$ decrypts and shuffles $\{c_1^{k-1}, \cdots, c_{(t-1)b}^{k-1}\}$ obtaining $\{c_1^k, \cdots, c_{(t-1)b}^k\}$, where $c^0 = c^t$; $\mathscr{P}_k$ sends $\{c_1^k, \cdots, c_{(t-1)b}^k\}$ to $\mathscr{P}_{k+1}$;
3. $\mathscr{P}_t$ constructs a empty set Z_t and decrypts $\{c_1^{t-1}, \cdots, c_{(t-1)b}^{t-1}\}$ obtaining $\{m_1, \cdots, m_{(t-1)b}\}$. For $k \in [(t-1)b]$: $\mathscr{P}_t$ 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, $\mathscr{P}_t$ obtains $Z_t = Z_t \cup X_t$.

Protocol 2. Our One-leader Multi-party Private Set Union Protocol $\Pi_{\text{MPSU}}^{\text{one-leader}}$

ARTICLE IN PRESS

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

In this section, we present the full protocol $\Pi_{\text{MPSU}}^{\text{one-leader}}$ in Protocol 2. Its core component, $\Pi_{\text{MPSU}}^{\text{core}}$, is given in Protocol 3. The preparation phase and the Joint Decryption + Shuffle procedure of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ were described in detail in our technical overview. We now describe the core protocol $\Pi_{\text{MPSU}}^{\text{core}}$ in detail.

- For each $j \in [t]$, $\mathscr{P}_j$ randomly selects string groups for its interactions with earlier indices: it samples $j-1$ string groups $\{r_k^{j,1}\}_{k\in[b]}$, $\{r_k^{j,2}\}_{k\in[b]}$, …, $\{r_k^{j,j-1}\}_{k\in[b]}$. $\mathscr{P}_j$ then interacts with each subsequent party $\mathscr{P}_q$ for $q \in \{j+1, \ldots, t\}$ via $\mathscr{F}_{\text{OPPRF}}$ as follows. $\mathscr{P}_j$ acts as the receiver and inputs $\{\mathsf{XC}_j[k]\}_{k\in[b]}$. $\mathscr{P}_q$ acts as the sender and inputs the pair set $\{(\mathsf{XS}_q[k], r_k^{q,j})\}_{k\in[b]}$, where $\mathsf{XS}_q[k]$ contains m items. At a result, $\mathscr{P}_q$ obtains $\{w_k^{q,j}\}_{k\in[b]}$ such that

$$w_k^{q,j} = \begin{cases} r_k^{q,j}, & \text{if } \mathsf{XC}_j[k] \in \mathsf{XS}_q[k], \\ \text{pseudorandom}, & \text{otherwise.} \end{cases}$$

  Intuitively, after hashing, any common item that lands in the same bin index induces the same label $r_k^{q,j}$ on both sides, while non-matching bins yield random-looking outputs.

- For each $j \in [t]$, $\mathscr{P}_j$ samples a random permutation $\pi_j : [b] \to [b]$ and invokes $\mathscr{F}_{\text{PS}}$ with each $\mathscr{P}_q$ for $q \in \{j+1, \ldots, t\}$. In this call, $\mathscr{P}_j$ acts as the sender and inputs π_j, while $\mathscr{P}_q$ acts as the receiver and inputs $\{r_1^{q,j}, \ldots, r_b^{q,j}\}$. The functionality outputs to $\mathscr{P}_j$ $\{s_k^{q,j}\}_{k\in[b]}$ and to $\mathscr{P}_q$ $\{\tilde{s}_k^{q,j}\}_{k\in[b]}$ such that

$$s_k^{q,j} \oplus \tilde{s}_k^{q,j} = r_{\pi_j^{-1}(k)}^{q,j}.$$

  Intuitively, this step applies π_j to $\mathscr{P}_q$'s labels and secret-shares the permuted labels between $\mathscr{P}_j$ and $\mathscr{P}_q$, thereby hiding the correspondence between bin indices and label values.

ARTICLE IN PRESS

1. Each $\mathscr{P}_j$ ($j \in [t]$) randomly selects j-1 string group $\{r_k^{j,1}\}, \cdots, \{r_k^{j,j-1}\} \xleftarrow{\$} (\{0,1\}^{\ell_2})^b$, where $k \in [b]$; Then each $\mathscr{P}_j$ ($j \in [t]$) severally invokes $\mathscr{F}_{\text{OPPRF}}$ with all parties in $\mathscr{I} = \{\mathscr{P}_{j+1}, \cdots, \mathscr{P}_t\}$ in parallel:

   a) $\mathscr{P}_q$ acts as sender with input $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), \cdots, (\mathsf{XS}_q[k][m], r_k^{q,j})\}$, and $\mathscr{P}_j$ acts as receiver with input $\{\mathsf{XC}_j[k]\}$, where $k \in [b]$ and $q \in \{j+1, \cdots, t\}$;
   b) $\mathscr{P}_j$ obtains $(\mathsf{hint}^{q,j}, \{w_1^{q,j}, \cdots, w_b^{q,j}\})$ and $\mathscr{P}_q$ obtains $(\mathsf{hint}^{q,j}, \{\mathscr{K}_1^{q,j}, \cdots, \mathscr{K}_b^{q,j}\})$, where $w_k^{q,j} = F(\mathscr{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k])$. Specifically, if $\mathsf{XC}_j[k] \in \mathsf{XS}_q[k]$, then $F(\mathscr{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k]) = r_k^{q,j}$; otherwise, $F(\mathscr{K}_k^{q,j}, \mathsf{hint}^{q,j}, \mathsf{XC}_j[k]) \neq r_k^{q,j}$;

2. Each $\mathscr{P}_j$ ($j \in [t]$) randomly selects a permutation function $\pi_j : [b] \to [b]$; Then each $\mathscr{P}_j$ ($j \in [t]$) severally invokes $\mathscr{F}_{\text{PS}}$ with all parties in $\mathscr{I} = \{\mathscr{P}_{j+1}, \cdots, \mathscr{P}_t\}$ in parallel:

   a) $\mathscr{P}_j$ acts as sender with input π_j, and $\mathscr{P}_q$ acts as receiver with input $\{r_1^{q,j}, \cdots, r_b^{q,j}\}$, where $q \in \{j+1, \cdots, t\}$;
   b) $\mathscr{P}_j$ obtains $\{s_1^{q,j}, \cdots, s_b^{q,j}\}$ and $\mathscr{P}_q$ obtains $\{\tilde{s}_1^{q,j}, \cdots, \tilde{s}_b^{q,j}\}$, where $s_k^{q,j} \oplus \tilde{s}_k^{q,j} = r_{\pi_j^{-1}(k)}^{q,j}$ and $k \in [b]$;

