Toward A Practical Multi-party Private Set Union

Jiahui Gao * Son Nguyen * Ni Trieu *

March 16, 2024

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 tools, namely, Membership Oblivious Transfer (mOT) and Conditional Oblivious Pseudorandom Function (cOPRF). We believe that the mOT and cOPRF may be of independent interest.

We implement our mPSU protocol and evaluate their performance. Our protocol shows an improvement of up to 37.82× in term of running time and 389.85× bandwidth cost compared to the existing state-of-the-art protocols.

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 [21], contract tracing [43, 2], and contact discovery [19], and cache sharing in IoT [32]. Similarly, PSU has numerous practical use cases. For example, PSU can be used to implement Private-ID functionality [7], cyber risk assessment and management via joint IP blacklists and joint vulnerability data [20], private database supporting full join [27], association rule learning [24], joint graph computation [6], and aggregation of multi-domain network events [9].

Over the last decade, a substantial body of research [37, 39, 8] has focused on PSI, whereas PSU has received relatively little attention. The majority of present practical PSU protocols [27, 16, 46, 23, 3] 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.

*Arizona State University, {jgao76, snguye63, nitrieu}@asu.edu

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. 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 [4, 44], or homomorphic encryption [25, 15, 41, 18], 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 [10, 31] with practical implementations, one might think that mPSU can be computed directly from multi-party PSI using DeMorgan's Law as $\bigcup_{i=1}^{n} X_i = U \setminus \big(\bigcap_{i=1}^{n}(U \setminus X_i)\big)$, 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 [37, 39, 8] 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 violate the mPSU functionality.

At this point, one may wonder why these leakages are of concern. Consider the application of "Cyber risk assessment and management via joint IP blacklists and joint vulnerability data" [20] previously mentioned and elaborated in [27]: "Organizations aim to optimize their security updates to minimize vulnerabilities in their infrastructure. Crucial role in the above is played by joint lists of blacklisted IP addresses. At the same time, organizations are understandably reluctant to reveal details pertaining to their current or past attacks or sensitive network data". This application requires minimal leakage from PSU. Specifically, if we implement mPSU using pairwise 2-party PSU, it could potentially leak blacklisted IP addresses of each organization. E.g., if $X_2 = X_3$ and $|X_2 \cap X_1| = 0$, then P_1 learns X_3. Moreover, in certain scenarios, revealing the cardinality *count* of a subset of parties' inputs poses an issue. For example, when $n = 3$ and P_1 receives $\mathit{count} = 2$ for $x \notin X_1$, the P_1 can learn X_3 if $|X_3| = 1$. Another instance is if $\mathit{count} = n - 1$ for $x \notin X_1$, P_1 learns that everyone except him has x.

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 [16, 46, 23, 3], which follow the framework of [27] 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 duplications 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 [25], 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 [25] has $O(n^3m^2)$ computation complexity. Relying on the polynomial presentation technique, Frikken [15] proposed an efficient mPSU protocol that requires the $O(n^2 m \log(m))$ number of multiplications. [41] presented input sets using rational polynomial functions and reversed Laurent series. As a result, it showed a more efficient protocol than previous works [25, 15], but the protocol is secure up to $n/2$ corrupted parties.

Blanton and Aguiar [4] 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

Protocol	Overall Communication	Computation			Round
		Overall	# HE		
			Leader P_1	Party $P_{i\in[2,n]}$	
[25]	$O(n^2m)$	$O(n^3m^2)$	$O(nm^2)$	$O(nm^2)$	$O(n)$
[15]	$O(n^2m)$	$O(n^2m\log(m))$	$O(nm)$	$O(nm\log(m))$	$O(n)$
[4]	$O(n^2m\log^2(mn))$	$O(n^2m\log^2(mn))$	0	0	$O(\log(nm))$
[18]	$O(n^2m\lambda)$	$O(n^2m\lambda)$	$O(nm\lambda)$	$O(nm\lambda)$	$O(1)$
[44]	$O(n^2m\log\lvert\mathcal{U}\rvert)$	$O(n^2m\log\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$
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 [18] consists of λ due to the usage of the Bloom filter. All [25, 15, 18] use Paillier encryption to compute addition on the encryptions (#HE) while our protocol uses ElGamal encryption scheme to re-randomize the ciphertexts.

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 [4] is inefficient when the m and n are large.

Recently, [42, 18] 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 [42] makes use of an outsourcing server to compute G, [18] is built on homomorphic encryption (HE), which requires a homomorphic computation per each entry of G, and might need the expensive multi-key HE. Moreover, the BF-based approach is associated with a high false positive rate.

In another work, Vos *at. el.*[44] proposed private OR protocols and build mPSU protocols upon it. They consider 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 invoke 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 [29] 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 [15, 4, 18].

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, Diffie-Hellman-based PRF [12] from the elliptic curve, and ElGamal encryption. This is in contrast with prior protocols, which require expensive Paillier encryption on the polynomial set representation [25, 15] or each entry of the Bloom filter [18]. Additionally, our approach eliminates the need for the inefficient oblivious sort/OR operations and generic MPC of [4, 44].

