Toward A Practical Multi-party Private Set Union

Jiahui Gao
Arizona State University
Tempe, Arizona, USA
jgao76@asu.edu

Son Nguyen
Arizona State University
Tempe, Arizona, USA
snguye63@asu.edu

Ni Trieu
Arizona State University
Tempe, Arizona, USA
nitrieu@asu.edu

ABSTRACT

This paper studies a multi-party private set union (mPSU), a fundamental cryptographic problem that allows multiple parties to compute the union of their respective datasets without revealing any additional information. We propose an efficient mPSU protocol which is secure in the presence of any number of colluding semi-honest participants. Our protocol avoids computationally expensive homomorphic operations or generic multi-party computation, thus providing an efficient solution for mPSU.

The crux of our protocol lies in the utilization of new cryptographic tool, namely, Membership Oblivious Transfer (mOT). We believe that the mOT may be of independent interest. We implement our mPSU protocol and evaluate its performance. Our protocol shows an improvement of up to 80.84× in terms of running time and 405.73× bandwidth cost compared to the existing state-of-the-art protocols.

KEYWORDS

Multi-party Private Set Union, Membership OT, Oblivious Shuffle

1 INTRODUCTION

Secure multi-party computation (MPC) enables multiple parties to compute an arbitrary function on their private input without revealing additional information. A special case of MPC is the private set operation, which provides a secure means for joining data distributed across disparate databases. Private set intersection (PSI) and private set union (PSU) are two common set operations in this category. PSI finds applications in a variety of privacy-sensitive scenarios such as measuring the effectiveness of online advertising [26], contract tracing [4, 48], and contact discovery [23], and cache sharing in IoT [37]. Similarly, PSU has numerous practical use cases. For example, PSU can be used to implement Private-ID functionality [9], cyber risk assessment and management via joint IP blacklists and joint vulnerability data [25], private database supporting full join [31], association rule learning [29], joint graph computation [8], and aggregation of multi-domain network events [11].

Over the last decade, a substantial body of research [10, 42, 44] has focused on PSI, whereas PSU has received relatively little attention. The majority of present practical PSU protocols [5, 18, 28, 31, 53] have only been optimized for the two-party setting. In this study, we investigate multi-party PSU (mPSU) in the semi-honest model, which allows more than two parties to compute the union of their private data sets without revealing additional information.

1.1 Multi-Party PSU vs 2-Party PSU

Multi-party PSU is a natural extension of the two-party PSU and enables much richer data sharing than a two-party PSU. Collection of data from more participants will surely improve the performance of the data-driven applications that we mentioned above. However, designing a multi-party protocol in secure computation is challenging as it usually requires a dishonest majority (e.g. provides security in the presence of a number of dishonest, colluding participants). Existing mPSU protocols in generic MPC [6, 49], or homomorphic encryption [16, 22, 30, 46], are considerably more complex and expensive in the multiparty case than in the two-party case.

A possible solution for computing mPSU is leveraging efficient multi-party PSI protocols. Given their recent PSI improvements [12, 36] with practical implementations, one might think that mPSU can be computed directly from multi-party PSI using DeMorgan's Law as $\bigcup_{i=1}^{n} X_i = U \setminus \left(\bigcap_{i=1}^{n}(U \setminus X_i)\right)$, where U is a universe of input items. While this approach correctly and securely computes the set union, it is inefficient when U is significantly larger than $\bigcup_{i=1}^{n} X_i$. Thus, this solution is still far from practical.

Another potential approach is to extend the aforementioned practical two-party PSU protocols [10, 42, 44] to the multi-party case. However, it remains unclear how to achieve a secure mPSU protocol through this extension since the intermediate result would leak information like the intersection or intersection cardinality or union of a subset of parties' inputs which not only violates the mPSU functionality but also harms the privacy of the data owner.

In the application of Cyber risk assessment [25] previously mentioned and elaborated in [28, 31, 53], organizations want to compute the union of the IP blacklist while requires minimal leakage from the union. Consider data-driven applications as Private-ID [9] and association rule learning [29], where companies aim to build a joint dataset, better performance can be achieved by having richer data from more participants [24]. However, leakage of the input dataset of any company leads to disastrous consequences. If we implement mPSU using pairwise 2-party PSU, in some cases, privacy may not be guaranteed at all! For example, in a 3-party cases where P_1 is the receiver, if $X_3 \bigcap (X_1 \bigcup X_2) = \phi$, P_1 learns the P_3's data set completely. This issue arises not only when applying 2-party PSU protocol in a multi-party setting but also when P_1 colludes P_2, as even the leakage of the count of elements can lead to the same problem. This risk is exacerbated when dealing with sensitive information such as healthcare data or financial records. To address these privacy concerns, an mPSU protocol that prevents any additional information leakage and is resilient to arbitrary collusion is highly needed for the multi-party scenario.

Proceedings on Privacy Enhancing Technologies 2024(4), 622–635

https://doi.org/10.56553/popets-2024-0133

To grasp the challenges of extending from two-party to multiple-party PSU, we begin by reviewing the state-of-the-art 2-party PSU protocols [5, 18, 28, 53], which follow the framework of [31] based on oblivious transfer (OT), which consists of two main stages:

(1) Reverse Private Membership Test (RPMT): The receiver learns the bit representing the membership of each element in the sender's set. (e.g. for an element x in sender's set, the receiver with set Y learns a bit $b = 1$ if $x \in Y$ and $b = 0$ otherwise.). Note that the bit b reveals no additional information about the sender's set X, apart from the intersection cardinality $|X \cap Y|$, which is already revealed by the final PSU output.
(2) Oblivious Transfer (OT): The sender obliviously sends each item x in its set X to the receiver using OT. Concretely, the sender and the receiver invoke an OT functionality in which the sender possesses messages $\{\perp, x\}$ while the receiver holds the choice bit b, where $\perp$ represents a predefined special character. The bit b is the membership indicator bit which is derived from the preceding stage. The result of the OT provides the receiver with either $\perp$ or the sender's item x which is not the intersection item. By merging this outcome with its set Y, the receiver can produce the set union. This OT step prevents the receiver from deducing the intersection set, thereby fulfilling the functionality of a two-party PSU.

To summarize, in the two-party protocol, the parties initially establish the sender's element membership, followed by the receiver obliviously obtaining only the set difference from the sender. While this framework functions effectively and securely for the 2-party PSU, it cannot be directly extended to multi-party settings due to various sources of information leakage. To be more precise, assume there are n parties, each with a set X_i, and P_1 is the one who receives the final output. Considering a single element $x \in \bigcup_{i=2}^{n} X_i \setminus X_1$ which will be learned by P_1 from a PSU protocol, there are two types of information leakage considered in the multi-party setting:

- **Which party sends this element x?** The initial potential leakage arises from the origin of x. If P_1 and $P_{i\in[2:n]}$ invoke OT in the same manner as 2-party protocols, P_1 will know the contribution for the received element which is indeed an information leakage in the multi-party setting.
- **How many x are there?** Another potential leakage is the number of element x. In a 2-party setting, this count is consistently one, as the sender is the sole provider of new elements to the receiver (assuming that the X_2 is not a multi-set). In a multi-party setting, for element $x \in \bigcup_{i=2}^{n} X_i \setminus X_1$, any $P_{i\in[2:n]}$ can have it in the input set. So the number of duplication's can range from 1 to $n-1$.

In general, any information that can not be derived from the final output is not allowed. In the case of mPSU, the potential information leakage can be the union or intersection of the input from a subset of participants which can be addressed by avoiding the two leakages mentioned above. Thus, in the multi-party setting, the definition and execution of RPMT and OT must differ from those in the 2-party setting. Furthermore, another main challenge in designing mPSU is to prevent leakages in the event of collusion among a subset of parties.

1.2 Related Work

In this section, we focus on the state-of-the-art of multi-party PSU protocols. The earliest construction of such a protocol was proposed by Kissner and Song [30], which relied heavily on homomorphic encryption (the Paillier encryption) and the idea of polynomial representation. Input sets are represented as polynomials where each party $P_{i\in[n]}$ represents an input set $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ as a polynomial whose roots are its elements, which we denote $f_i(x) = \prod_{j=1}^{m}(x - x_{i,j})$. All parties together compute the encryption of polynomial $p = \prod_{i=1}^{n} f_i$ which presents the polynomial of the union $\bigcup_{i=1}^{n} X_i$. Using polynomial evaluation on the encrypted p, all parties are able to extract the union items without disclosing additional information. The protocol proposed in [30] has $O(n^3m^2)$ computation complexity. Relying on the polynomial presentation technique, Frikken [16] proposed an efficient mPSU protocol that requires the $O(n^2m\log(m))$ number of multiplications. [46] presented input sets using rational polynomial functions and reversed Laurent series. As a result, it showed a more efficient protocol than previous works [16, 30], but the protocol is secure up to $n/2$ corrupted parties.

Blanton and Aguiar [6] presented a new direction to compute mPSU that avoids expensive homomorphic encryption but heavily relies on MPC. Their idea is to combine the input sets of all parties under a secret-shared form, perform an oblivious sort on the resulting set, and then remove the duplications by comparing the adjacent elements. In the context of MPC, a more practical sorting algorithm is Batcher's network which requires $O(mn\log(mn))$ comparisons to sort the union sets. Due to the underlying MPC techniques, the protocol of [6] is inefficient when the m and n are large.

[22, 47] compute the mPSU using Bloom filter (BF). Specifically, each party $P_{i\in[1,n]}$ inserts its input items into a local BF and transmits the encrypted version of the resulting BF to a designated leader party P_1. Subsequently, the P_1 aggregates the encrypted local BFs from all parties to generate a global BF, denoted by G, from which the union items are computed. While the protocol presented in [47] makes use of an outsourcing server to compute G, [22] is built on homomorphic encryption (HE), which requires a homomorphic computation per each entry of G, and might need expensive multi-key HE. Moreover, the BF-based approach is associated with a high false positive rate.

In another work, Vos *at. el.*[49] proposed private OR protocols and build mPSU protocols upon it. They consider a relatively small universe (e.g. up to 32-bit long element). At the high-level idea, their approach presents the input set in a bit vector of length $|\mathcal{U}|$. The bit is set to 1 if its corresponding element belongs to the given input set and 0 otherwise. By invoking the proposed private OR protocol, the leader learns the bit vector of the union. While optimization is given by applying divide-and-conquer so that the long vector can be divided into small ones, it is still inefficient, especially for the standard input of 128-bit elements. Concurrently with our work, Liu and Gao [34] presented an efficient mPSU protocol but requires a weak security assumption wherein *the leader is not in collusion with any other participating parties.*

For a comprehensive analysis of representative multi-party PSU protocols that are resilient to *the presence of any number of colluding*

semi-honest participants, we provide a summary of their theoretical complexity in Table 1. Additionally, in Section 5.2, we present a numerical performance comparison of our proposed protocols with prior works [6, 16, 22].