3. For $j \in [t-1]$, t parties sequentially execute the following processes:

   a) $\mathscr{P}_j$ invokes $\mathscr{F}_{\text{BEtORG}}$ with $\mathscr{P}_{j+1}$:
      – $\mathscr{P}_j$ acts as sender with input $\{\{sw_k^{j+1,q}\}_{k\in[b]}\}_{q\in[j]}$, where $sw_k^{j+1,q} = s_k^{j+1,q} \oplus w_{\pi_q^{-1}(k)}^{j+1,q} \oplus_{p=q+1}^{j\geq p+1} (a_k^{p,q} \oplus b_k^{p,q})$, and $\mathscr{P}_{j+1}$ acts as receiver with input $\{\{\tilde{s}_k^{j+1,q}\}_{k\in[b]}\}_{q\in[j]}$;
      – $\mathscr{P}_j$ obtains $\{\{(a_k^{j+1,q}, \tilde{a}_k^{j+1,q})\}_{k\in[b]}\}_{q\in[j]}$ and $\mathscr{P}_{j+1}$ 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 $b_k^{j+1,q} = \tilde{a}_k^{j+1,q}$; otherwise $b_k^{j+1,q} = a_k^{j+1,q}$;
   b) $\mathscr{P}_j$ sends $\{\{A_k^{j,q}\}_{k\in[b]}\}_{q\in[j]}$ and $\{\{\{s\tilde{w}_k^{l,q}\}_{k\in[b]}\}_{q\in[j]}\}_{l\in\{j+2,\cdots,t\}}$ to $\mathscr{P}_{j+1}$, where

$$A_k^{q,q} = \{\mathsf{EXC}_q[\pi_q^{-1}(k)] \oplus a_k^{q+1,q}\}$$
$$A_k^{j,q} = \{\mathsf{ReRand}(A_k^{j-1,q} \oplus b_k^{j,q}) \oplus a_k^{j+1,q}\}$$
$$s\tilde{w}_k^{l,q} = s_k^{l,q} \oplus w_{\pi_q^{-1}(k)}^{l,q} \oplus_{p=q+1}^{j\geq q+1} (a_k^{p,q} \oplus b_k^{p,q}) \oplus a_k^{j+1,p}.$$

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

ARTICLE IN PRESS

Protocol 3. The core protocol $\Pi_{\text{MPSU}}^{\text{core}}$ of $\Pi_{\text{MPSU}}^{\text{one-leader}}$

- The parties now execute a sequential interaction for $j = 1, \ldots, t-1$ along the chain $\mathscr{P}_1 \to \mathscr{P}_2 \to \cdots \to \mathscr{P}_t$. In round j, parties $\mathscr{P}_j$ and $\mathscr{P}_{j+1}$ use $\mathscr{F}_{\text{BEtORG}}$ to privately decide, for each bin, whether the item forwarded from earlier parties should be kept or replaced by random padding, while maintaining all set items under TR-PKE encryption.

  – *BEtORG-driven masking.* For each $q \in [j]$ and each bin $k \in [b]$, $\mathscr{P}_j$ forms a masked comparison value $sw_k^{j+1,q}$ and $\mathscr{P}_{j+1}$ holds the corresponding share $\tilde{s}_k^{j+1,q}$. The parties invoke $\mathscr{F}_{\text{BEtORG}}$ on these vectors. As a result, $\mathscr{P}_j$ obtains pairs $(a_k^{j+1,q}, \tilde{a}_k^{j+1,q})$ and $\mathscr{P}_{j+1}$ obtains $b_k^{j+1,q}$ for all k. BEtORG guarantees that $b_k^{j+1,q}$ equals one of the two values depending on whether $sw_k^{j+1,q} = \tilde{s}_k^{j+1,q}$.

  – *Packaging and forwarding.* $\mathscr{P}_j$ then re-randomizes and packages the encrypted payloads, and forwards them to $\mathscr{P}_{j+1}$. Concretely, it updates the ciphertext accumulators $\{A_k^{q,j}\}$ (which carry encrypted items masked by the one-time pads $a_k^{\cdot,\cdot}$ and $b_k^{\cdot,\cdot}$) and sends the updated values together with the masked labels needed for the next hop. The last hop ($j = t-1$) produces a ciphertext multiset $\{c_1, \ldots, c_{(t-1)n}\}$ at $\mathscr{P}_t$, where each ciphertext is a re-randomized TR-PKE encryption of either a genuine set item or a random padding value.

By **Remark 1**, if all masks generated in Step 3 satisfy $a_k^{j+1,q} \oplus b_k^{j+1,q} = 0$, then the corresponding ciphertext in $\{c_1, \ldots, c_{(t-1)n}\}$

ARTICLE IN PRESS

is a sequence of re-randomizations of an encryption of a well-formed plaintext $m = H(x)||x$ originating from one of $\mathsf{EXC}_1, \ldots, \mathsf{EXC}_{t-1}$. Hence, the parties can jointly decrypt and recover m. In contrast, if for some hop the mask satisfies $a_k^{j+1,q} \oplus b_k^{j+1,q} \neq 0$, then the plaintext becomes malformed after unmasking and will not pass the format check, so it is discarded. This mechanism ensures that items appearing multiple times across parties are retained only once, while the remaining duplicates are converted into random padding.

Security. We state the security of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ in Theorem 2.

Theorem 2. *The protocol in Protocol 2 securely computes $\mathcal{F}_{MPSU}$ of one-leader scenario in Functionality 5 against any number of corrupt, colluding, semi-honest parties in the ($\mathcal{F}_{OPPRF}$, $\mathcal{F}_{PS}$, and $\mathcal{F}_{BEtORG}$)-hybrid model if the TR-PKE scheme is IND-CPA security.*

Proof. All interactions between parties in $\Pi_{\text{MPSU}}^{\text{one-leader}}$ occur within interaction phase, that is $\Pi_{\text{MPSU}}^{\text{core}}$ and joint decryption + shuffling operation. Assuming $\mathscr{P}_i$ (where $i \in [t]$) is honest and $I = \{(\mathscr{P}_1, \cdots, \mathscr{P}_t) \backslash \mathscr{P}_i\}$ is a colluding coalition. To satisfy the security of Theorem 2, we need to prove that $\mathscr{P}_i$ can resist coalition attacks of I when it respectively assumes the roles $\mathscr{P}_1, \cdots, \mathscr{P}_t$. Specifically, for any adversary $\mathscr{A}$, 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 indistinguishable from that in the real-world in any PPT environment. Since $\mathscr{P}_1, \cdots, \mathscr{P}_{t-1}$ perform the same role in the protocol, we will prove in two different cases: when $\mathscr{P}_i$ acts as the leader $\mathscr{P}_t$, when $\mathscr{P}_i$ acts as the other party roles.