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 party 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) and Conditional Oblivious Pseudorandom Function(cOPRF) to 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 [38], 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 $x_{i,j} \in \bigcup_{i=1}^{n} X_i$ 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. At this point, the union set 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 ElGamal encryption scheme (ref. Section 2.6). 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 3.3.

The brief overview of mOT executions above ignores many important concerns — in particular, how the P_1 obtains encryption of zero from $P_{j\in[i+1,n]}$ for a duplicated item x which P_i also has. We propose utilizing an Oblivious PRF (OPRF), wherein P_1 obliviously learns and maintains a list of PRF values for the union sets $\bigcup_{i=1}^{t} X_i, \forall t \in [2, n]$. The PRF values hide the actual input items and are used as the input into mOT, rather than the input sets as previously described. In our protocol, we require an extended variant of OPRF to prevent a coalition of corrupt parties from learning the partial union (we describe the attack explicitly in Section 4). Thus, we introduce a new gadget called Conditional OPRF (cOPRF). Similar to the classical OPRF, the cOPRF has the additional feature that the sender P_i has a set of elements X_i, and the receiver $P_{t>i}$ only obtains the correct/active PRF value if its input query x_t is not present in the sender's set X_i, and "fake" or inactive PRF value otherwise. Thus, the parties only need to maintain the list of active PRF values. We describe the cOPRF ideal functionality and its instantiation 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), Conditional OPRF (cOPRF), 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 [15, 4, 18]. For example, for four parties with data-set of 2^{16} item each, our mPSU protocol shows an improvement up to 37.82× in terms of running time and up to 306.45× less bandwidth requirement when compared to the state-of-the-art protocols. Our implementation will be made publicly available on GitHub.

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. We use $F(k, x)$ to denote the evaluation of x on a pseudorandom function $F : \{0,1\}^\kappa \times \{0,1\}^\star \to \{0,1\}^\kappa$ given key k.

In this work, we rely on Diffie-Hellman OPRF protocol [12] in which the PRF value of x has a form $\bar{F}(k, x) = H(x||0)^k$ for a random hash function $H : \{0,1\}^\star \to \{0,1\}^\kappa$. Moreover, within our mPSU protocol, we establish $F(k, x) = H(x||0)^k$ as the designated "active" PRF from cOPRF, which is used for managing the list of union items.

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.

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.

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

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 PRF

An oblivious pseudorandom function (OPRF) [14] is a 2-party protocol in which the sender has a PRF key k and the receiver learns $F(k, q)$. Here, F is a PRF, and q is a query input chosen by the receiver. Figure 9 formally presents a variant of the OPRF where the receiver obtains outputs of multiple chosen queries.

2.3 Oblivious Transfer

Oblivious Transfer (OT) is a fundamental primitive of secure computation, and introduced by Rabin [38]. 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 10 presents the OT functionality.

2.4 Secret-shared Private Membership Test

Secret-shared Private Membership Test ($\mathsf{SS\text{-}PMT}$) is the main building block in different applications [35, 28, 10, 36, 29, 46]. 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. $\mathsf{SS\text{-}PMT}$ 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 11 presents the $\mathsf{SS\text{-}PMT}$ functionality.

2.5 Bin-and-ball Scheme

Our protocols employ hashing schemes such as the Cuckoo and Simple hashing schemes [34, 37] to allocate items into bins. We review the basics of the Cuckoo hashing and Simple hashing schemes [34, 37] 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 [34, 11] 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 $(1 - 2^{-\lambda})$.

2.6 Multi-key Cryptosystem

We revise the multi-key cryptosystem that needs 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 ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{FulDec}, \mathsf{ReRand}$) 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 $\mathsf{pk} = \mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, \dots, \mathsf{sk}_n)$, where $\mathsf{Combine}$ 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 $\mathsf{ct} \leftarrow \mathcal{C}$ encrypted under the partial pubic key $\mathsf{pk}_A = \mathsf{Combine}(\{\mathsf{sk}_j \mid j \in A\})$ and outputs a ciphertext $\mathsf{ct}' \leftarrow \mathcal{C}$ 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 $\mathsf{ct} \leftarrow \mathcal{C}$ encrypted under pk and all the secret keys and outputs a message $\mathsf{m} \leftarrow \mathcal{M}$.

- **Re-randomization**: $\mathsf{ct}' \leftarrow \mathsf{ReRand}(\mathsf{ct}, \mathsf{pk})$. A re-randomization algorithm takes a ciphertext $\mathsf{ct} \leftarrow \mathcal{C}$ encrypted under pk and gives a new ciphertext $\mathsf{ct}' \leftarrow \mathcal{C}$ 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 [17, 1, 5]. Informally, the multi-key cryptosystem satisfies correctness if $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, ..., \mathsf{sk}_n, \mathsf{ct})$ or $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, \dots, \mathsf{sk}_{i-1}, \mathsf{sk}_{i+1}, \dots, \mathsf{sk}_n, \mathsf{ct}')$ for $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{ct}' = \mathsf{ParDec}(\mathsf{sk}_i, \mathsf{ct}_{\{1,\dots,i-1,i+1,\dots,n\}})$. 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[13].