Very recently, Dong et al. [14] points out a security flaw in our original protocol. Our updated protocol (Figure 8) is an easy fix by removing the redundant step for computing PRF. Refer to our full version [17] for details. [14] proposed a new batched SS-PMT construction which outperforms the multi-query SS-PMT proposed in [34] in the usage of mPSU. Following our mPSU framework where P_1 collects encryptions of the union items from other participants with the multi-key cryptosystem (Section 2.5) and Shuffle&Decrypt (Section 3.2), they presented a new efficient mPSU protocol from the batched SS-PMT and random OT. Their protocol is secure against arbitrary collusion and achieves linear computation and communication cost in terms of number of elements which outperform our work.

1.3 Technical Overview of Our Protocols

We present an efficient protocol for mPSU that guarantees security in the semi-honest setting. We demonstrate the practicality of our mPSU protocol with an implementation. It is shown to be efficient even for large sets with 2^{20} items distributed among 8 parties. The main reason for our protocol's high performance is its reliance on fast symmetric-key primitives and ElGamal encryption. This is in contrast with prior protocols, which require expensive Paillier encryption on the polynomial set representation [16, 30] or each entry of the Bloom filter [22]. Additionally, our approach eliminates the need for the inefficient oblivious sort/OR operations and generic MPC of [6, 49].

Technical Overview. In our protocol, we assume the existence of a leader party denoted as P_1. This party learns the final result by growing the union starting with X_1. To be specific, P_1 learns $X_t \setminus \bigcup_{i=1}^{t-1} X_i$ from P_t. This is achieved by interacting sequentially with each party $P_2, \ldots, P_n$. All the parties agree on a multi-key cryptosystem for encryption, and sets are encrypted to prevent the leader from learning the partial union. Moreover, we propose new primitives Membership Oblivious Transfer(mOT) to ensure the correctness of the final result as well as prevent the information leakage introduced earlier in the Section 1.1.

Briefly, the mOT is a two-party protocol, in which a leader P_1 (also referred to as the receiver) holding a set X_1 interacts with the sender P_t who possesses an input item $x_{t,j}, j \in [m]$, and two associated values $\{v_0, v_1\}$. Similar to the traditional OT [43], the result is that the sender P_t learns nothing whereas the receiver P_1 obtains one of the two sender's associated values depending on whether $x_{t,j} \in X_1$.

In our mPSU protocol, sender P_t prepares $v_0 = \mathsf{Enc}(\mathsf{pk}, 0)$ and $v_1 = \mathsf{Enc}(\mathsf{pk}, x_{t,j})\}$, where pk is the public key for the multi-key cryptosystem. If $x_{t,j} \in X_1$, P_1 learns $\mathsf{Enc}(\mathsf{pk}, 0)$; otherwise it learns $\mathsf{Enc}(\mathsf{pk}, x_{t,j})$. By executing the mOT multiple times with $P_{t\in[2,n]}$ for each item $x_{t,j} \in X_t$, the leader P_1 obtains a set E of encryptions $\mathsf{Enc}(\mathsf{pk}, x_{i,j})$ for $x_{i,j} \in \bigcup_{i=1}^{n} X_i \setminus X_1$ and the number of encryptions of zero. At this point, the set E still contains the encryption of the $X_i \cup X_t$ for $i, t > 1$. To remove these encryptions, before executing with P_1, we require P_t executes an mOT with each $P_{i\in[2,t-1]}$.

PARAMETERS: n parties $P_1, ..., P_n$, and the set size m.

FUNCTIONALITY:

- Wait for input set X_i of size m from P_i.
- Give P_1 the union $\bigcup_{i=1}^{n} X_i$.

Figure 1: Multi-party Private Set Union Ideal Functionality

As the result, P_t holds $\mathsf{Enc}(\mathsf{pk}, x_{t,j})$ if the item $x_{t,j}$ is not in any set $X_2, \ldots, X_{t-1}$, and $\mathsf{Enc}(\mathsf{pk}, 0)$ otherwise. Now, the union can be obtained by decrypting E and removing the zeros.

For the dishonest majority setting in which the protocol is secure against an arbitrary number of colluding parties, the decryption should be executed by all the parties. Thus, we employ the multi-key cryptosystem based on the ElGamal encryption scheme (ref. Section 2.5). The decryption process involves a partial decryption that requires the individual party's secret key. In our protocol, each party is required to perform its own private permutation on the partial decryption result before sending it to another party. This step aims to prevent a coalition of corrupt parties (including the leader P_1) from learning which parties hold which elements. We implement the permutation and decryption using our simple building block "Oblivious Shuffle and Decryption" (Shuffle&Decrypt), which is described in Section 3.2.

In brief, our contributions can be summarized as follows:

- We present an efficient mPSU construction, which eliminates the need for computationally expensive homomorphic operations or generic multi-party computation and is secure in the presence of any number of colluding semi-honest participants.
- We introduce new building blocks, namely Membership Oblivious Transfer (mOT) and Oblivious Shuffle and Decryption (Shuffle&Decrypt), which may be of independent interest and can be used in other related protocols.
- We show that our protocol is significantly faster than previous work [6, 16, 22]. For example, for four parties with a dataset of 2^{16} item each, our mPSU protocol shows an improvement up to 80.84× in terms of running time and up to 405.73× less bandwidth requirement when compared to the state-of-the-art protocols. Our implementation is publicly available at https://github.com/asu-crypto/mpsu.

2 PRELIMINARIES

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. We use $[m]$ to refer to the set $\{1, \ldots, m\}$, and $[i, j]$ to denote the set $\{i, \ldots, j\}$. We denote the concatenation of two strings x and y by $x||y$. We use $f \circ g$ to denote the composition of the functions f and g.

2.1 Multi-party Private Set Union

The ideal functionality of multi-party PSU (mPSU) is given in Figure 1. It allows n parties, each holding a set X_i of the input items, to learn the union $\bigcup_{i=1}^{n} X_i$ and nothing else. For simplicity, we assume that all parties have the same set size m, which is publicly known.

Protocol	Overall Communication	Computation			Round
		Overall	# HE		
			Leader P_1	Party $P_{i\in[2,n]}$	
[30]	$O(n^2m)$	$O(n^3m^2)$	$O(nm^2)$	$O(nm^2)$	$O(n)$
[16]	$O(n^2m)$	$O(n^2m\log(m))$	$O(nm)$	$O(nm\log(m))$	$O(n)$
[6]	$O(n^2m\log^2(mn))$	$O(n^2m\log^2(mn))$	0	0	$O(\log(nm))$
[22]	$O(n^2m\lambda)$	$O(n^2m\lambda)$	$O(nm\lambda)$	$O(nm\lambda)$	$O(1)$
[49]	$O(n^2m\log\lvert\mathcal{U}\rvert)$	$O(n^2m\log\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$
Ours	$O(n^2m\log m/\log\log m)$	$O(n^2m\log m/\log\log m)$	$O(nm)$	$O(nm)$	$O(n)$

Table 1: Communication (overall), computation (overall and number of homomorphic operation), and round complexities of n-party PSU protocols which are secure in *the presence of any number of colluding semi-honest participants*. #HE represents the number of additive homomorphic operations. n is number of parties, each with set size m; λ is the statistical security parameter; $\mathcal{U}$ is the universal domain of the input; σ is the bit length of input element;t is the number of AND gates in the SKE decryption circuit. Notably, the complexity of [22] consists of λ due to the usage of the Bloom filter. All [16, 22, 30] use Paillier encryption to compute addition on the encryptions (#HE) while our protocol uses ElGamal encryption scheme to re-randomize the ciphertexts. Note that we can eliminate the term $\log m/\log\log m$ in the complexity of our protocol by applying the recent batch Secret-shared Private Membership Test of [34].

Threat Model and Security Goal. From the ideal functionality of mPSU, we can see that the mPSU protocol is secure if the mPSU protocol is considered secure as long as it does not disclose any additional information beyond the union and m to the parties, encompassing partial set union/intersection.

Note that our protocol can be easily extended to accommodate varying set sizes, while also protecting the set size of each party. This can be accomplished if all parties agree on an upper bound set size m and utilize it as the input set size. Before initiating the protocol, each party can pad their set with a particular item, such as zero, to reach the size of m. It is customary in private set operation literature to assume that all parties have the same set size.

In this paper, we focus on the semi-honest setting, where it is assumed that parties adhere to the protocol description but attempt to glean additional information from the protocol's transcript.

2.2 Oblivious Transfer

Oblivious Transfer (OT) is a fundamental primitive of secure computation and was introduced by Rabin [43]. It refers to the problem where a sender with two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. The OT gives the receiver x_b and nothing to the sender. Figure 2 presents the OT functionality.

Parameters: Two parties: Sender and Receiver

Functionality:
- Wait for input strings $(x_0, x_1) \subset (\{0,1\}^\star)^2$ from the sender.
- Wait for input choice bit $b \in \{0,1\}$ from the receiver.
- Give x_b to the receiver.

Figure 2: Oblivious Transfer (OT) Ideal Functionality.

2.3 Secret-shared Private Membership Test

Secret-shared Private Membership Test (SS-PMT) is the main building block in different applications [12, 33, 34, 40, 41, 53]. It refers to the two-party setting where a P_0 with input a set of items $X = \{x_1, \ldots, x_n\}$ interacts with a P_1 who has an input single item y. SS-PMT gives both parties a secret share of a membership bit, i.e. the two parties obtain XOR shares of 1 if $y \in X$ and 0 otherwise. Figure 3 presents the SS-PMT functionality.

Parameters: Two parties: P_0 and P_1, and the set size n.

Functionality:
- Wait for input a set of items $X = \{x_1, \ldots, x_n\} \subset (\{0,1\}^\star)^n$ from the P_0.
- Wait for input item $y \in \{0,1\}^\star$ from the P_1.
- Give b_i to the $P_{i\in\{0,1\}}$ where $b_0 \oplus b_1 = 1$ if $y \in X$ and 0 otherwise.

Figure 3: Secret-shared Private Membership Test (SS-PMT) Ideal Functionality.

2.4 Bin-and-ball Scheme

Our protocols employ hashing schemes such as the Cuckoo and Simple hashing schemes [39, 42] to allocate items into bins. We review the basics of the Cuckoo hashing and Simple hashing schemes [39, 42] as follows.

Cuckoo hashing. In basic Cuckoo hashing, there are μ bins denoted $B[1 \ldots \mu]$, a stash, and h random hash functions $H_1, \ldots, H_h : \{0,1\}^\star \to [\mu]$. One can use a variant of Cuckoo hashing such that each item $x \in X$ is placed in exactly one of μ bins. Using the Cuckoo analysis [13, 39] based on the set size $|X|$, the parameters μ, h are chosen so that with high probability $(1 - 2^{-\lambda})$ every bin contains at most one item, and no item has to be placed in the stash during the Cuckoo eviction (i.e. no stash is required).

Simple hashing. One can map its input set Y into μ bins using the same set of h Cuckoo hash functions (i.e, each item $y \in Y$ appears h times in the hash table). Using a standard ball-and-bin analysis

based on h, μ, and $|X|$, one can deduce an upper bound η such that no bin contains more than β items with high probability $(1 - 2^{-\lambda})$.