(1) $\mathscr{P}_i$ acts as the leader $\mathscr{P}_t$ and I acts as the $\{\mathscr{P}_1, \cdots, \mathscr{P}_{t-1}\}$. The simulator Sim simulates a real execution. Since $\mathscr{A}$ is semi-honest, Sim can obtain the input $I_X = \{X_1, \cdots, X_{t-1}\}$ of I directly, and externally send I_X to $\mathscr{F}_{\text{MPSU}}$. When receiving $\{\mathsf{XC}_j[1], \cdots, \mathsf{XC}_j[b]\}$ ($j \in [t-1]$) from $\mathscr{A}$, Sim randomly selects $(\mathsf{hint}^{t,j}, \{w_k^{t,j}\})$ (where $k \in [b]$), and simulates the execution of Π_{OPPRF}, where $\mathscr{P}_j$ acts as the receiver and $\mathscr{P}_t$ acts as the sender. Once receiving the permutation function π_j from $\mathscr{A}$, Sim checks if it is a permutation of b items. If yes, Sim randomly selects $\{s_1^{t,j}, \cdots, s_b^{t,j}\}$, where $s_k^{t,j} \in \{0,1\}^{\ell_2}$, and simulates the execution of Π_{PS}, where $\mathscr{P}_j$ acts as the sender and $\mathscr{P}_t$ acts as the receiver. After obtaining the $\{\{sw_k^{t,q}\}_{k\in[b]}\}_{q\in[t-1]}$ from $\mathscr{A}$, 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 $\mathscr{P}_{t-1}$ acts as the sender and $\mathscr{P}_t$ acts as the receiver. Sim samples random values and encrypts them to generate $\left\{\{A_k^{t-1,q}\}_{k\in[b]}\right\}_{q\in[t-1]}$. After receiving Z_t from $\mathscr{F}_{\text{MPSU}}$, Sim computes $M = Z_t \setminus X_t$. Then, Sim pads M with dummy value d until $|M| = (t-1)b$, and encrypts each item of M to obtain a ciphertext set $\tilde{M}$. Finally, Sim random shuffles $\tilde{M}$ and sends it to $\mathscr{A}$.

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 $(\mathsf{hint}^{t,j}, \{w_1^{t,j}, \cdots, w_b^{t,j}\})$, chosen by Sim, when $\mathscr{P}_j$ (where $j \in [t-1]$) acts as the receiver and jointly invokes with $\mathscr{P}_t$. Afterwords, Sim invokes the Π_{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 $\mathscr{P}_1, \cdots, \mathscr{P}_{t-1}$ and $\mathscr{P}_t$ 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 $\{s_1^{t,j}, \cdots, s_b^{t,j}\}$, chosen by Sim, when $\mathscr{P}_j$ (where $j \in [t-1]$) acts as the sender and jointly calls with $\mathscr{P}_t$. Afterwords, Sim invokes the Π_{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 $\mathscr{P}_1, \cdots, \mathscr{P}_{t-1}$ and $\mathscr{P}_t$ 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 $\{(a_1^{t,q}, \tilde{a}_1^{t,q}), \cdots, (a_b^{t,q}, \tilde{a}_b^{t,q})\}$, chosen by Sim, when $\mathscr{P}_{t-1}$ acts as the sender and jointly calls with $\mathscr{P}_t$. Afterwords, Sim invokes the Π_{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 $\left\{\{A_k^{t-1,q}\}_{k\in[b]}\right\}_{q\in[t-1]}$ is computed by Sim. The computational indistinguishability between the simulated view and the real protocol view is ensured by the IND-CPA security of TR-PKE. This hybrid is the view output by the simulator Sim.

(2) $\mathscr{P}_i$ acts as the $\mathscr{P}_j$ (where $j \in [t-1]$) and I acts as the $\{\mathscr{P}_1, \cdots, \mathscr{P}_t\} \setminus \mathscr{P}_j$. The simulator Sim simulates a real execution. Since $\mathscr{A}$ is semi-honest, Sim can obtain the input $I_X = \{X_1, \cdots, X_t\} \setminus X_j$ of I directly, and externally send I_X to $\mathscr{F}_{\text{MPSU}}$. When obtaining $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), \cdots, (\mathsf{XS}_q[k][m], r_k^{q,j})\}$ (where $q \in \{j+1, \cdots, t\}$ and $k \in [b]$) and $\{\mathsf{XC}_p[1], \cdots, \mathsf{XC}_p[b]\}$ (where

ARTICLE IN PRESS

$p \in [j-1]$), Sim randomly selects $(\mathsf{hint}^{q,j}, \{\mathscr{K}_1^{q,j}, \cdots, \mathscr{K}_b^{q,j}\}$ and $(\mathsf{hint}^{j,p}, \{w_1^{j,p}, \cdots, w_b^{j,p}\})$, respectively. Following, Sim simulates the execution of Π_{OPPRF}. Specifically, for $\{(\mathsf{XS}_q[k][1], r_k^{q,j}), \cdots, (\mathsf{XS}_q[k][m], r_k^{q,j})\}$, Sim simulates the scenario where $\mathscr{P}_q$ acts as the sender and $\mathscr{P}_j$ acts as the receiver. For $\{\mathsf{XC}_p[1], \cdots, \mathsf{XC}_p[b]\}$, Sim simulates the scenario where $\mathscr{P}_p$ acts as the receiver and $\mathscr{P}_j$ acts as the sender. Once obtaining π_q (where $q \in \{j+1, \cdots, t\}$) and $\{r_1^{j,p}, \cdots, r_b^{j,p}\}$ (where $p \in [j-1]$), Sim randomly selects $\{s_1^{q,j}, \cdots, s_b^{q,j}\}$ and $\{\tilde{s}_1^{j,p}, \cdots, \tilde{s}_b^{j,p}\}$, respectively. Then, Sim simulates the execution of Π_{PS}. Specifically, for π_q, Sim simulates the scenario where $\mathscr{P}_q$ acts as the sender and $\mathscr{P}_j$ acts as the receiver. For $\{r_1^{j,p}, \cdots, r_b^{j,p}\}$, Sim simulates the scenario where $\mathscr{P}_p$ acts as the receiver and $\mathscr{P}_j$ acts as the sender. Upon obtaining $\{\{sw_k^{j,q}\}_{k\in[b]}\}_{q\in[j-1]}$ from $\mathscr{A}$, Sim randomly selects $\{(a_1^{j,q}, \tilde{a}_1^{j,q}), \cdots, (a_b^{j,q}, \tilde{a}_b^{j,q})\}$, and simulates the Π_{BEtORG}, where $\mathscr{P}_{j-1}$ acts as the sender and $\mathscr{P}_j$ acts as the receiver. Upon obtaining $\{\tilde{s}_1^{j+1,j}, \cdots, \tilde{s}_b^{j+1,j}\}$ from $\mathscr{A}$, Sim randomly selects $\{a_1^{j+1,j}, \cdots, a_b^{j+1,j}\}$, and simulates the Π_{BEtORG}, where $\mathscr{P}_j$ acts as the sender and $\mathscr{P}_{j+1}$ acts as the receiver. Sim samples random values and encrypts them to generate $\left\{\{A_k^{j-1,q}\}_{k\in[b]}\right\}_{q\in[j-1]}$. Finally, Sim samples random values and encrypts them to generate $\{c_1^{j-1}, \cdots, c_{(t-1)b}^{j-1}\}$.