A Construction While there are many multi-key cryptosystems, we choose ElGamal system[13] 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 $P_1, ..., P_n$.

- **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 $\mathsf{pk} = \mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_n) = g^{\Sigma_{i=1}^n \mathsf{sk}_i} = \prod_{i=1}^n h_i$.

- **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 can 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 $\mathsf{m} = \mathsf{FulDec}(\mathsf{sk}_1, \mathsf{sk}_2, ..., \mathsf{sk}_n; \mathsf{ct}) = \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\Sigma_{i=1}^n \mathsf{sk}_i}$.

- **Re-randomization**: To rerandomize a ciphertext ct encrypted under the pk, one can choose a random value $r' \leftarrow \{0,1\}^\kappa$, and compute $(\mathsf{ct}'_1, \mathsf{ct}'_2) = \mathsf{ReRand}((\mathsf{ct}_1, \mathsf{ct}_2), \mathsf{pk})$ where $\mathsf{ct}'_1 = \mathsf{ct}_1 \cdot g^{r'}$ and $\mathsf{ct}'_2 = \mathsf{ct}_2 \cdot \mathsf{pk}^{r'}$.

3 Our mPSU Building Blocks

We introduce three 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.

- The second gadget is called "Conditional OPRF" (cOPRF) which allows the receiver to obtain a PRF value of its query x if and only if x satisfies the pre-defined condition (i.e., membership test). The cOPRF eliminates duplicated items from the final mPSU output by giving the "fake" PRF value of the intersection item, which will be ignored in subsequent computations (i.e., an inactive PRF value).

- 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 2.6, 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.

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 2: Membership Oblivious Transfer (mOT) Ideal Functionality

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 $X = \{x_1, ..., x_m\} \in (\{0,1\}^\star)^m$. The mOT 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 their domain $\{0,1\}^\ell$, so that the membership of y in terms of X is also not revealed to the receiver. 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 2.

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 [38], 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 $y \in X$ 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 [35, 36], 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 3.

PARAMETERS:

- Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, the receiver set size m, the length ℓ.
- The OT and $\mathsf{SS\text{-}PMT}$ 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 $\mathsf{SS\text{-}PMT}$ functionality where:
   - $\mathcal{R}$ has an inputs 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 3: Membership Oblivious Transfer (mOT) Construction

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 proof of the below theorem in Appendix A.1.

Theorem 2. *The mOT protocol described in Figure 3 securely implements the mOT functionality defined in Figure 2 in the semi-honest setting, given the OT and $\mathsf{SS\text{-}PMT}$ functionalities described in Section 2.*

3.2 Conditional OPRF (cOPRF)

As described in Section 2.2, an OPRF [14] enables the receiver to learn a PRF value on its input query q without knowing the sender's PRF key k. In this work, we introduce a new notion of a conditional oblivious PRF (cOPRF). Intuitively, the functionality is similar to OPRF, with the additional feature that the sender has a set of elements X, and the receiver obtains a designated PRF value depending on whether its query q is within the sender's set X.

Definition 3. *Conditional OPRF (cOPRF) is a two-party protocol, in which a sender $\mathcal{S}$ has a PRF key $k \in \{0,1\}^\kappa$ and an associated set $X = \{x_1, ..., x_m\} \in (\{0,1\}^\star)^m$, and the receiver learns $\bar{F}(k, q||b)$ where $b = 0$ if $q \in X$ and $b = 1$ otherwise. Here, $\bar{F}$ is a PRF, and q is a query input chosen by the receiver.*

PARAMETERS: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, the receiver set size m, and the PRF F.
FUNCTIONALITY:

- Wait for input set $X = \{x_1, ..., x_m\}$, a PRF key k.
- Wait for input a query q from $\mathcal{R}$.
- Give $\mathcal{R}$ the PRF value $\bar{F}(k, q||i)$ where $i = 0$ if $q \notin X$?, and $i = 1$ otherwise.

Figure 4: Conditional OPRF (cOPRF) Ideal Functionality

PARAMETERS:

- Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, the receiver set size m, the PRF F
- The mOT functionalities described in Figure 2
- The hash function $H : \{0,1\}^\star \to \{0,1\}^\kappa$

INPUT:

- Sender $\mathcal{S}$: $X = \{x_1, \ldots, x_m\} \subset (\{0,1\}^\star)^m$ and the PRF key k
- Receiver $\mathcal{R}$: $q \in \{0,1\}^\star$

PROTOCOL:

1. The receiver chooses a random $\alpha \leftarrow \mathbb{Z}$ and computes $v_i = H(q||i)^\alpha$ for $i \in \{0,1\}$.
2. The sender $\mathcal{S}$ and the receiver $\mathcal{R}$ invoke a mOT functionality where:
    - $\mathcal{R}$ acts as the sender with input (q, v_0, v_1).
    - $\mathcal{S}$ acts as the receiver with input X, and obtains v.
3. The sender computes $w = v^k$ and sends it to the receiver who outputs $w^{1/\alpha}$.