2.5 Multi-key Cryptosystem

We revise the multi-key cryptosystem that is needed for our mPSU protocol. We first give an overview of each component of the cryptosystem. We then present a construction based on the ElGamal scheme. A **multi-key cryptosystem** is defined as a tuple of PPT algorithm (KeyGen, Enc, ParDec, FulDec, ReRand) with properties as follows:

- **Key Generation**: $\mathsf{KeyGen}(1^\kappa, n)$. In a setting with n parties, a key generation algorithm takes security parameter κ as input and gives each party P_i a secret key sk_i and a joint public key $\mathsf{pk} = \mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_n)$, where Combine is an algorithm to generate the public key from the input secret keys depending on the construction.
- **Encryption**: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}; \mathsf{m})$. Given a joint public key pk and a message $\mathsf{m} \leftarrow \mathcal{M}$ from the plaintext space $\mathcal{M}$, an encryption algorithm outputs a ciphertext ct.
- **Decryption**: There are two types of decryption:
  - Partial decryption $\mathsf{ct}' \leftarrow \mathsf{ParDec}(\mathsf{sk}_i, \mathsf{ct}, \mathsf{A})$. A partially decryption algorithm takes a secret key sk_i and a ciphertext $\mathsf{ct} \leftarrow C$ encrypted under the partial pubic key $\mathsf{pk}_A = \mathsf{Combine}(\{\mathsf{sk}_j \mid j \in A\})$ and outputs a ciphertext $\mathsf{ct}' \leftarrow C$ which is encrypted under the partial pubic key $\mathsf{pk}_{A\setminus\{i\}} = \mathsf{Combine}(\{\mathsf{sk}_j \mid j \in A, j \neq i\})$. Note that in the context of the multi-key encryption system, we utilize set A to represent the collection of public keys belonging to the parties within A.
  - Full decryption: $\mathsf{m} \leftarrow \mathsf{FulDec}(\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_n; \mathsf{ct})$. A full decryption algorithm takes a ciphertext $\mathsf{ct} \leftarrow C$ encrypted under pk and all the secret keys and outputs a message $\mathsf{m} \leftarrow \mathcal{M}$.
- **Re-randomization**: $\mathsf{ct}' \leftarrow \mathsf{ReRand}(\mathsf{ct}, \mathsf{pk})$. This algorithm takes a ciphertext $\mathsf{ct} \leftarrow C$ encrypted under pk and gives a re-randomized ciphertext $\mathsf{ct}' \leftarrow C$ encrypted under the same pk such that they are both encryptions of the same message $\mathsf{m} \leftarrow \mathcal{M}$.

The multi-key cryptosystem should satisfy correctness and security as defined in [1, 7, 20]. Informally, the multi-key cryptosystem satisfies correctness if $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, ..., \mathsf{sk}_n, \mathsf{ct})$ or $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_{i-1}, \mathsf{sk}_{i+1}, \ldots, \mathsf{sk}_n, \mathsf{ct}')$ for $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{ct}' = \mathsf{ParDec}(\mathsf{sk}_i, \mathsf{ct}_{\{1,\ldots,i-1,i+1,\ldots,n\}})$. For security, the ciphertext ct or ct' is random and reveals nothing about the plaintext. When $n = 1$, we have a single-key encryption scheme which is indeed the traditional ElGamal system[15].

A Construction. While there are many multi-key cryptosystems, we choose ElGamal system[15] as it is easy to implement and efficient (we do not perform any arithmetic computation on the encryption). In the following, we present the ElGamal scheme in the multi-key setting with n parties $P_1, \ldots, P_n$.

- **Key Generation**: Given a security parameter κ and number of parties n. A cyclic group $\mathcal{G}$ of order p is chosen, and all the parties agree on a common generator g. Each party $P_{i\in[n]}$ chooses a random secret key $\mathsf{sk}_i \leftarrow \{0,1\}^\kappa$ and publishes the value of $h_i = g^{\mathsf{sk}_i}$. We can define the public key $\mathsf{pk} = \mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_n) = g^{\sum_{i=1}^n \mathsf{sk}_i} = \prod_{i=1}^n h_i$.
- **Encryption**: To encrypt a message m, one can compute $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (g^r, m \cdot \mathsf{pk}^r)$ where r is a randomly chosen value from $\{0,1\}^\kappa$.
- **Decryption**: The two decryption algorithms are as follows:
  - Partial decryption: To partially decrypt a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ encrypted under the partial pubic key $\mathsf{pk}_A = \prod_{j\in A} h_j$, one output $\mathsf{ct}' = \mathsf{ParDec}(\mathsf{sk}_i, \mathsf{ct}, A) = (\mathsf{ct}'_1, \mathsf{ct}'_2)$, where $\mathsf{ct}'_1 = \mathsf{ct}_1 \cdot g^{r'}, \mathsf{ct}'_2 = \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}_i} \cdot (\mathsf{pk}_{A\setminus\{i\}})^{r'}$, the $r' \leftarrow \{0,1\}^\kappa$ is a random value, and $\mathsf{pk}_{A\setminus\{i\}} = \prod_{j\in A\setminus\{i\}} h_j$. Note that the use of the random r' aims to re-randomize the ct_1.
  - Full decryption: To fully decrypt a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ encrypted under $\mathsf{pk} = \prod_{i\in[n]} h_i$, one can compute $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_n; \mathsf{ct}) = \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\sum_{i=1}^n \mathsf{sk}_i}$.
- **Re-randomization**: To re-randomize a ciphertext ct encrypted under the pk, one can choose a random value $r' \leftarrow \{0,1\}^\kappa$, and compute $(\mathsf{ct}'_1, \mathsf{ct}'_2) = \mathsf{ReRand}((\mathsf{ct}_1, \mathsf{ct}_2), \mathsf{pk})$, in which $\mathsf{ct}'_1 = \mathsf{ct}_1 \cdot g^{r'}$ and $\mathsf{ct}'_2 = \mathsf{ct}_2 \cdot \mathsf{pk}^{r'}$.

3 OUR MPSU BUILDING BLOCKS

We introduce two simple cryptographic gadgets that will serve as the fundamental building blocks in our mPSU protocol.

- The first gadget is called "Membership Oblivious Transfer" (mOT) which enables a receiver to obtain one of two associated values from the sender based on the set membership. The mOT allows a leader party in our mPSU protocol to obliviously retrieve the items of other parties that are not in the intersection while maintaining privacy.
- In our mPSU protocol, the union result is stored under the multi-key encryption until the final step, which requires all parties to decrypt the ciphertexts together. The encryption protects against corrupted parties from learning partial union. We revise a multi-key cryptosystem in Section 2.5, and introduce a simple tool called "Shuffle and Decryption" (Shuffle&Decrypt) to implement the last step of our mPSU construction.

In the following, we present the definition and ideal functionality of each building block, which specify the input and output. Parties should not gain any additional knowledge beyond the desired output, ensuring the security of each introduced primitive.

3.1 Membership Oblivious Transfer (mOT)

Definition 1. *Membership Oblivious Transfer (mOT) is a two-party protocol, in which a sender $\mathcal{S}$ with a keyword $y \in \{0,1\}^\star$ and two associated values $\{v_0, v_1\} \in (\{0,1\}^\ell)^2$ interacts with a receiver $\mathcal{R}$ who has a set of keywords $X = \{x_1, ..., x_m\} \in (\{0,1\}^\star)^m$. Except randomnesses, the mOT functionality gives the receiver the value v_b where $b = 0$ if $y \in X$ and $b = 1$ otherwise, and nothing to the sender.*

Similar to the traditional OT, the associated values v_0, v_1 are indistinguishable with respect to their domain $\{0,1\}^\ell$, so that the membership of y in terms of X is also not revealed to the receiver.

PARAMETERS: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, the receiver set size m, the length ℓ.

FUNCTIONALITY:

- Wait for input keyword y and a pair $(v_0, v_1) \in \{0,1\}^\ell \times \{0,1\}^\ell$ from $\mathcal{S}$.
- Wait for input set $X = \{x_1, ..., x_m\}$ from $\mathcal{R}$.
- Give $\mathcal{R}$ the value v where v equals to v_0 if $y \in X$, and v_1 otherwise.

Figure 4: Membership Oblivious Transfer (mOT) Ideal Functionality

We name our gadget "Membership Oblivious Transfer" as the receiver's obtained value depends on whether $y \in X$. We formally describe the mOT ideal functionality in Figure 4.

From Definition 1, we see that if a construction for mOT is secure, it should satisfy two following properties:

- Similar to the traditional one-out-of-two oblivious transfer [43], the receiver $\mathcal{R}$ only learns one of the two associated values of the sender $\mathcal{S}$. In addition, the receiver $\mathcal{R}$ has no information about whether $y \in X$ is from the protocol's output. In fact, the latter is satisfied if the associated values (v_0, v_1) are sampled according to the same distribution.
- The sender $\mathcal{S}$ learns nothing about the receiver's input and output.

To sum up, our security objective for mOT is to enable the sender to anonymously transmit one of its associated values to the receiver based on the membership condition.

Our mOT Protocol. Our mOT construction consists of two main phases. The first phase follows the popular steps in the circuit-PSI protocols [40, 41], which enables the sender and the receiver to compute a secret share of a membership bit, i.e. the two parties obtain XOR shares of 1 or 0 if the sender's keyword y is or is not in the receiver's set X.

The second phase allows the receiver to obtain the corresponding associated value from the sender, depending on whether the output of the first phase was shares of 0 or 1. Typically, this step can be done using generic two-party secure computation (e.g., garbled circuit) in the literature. However, it is relatively inefficient. Instead, we propose a simple solution that relies on OT. More precisely, the sender randomly chooses a value $r \leftarrow \{0,1\}^\ell$ and masks its associated values by computing $(r \oplus v_0, r \oplus v_1)$. Denote a secret share bit of $\mathcal{S}$ and $\mathcal{R}$ to be $b_\mathcal{S}$ and $b_\mathcal{R}$ received from the first phase, respectively. Using the choice bit $b_\mathcal{R}$, the receiver obliviously obtains $w = r \oplus v_{b_\mathcal{R}}$ when interacting with the sender with input $(r \oplus v_0, r \oplus v_1)$ via OT. Next, the sender sends $u = r \oplus b_\mathcal{S} \cdot (v_1 \oplus v_0)$ to the receiver $\mathcal{R}$. The value u helps to remove the mask r from the w by computing $v = u \oplus w$, which is the receiver's output. We formally present the construction of our mOT in Figure 5.

For the correctness of the mOT construction, one can rewrite $w = r \oplus b_\mathcal{R} \cdot v_1 \oplus (1 \oplus b_\mathcal{R}) \cdot v_0$. Hence, $v = u \oplus w = (b_\mathcal{R} \oplus b_\mathcal{S}) \cdot v_1 \oplus (1 \oplus b_\mathcal{R} \oplus b_\mathcal{S}) \cdot v_0$ which equals to $v_{b_\mathcal{R} \oplus b_\mathcal{S}}$ as desired (recall that $b_\mathcal{R} \oplus b_\mathcal{S} = 1$ if $y \in X$ and 0 otherwise). We present the security statement of our mOT protocol below.