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 $\mathscr{P}_{j+1}, \cdots, \mathscr{P}_t$ act as the sender and jointly invoke Π_{OPPRF} with $\mathscr{P}_j$, the output is replaced by $(\mathsf{hint}^{q,j}, \{\mathscr{K}_1^{q,j}, \ldots, \mathscr{K}_b^{q,j}\})$ (where $q \in \{j+1, \cdots, t\}$), chosen by Sim. Similarly, when $\mathscr{P}_1, \ldots, \mathscr{P}_{j-1}$ act as the receiver and jointly invoke Π_{OPPRF} with $\mathscr{P}_j$, 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 Π_{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 $\mathscr{P}_{j+1}, \cdot, \mathscr{P}_t$ act as the receiver and jointly invoke Π_{PS} with $\mathscr{P}_j$, the output is replaced by $\{\tilde{s}_1^{q,j}, \cdots, \tilde{s}_b^{q,j}\}$ (where $q \in \{j+1, \cdots, t\}$), chosen by Sim. Similarly, when $\mathscr{P}_1, \ldots, \mathscr{P}_{j-1}$ act as the sender and jointly invoke Π_{PS} with $\mathscr{P}_j$, the output is replaced by $\{s_1^{j,p}, \cdots, s_b^{j,p}\}$ (where $p \in [j-1]$), also chosen by Sim. Then, Sim invokes the Π_{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 $\mathscr{P}_{j-1}$ acts as the sender and jointly invokes Π_{BEtORG} with $\mathscr{P}_j$, 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 $\mathscr{F}_{\mathrm{BEtORG}}$ simulator to simulate the invocation and generate a simulated view for I. Additionally, when $\mathscr{P}_{j+1}$ acts as the receiver and jointly invokes Π_{BEtORG} with $\mathscr{P}_j$, 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 Π_{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 $\left\{\{A_k^{j-1,q}\}_{k\in[b]}\right\}_{q\in[j-1]}$ is computed by Sim. The computational indistinguishability between the simulated view and the real protocol view is ensured by the IND-CPA security of TR-PKE.

Hybrid **5** This hybrid is identical to Hybrid 4, except that $\{c_1^{j-1}, \cdots, c_{(t-1)b}^{j-1}\}$ is computed by Sim. The IND-CPA security of TR-PKE 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.

□

New MPSU Protocol $\Pi_{\mathbf{MPSU}}^{\mathbf{leaderless}}$

The $\Pi_{\mathrm{MPSU}}^{\mathrm{leaderless}}$ protocol realizes the leaderless MPSU. It is an extension of the $\Pi_{\mathrm{MPSU}}^{\text{one-leader}}$ protocol, with the same preparation phase. The distinction lies in the extended interaction phase, as detailed in Protocol 4.

Since the preparation phase of the $\Pi_{\mathrm{MPSU}}^{\mathrm{leaderless}}$ is identical to that of the $\Pi_{\mathrm{MPSU}}^{\text{one-leader}}$, and its interaction phase can be regarded as t independent executions of the $\Pi_{\mathrm{MPSU}}^{\text{one-leader}}$'s interaction phase, proving the security of the $\Pi_{\mathrm{MPSU}}^{\text{one-leader}}$ is sufficient to establish the security of the $\Pi_{\mathrm{MPSU}}^{\mathrm{leaderless}}$. The security of the $\Pi_{\mathrm{MPSU}}^{\text{one-leader}}$ have been proved previously. Therefore, we can directly derive the following theorem.

Theorem 4.3. *The protocol in Protocol 4 securely computes $\mathscr{F}_{MPSU}$ of leaderless scenario in Functionality 5 against any number of corrupt, colluding, semi-honest parties in the ($\mathscr{F}_{OPPRF}$, $\mathscr{F}_{PS}$, and $\mathscr{F}_{BEtORG}$)-hybrid model if the TR-PKE scheme is IND-CPA security.*

ARTICLE IN PRESS

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

Parameters, **Inputs**, and **Preparations** are as in Protocol 2.
Interaction:

1. All parties execute $\Pi_{\text{MPSU}}^{\text{core}}$ for t rounds. In the i-th round (where $i \in [t]$), $\mathscr{P}_i, \mathscr{P}_{i+1}, \cdots, \mathscr{P}_t, \mathscr{P}_1, \cdots, \mathscr{P}_{i-1}$ sequentially take on the roles of $\mathscr{P}_t, \mathscr{P}_1, \ldots, \mathscr{P}_{t-1}$ within $\Pi_{\text{MPSU}}^{\text{core}}$. At the end of each round of $\Pi_{\text{MPSU}}^{\text{core}}$, $\mathscr{P}_i$ obtains the output ciphertext set and sends it to $\mathscr{P}_{i+1}$.
2. Execute the following operations in parallel: For $i \in [t]$, $\mathscr{P}_j$ performs decryption and shuffle operations on $\{c_1^{j-1}, \cdots, c_{(t-1)b}^{j-1}\}$, generating a new set $\{c_1^j, \cdots, c_{(t-1)b}^j\}$, 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 $\mathscr{P}_i$ (where $i \in [n]$) constructs a empty set Z_i and decrypts $\{c_1^{i-1}, \cdots, c_{(t-1)b}^{i-1}\}$ obtaining $\{m_1^i, \cdots, m_{(t-1)b}^i\}$. For $k \in [(t-1)b]$: $\mathscr{P}_i$ checks whether $m_k^i = H(s_k^i)||s_k^i$ and $s_k^i \neq d$, if yes, setting $Z_i = Z_i \cup \{s_k^i\}$; Finally, $\mathscr{P}_i$ obtains $Z_i = Z_i \cup X_i$.