Figure 5: Conditional OPRF (cOPRF) Construction

From Definition 3, we see that a cOPRF remains secure when the underlying PRF function $\bar{F}$ reveals nothing about the query, akin to the traditional OPRF. Moreover, the receiver learns nothing about the sender's input from the cOPRF output, even when sending the same query or multiple queries. The formal description of a conditional oblivious PRF (cOPRF) functionality is given in Figure 4. The primary security goal of cOPRF is to enable the receiver to acquire a designated PRF value according to a defined condition. These PRF values can then be employed in the following computation phase tailored to that condition, as seen in our mPSU protocol.

Our cOPRF Protocol. We present the construction of an cOPRF, which is built on our mOT primitive. While many OPRF protocols exist such as the BaRK OPRF [26], we use the Diffie-Hellman OPRF protocol [12] in which the PRF value of x has a form $H(x||0)^k$ for a random hash function $H : \{0,1\}^\star \to \{0,1\}^\kappa$

The protocol starts with the receiver $\mathcal{R}$ picking a random number $\alpha \leftarrow \{0,1\}^\kappa$ and computing $v_i = H(q||i)^\alpha$ for $i \in \{0,1\}$. The goal is to allow the receiver $\mathcal{R}$ obliviously send v_i to the sender $\mathcal{S}$ depending on whether $q \in X$. This can be done using the mOT in which the receiver $\mathcal{R}$ acts as

Parameters: n parties, parameter m, and a multi-key encryption scheme defined in Section 2.6
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 $\{\mathsf{ct}_1, \dots, \mathsf{ct}_m\}$ where $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, x_i)$ from all parties $\{P_1, \dots, P_n\}$.
- Give $\{x_{\pi(1)}, \dots, x_{\pi(m)}\}$ to P_1 where $\pi = \pi_n \circ \pi_{n-1} \circ \dots \circ \pi_1$.

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

the mOT's sender with input (q, v_0, v_1) while the sender $\mathcal{S}$ acts as the mOT's receiver with input X. As a result, $\mathcal{S}$ obtains v. Next, the $\mathcal{S}$ raises v to the k power as $w = v^k$ and sends the result w back to the $\mathcal{R}$. Now, the receiver can raise the w to the $1/\alpha$ to obtain the final output y. We formally present our cOPRF construction in Figure 5.

It is not hard to see that the output of the cOPRF protocol satisfies correctness. More precisely, if $q \notin X$, $v = v_1 = H(q||0)^\alpha$, thus, the protocol's output $y = w^{1/\alpha} = H(q||0)^k$ as desired. In case the $q \in X$, the value $y = H(q||1)^k$ is a pseudorandom value which is computationally indistinguishable to $H(q||0)^k$ when the PRF key is unknown. In general, our cOPRF protocol is secure against the same query (i.e. the same query will always leads to the same pseudorandom value no matter its membership related to sender's set). We present the security proof of the below theorem in Appendix A.2.

Theorem 4. *The cOPRF protocol described in Figure 5 securely implements the cOPRF functionality defined in Figure 4 in the semi-honest setting, given the mOT functionalities described in Section 2.*

3.3 Oblivious Shuffle and Decryption (Shuffle&Decrypt)

Definition 5. *Oblivious Shuffle and Decryption (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 $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(1^\kappa, n)$. Given a set of ciphertexts $\{\mathsf{ct}_1, \dots, \mathsf{ct}_m\}$ where $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, x_i)$, the Shuffle&Decrypt functionality gives $\{x_{\pi(1)}, \dots, x_{\pi(m)}\}$ to the party P_1 where $\pi = \pi_n \circ \pi_{n-1} \circ \dots \circ \pi_1$, 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

PARAMETERS:

- n parties, the set size m
- A multi-key cryptosystem ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{FulDec}, \mathsf{ReRand}$) defined in Section 2.6

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: $\{\mathsf{ct}_1, \dots, \mathsf{ct}_m\}$ 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 $C_1 = \{\mathsf{ct}^1_1, \dots, \mathsf{ct}^1_m\}$ 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 $A_i = \{1, i, i+1, \dots, n\}$.
   - P_i permutes the set $\{\bar{\mathsf{ct}}^i_1, \dots, \bar{\mathsf{ct}}^i_m\}$ as $\mathsf{ct}^i_j = \bar{\mathsf{ct}}_{\pi_i(j)}, \forall j \in [m]$.
   - P_i sends $C_i = \{\mathsf{ct}^i_1, \dots, \mathsf{ct}^i_m\}$ 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 ($\mathsf{Shuffle\&Decrypt}$) Construction

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 $P_4, \dots, P_n$. 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 proof of the below theorem in Appendix A.3.