PARAMETERS:

- Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, the receiver set size m, the length ℓ.
- The OT and SS-PMT functionalities described in Section 2.

INPUT:

- Receiver $\mathcal{R}$: $X = \{x_1, \ldots, x_m\} \subset (\{0,1\}^\star)^m$
- Sender $\mathcal{S}$: $y \in \{0,1\}^\star$ and two associated values $(v_0, v_1) \subset (\{0,1\}^\ell)^2$

PROTOCOL:

(1) The sender $\mathcal{S}$ and the receiver $\mathcal{R}$ invoke a SS-PMT functionality where:
   - $\mathcal{R}$ has an input set X, and $\mathcal{S}$ has an input y.
   - $\mathcal{S}$ and $\mathcal{R}$ obtain the bit $b_\mathcal{S}$ and $b_\mathcal{R}$, respectively. Here, $b_\mathcal{S} \oplus b_\mathcal{R} = 1$ if $y \in X$ and 0 otherwise.
(2) $\mathcal{S}$ and $\mathcal{R}$ invoke an OT instance where:
   - $\mathcal{S}$ acts as an OT sender with input two strings $\{r \oplus v_0, r \oplus v_1\}$, where $r \leftarrow \{0,1\}^\ell$ is a random chosen value.
   - $\mathcal{R}$ acts as an OT receiver with input a choice bit $b_\mathcal{R}$.
   - $\mathcal{R}$ obtains $w = r \oplus v_{b_\mathcal{R}}$.
(3) $\mathcal{S}$ sends $u = r \oplus b_\mathcal{S} \cdot (v_1 \oplus v_0)$ to $\mathcal{R}$ who outputs $v = u \oplus w$.

Figure 5: Membership Oblivious Transfer (mOT) Construction

THEOREM 2. *The mOT protocol described in Figure 5 securely implements the mOT functionality defined in Figure 4 in the semi-honest setting, given the OT and SS-PMT functionalities described in Section 2.*

PROOF. We construct simulators $\mathsf{Sim}_\mathcal{S}$ and $\mathsf{Sim}_\mathcal{R}$ to simulate the view of corrupted sender $\mathcal{S}$ and corrupted receiver $\mathcal{R}$, respectively. We argue the indistinguishability of the simulator and the real execution.

Simulating $\mathcal{S}$: The simulator $\mathsf{Sim}_\mathcal{S}$ has input (y, v_0, v_1) and receives output from the SS-PMT ideal functionality, consisting of a secret-shared membership bit $b_\mathcal{S}$. For the OT execution, the simulator $\mathsf{Sim}_\mathcal{S}$ obtains nothing, except the random OT transcript which is random. Since the output of SS-PMT is secret-shared amongst the corrupt sender and honest receiver, one can replace the bit $b_\mathcal{S}$ with a random. It is straightforward to check that the simulation is perfect.

Simulating $\mathcal{R}$: $\mathsf{Sim}_\mathcal{R}$ with input X receives nothing from the SS-PMT ideal functionality, expect a secret-shared membership bit $b_\mathcal{R}$. $\mathsf{Sim}_\mathcal{R}$ obtains w from the OT and u from the sender in the last step. We show that the output of the simulator $\mathsf{Sim}_\mathcal{R}$ is indistinguishable from the real execution. For this, we formally show the simulation by proceeding with the sequence of hybrid transcripts T_0, T_1, T_2 where T_0 is real view of the receiver, and T_2 is the output of $\mathsf{Sim}_\mathcal{R}$.

- Let T_1 be the same as T_0, except the SS-PMT output which can be replaced with random as the honest sender holds a secret-shared of the output. Thus, T_0 and T_1 are indistinguishable.
- Let T_2 be the same as T_1, except the OT execution and obtaining u. Due to the underlying security property of OT, the

PARAMETERS: n parties, parameter m, and a multi-key encryption scheme defined in Section 2.5

FUNCTIONALITY:

- Wait for input secret key sk_i and a permutation function $\pi_i : [m] \to [m]$ from each party $P_{i\in[n]}$. Here, $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(1^\kappa, n)$.
- Wait for a combined input a set of ciphertexts $\{\mathsf{ct}_1, \ldots, \mathsf{ct}_m\}$ where $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, x_i)$ from all parties $\{P_1, \ldots, P_n\}$.
- Give $\{x_{\pi(1)}, \ldots, x_{\pi(m)}\}$ to P_1 where $\pi = \pi_n \circ \pi_{n-1} \circ \ldots \circ \pi_1$.

Figure 6: Oblivious Shuffle and Decryption (Shuffle&Decrypt) Ideal Functionality

receiver only learns one of the two strings related to v_0 or v_1. In addition, the sender's associated values were masked with a random value r before the OT execution. Thus, w reveals nothing about $v_{i\in\{0,1\}}$. When having $u = r \oplus b_{\mathcal{S}} \cdot (v_1 \oplus v_0)$, the corrupt receiver might try to unmask r by computing $u \oplus w$. However, the resulting value is indeed the protocol's output which can be simulated. Therefore, we can replace both w and u with random (the receiver sees a system of two equations that contains three unknown variables). In summary, T_2 and T_1 are indistinguishable.

□

3.2 Oblivious Shuffle and Decryption (Shuffle&Decrypt)

DEFINITION 3. *Oblivious Shuffle and Decryption (refers as Shuffle&Decrypt) is a n-party protocol, in which each party $P_{i\in[n]}$ holds a permutation $\pi_i : [m] \to [m]$ and a secret key sk_i of the multi-key cryptosystem as $(pk, \{sk_i\}_{i\in[n]}) \leftarrow KeyGen(1^\kappa, n)$. Given a set of ciphertexts $\{ct_1, \ldots, ct_m\}$ where $ct_i = Enc(pk, x_i)$, except randomnesses, the Shuffle&Decrypt functionality gives $\{x_{\pi(1)}, \ldots, x_{\pi(m)}\}$ to the party P_1 where $\pi = \pi_n \circ \pi_{n-1} \circ \ldots \circ \pi_1$, and nothing to other parties.*

The private permutation aims to remove the linkage between the ciphertext ct_i and the plaintext x_i. We formally describe the Shuffle&Decrypt ideal functionality in Figure 6.

Our Shuffle&Decrypt Protocol. The Shuffle&Decrypt construction is simple and directly built from calling algorithms provided in the multi-key cryptosystem. First, the P_1 re-randomizes the ciphertexts and then permutes the result. P_1 then sends the permuted set C_1 to P_2. The re-randomization aims to hide the permutation function from P_2. The P_2 now performs partial decryption using its secret key sk_2. This decryption removes the role of sk_2 from the original ciphertext. P_2 then applies the permutation π_2 on the resulting ciphertexts C_2 and forwards them to P_3. Note that P_2 does not need to rerandomize C_2 as the C_2 is in the random distribution and thus it hides the permutation of P_2. The process repeats sequentially through $P_4, \ldots, P_n$. After the partial decryption was executed by P_n, the ciphertexts require only the secret key sk_1 for the final decryption. P_n now sends these ciphertexts in the permuted order to P_1 which performs the partial decryption and outputs the final result. Figure 7 presents the Shuffle&Decrypt construction. From the high-level description, it is clear that the protocol is correct given the correctness of the underlying multi-key cryptosystem. We present the security statement of the Shuffle&Decrypt protocol below.

PARAMETERS:

- n parties, the set size m
- A multi-key cryptosystem (KeyGen, Enc, ParDec, FulDec, ReRand) defined in Section 2.5

INPUT:

- Each party $P_{i\in[n]}$: The secret key sk_i and a permutation function $\pi_i : [m] \to [m]$. Here, $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(1^\kappa, n)$
- All parties: $\{\mathsf{ct}_1, \ldots, \mathsf{ct}_m\}$ where $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, x_i)$

PROTOCOL:

(1) P_1 re-randomizes $\mathsf{ct}_j = \mathsf{ReRand}(\mathsf{ct}_j, \mathsf{pk}), \forall j \in [m]$, and sends $C_1 = \{\mathsf{ct}^1_1, \ldots, \mathsf{ct}^1_m\}$ to P_2 where $\mathsf{ct}^1_j = \mathsf{ct}_{\pi_1(j)}, \forall j \in [m]$.
(2) For $i = 2$ to n:
   - P_i computes a partial decryption $\bar{\mathsf{ct}}^i_j = \mathsf{ParDec}(\mathsf{sk}_i, \mathsf{ct}^{i-1}_j, A_i), \forall j \in [m]$, where $A_i = \{1, i, i+1, \ldots, n\}$.
   - P_i permutes the set $\{\bar{\mathsf{ct}}^i_1, \ldots, \bar{\mathsf{ct}}^i_m\}$ as $\mathsf{ct}^i_j = \bar{\mathsf{ct}}_{\pi_i(j)}, \forall j \in [m]$.
   - P_i sends $C_i = \{\mathsf{ct}^i_1, \ldots, \mathsf{ct}^i_m\}$ to $P_{(i+1)\%n}$.
(3) P_1 outputs $\mathsf{ParDec}(\mathsf{sk}_1, \mathsf{ct}^n_j, \{1\}), \forall j \in [m]$.

Figure 7: Oblivious Shuffle and Decryption (Shuffle&Decrypt) Construction

THEOREM 4. *Given the multi-key cryptosystem defined in Section 2.5, the Shuffle&Decrypt protocol described in Figure 7 securely implements the Shuffle&Decrypt functionality defined in Figure 6 in the semi-honest model, against any number of corrupt, colluding, semi-honest parties.*

PROOF. Let A be a coalition of corrupt parties. The view of A is a set of ciphertexts $\{C_i \mid P_i \in A\}$, and the output of the Shuffle&Decrypt which is $\{x_{\pi(1)}, \ldots, x_{\pi(m)}\}$ if the leader $P_1 \in A$.

Thanks to the property of the multi-key cryptosystem, $C_{i\in[n]}$ reveals nothing about the underlying plaintexts. If P_1 is honest, the randomization hides the party's permutation function. Moreover, when assuming $\{P_i, P_j\} \in A$ but $\{P_{i+1}, \ldots, P_{j-1}\} \notin A$, one might think that A might learn the permutation functions of honest parties $\{P_{i+1}, \ldots, P_{j-1}\}$. However, the output of the partial decryption gives ciphertexts in the random distribution. Thus, the resulting view is random to A (i.e., the corrupt coalition's view is simulated). □

4 OUR MPSU CONSTRUCTION

Figure 8 presents our main mPSU protocol, which guarantees security against any number of corrupt, colluding, semi-honest parties.

The protocol makes use of our new mOT and Shuffle&Decrypt gadgets.