Protocol 4. Our Leaderless Multi-party Private Set Union Protocol $\Pi_{\text{MPSU}}^{\text{leaderless}}$ Protoco

Complexity Analysis

For $\Pi_{\text{MPSU}}^{\text{one-leader}}$, after running TR-PKE key generation, the parties preprocess their inputs using β hash functions to build a simple hash table and a cuckoo hash table (and its encrypted version). The cuckoo table has $b = \varepsilon n$ bins, while the simple hash table stores up to bm items in total. Then, the t parties proceed with executing $\Pi_{\text{MPSU}}^{\text{core}}$ and Joint Decryption + shuffle, involving the execution of Π_{OPPRF}, Π_{PS}, Π_{BEtORG}, as well as encryption and decryption operations of the TR-PKE.

- Π_{OPPRF}**.** We use the protocol of[38], which costs $O(n\log(n))$ computation and $O(n)$ communication per invocation. Since $\Pi_{\text{MPSU}}^{\text{core}}$ invokes it $\sum_{i=1}^{t-1} i = O(t^2)$ times, the total complexity contributed by Π_{OPPRF} is $O(t^2 n\log(n))$ computation and $O(t^2 n)$ communication.
- Π_{PS}**.** We use the protocol of[39], which costs $O(n\log(n))$ per invocation. Aggregated over $O(t^2)$ invocations in $\Pi_{\text{MPSU}}^{\text{core}}$, this yields $O(t^2 n\log(n))$ computation and $O(t^2 n\log(n))$ communication.
- Π_{BEtORG}**.** Our Π_{BEtORG} consists of Π_{PEqT} and $\Pi_{\binom{2}{1}\text{-OTe}}$, both with linear cost in n. Hence, one BEtORG invocation costs $O(n)$ computation and $O(n)$ communication, and across $O(t^2)$ invocations it contributes $O(t^2 n)$ computation and $O(t^2 n)$ communication.
- In addition, the interaction phase performs $O(t^2 n)$ TR-PKE encryptions and $O(t^2 n)$ TR-PKE decryptions.

Table 3 summarizes the computational and communication complexities of each module in $\Pi_{\text{MPSU}}^{\text{one-leader}}$. Overall, both the total computation and communication of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ are $O(t^2 n\log(n))$.

	$\mathscr{F}_{\text{OPPRF}}$	$\mathscr{F}_{\text{PS}}$	$\mathscr{F}_{\text{BEtORG}}$	Encryption	Decryption
Comp.	$O(t^2 n\log(n))$	$O(t^2 n\log(n))$	$O(t^2 n)$	$O(t^2 n)$	$O(t^2 n)$
Comm.	$O(t^2 n)$	$O(t^2 n\log(n))$	$O(t^2 n)$	$O(t^2 n)$	$O(t^2 n)$

Table 3. Computational and communication complexities of each module in $\Pi_{\text{MPSU}}^{\text{one-leader}}$. Note: Comm./Comp.: Communication /Computational complexity; t: Number of parties; n: Size of sets.

For $\Pi_{\text{MPSU}}^{\text{leaderless}}$, the preparation phase is identical to that of $\Pi_{\text{MPSU}}^{\text{one-leader}}$, while the interaction phase executes the one-leader interaction t times with a rotating output recipient. Hence, the total computation and communication of $\Pi_{\text{MPSU}}^{\text{leaderless}}$ are t times those of $\Pi_{\text{MPSU}}^{\text{one-leader}}$. Moreover, since roles are rotated evenly across the t interaction rounds, each party's computation and communication are of the same as a single interaction of $\Pi_{\text{MPSU}}^{\text{one-leader}}$.

Performance Evaluation

In this section, we experimentally evaluate the performance of our MPSU protocols and compare with state-of-the-art.

ARTICLE IN PRESS

Party Number	Set Size	FF	Preparation	Interaction ($\Pi^{\text{one-leader}}_{\text{MPSU}}$)	Comm. Cost ($\Pi^{\text{one-leader}}_{\text{MPSU}}$)	Interaction ($\Pi^{\text{leaderless}}_{\text{MPSU}}$)	Comm. Cost ($\Pi^{\text{leaderless}}_{\text{MPSU}}$)
3	2^8	1024	0.62	2.06	1	4.62	3
		2048	11.33	6.39	2	9.01	6
	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	2^8	1024	1.03	4.08	2	11.56	8
		2048	12.02	12.77	4	20.56	16
	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	2^8	1024	1.04	10.15	7	41.54	42
		2048	12.26	32.93	10	64.37	60
	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	2^8	1024	1.19	18.86	13	100.23	104
		2048	12.77	59.54	20	144.14	160
	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 preparation and interaction runtime (seconds) and communication cost (MB) of $\Pi^{\text{one-leader}}_{\text{MPSU}}$, as well as the interaction runtime and communication cost of $\Pi^{\text{leaderless}}_{\text{MPSU}}$ (Finite field (FF): 1024-bit and 2048-bit). Each party's set contains of 64-bit items. The decimal portion of the communication cost is disregarded.

ARTICLE IN PRESS

- *Benchmarking Environment.* We conduct experiments on a MacBook Pro running a macOS 14.0, equipped with an Apple M3 Pro chip and 36GB RAM. The protocols are evaluated in WAN setting (100Mbps, 10ms RTT).
- *Implementation Details.* We instantiate TR-PKE using threshold ElGamal over a finite field. We evaluate two parameter sets, using a 1024-bit and a 2048-bit field, which roughly correspond to computational security levels of $\kappa \approx 80$ and $\kappa \approx 116$ bits, respectively. We set the statistical security parameter to $\lambda = 40$. For hashing, we use $\beta = 3$ hash functions to map each input set X_i ($i \in [t]$) into the simple hash table, the cuckoo hash table, and the encrypted cuckoo table. We set the number of bins to $b = \varepsilon n = 1.27n$. Our implementation is in C/C++ and builds on the following libraries.
  - **OPPRF.** We use the BSTY19 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 MS13 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 CGS22 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.

Performance of Our Proposed

We evaluate runtime and communication of our proposed protocols under set sizes $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$ and party counts $t \in \{3,4,6,8\}$. Table 4 reports the runtime of the preparation and interaction phases and the total communication of $\Pi^{\text{one-leader}}_{\text{MPSU}}$.

ARTICLE IN PRESS

It also reports the interaction runtime and communication of $\Pi_{\text{MPSU}}^{\text{leaderless}}$. All experiments are conducted with threshold finite-field ElGamal instantiated over 1024-bit and 2048-bit fields.