Theorem 6. *Given the multi-key cryptosystem defined in Section 2.6, 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.*

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, cOPRF 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

	$P_1 : X_1$		$P_2 : \{x_2\}$		$P_3 : \{x_3\}$		$P_4 : \{x_4\}$
Round 1		$\Leftarrow$ OPRF & mOT $\Rightarrow$		$\Leftarrow$ cOPRF $\Rightarrow$			
	$F_1(X_1)$		$F_1(x_2)$		r_3		
S	$F_1(X_1) \cup \{F_1(x_2)\}$			$\Leftarrow$	cOPRF	$\Rightarrow$	
E	$\{\mathsf{Enc}(x_2)\}$						$F_1(x_4)$
Round 2		$\Leftarrow$ OPRF & mOT $\Rightarrow$				$\Leftarrow$ cOPRF $\Rightarrow$	
	$F_2(X_1) \cup \{F_2(x_2)\}$				$F_2(r_3)$		$F_2(x_4)$
S	$F_2(X_1) \cup \{F_2(x_2)\} \cup \{F_2(r_3)\}$						
E	$\{\mathsf{Enc}(x_2), \mathsf{Enc}(0)\}$						
Round 3			$\Leftarrow$ OPRF & mOT $\Rightarrow$				
	$F_3(X_1) \cup \{F_3(x_2)\} \cup \{F_3(r_3)\}$						$F_3(x_4)$
S	$F_3(X_1) \cup \{F_3(x_2)\} \cup \{F_3(r_3)\} \cup \{F_4(x_4)\}$						
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 $x_4 \in X_1$. The $F_{i-1}(\cdot)$ denotes the multi-key PRF $F((k_{i+1}, \ldots, k_2), \cdot)$, which represents the PRF value received in the i-th round. $\Leftarrow \mathsf{P} \Rightarrow$ denotes the execution of protocol P between two parties. Colors indicating the corresponding output for each invocation of protocol. The same color for OPRF and cOPRF means they use the same key. For example, in round 1, P_1 and P_2 invoke the OPRF and mOT (Step (3,a) and (3,b) in Figure 8). P_1 updates its set by the PRF value $F_1(X_1)$ and receives the message $(F_1(x_2), \mathsf{Enc}(x_2))$ from P_2. P_2 invokes cOPRF protocol with P_3 and P_4 concurrently. P_3 receives a random value r_3 since $x_3 = x_2$ and P_4 receives the PRF value $F_1(x_4)$ for x_4. The following rounds are similar.

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. To overcome the limitation, we leverage the OPRF in the following manner. The leader party P_1 with input X_1 interacts with P_2 holding the PRF key k and receives the PRF values $S = \{y \mid y = F(k, x), x \in X_1\}$. They then execute the mOT where P_2 has keyword $F(k, x_2)$ and messages $(r_2 || \mathsf{Enc}(\mathsf{pk}, 0), F(k, x_2) || \mathsf{Enc}(\mathsf{pk}, x_2))$ so that P_1 can obliviously obtain $v_2 || e_2$ from P_2 where $v_2 || e_2$ equals $F(k, x_2) || \mathsf{Enc}(\mathsf{pk}, x_2)$ if $x_2 \notin X_1$, and $r_2 || \mathsf{Enc}(\mathsf{pk}, 0)$ otherwise for a random r_2. The P_1 appends v_2 to S and e_2 to an empty set E. At this point, $S = \{y \mid y = F(k, x), x \in X_1\} \cup \{F(k, x_2)\}$ if $x_2 \notin X_2$, and $S = \{y \mid y = F(k, x), x \in X_1\} \cup \{r_2\}$ otherwise. The updated set S will be used as the input to the next mOT between P_1 and P_3. It helps to remove the duplication items of P_2 and P_3 from the final output. Concretely, P_3 prepare the keyword and messages in the same way for the mOT as $\big(F(k, x_3), \mathsf{Enc}(\mathsf{pk}, 0), \mathsf{Enc}(\mathsf{pk}, x_3)\big)$. The mOT functionality checks the membership condition whether $F(k, x_3)$ is in the updated S and then gives P_1 the corresponding result $v_3||e_3$ which equals to either $r_3||\mathsf{Enc}(\mathsf{pk}, 0)$ for a random r_3 or $F(k, x_3)||\mathsf{Enc}(\mathsf{pk}, x_3)$. If $x_2 = x_3 \notin X_1$, the P_1 obtains the encryption $e_2 = \mathsf{Enc}(\mathsf{pk}, x_2)$ from P_2, but $e_3 = \mathsf{Enc}(\mathsf{pk}, 0)$ from P_3. The r_3 is indistinguishable from the PRF value $F(k, x_3)$, thus r_3 reveals nothing to P_1 about whether $x_2 = x_3$. The P_1 continues to update the PRF set S and the encryption set E by repeating the above process sequentially with P_4.

When considering P_1 colludes with P_3, there is a security problem when $x_2 = x3$. Concretely, P_1 learns the value $F(k, x_2)$ as the output from the mOT with P_2, and P_3 receives the same value from invoking OPRF with P_2, thus, the P_1 and P_3 infer the fact that P_2 has the element $x_3 = x_2$. To address this vulnerability, i.e., the challenge (2), we propose to use multi-key OPRF and our new gadget cOPRF. The detail is described in Section 4.1.1.

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 $x \in \bigcup_{i=1}^{n} X_i$ 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 $\mathsf{Shuffle\&Decrypt}$ 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.

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 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.