4.1 Our Protocol

The design of a secure mPSU protocol presents significant challenges, specifically with regard to (1) ensuring that the output does not contain duplicate items, (2) preventing the disclosure of partial union results, and (3) hiding which items from which parties. To illustrate the high-level idea of our protocol, we consider a simple 4-party case where the leader party P_1 has a set X_1 of items while each of the remaining party P_i for $i \in [2,4]$ possesses a single item $X_i = \{x_i\}$. We assume that the item x_2 of P_2 and x_3 of P_3 are not in the X_1 but x_4 of P_4 is (i.e. $x_2 = x_3 \notin X_1$ and $x_4 \in X_1$).

Regarding (1), a potential approach is to enable the leader P_1 to engage with the other parties and obtain an encryption of x_i if $x_i \notin X_1$ and an encryption of the zero otherwise. An encryption of zero indicates the presence of common items between P_1 and P_i, which can be removed after decryption. To this end, the P_1 and P_i invoke the mOT instance in which P_i acts as the sender with input $\big(x_i, \mathsf{Enc}(\mathsf{pk}, 0), \mathsf{Enc}(\mathsf{pk}, x_i)\big)$ and P_1 acts the receiver with input X_1, thereby obtaining the desired encryption. After executing the mOT instances, the leader party P_1 acquires $E = \{\mathsf{Enc}(\mathsf{pk}, x_2), \mathsf{Enc}(\mathsf{pk}, x_3), \mathsf{Enc}(\mathsf{pk}, 0)\}$ from the party P_2, P_3 and P_4, respectively. The set E allows the leader P_1 to obtain the set union after decryption.

The above protocol description does not entirely address the issue of removing duplicate items since x_2 could be identical to x_3. This causes the challenge (2) as mentioned above. To overcome the limitation, we leverage the mOT in the following manner.

Similar as before, the protocol starts with the leader party P_1 which has an input X_1 and learns $\mathsf{Enc}(\mathsf{pk}, x_2)$ after executing an mOT with P_2. Next, the P_2 performs an mOT with each of the parties $P_{i\in\{3,4\}}$. Specifically, $P_{i>2}$ acts as the sender with inputs $\big(x_i, \mathsf{Enc}(\mathsf{pk}, 0), \mathsf{Enc}(\mathsf{pk}, x_i)\big)$, while P_2 acts as the receiver with input X_2. As the result, the P_2 learns e_2^i, where e_2^i is the encryption of zero if $x_i \in X_2$ and the encryption of x_i otherwise. The obtained encryption e_2^i is then randomized by P_2 before being sent back to P_i, which is used as the input to the next mOT between P_1 and P_i.

Using the above mOT, we can remove the intersection items between X_2 and $X_{i>2}$. Therefore, when P_1 with input X_1 and P_3 with input $\big(x_3, \mathsf{Enc}(\mathsf{pk}, 0), e_2^3\big)$ execute an mOT, the receiver P_1 does not obtains the encryption of the items x_3 that is in the intersection $X_2 \cap X_3$. Concretely, $e_2^3 = \mathsf{Enc}(\mathsf{pk}, 0)$ as $x_3 \in X_2$.

At this point, if the P_4 repeats the previous step performed by P_3 using the encryption e_2^4 as the input to mOT with the receiver P_1, there is a risk that the P_1 might obtain the encryption of the same item twice if $x_3 = x4$. Therefore, it is necessary for P_4 executes an mOT with P_3 to remove the intersection between X_3 and X_4.

In summary, our protocol can be viewed as a sequence of mOT instances between P_1 and P_i, where P_1 has input X_1 while P_i inputs is $\big(x_i, \mathsf{Enc}(\mathsf{pk}, 0), e_{i-1}^i\big), \forall i \in X_i$. Here e_{i-1}^i is the result obtained from a sequence of mOT executions between between P_i and each $P_{t\in[2:i-1]}$. This process helps to eliminate duplications between X_i and X_t.

Upon the completion of $(n-1)$ instances of the mOT protocol between the leader party P_1 and other parties P_i, the leader P_1 has acquired an encryption set E, containing encryptions $\mathsf{Enc}(\mathsf{pk}, x)$ for $x \in \bigcup_{i=2}^n X_i$ and τ encryptions of zero, where $\tau = \sum_{i=1}^n |X_i| - |\bigcup_{i=1}^n X_i|$ indicates the number of duplicate items. In order to satisfy requirement (3) of the mPSU protocol, we employ the Shuffle&Decrypt functionality, which permits each party to apply its own permutation function on the encryption set E.

We demonstrate our mPSU protocol execution in Table 2 pertaining to the 4-party scenario described above. Our protocol maintains security even if some parties collude. For example, the adversary cannot determine the numbers of zero encryption before Shuffle&Decrypt execution due to the IND-CPA security provided by the multi-key encryption.

At this stage, we are currently focusing on a simple scenario where each $P_{i\in[2,n]}$ possesses only one item. In order to generalize our method to a set X_i, we apply a popular technique known as the bin-and-ball technique. At the high level, the party $P_{i\in[2,n]}$ places its input values into β bins through the use of Cuckoo hashing, where each bin is allowed to contain at most one item. The leader P_1 utilizes the same set of Cuckoo hash functions to map the input values in S into β bins using Simple hashing. The mapping allows the parties to execute the simple case above bin-by-bin efficiently. As a result, for each bin, the P_1 obtains encryptions of the partial union set which are subsequently combined into a big encryption set E before being subjected to decryption.

4.2 Correctness and Security

Correctness. We consider three following cases depending on whether a specific item $x_{i,k} \in X_i$ of the smallest-index party P_i is in P_1 or other parties P_t for $n \geq t > i > 1$. Since P_i is the smallest index that has $x_{i,k}$, no previous parties have $x_{i,k}$. Thus, P_i obtains $e_{b,i-1}^i = \mathsf{Enc}(\mathsf{pk}, x_{i,k})$ after interacting with $P_{t\in[2,i-1]}$ via a sequence of the mOT instances.

- Case 1 ($x_{i,k} \in X_1$) – the P_1 has $x_{i,k}$: As $x_{i,k} \in X_1$, the mOT with P_i in Step (3,a) gives P_1 the encryption of zero $\mathsf{Enc}(\mathsf{pk}, 0)$. As a result, $x_{i,k}$ does not appear in the final result from the Shuffle&Decrypt execution.
- Case 2 ($x_{i,k} \notin X_1$ and $x_{i,k} \in X_t$) – the P_1 does not have $x_{i,k}$, but another party P_t has $x_{t,j} = x_{i,k}$ for $t > i$: The mOT execution between P_i and P_t in Step (3,b), on input including $e_{b,i-1}^t$), provides P_i with the encryption of zero. This encryption is then rerandomized before being sent back to P_t. In other words,P_t obtains $e_{b,i}^t$) $= \mathsf{Enc}(\mathsf{pk}, 0)$ after Step (3,b). Thus, when executing with P_1 in the following round, P_1 obtains $\mathsf{Enc}(\mathsf{pk}, 0)$, ensuring that the $x_{i,k}$ will appear in the final union output.
- Case 3 ($x_{i,k} \notin \bigcup_{j=1,j\neq i}^n X_j$) – no party has $x_{i,k}$: The mOT executions between P_i and P_t in Step (3) maintain the encryption of $\mathsf{Enc}(\mathsf{pk}, x_{i,k})$. Thus, P_1 then obtains an $\mathsf{Enc}(\mathsf{pk}, x_{i,k})$ from P_t in Step (3,a). Consequently, $x_{i,k}$ appears in the final result from the Shuffle&Decrypt functionality.

Security. The security of our mPSU protocol is given as below. At a high level, the multi-key encryption system is secure because all the information viewed by a malicious adversary remains encrypted when the mOT is secure.

	$P_1 : X_1$	$P_2 : \{x_2\}$	$P_3 : \{x_3\}$	$P_4 : \{x_4\}$
Round 1	$\Leftarrow$ mOT $\Rightarrow$	$\Leftarrow$ mOT $\Rightarrow$		
		$\Leftarrow$	mOT	$\Rightarrow$
E	$\{\mathsf{Enc}(x_2)\}$			
Round 2	$\Leftarrow$	mOT	$\Rightarrow$	$\Leftarrow$ mOT $\Rightarrow$
E	$\{\mathsf{Enc}(x_2), \mathsf{Enc}(0)\}$			
Round 3	$\Leftarrow$	mOT		$\Rightarrow$
E	$\{\mathsf{Enc}(x_2), \mathsf{Enc}(0), \mathsf{Enc}(x_4)\}$			
Shuffle	$\{\mathsf{Enc}(0), \mathsf{Enc}(x_4), \mathsf{Enc}(x_2)\}$			
Decrypt	$\{0, x_4, x_2\}$			
Output	$X_1 \cup \{x_4, x_2\}$			

Table 2: Illustration of our mPSU protocol for 4 parties. P_1 has an input set of X_1 while P_i have input set of only one item x_i for $i \in [2, 4]$. In addition, we assume that $x_2 = x_3 \notin X_1$ and $x_4 \in X_1$. $\Leftarrow \mathsf{P} \Rightarrow$ denotes the execution of protocol P between two parties. Colors indicating the corresponding output for each invocation of protocol. For example, in round 1, P_1 and P_2 invoke the mOT (Step (3,a) in Figure 8). P_1 updates its set E by the received the message $\mathsf{Enc}(x_2)$) from P_2. P_3 and P_4 invoke mOT protocol with P_2 concurrently to update their encryption (Step (3,b) in Figure 8). P_3 receives an encryption of zero $\mathsf{Enc}(0)$ since $x_3 = x_2$ and P_4 receives the same encryption of x_4 as $\mathsf{Enc}(x_4)$. The following rounds are similar.