- *Effect of field size.* Fixing n and t, moving from a 1024-bit field to a 2048-bit field increases both runtime and communication, due to more expensive public-key group operations and larger ciphertexts. The impact is concentrated in TR-PKE-heavy components, including
  - the preparation phase that builds encrypted cuckoo tables.
  - the interaction step that transmits and rerandomizes TR-PKE ciphertexts (Step 3.b in $\Pi_{\text{MPSU}}^{\text{core}}$).
  - the Joint Decryption + Shuffle step.

  Empirically, wall-clock time is markedly more sensitive than bandwidth to this parameter change, indicating that TR-PKE operations dominate computation. For example, when $(t,n) = (4,2^{16})$, the interaction runtime increases from 582.77s to 2764.91s ($\approx 4.7\times$), while communication increases from 719MB to 1085MB ($\approx 1.5\times$). When $(t,n) = (6,2^{20})$, the interaction runtime increases from 24395.28s to 106843.80s ($\approx 4.4\times$), while communication increases from 31306MB to 45936MB ($\approx 1.5\times$).

- *Leaderless overhead and wall-clock behavior.*
  - $\Pi_{\text{MPSU}}^{\text{leaderless}}$ reuses the same preparation as $\Pi_{\text{MPSU}}^{\text{one-leader}}$ and rotates the output recipient across t interaction invocations, so that every party learns the union directly. This design improves robustness and fairness in decentralized deployments (e.g., it avoids depending on a single leader to disseminate the result), at the cost of higher bandwidth. Accordingly, the total communication of $\Pi_{\text{MPSU}}^{\text{leaderless}}$ is approximately t times that of $\Pi_{\text{MPSU}}^{\text{one-leader}}$, which is consistent with the measurements in Table 4.
  - In contrast, the interaction time grows more slowly than t because the t Joint Decryption + Shuffle procedures can be interleaved across rounds. Thus, their contribution remains close to that of a single execution rather than scaling linearly. A useful approximation is $T_{\text{leaderless}} \approx t \cdot T_{\Pi_{\text{MPSU}}^{\text{core}}} + T_{\text{JDS}}$, where T_{JDS} denotes the time of one Joint Decryption + Shuffle. For example, at $n = 2^{20}$ under the 1024-bit field, the observed ratio $T_{\text{leaderless}}/T_{\text{one-leader}}$ ranges from $1.66\times$ ($t = 3$) to $3.40\times$ ($t = 8$), which is substantially smaller than the corresponding $\approx t$ increase in total communication.

- *Practical operating regime and peak configuration.* Table 4 provides absolute runtimes and bandwidths across a wide range of parameters, which can guide practitioners in choosing security levels and deployment settings. In particular, the largest configuration $(t,n) = (8,2^{20})$ under the 2048-bit field is resource-intensive; we report one-leader results and omit the leaderless result for this extreme point due to testbed limitations.

Party Number	Set Size	Runtime		Comm. Cost	
		Ours	[28]	Ours	[28]
3	2^8	15.26	19.42	1	4
	2^{12}	37.66	67.27	12	61
	2^{16}	555.11	716.79	222	694
	2^{20}	11603.48	13564.51	4043	15427
4	2^8	21.5	31.63	2	9
	2^{12}	61.96	107.04	25	122
	2^{16}	951.13	1209.28	452	1935
	2^{20}	19334.63	23143.25	8209	30956
6	2^8	38.56	59.16	4	22
	2^{12}	125.58	204.56	65	306
	2^{16}	2072.08	2632.3	1169	4853
	2^{20}	35400.21	48033.24	17037	77644
8	2^8	63.24	91.93	9	41
	2^{12}	229.84	347.15	126	572
	2^{16}	3684.01	5132.42	2254	9074
	2^{20}	59733.91	86269.36	32394	145179

Table 5. Performance comparison of the runtime and communication cost of $\Pi_{\text{MPSU}}^{\text{one-leader}}$ with Gao et al.'s protocol (Curve25519-based).

ARTICLE IN PRESS

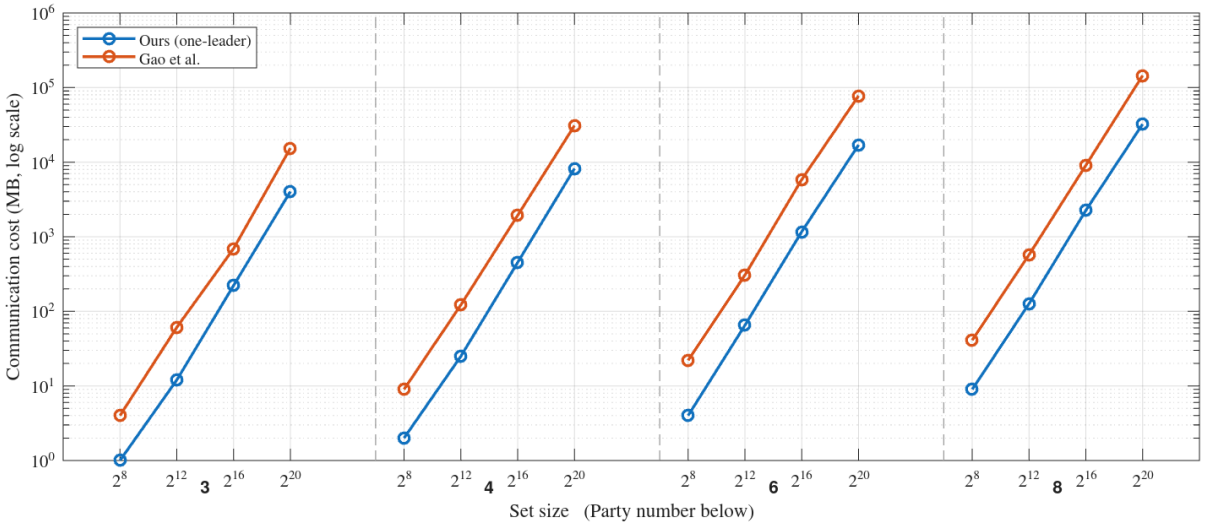


Figure 4. Communication cost scaling for our one-leader MPSU and Gao et al.'s protocol over varying party numbers and set sizes.