4.1.1 The Usage of OPRF and cOPRF

OPRF is widely used in the design for private set operation and we use it for our design as well. All the inputs are converted into corresponding PRF values for the comparison, this prevents leakage of the original input elements. In the OPRF protocol, the sender learns the key and the receiver can only evaluate the PRF values for his chosen inputs. This prevents the dictionary attack from the receiver to evaluate arbitrary element. In our mPSU protocol, the leader P_1 maintained a list S to store and update the (active) PRF values for the set while growing the union. To be specific, in round $i-1$ for $i \in [2, n]$, P_i acts as the OPRF sender with a key k_i and updates the PRF list S for P_1. From now, the PRF of element x given key k is denoted as $F(k, x)$. We use Diffie-Hellman PRF defined as $F(k, x) = H(x||0)^k$ so that the output of OPRF and cOPRF are compatible in our mPSU protocol. Note that the formulation of F does not affect the correctness of our mPSU protocol asthe probability of two queries sharing the same PRF F remains negligible within κ.

However, as mentioned in the 4-party case earlier, there is a problem and we formalize it here. In round $i-1$, after updating the PRF value, P_i invoke the mOT with P_1 so that P_1 learns the PRF values for any $x_i \in X_i \setminus \bigcup_{j=1}^{i-1} X_j$. If P_1 and any $P_{j\in[i+1,n]}$ collude, P_1 can check P_j's PRF

PARAMETERS:

- n parties $P_{i\in[n]}$ for $n > 1$.
- The mOT, OPRF, $\mathsf{Shuffle\&Decrypt}$ functionalities described in Figures 2&9&6, respectively.
- A multi-key cryptosystem ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{FulDec}, \mathsf{ReRand}$) defined in Section 2.6.
- 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 $X_i = \{x_{i,1}, \dots, x_{i,m}\}$.

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 $C^i_{b,1}$ denote the items in the P_i's bth bin. P_i computes the encryption $e^i_{b,1} = \mathsf{Enc}(pk, C^i_{b,1})$, for $b \in [\mu]$.
   (b) $P_{i\in[n]}$ hashes X_i into μ bins under k hash functions. Let $S^i_{b,1}$ denote the set of items in the P_i's bth bin. P_i pads $S^i_{b,1}$ with dummy values to the maximum bin size $\beta(m)$.
   (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 $b \in [\mu]$, the P_1 and P_i invoke the functionality of OPRF where:
      - P_1 acts as the receiver with input the set $S^1_{b,i-1}$.
      - P_i acts as the sender with input the random-chosen PRF key k_i.
      - P_1 obtains a set $S^1_{b,i}$ of the PRF values $F(k_i, y)$ for $y \in S^1_{b,i-1}$. Note that $F(k_i, y) = H(k_i, y||1)^{k_i}$ for a random hash function H.

   (b) For each bin $b \in [\mu]$, P_1 and P_i invoke a mOT instance where:
      - P_1 acts as the receiver with input $S^1_{b,i}$.
      - P_i acts as the sender with input $(y_{b,i}, r_{b,i}||\mathsf{Enc}(\mathsf{pk}, 0), y_{b,i}||\bar{e}_{b,i})$. Here, $r_{b,i}$ is a random value; $y_{b,i} = F(k_i, C^i_{b,i-1})$ if $C^i_{b,i-1} \neq \varnothing$ and random otherwise; $\bar{e}_{b,i} = \mathsf{Enc}(\mathsf{pk}, 0)$ if $C^i_{b,i-1} = \varnothing$, otherwise, $\bar{e}_{b,i} = e^i_{b,i-1}$.
      - P_1 obtains $v_b||e_b$.

      P_1 appends e_b to E_b. P_1 hashes $\bigcup_{b=1}^{\mu} \left(S^1_{b,i} \cup v_b\right)$ into μ bins under h hash functions. The P_1 redefines $S^1_{b,i}$ to be the set of items in its bth bin, and then pads $S^1_{b,i}$ with dummy values to the maximum bin size $\beta(im)$.
   (c) For each bin $b \in [\mu]$, P_i and $P_{t\in[i+1,n]}$ invoke the cOPRF where:
      - P_t acts as the receiver with input $C^t_{b,i-1}$ or a dummy if $C^t_{b,i-1} = \varnothing$.
      - P_i acts as the sender with input the PRF key k_i and the set $S^i_{b,i-1}$.
      - P_t obtains w_b, and sets $w_b = \varnothing$ if $C^t_{b,i-1} = \varnothing$.