Theorem 5. *Given the multi-key cryptosystem, mOT and Shuffle&Decrypt functionalities described in Section 2.5, Figure 4, and Figure 6, respectively, the mPSU protocol described in Figure 8 securely implements the mPSU functionality defined in Figure 1 in the semi-honest model, against any number of corrupt, colluding, semi-honest parties.*

Proof. Let C and H be a coalition of corrupt and honest parties, respectively. We must show how to simulate C's view in the ideal model. We consider three following cases based on whether C has an item x:

(1) C does not have x, but H has x: We consider two cases. First, if H contains only one honest party P_i, then P_i has x. Consider the case where P_i is P_1. During the protocol execution, P_1 only acts as the receiver via mOT in Step (3,a) and participates in the shuffle and decryption in Step (4). Assuming that these two building blocks are secure, P_1 does not reveal anything to C. If $i \neq 1$, the corrupted parties C should contain the leader P_1, thus, they can deduce that the honest party P_i has x from the output of the set union. Hence, there is nothing to hide about whether P_i has x in this case. Second, if H has more than one honest party (say P_i and $P_{j>i}$). We consider two following subcases:
   - Only P_i has x: we must show that the protocol must hide the identity of P_i. If $P_1 \in H$, only the honest party P_1 learns the union $\bigcup_{i=1}^{n} X_i$ in Step 5. In addition, the mOT between P_i and previous corrupt parties $P_t \in C$ reveals nothing to C as the obtained output is under the multi-key encryption and rerandomized before the next execution. Thus, the simulation is simple.
     If $P_1 \in C$, the corrupt P_1 obtains $\mathsf{Enc}(\mathsf{pk}, x)$ from P_i. Since the encryption is protected under the Shuffle&Decrypt functionality until the P_1 learns the union sets which was permuted by the honest party P_i, the encryption reveals nothing to C.
   - Both P_i and P_j have x: If $i = 1$, then the honest leader P_1 receives encryptions of zeros $\mathsf{Enc}(\mathsf{pk}, 0)$ when executing mOT with P_j. If another party $P_{t>j}$ possesses x, the mOT execution between P_j and P_t results in P_t receiving the $\mathsf{Enc}(\mathsf{pk}, 0)$, while P_t learns nothing. Thus, when doing the permutation in Step 5, the C learns nothing about which parties in H have x. If $P_1 \in C$, the corrupt P_1 receives the encryption $\mathsf{Enc}(\mathsf{pk}, x)$ from P_i and $\mathsf{Enc}(\mathsf{pk}, 0)$ from P_j. Similarly, thanks to the CCA property of the encryption scheme and the permutation in Shuffle&Decrypt, C cannot distinguish the two encryptions. Thus, the protocol hides the identity of which honest party has x.

(2) C have x, but H does not have x: We must show that the protocol must hide the information that H does not have x. Consider the mOT execution where a party in H acts as the sender and a party in C acts as the receiver, the corrupt set C receives $\mathsf{Enc}(\mathsf{pk}, x)$ which is rerandomized by H. Thus, C learns nothing. In the final step, the encryption set E contains $\mathsf{Enc}(\mathsf{pk}, x)$, which was permuted by the honest parties H. Hence, all honest parties have an indistinguishable effect on the Shuffle&Decrypt step.

(3) Both C and H have x. When C acts as the receiver and invokes the mOT with an honest sender, if the sender's keyword x is not in the receiver's set, the receiver obtains the encryption of the keyword x. Otherwise, the receiver obtains the encryption of zero. The C cannot differentiate between the two cases. When C acts as the sender in mOT, C obtains nothing but might receive the rerandomization of the output from the receiver in Step (3,b). Since the message was rerandomized by H, C cannot infer the underlying encryption. Moreover, $\mathsf{Enc}(\mathsf{pk}, x)$ appears only once in the encryption set E, so the C learns nothing about whether the H has x.

□

PARAMETERS:

- n parties $P_{i\in[n]}$ for $n > 1$.
- The mOT and Shuffle&Decrypt functionalities described in Figure 4 and Figure 6, respectively.
- A multi-key cryptosystem (KeyGen, Enc, ParDec, FulDec, ReRand) defined in Section 2.5.
- Hashing parameters: a number of bins μ, maximum bin sizes $\beta : \mathbb{Z} \to \mathbb{Z}$ for simple-hashing bins, the h hash functions $H_{j\in[h]} : \{0,1\}^\star \to [\mu]$.

INPUT:

- Party $P_{i\in[n]}$ has $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$.

PROTOCOL:

(1) All n parties call the key generation algorithm $\mathsf{KeyGen}(1^\lambda, 1^\kappa)$. Each P_i receives a private key sk_i and a joint public key pk.
(2) Local Execution:
  (a) $P_{i\in[2,n]}$ hashes items X_i into μ bins using the Cuckoo hashing. Let C_b^i denote the items in the P_i's bth bin. P_i computes the encryption $e_{b,1}^i = \mathsf{Enc}(pk, C_b^i)$, for $b \in [\mu]$.
  (b) $P_{i\in[n]}$ hashes X_i into μ bins under k hash functions. Let S_b^i denote the set of items in the P_i's bth bin. P_i pads S_b^i with dummy values to the maximum bin size β.
  (c) For bin $b \in [\mu]$, the P_1 initials an empty set E_b.
(3) P_1 sequentially interacts with P_i for $i \in [2, n]$ as follow.
  (a) For each bin $b \in [\mu]$, P_1 and P_i invoke a mOT instance where:
    - P_1 acts as the receiver with input S_b^1.
    - P_i acts as the sender with input $(C_b^i, \mathsf{Enc}(\mathsf{pk}, 0), \bar{e}_b)$. Here, $\bar{e}_b = \mathsf{Enc}(\mathsf{pk}, 0)$ if $C_b^i = \varnothing$, otherwise, $\bar{e}_b = e_{b,i-1}^i$.
    - P_1 obtains e_b.

    P_1 appends e_b to E_b.
  (b) For each bin $b \in [\mu]$, the P_i and $P_{t\in[i+1,n]}$ invoke a mOT instance where:
    - P_i acts as the receiver with input S_b^i
    - P_t acts as the sender with input $(C_b^t, \mathsf{Enc}(\mathsf{pk}, 0), e_{b,i-1}^t)$.
    - P_i obtains c and sends $c' = \mathsf{ReRand}(c, \mathsf{pk})$ to P_t

    P_t computes $e_{b,i}^t = \mathsf{ReRand}(c', \mathsf{pk})$
(4) All the parties invoke the Shuffle&Decrypt functionality where:
  - P_1 inputs $E = \bigcup_{b=2}^{\mu} E_b$, the sk_1 and a random permutation $\pi_1 : [m] \to [m]$.
  - P_i inputs the private key sk_i and a random permutation $\pi_i : [m] \to [m]$.
  - P_1 obtains a set U.
(5) P_1 removes all zero from U, and outputs $U \cup X_1$.

Figure 8: Our mPSU Protocol

4.3 Complexity

We presented the communication, computation, and round complexities of our mPSU protocol in Figure 1 and elaborate on them here. It is clear that our protocol has n rounds for both Step (3) and Step (4). Leveraging the bin-and-ball technique introduced in [39, 42], parties hash elements into Cuckoo and Simple hashing tables consisting of $O(m)$ bins. Each bin of the Simple hashing table accommodates up to $O(\log m/\log\log m)$ elements. In round $i-1$, party P_i engages in mOT with P_1 and $P_{t>i}$, each incurring a cost of $O(\log m/\log\log m)$ in terms of communication and computation per bin. This yields a total cost of $O(n^2 m \log m/\log\log m)$.

Remark. In our protocol, the non-linearity with respect to m comes from the mOT, specifically the SS-PMT. We discuss the construction in Appendix A. [14, 34] proposed SS-PMT protocol with linear complexity. By incorporating these new constructions into our mOT, our mPSU framework can achieve linear complexity in terms of the number of parties and set size.

5 IMPLEMENTATION AND PERFORMANCE

We implement our protocol and evaluate it with various number of parties, set sizes, and number of threads. All evaluations were performed with an item input length of 128 bits, a statistical security parameter $\lambda = 40$, and a computational security parameter $\kappa = 128$. We do a number of experiments on a single server that has AMD EPYC 74F3 processors and 256GB of RAM. We run all parties in the same network, but simulate a network connection using the Linux tc command: a LAN setting with 0.02ms round-trip latency, 10 Gbps network bandwidth; two WAN settings: WAN_1 with 10ms and WAN_2 with 80ms round-trip latency, and both have 400 Mbps network bandwidth.

Our mPSU protocol is built on ElGamal encryption scheme (multi-key cryptosystem), SS-PMT, and OT (mOT). We implement the multi-key ElGamal encryption scheme using the elliptic curve code (Curve25519) from Relic [45]. For the SS-PMT implementation which requires garbled circuit for two strings comparison, we use the EMP-toolkit library [50]. Finally, we use the OT-extension [27]

Our Protocol		$m = 2^8$			$m = 2^{12}$			$m = 2^{16}$			$m = 2^{20}$		
		$t = 1$	$t = 4$	$t = 16$	$t = 1$	$t = 4$	$t = 16$	$t = 1$	$t = 4$	$t = 16$	$t = 1$	$t = 4$	$t = 16$
LAN (s)	$n = 3$	1.10	0.35	0.23	16.49	4.33	1.56	284.47	73.57	20.77	4546.74	1179.99	337.02
	$n = 4$	1.88	0.58	0.34	27.96	7.36	2.25	490.53	127.14	35.38	7841.32	2038.63	573.13
	$n = 6$	3.94	1.18	0.60	59.65	15.52	4.58	1061.45	271.60	74.80	16971.22	4352.98	1208.59
	$n = 8$	6.72	1.94	0.92	103.86	26.62	7.65	1838.59	470.44	127.95	29400.65	7537.26	2063.72
WAN_1 (s)	$n = 3$	5.03	4.45	4.37	24.58	12.66	10.02	309.70	98.93	46.18	4854.77	1488.01	645.04
	$n = 4$	8.38	7.16	6.94	39.76	19.37	14.40	528.25	165.07	73.39	8302.03	2499.34	1033.85
	$n = 6$	15.54	12.80	12.22	79.51	35.75	24.99	1124.23	334.73	138.05	17737.32	5119.07	1974.68
	$n = 8$	23.11	18.38	17.40	129.62	54.58	36.07	1926.38	558.72	216.41	30472.12	8608.74	3135.19
WAN_2 (s)	$n = 3$	9.49	8.91	8.83	31.61	19.69	17.05	336.84	126.08	73.33	5099.44	1732.69	889.71
	$n = 4$	15.07	13.85	13.63	50.30	29.91	24.94	568.27	205.09	113.41	8666.94	2864.25	1398.75
	$n = 6$	26.69	23.95	23.37	97.07	53.31	42.55	1189.99	400.50	203.82	18342.69	5724.45	2580.06
	$n = 8$	38.72	33.99	33.01	154.20	79.17	60.65	2017.89	650.23	307.92	31304.27	9440.89	3967.34
Comm. Cost (MB)	$n = 3$	1.46			20.25			321.40			5142.39		
	$n = 4$	2.20			30.48			483.69			7739.04		
	$n = 6$	3.68			50.96			808.79			12940.64		
	$n = 8$	5.16			71.47			1134.21			18147.41		

Table 3: The running time and communication cost of our mPSU protocol: the number of parties $n \in \{3, 4, 6, 8\}$, set size $m \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$, and numbers of thread $t = \{1, 4, 16\}$. The reported running time represents the time taken for the entire protocol to complete. Communication cost is computed as the average cost across all parties.