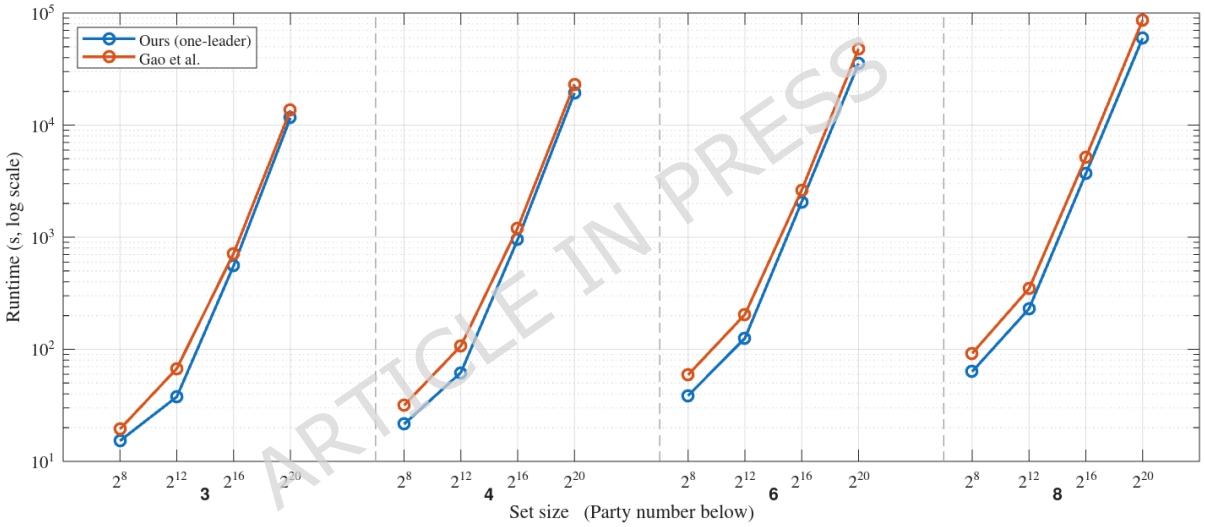


Figure 5. Runtime scaling for our one-leader MPSU and Gao et al.'s protocol over varying party numbers and set sizes.

Comparison with Previous Work

To demonstrate the performance, we benchmark our $\Pi_{\text{MPSU}}^{\text{one-leader}}$ protocol against the state-of-the-art one-leader MPSU protocol of Gao et al.[28].

Our results in Table 4 use threshold ElGamal over a finite field, whereas Gao et al. instantiate TR-PKE with threshold ElGamal over Curve25519, targeting $\approx$ 128-bit security and offering substantially faster group operations. A caveat of the Curve25519 instantiation is that plaintexts must be represented as curve points (or encoded into the curve group), which constrains the input domain and introduces additional encoding considerations. To ensure a fair apples-to-apples comparison, we re-implement our protocol using the same threshold Curve25519-based ElGamal employed by Gao et al., and we evaluate both protocols under this common instantiation. The detailed evaluated result is reported in Table 5.

- *Communication comparison.* Figure 4 shows that $\Pi_{\text{MPSU}}^{\text{one-leader}}$ consistently communicates less than Gao et al.'s protocol across all tested set sizes and party counts. As summarized in Table 5, when n increases from 2^8 to 2^{20}, our protocol reduces communication by approximately 4–5$\times$.

- *Runtime comparison.* Figure 5 shows that $\Pi_{\text{MPSU}}^{\text{one-leader}}$ consistently runs faster than Gao et al.'s protocol, with the gap becoming more pronounced as the number of parties increases. As reported in Table 5, when n increases from 2^8 to 2^{20},

ARTICLE IN PRESS

our protocol achieves a 1.3–1.8× speedup.

Discussion: Threat Model and Toward Malicious Security

Our security analysis follows the standard semi-honest model with maximum collusion, where an adversary may corrupt an arbitrary subset of parties but corrupted parties are assumed to follow the prescribed protocol. This is the same threat model adopted by Gao et al., who also target security against any number of colluding semi-honest parties in the one-leader setting. Accordingly, both their protocol and ours are well suited to consortium-style deployments in which parties are mutually distrustful yet operate under compliance, auditing, or contractual constraints that make deviations unlikely.

In the malicious model, corrupted parties may deviate arbitrarily, e.g., by constructing malformed hash-table encodings, injecting inconsistent ciphertexts, biasing randomness, or returning incorrect partial decryptions and shuffles. Our current proofs are restricted to semi-honest behavior and therefore do not cover such deviations. Nevertheless, the protocol admits a natural hardening strategy that focuses on three aspects: input consistency, correctness of cryptographic operations, and resilience to aborts. A maliciously secure variant can be obtained by combining standard techniques such as the following.

- **Malicious-secure sub-protocols.** Replace the semi-honest sub-protocols with malicious-secure counterparts. Under a modular composition theorem, this yields a protocol secure in the malicious model in the corresponding hybrid setting.
- **Input consistency checks.** Require each party to commit to its hashed representations and prove (e.g., via efficient zero-knowledge proofs or consistency checks) that the encrypted cuckoo table EXC_i is correctly formed from X_i and that rerandomizations are well formed.
- **Verifiable shuffle and threshold decryption.** Strengthen Joint Decryption + Shuffle with verifiability, e.g., by attaching publicly verifiable proofs that each partial decryption is correct and each shuffle is a valid permutation/re-randomization of the input multiset. This prevents a malicious party from biasing or corrupting the output while keeping item origins hidden.
- **Abort and fairness.** Even with verifiable steps, malicious parties may still abort strategically. In practice, abort resilience is often handled at the system level (e.g., retries, accountability/logging, or penalties). Our leaderless construction mitigates reliance on a single output holder, but full fairness under aborts remains a separate goal and would require additional mechanisms.

Conclusion

This paper presents an improved one-leader MPSU protocol that is secure against the maximum semi-honest collusion model and delivers the union only to a designated leader. Our design reduces bandwidth by using SKE for most online payload transport, while reserving TR-PKE for the few steps that require re-randomization and joint threshold decryption. To better support decentralized deployments and avoid reliance on a single output holder, we further extend the construction to a leaderless MPSU protocol in which every party learns the union. Our implementation and experiments show that the proposed protocols are practical and achieve substantial communication savings and runtime improvements compared to prior work while preserving security. As future work, we aim to design more efficient MPSU protocols, such as constructions fully based on SKE while tolerating the maximum semi-honest collusion, as well as MPSU protocols secure against malicious adversaries.

Data availability

The datasets used and analyzed during the present study are available from the corresponding author upon reasonable request.