      P_t hashes $W = \{w_b \mid b \in [\mu] \ \& \ w_b \neq \varnothing\}$ into μ bins using the Cuckoo and Simple hashing. Let $S^t_{b,i}$ and $C^t_{b,i}$ denote the items in the Simple and Cuckoo b-th bin, respectively. P_t pads $S^t_{b,i}$ with dummy to maximum bin size $\beta(m)$.
   (d) The P_i and $P_{t\in[i+1,n]}$ invoke a mOT instance where:
      - P_i acts as the receiver with input $\{F(k_i, y) \mid y \in S^i_{b,i-1}\}$
      - P_t acts as the sender with input $(C^t_{b,i}, \mathsf{Enc}(\mathsf{pk}, 0), e^t_{b,i-1})$.
      - P_i obtains c and sends $c' = \mathsf{ReRand}(c, \mathsf{pk})$ to P_t

      P_t computes $e^t_{b,i} = \mathsf{ReRand}(c', \mathsf{pk})$
4. All the parties invoke the $\mathsf{Shuffle\&Decrypt}$ 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

values and the output of mOT with P_i. The intersection recovers a subset of P_i's input with P_j's knowledge of the correspondence of the PRF value and the original input. To address the leakage, we ensure that the corrupt party P_i obtains an fake/"inactive" PRF value of their input item x_i (which has a form $H(x||1)^k$) if the x_i appears in any input set of previous parties $P_{t\in[2,i-1]}$ so that the intersection is always empty. This objective can be achieved through the use of our cOPRF primitive.

Recall that our cOPRF is a single-query PRF, thus, we use the bin-and-ball technique to enhance the performance of our protocol. We now demonstrate how to execute the cOPRF using the recursive method. Initially, each party $P_{t\in[2,n]}$ hashes input set X_t into a Cuckoo hashing table $C_1^t$and a Simple hashing table S_1^t using the same h hash functions. Both tables are updated in each round. We use $C_{b,i}^t$ and $S_{b,i}^t$ to denote the b^{th} bin of party P_t's Cuckoo hashing table and Simple hashing table in round i correspondingly. In round $i-1$, P_i invokes cOPRF protocol with $P_{t\in[i+1,n]}$ bin by bin, where P_i acts as the sender with key k_i and input set $S_{b,i-1}^i$ while P_t acts as the receiver with query $c \in C_{b,i-1}^t$. The cOPRF provides P_t the active/correct PRF value which equals $F((k_i, k_{i-1}, \ldots, k_2), x_{t,j})$ for an input $x_{t,j}$ if $x_{t,j} \notin X_i$ and c has a form $F((k_{i-1}, \ldots, k_2), x_{t,j})$ – in other words, $x_{t,j} \notin X_2 \cup \ldots \cup X_{i-1}$. The cOPRF gives P_t an inactive PRF value as $H(c||1)_i^k$ if $x_{t,j} \in X_i$ or c is random from the earlier round. For the baseline starting in round 1, P_2 plays the role of cOPRF sender with key k_2. For a query element $x_{t,j} \in C_{b,1}^t$ from $P_{t\in[3,n]}$, cOPRF outputs w_b as the correct/active PRF value $F(k_2, x_{t,j})$ if $x_{t,j} \notin X_2$ or a random/inactive value $r_{t,j}$ otherwise. P_t creates the Simple hashing table S_2^t and Cuckoo hashing table C_2^t for round 2 by hashing the received PRF values.

We present the cOPRF execution in Step (3,c), Figure 8. It should be noted that, by invoking OPRF and cOPRF in each round, the input for the mOT from party P_i in round $i-1$ is a multi-key PRF value. For an element $x_{i,j}$, the associated PRF is of form $F((k_i, \ldots, k_2), x_{i,j})$ if $x_{i,j} \notin \bigcup_{t=2}^{i-1} X_t$. The randomness always has a contribution from the key k_i chosen by party P_i himself. This prevents the information leakage from the collusion between the P_1 and the PRF key selector for the usage of single-key PRF scheme.

When using the bin-and-ball scheme, parties are required to apply the mapping to their multi-key PRF values. In round $i-1$, each individual $P_{i\in[2,n]}$ must also map its PRF values using Cuckoo hashing, and the resulting Cuckoo-hashing bin serves as the input of the sender P_i in the mOT process. Since the P_1 and P_i invoke mOT in Step (3,b) which takes care of the membership test, these two parties only need to execute the OPRF computation (instead of cOPRF). Note that P_1 does not need to query the PRF values for dummy items, but instead aggregates all non-dummy PRF values in the set $S_{b,i-1}^1$ and employs them as input to the OPRF execution.

4.1.2 The Usage of mOT

As per the overview description, each party $P_{i\in[2,n]}$ should oblivious transmit element $x \in X_i$ to P_1 if $x \notin \bigcap_{t=1}^{i-1} X_t$. In round i, the parties P_1 and P_i engage in the mOT to incrementally acquire the PRF values and encrypted union items. We now introduce the usage of mOT in detail.