		P_1			P_2			P_3			P_4		
		Comm.Cost	Running Time		Comm.Cost	Running Time		Comm.Cost	Running Time		Comm.Cost	Running Time	
			LAN	WAN_2		LAN	WAN_2		LAN	WAN_2		LAN	WAN_2
$m = 2^8$	**Total**	2.18	1.88	15.07	2.21	1.88	15.07	2.21	1.88	15.07	2.21	1.88	15.07
	mOT	2.11	0.86	13.07	2.13	0.86	13.07	2.13	0.86	13.07	2.13	0.86	13.07
	Shuffle&Decrypt	0.08	1.02	2.00	0.08	1.02	2.00	0.08	1.02	2.00	0.08	1.02	2.00
$m = 2^{12}$	**Total**	30.18	27.96	50.30	30.58	27.96	50.30	30.58	27.96	50.30	30.58	27.96	50.30
	mOT	28.99	11.91	30.98	29.39	11.91	30.98	29.39	11.91	30.98	29.39	11.91	30.98
	Shuffle&Decrypt	1.19	16.05	19.32	1.19	16.05	19.32	1.19	16.05	19.32	1.19	16.05	19.32
$m = 2^{16}$	**Total**	478.92	490.53	568.27	485.28	490.53	568.27	485.28	490.53	568.27	485.28	490.53	568.27
	mOT	459.88	187.23	258.45	466.24	187.23	258.45	466.24	187.23	258.45	466.24	187.23	258.45
	Shuffle&Decrypt	19.04	303.31	309.82	19.04	303.31	309.82	19.04	303.31	309.82	19.04	303.31	309.82

Table 4: The breakdown running time and communication cost for each party in our mPSU protocol ($n = 4$).

provided in [38] to implement mOT. Our complete implementation is available on GitHub.

Our protocol scales well using multi-threading between the parties. In each round, the party $P_{i\in[2,n-1]}$ can use $n - i + 1$ threads so that each party operates mOT building block with other parties P_1 and $P_{j\in[i+1,n]}$ at the same time. In addition, each pair of parties can use multiple threads to execute these building blocks bin-by-bin in parallel. We evaluate it on the number of threads $t \in \{1, 4, 16\}$ to show the performance of our protocols running with multi-threading.

5.1 Performance of Our mPSU Protocol

Table 3 presents the overall runtime and communication overhead of our mPSU protocol. From the empirical numbers, we can see that the performance difference between WAN_1, WAN_2, and LAN is primarily due to the latency for the smaller input. The gap increases with the number of parties which is also observed in other protocols with an $O(n)$ or higher round complexity.

Additionally, we present the breakdown cost of our protocol for each party in 4-party scenarios with varying set sizes in Table 4. Specifically, we present the performance metrics of the mOT in Step (3) and the Shuffle&Decrypt in Step (4) in our protocol in terms of running time and communication cost. All reported running time values in Table 3 and Table 4 represent end-to-end time.

5.2 Comparison with Previous Work

To demonstrate the performance of our mPSU protocols with a comparison, we have implemented the semi-honest protocols proposed in [6, 16] and estimate the performance for protocol proposed in [22]. Table 5 and Figure 9 present the running time and communication cost of various mPSU protocols [6, 16, 22] which are secure in the dishonest majority [1] and semi-honest setting. We do not incorporate the results from [49] into our comparison, as their protocol only works for a small universe. Even in their largest setting, with a universe size of 2^{32}, it is considerably smaller than the general

[1]The recent mPSU protocol [34] provides a weak security guarantee wherein the leader does not collude with any parties.

scenario involving 128-bit elements. According to [49, Figure 7], in a scenario involving 5 parties, each with only 32 elements of 32-bit length, their protocol takes around 10 seconds. Interestingly, this is comparable to the runtime of our protocol involving 6 parties, each with 256 elements in a 128-bit universe.

In [6], each input set X_i is initially shared among n parties using a secret-sharing scheme. Subsequently, these parties employ a generic secure computation technique to compute the union on the shares. Our implementation of the [6]'s method, however, is limited to the two-party scenario where each X_i is secret-shared between only two parties (which is in favor of [6]). Consequently, the secure computation takes place exclusively between these two parties. We implement [6] using EMP-toolkit library [50] which provides most of the state-of-the-art techniques for two-party secure computation in the semi-honest setting. As shown in Table 5, for $n = 4$, our protocol is $1.44 - 3.22$ times faster in the LAN setting and $8.50 - 80.84\times$ faster than [6] in the WAN_2. Additionally, the cost for [6] is significantly $(168.33 - 405.73\times)$ higher than our protocol for set size $m \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$.

We report the partial running time and communication cost of the mPSU protocol proposed by [22]. The first step of their protocol is for each party to locally compute an encryption of a local Bloom filter. To achieve a false positive rate of 2^{-40}, the table size should be at least $60nm$. We estimate the time and communication cost for this single step of each party based on the performance shown in [35] (as well as our [16]'s implementation), where each Paillier encryption takes about 2.5 ms with a key length of 2048 bits, and report the numbers in Table 5.

Our mPSU protocol outperforms previous works in the LAN setting. Despite the low communication cost due to the usage of homomorphic encryption, the running time of [16, 22] is not practical even for small set sizes. Thus, we skip the evaluation of the [16, 22] in the WAN setting. We also show the concrete number for each protocol in Table 5. [22] has a constant round of 7 and [6] has a round complexity of $O(\log(nm))$ sensitive to m. Both [16] and our protocol gives $O(n)$ rounds independent of input size. We believe that our protocol provides the best trade-off between the running time, bandwidth cost, and round complexity.

	m	Ours	[22]	[6]	[16]
Running Time LAN (second)	2^8	**1.88**	155.63	2.70	6009.00
	2^{12}	**27.96**	2490.04	57.73	-
	2^{16}	**490.53**	39840.65	1158.53	-
	2^{20}	**7841.32**	637450.40	25279.39	-
Running Time WAN_2 (second)	2^8	**15.07**	-	128.05	-
	2^{12}	**50.30**	-	2387.70	-
	2^{16}	**568.27**	-	45939.46	-
Comm. Cost (MB)	2^8	8.80	15.72	1481.34	**4.25**
	2^{12}	121.92	251.65	30116.50	**68.00**
	2^{16}	1934.76	4026.53	617047.00	**1088.00**
	2^{20}	30956.16	64424.48	12559743.00	**17408.00**
Number of Rounds	2^8	17	**7**	10	11
	2^{12}			14	
	2^{16}			18	

Table 5: Performance comparison of different mPSU protocols with $n = 4$ parties, each having $m \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$. The communication cost is computed as the overall cost across all parties. Concrete number of round is given here. The numbers for [6] are lower-bounds based on complexity $O(\log(nm))$ in Table 1.

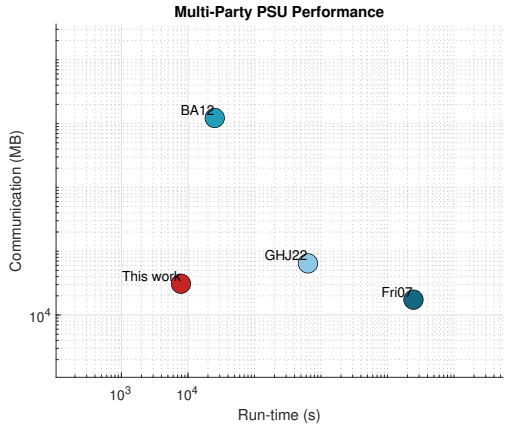

Figure 9: Multi-party PSU protocols with set size of 2^{20} among 4 parties on the network 10Gbps with 0.02ms latency.

6 CONCLUSION

In this work, we propose an efficient mPSU protocol in the semi-honest setting against an adversary that colludes an arbitrary number of participants. Our protocol is built on mOT which we believed of independent interests. Our protocol significantly outperforms prior mPSU works in the same security setting in terms of running time and communication cost. Our mPSU framework is the generalization of the well-studied 2-party PSU protocols to the multi-party setting. We highlight some directions for future work:

- Improving scalability: Unlike the 2-party PSU and some other efficient private set intersection protocols, our protocol still heavily relies on public key techniques which is the bottleneck of the performance. It is possible to use symmetric-key techniques such as secret-sharing to replace the encryption scheme and implement Shuffle&Decrypt. We leave the mPSU protocol constructed mainly on the symmetric key techniques as the future work.
- This study concentrates on semi-honest mPSU, which we consider a preliminary stage in advancing towards efficient malicious MPSU. To the best of our knowledge, there is no existing tailored protocols for malicious PSU in both two-party and multi-party settings. To achieve malicious PSU, one can employ cryptographic commitment techniques at each step of the protocol, albeit with added costs.

ACKNOWLEDGEMENT

The authors were partially supported by NSF awards #2101052, #2115075, and ARPA-H SP4701-23-C-0074. We are grateful to the PoPETs 2024 anonymous reviewers for their invaluable feedback. Dong et al. [14] identified a security flaw in our original protocol against arbitrary collusion through a valid attack. We appredicate them for pointing out a bug in the original version.

REFERENCES

[1] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 483–501. https://doi.org/10.1007/978-3-642-29011-4_29

[2] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient Garbling from a Fixed-Key Blockcipher. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 478–492. https://doi.org/10.1109/SP.2013.39

[3] Assaf Ben-David, Noam Nisan, and Benny Pinkas. 2008. FairplayMP: a system for secure multi-party computation. In *ACM CCS 2008*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, 257–266. https://doi.org/10.1145/1455770.1455804

[4] Alex Berke, Michiel Bakker, Praneeth Vepakomma, Kent Larson, and Alex 'Sandy' Pentland. 2020. Assessing Disease Exposure Risk with Location Data: A Proposal for Cryptographic Preservation of Privacy. arXiv:2003.14412 [cs.CR]

[5] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 301–318. https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock

[6] Marina Blanton and Everaldo Aguiar. 2012. Private and oblivious set and multiset operations. In *ASIACCS 12*, Heung Youl Youm and Yoojae Won (Eds.). ACM Press, 40–41.

[7] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50

[8] Justin Brickell and Vitaly Shmatikov. 2005. Privacy-Preserving Graph Algorithms in the Semi-honest Model. In *ASIACRYPT 2005 (LNCS, Vol. 3788)*, Bimal K. Roy (Ed.). Springer, Heidelberg, 236–252. https://doi.org/10.1007/11593447_13

[9] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. 2020. Private Matching for Compute. Cryptology ePrint Archive, Paper 2020/599. https://eprint.iacr.org/2020/599 https://eprint.iacr.org/2020/599.

[10] Dung Bui and Geoffroy Couteau. 2023. Improved Private Set Intersection for Sets with Small Entries. PKC. https://eprint.iacr.org/2022/334 https://eprint.iacr.org/2022/334.

[11] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. 2010. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *19th USENIX Security Symposium (USENIX Security 10)*. USENIX Association, Washington, DC. https://www.usenix.org/conference/usenixsecurity10/sepia-privacy-preserving-aggregation-multi-domain-network-events-and

[12] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 1182–1204. https://doi.org/10.1145/3460120.3484591

[13] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. 2018. PIR-PSI: Scaling Private Contact Discovery. Cryptology ePrint Archive, Paper 2018/579. https://eprint.iacr.org/2018/579 https://eprint.iacr.org/2018/579.

[14] Minglang Dong, Yu Chen, Cong Zhang, and Yujie Bai. 2024. Breaking Free: Efficient Multi-Party Private Set Union Without Non-Collusion Assumptions. arXiv:2406.07011 [cs.CR] https://arxiv.org/abs/2406.07011

[15] Taher ElGamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84 (LNCS, Vol. 196)*, G. R. Blakley and David Chaum (Eds.). Springer, Heidelberg, 10–18.