References

1. Hogan, K. et al. Secure multiparty computation for cooperative cyber risk assessment. *SecDev*, 75-76 (2016).
2. Ramanathan, S., Mirkovic, J. & Yu, M. BLAG: Improving the Accuracy of Blacklists. *NDSS* (2020).
3. Kolesnikov, V., Rosulek, M., Trieu, N. & Wang, X. Scalable Private Set Union from Symmetric-Key Techniques. *ASIACRYPT*, 11922, 636–666 (2019).
4. Nomura, K., Shiraishi, Y., Mohri, M. & Morii, M. Secure association rule mining on vertically partitioned data using private-set intersection. *IEEE Access*, 8, 144458-14467 (2020).
5. Kolesnikov, V., Kumaresan, R., Rosulek, M. & Trieu, N. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. *CCS*, 818-829 (2016).

ARTICLE IN PRESS

6. Pinkas, B., Schneider, T. & Zohner, M. Scalable Private Set Intersection Based on OT Extension. *TOPS*, 21, 1-35 (2018).

7. Pinkas, B., Rosulek, M., Trieu, N. & Yanai, A. SpOT-light: lightweight private set intersection from sparse OT extension. *CRYPTO*, 401-431 (2019).

8. Chase, M. & Miao, P. Private set intersection in the internet setting from lightweight oblivious PRF. *CRYPTO*, 34-63 (2020).

9. Pinkas, B., Rosulek, M., Trieu, N. & Yanai, A. PSI from PaXoS: Fast, Malicious Private Set Intersection. *EUROCRYPT*, 739-767 (2020).

10. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N. & Yanai, A. Oblivious Key-Value Stores and Amplification for Private Set Intersection. *CRYPTO*, 395-425 (2021).

11. Aranha, DF., Lin, CW., Orlandi, C. & SimkinAuthors, M. Laconic private set-intersection from pairings. *CCS*, 111-124 (2022).

12. Raghuraman, S. & Rindal, P. Blazing Fast PSI from Improved OKVS and Subfield VOLE. *CCS*, 2505-2517 (2022).

13. Bui, D. & Couteau, G. Improved private set intersection for sets with small entries. *PKC*, 190-220 (2023).

14. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M. & Trieu, N. Practical multi-party private set intersection from symmetric-key techniques. *CCS*, 1257-1272 (2017).

15. Inbar, R., Omri, E. & Pinkas, B. Efficient scalable multiparty private set-intersection via garbled bloom filters. *SCN*, 235-252 (2018).

16. Kavousi, A., Mohajeri, J. & Salmasizadeh, M. Efficient scalable multi-party private set intersection using oblivious PRF. *STM*, 81-99 (2021).

17. Gordon, S.D., Hazay, C. & Le, P.H. Fully Secure PSI via MPC-in-the-Head. *PETS*, 2022, 291-313 (2022).

18. Ben-Efraim, A., Nissenbaum, O., Omri, E. & Paskin-Cherniavsky, A. Psimple: Practical multiparty maliciously-secure private set intersection. *ASIA CCS*, 1098-1112 (2022).

19. Kissner, L. & Song, D. Privacy-Preserving Set Operations. *CRYPTO*, 241-257 (2005).

20. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S. & Singh, J. Private Set Operations from Oblivious Switching. *PKC*, 591-617 (2021).

21. Jia, Y.X., Sun, S.F., Zhou, H.S., Du, J.J. & Gu, D.W. Shuffle-based Private Set Union: Faster and More Secure. *USENIX Security*, 2947-2964 (2022).

22. Zhang, C., Chen, Y., Liu, W.R., Zhang, M. & Lin, D.D. Linear Private Set Union from Multi-Query Reverse Private Membership Test. *SEC*, 337-354 (2023).

23. Frikken, K. Privacy-Preserving Set Union. *ACNS*, 237-252 (2007).

24. Seo, J.H., Cheon, J.H. & Katz, J. Constant-round multi-party private set union using reversed laurent series. *PKC*, 7293, 398-412 (2012).

25. Gong, X.H., Hua, Q.S. & Jin, H. Nearly optimal protocols for computing multi-party private set union. *IWQoS*, 1-10 (2022).

26. Blanton, M. & AguiarAuthors, E. Private and oblivious set and multiset operations. *ASIACCS*, 40-41 (2012).

27. Liu, X. & Gao, Y. Scalable multi-party private set union from multi-query secret-shared private membership test. *ASIACRYPT*, 237-271 (2023).

28. Gao, J.H., Nguyen, S. & Trieu, N. Toward A Practical Multi-party Private Set Union. *PETS*, 4, 622-635 (2024).

29. Davidson, A. & Cid, C. An Efficient Toolkit for Computing Private Set Operations. *ACISP*, 10343, 261-267 (2017).

30. Batcher, K.E. Sorting networks and their applications. *AFIPS*, 307-314 (1986).

31. Goldreich, O. The Foundations of Cryptography - Volume 2: Basic Applications. *Cambridge University Press* (2004).

32. Freedman, M.J., Ishai, Y., Pinkas, B. & Reingold, O. Keyword Search and Oblivious Pseudorandom Functions. *TCC*, 303-324 (2005).

33. Chase, M., Ghosh, E. & Poburinnaya, O. Secret-Shared Shuffle. *ASIACRYPT*, 12493, 342-272 (2020).

34. Kolesnikov, V. & Kumaresan, R. Improved OT extension for transferring short secrets. *CRYPTO*, 8043, 54-70 (2013).

35. Motwani, R. & Raghavan, P. Randomized Algorithms. *CSUR*, 28(1), 33-37 (1996).

36. Pagh, R. & Rodler, F.F. Cuckoo hashing. *J. Algorithms*, 51(2), 122–144 (2004).

37. Bay, A., Erkin, Z., Hoepman, J.H., Samardjiska, S. & Vos, J. Practical multi-party private set intersection protocols. *IEEE Trans. Inf. Forensics Secur*, 17, 1-15 (2021).

38. Pinkas, B., Schneider, T., Tkachenko, O. & Yanai, A. Efficient circuit-based PSI with linear communication. *EUROCRYPT*, 122-153 (2019).

39. Mohassel, P. & Sadeghian, S.S. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. *EUROCRYPT*, 7881, 557-574 (2013).

40. Chandran, N., Gupta, D. & Shah, A. Circuit-PSI with linear complexity via relaxed batch OPPRF. *PETS*, 2022, 353-372 (2022).

ARTICLE IN PRESS

ARTICLE IN PRESS

Funding

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)

Author contributions

Qiang Liu conceived and designed the one-leader and leaderless MPSU protocol frameworks, completed the security proofs, set up the experimental environment and simulations, and wrote the initial draft of the manuscript. Joon-Woo Lee supervised the entire study and provided critical revisions to the manuscript; he contributed to the methodological framework, result validation, and the comparative analysis against existing MPSU models, and was responsible for finalizing and submitting the manuscript. Both authors have read and approved the final version of the manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to J.-W.S.

ARTICLE IN PRESS