[16] Keith B. Frikken. 2007. Privacy-Preserving Set Union. In *ACNS 07 (LNCS, Vol. 4521)*, Jonathan Katz and Moti Yung (Eds.). Springer, Heidelberg, 237–252. https://doi.org/10.1007/978-3-540-72738-5_16

[17] Jiahui Gao, Son Nguyen, and Ni Trieu. 2023. Toward A Practical Multi-party Private Set Union. Cryptology ePrint Archive, Paper 2023/1930. https://eprint.iacr.org/2023/1930 https://eprint.iacr.org/2023/1930.

[18] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. 2021. Private Set Operations from Oblivious Switching. In *Public-Key Cryptography – PKC 2021*, Juan A. Garay (Ed.). Springer International Publishing, Cham, 591–617.

[19] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In *CRYPTO 2021, Part II (LNCS, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 395–425. https://doi.org/10.1007/978-3-030-84245-1_14

[20] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[21] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420

[22] Xuhui Gong, Qiang-Sheng Hua, and Hai Jin. 2022. Nearly Optimal Protocols for Computing Multi-party Private Set Union. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS54832.2022.9812897

[23] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. 2022. Contact Discovery in Mobile Messengers: Low-Cost Attacks, Quantitative Analyses, and Efficient Mitigations. *ACM Trans. Priv. Secur.* 26, 1, Article 2 (nov 2022), 44 pages. https://doi.org/10.1145/3546191

[24] Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems* 24, 2 (2009), 8–12. https://doi.org/10.1109/MIS.2009.36

[25] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. 2016. Secure Multiparty Computation for Cooperative Cyber Risk Assessment. In *2016 IEEE Cybersecurity Development (SecDev)*. 75–76. https://doi.org/10.1109/SecDev.2016.028

[26] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 370–389. https://doi.org/10.1109/EuroSP48549.2020.00031

[27] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003 (LNCS, Vol. 2729)*, Dan Boneh (Ed.). Springer, Heidelberg, 145–161. https://doi.org/10.1007/978-3-540-45146-4_9

[28] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. 2022. Shuffle-based Private Set Union: Faster and More Secure. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2947–2964. https://www.usenix.org/conference/usenixsecurity22/presentation/jia

[29] M. Kantarcioglu and C. Clifton. 2004. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering* 16, 9 (2004), 1026–1037. https://doi.org/10.1109/TKDE.2004.45

[30] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *CRYPTO 2005 (LNCS, Vol. 3621)*, Victor Shoup (Ed.). Springer, Heidelberg, 241–257. https://doi.org/10.1007/11535218_15

[31] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. In *ASIACRYPT 2019, Part II (LNCS, Vol. 11922)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, Heidelberg, 636–666. https://doi.org/10.1007/978-3-030-34621-8_23

[32] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *Automata, Languages and Programming*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 486–498.

[33] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. 2021. Private Join and Compute from PIR with Default. In *ASIACRYPT 2021, Part II (LNCS, Vol. 13091)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Heidelberg, 605–634. https://doi.org/10.1007/978-3-030-92075-3_21

[34] Xiang Liu and Ying Gao. 2023. Scalable Multi-party Private Set Union from Multi-Query Secret-Shared Private Membership Test. Cryptology ePrint Archive, Paper 2023/1413. https://eprint.iacr.org/2023/1413 https://eprint.iacr.org/2023/1413.

[35] Huanyu Ma, Shuai Han, and Hao Lei. 2021. Optimized Paillier's Cryptosystem with Fast Encryption and Decryption. In *Annual Computer Security Applications Conference* (Virtual Event, USA) *(ACSAC '21)*. Association for Computing Machinery, New York, NY, USA, 106–118. https://doi.org/10.1145/3485832.3485842

[36] Ofri Nevo, Ni Trieu, and Avishay Yanai. 2021. Simple, Fast Malicious Multiparty Private Set Intersection. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 1151–1165. https://doi.org/10.1145/3460120.3484772

[37] Duong Tung Nguyen and Ni Trieu. 2021. MPCCache: Privacy-Preserving Multi-Party Cooperative Cache Sharing at the Edge. Cryptology ePrint Archive, Report 2021/317. https://eprint.iacr.org/2021/317.

[38] Lance Roy Peter Rindal. 2021. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[39] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security 2015*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 515–530.

[40] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT 2019, Part III (LNCS, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, 122–153. https://doi.org/10.1007/978-3-030-17659-4_5

[41] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT 2018, Part III (LNCS, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer,

Heidelberg, 125–157. https://doi.org/10.1007/978-3-319-78372-7_5
[42] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.* 21, 2, Article 7 (jan 2018), 35 pages. https://doi.org/10.1145/3154794
[43] Tal Rabin. 1998. A Simplified Approach to Threshold and Proactive RSA. In *CRYPTO'98 (LNCS, Vol. 1462)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 89–104. https://doi.org/10.1007/BFb0055722
[44] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 2505–2517. https://doi.org/10.1145/3548606.3560658
[45] RELIC. 2020. A modern research-oriented cryptographic meta-toolkit with emphasis on efficiency and flexibility. https://github.com/relic-toolkit.
[46] Jae Hong Seo, Jung Cheon, and Jonathan Katz. 2012. Constant-Round Multi-party Private Set Union Using Reversed Laurent Series, Vol. 7293. 398–412. https://doi.org/10.1007/978-3-642-30057-8_24
[47] Katsunari Shishido and Atsuko Miyaji. 2018. Efficient and Quasi-accurate Multi-party Private Set Union. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. 309–314. https://doi.org/10.1109/SMARTCOMP.2018.00021
[48] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. 2020. Epione: Lightweight Contact Tracing with Strong Privacy. *CoRR* abs/2004.13293 (2020). arXiv:2004.13293 https://arxiv.org/abs/2004.13293
[49] Jelle Vos, Mauro Conti, and Zekeriya Erkin. 2022. Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs. Cryptology ePrint Archive, Paper 2022/721. https://eprint.iacr.org/2022/721 https://eprint.iacr.org/2022/721.
[50] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit.
[51] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, 162–167. https://doi.org/10.1109/SFCS.1986.25
[52] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 220–250. https://doi.org/10.1007/978-3-662-46803-6_8
[53] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. 2023. Linear Private Set Union from Multi-Query Reverse Private Membership Test. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 337–354. https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-cong

A SECRET-SHARED PRIVATE MEMBERSHIP TEST

In this section, we present the secret-shared private membership test (SS-PMT) construction [19, 40] used in our mPSU implementation. The functionality of SS-PMT is given in Figure 3. SS-PMT is a two-party where P_0 have a set $X = \{x_1, \ldots, x_n\} \subset \{0,1\}^*$ and P_1 have an item $y \in \{0,1\}^*$. As the result, each $P_{i\in\{0,1\}}$ learns a bit b_i so that $b_0 \oplus b_1 = 1$ if $y \in X$ and $b_0 \oplus b_1 = 0$ otherwise. The security requirement of SS-PMT is that the participants only learn the desired output and nothing else.

The SS-PMT protocol can be built from the usage of oblivious key-value pair (OKVS) [19] with the help of generic equality test techniques such as garble circuit and secret sharing. A Key Value Store (KVS) consists of two algorithms: i) Encode takes as input a set of (k_i, v_i) key-value pairs from the key-value domain, $\mathcal{K} \times \mathcal{V}$, and outputs an object S (or, with negligible probability, an error indicator $\perp$); ii) Decode takes as input an object S, a key x and outputs a value y. A KVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys: i) $Pr[\mathsf{Encode}(A) = \perp]$ is negligible, and ii) if $\mathsf{Encode}(A) = S \neq \perp$ and $(k, v) \in A$ then $\mathsf{Decode}(S, k) = v$. We say that a KVS is oblivious, if the values v_i are chosen uniformly then the output of Encode hides the choice of the keys k_i. We refer to the [19] for more details about the security and other properties of OKVS.

The SS-PMT protocol is given in Figure 11. For each instance of SS-PMT execution, P_0 randomly chooses a secret value s and encodes an OKVS structure with key-value pairs $\{(x_1, s), \ldots, (x_n, s)\}$. P_0 send this OKVS structure to P_1 and P_1 learns a decoded value s' with input x_1. Then P_0 and P_1 invoke a two-party computation protocol for equality check, which functionality is described in Figure 10. The protocol of equality check can be constructed by garbled circuit (GC) [21, 51] with several optimizations such as point-and-permute [3], Free-XOR [32], the half-gate [52], and fixed-key AES garbling optimizations [2].

To enable multi-point query which is desirable in our mPSU protocol. Naively, for P_0 with set $X = \{x_1, \ldots, x_m\}$ and P_1 with set $Y = \{y_1, \ldots, y_m\}$, we would like to learn the share of a sequence of bits $b_{1,0}, \ldots, b_{m,0}$ and $b_{1,1}, \ldots, b_{m,1}$ for P_0 and P_1 such that $b_{i,0} \oplus b_{i,1} = 1$ if $x_i \in Y$ and 0 otherwise. We execute the SS-PMT protocol following the methods in [18, 40]. Instead of invoking m times SS-PMT protocol between P_0 with $x_i \in X$ for $i \in [1, m]$ and P_1 with Y, we pre-process the set of with cuckoo and simple hashing scheme described in Section 2.4. Then the SS-PMT protocol is performed for each bin of the hash table. Very recently, [34] proposed a multi-query SS-PMT protocol. The performance of our mPSU protocol can be improved by using their SS-PMT protocol. For easy of implementation, we opt for the SS-PMT construction described above.

Parameters: The Boolean circuit C computing the equality for two strings from $\{0,1\}^\star$, with I_1, I_2 inputs and O_1, O_2 outputs associated with P_1 and P_2 resp.
Functionality:
- Wait for input $x_0 \in \{0,1\}^\star$ from P_0 and $x_1 \in \{0,1\}^\star$ from P_1.
- Give b_0 and b_1 to P_0 and P_1 respectively such that $b_0 \oplus b_1 = 1$ if $x_0 = x_1$ and 0 otherwise.

Figure 10: Secure Two-Party Computation for Equality Check

Parameters:
- P_0 and P_1, the receiver set size n, the PRF F

Input:
- P_0: $X = \{x_1, \ldots, x_m\} \subset (\{0,1\}^\star)^n$ and the PRF key k
- P_1: $y \in \{0,1\}^\star$

Protocol:
(1) P_0 chooses a random value $s \leftarrow \{0,1\}^\star$ and constructs an OKVS $S \leftarrow \mathsf{Encode}(\{(x_1, s), \ldots, (x_n, s)\})$ and sends S to P_1.
(2) The sender decode compute the value $s' = Decode(S, x)$.
(3) P_0 and P_1 invoke the protocol of secure two-party equality check protocol (Functionality 10) with input s and s' and learns the bit b_0 and b_1.

Figure 11: Secret-shared Private Membership Test Protocol [19, 40]