Instead of using the original element, P_1 and P_i use PRF values as the input set and value for the mOT execution. Same as what we do for the cOPRF, mOT is executed in a bin-by-bin manner. In round $i-1$, each party P_i has Cuckoo hashing table C_{i-1}^i and a simple hashing table S_{i-1}^i while P_1 having a Simple hash table S_{i-1}^1 to store the PRF values. As introduced in Section 4.1.1, P_1 updates his PRF values by OPRF. Concretely, assuming that after the mOT with P_{i-1}, the set $S_{b,i-1}^1$ contains either random values r or $F((k_{i-1}, \ldots, k_2), x)$ for $x \in \bigcup_{t=1}^{i-1} X_i$. In the next round, P_i selects the PRF key k_i. The P_1 submits OPRF queries on S^1_{i-1} to P_i and obtains $S^1_{b,i} = \{F(k_i, s) \mid s \in S^1_{i-1}\}$ while P_i learns nothing. Note that the $F(k_i, s)$ has a form $H(k_i, s||0)$ which is the same as the "active" PRF in cOPRF. Clearly, the set $S^1_{b,i}$ consists of $F((k_i, \ldots, k_2), x)$ for $x \in \bigcup_{t=1}^{i-1} X_i$ and $F(k_i, r)$ for random $r \in S^1_{i-1}$. We present the OPRF in Step (3,a), Figure 8. The elements of each bin in S^1_i serves as the input set For P_1 for each call of mOT.

The mOT execution between P_1 and P_i should allow P_1 to add the PRF values $F((k_i, \ldots, k_2), x)$ of the P_i's item $x \in X_i$ to S^1_i if x is not in the union of $\{X_1, \ldots, X_{i-1}\}$. To achieve this, they invoke the mOT protocol where P_1 plays the role of receiver and P_i as sender. Considering the b-th bin, P_1's input set for the mOT is $S^1_{b,i}$ while P_i's input should be of form $(c_{b,i}, r_{b,i}||\mathsf{Enc}(\mathsf{pk}, 0), c_{b,i}||e_{b,i})$. The $c_{b,i} = F(k_i, c)$ is the PRF value stored in the b-th bin of his Cuckoo hashing table where c is the value obtained from cOPRF executions with previous parties $P_{i\in[2,i-1]}$. The $r_{b,i}$ is randomly chosen. The main tricky part is how to define $e^i_{b,i}$. If $e_{b,i}$ is an encryption $\mathsf{Enc}(\mathsf{pk}, x)$ of the P_i's input item x, and if the the $c_{b,i}$ is an inactive/fake PRF value, then $e_{b,i}$ is added to E_b as $c_{b,i}$ never appears in the P_1's set $S^1_{b,i}$. Recall that the inactive PRF $c_{b,i}$ indicates that the corresponding x is in the set of previous parties $P_{t\in[2,i-1]}$. Hence, if we set $\bar{e}_{b,i} = \mathsf{Enc}(\mathsf{pk}, x)$, the E_b contains two ciphertexts that are encryptions of the same x. This will reveal to P_1 the multiplicity of each element in the union after decrypting E_b. To avoid this issue, we propose the following method to compute $e_{b,i}$.

Our objective is to ensure that $e^i_{b,i}$ is the encryption of the P_i's input item x if x does not appear in any set of $X_2, \ldots, X_{i-1}$; otherwise it should be the encryption of zero. In brief, the value of $e^i_{b,i}$ depends on the membership test of x with set $X_{t\in[2,i-1]}$. This can be achieved by having P_i participate with each party $P_{t\in[2,i-1]}$ using mOTin round $t-1$ alongside with the execution of cOPRF. For the b-th bin of the hash table in round $t-1$, P_t and P_i invoke mOT where P_t acts as Receiver with input $S^t_{b,t-1}$ while P_i acts as Sender with input $(C^i_{b,t-1}, v_0 = \mathsf{Enc}(pk, 0), v_1 = e^i_{(b,t-1)})$. We use $e^i_{b,t}$ denotes the encryption corresponding to the value x whose PRF value stored in the $C^i_{b,t-1}$ if x does not appear in any set of $X_2, \ldots, X_{t-1}$. Initially $e^i_{b,1} = \mathsf{Enc}(pk, x)$. The mOT functionality returns the value $v = v_0$ to P_t if $x \in X_t$ and $v = v_1$ otherwise. The P_t then rerandomizes v by computing $v' \leftarrow \mathsf{ReRand}(v, \mathsf{pk})$ and returns the result to P_i. The P_i then computes $e^i_{b,t} = \mathsf{ReRand}(v', \mathsf{pk})$ which is the input to mOT. The first re-randomization from P_t aims to prevent the P_i from determining the output of P_t, which would reveal whether $x \in X_t$.The second re-randomization prevents the P_i to learn whether $x \in X_{t\in[2,t-1]}$ if colluding with P_1. The computation described above is presented in Step (3,d), Figure 8.

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 $C^i_{b,i-1} = F(k_{i-1}, \ldots, k_2, x_{i,k})$ after interacting with $P_{t\in[2,i-1]}$ via the cOPRF.

- Case 1 ($x_{i,k} \in X_1$) – the P_1 has $x_{i,k}$: As $x_{i,k} \in X_1$, the OPRF with $P_{t\in[2,i]}$ in Step (3,a) gives P_1 the multi-key PRF value $F((k_i, \ldots, k_2), x_{i,k})$. In the mOT execution between P_1 and P_i, the input keyword of P_i as $y_{b,i} = F(k_i, C^i_{b,i-1})$ is in $S^1_{b,i}$. Thus, P_1 receives the encryption of zero $\mathsf{Enc}(\mathsf{pk}, 0)$ from the mOT functionality. As a result, $x_{i,k}$ does not appear in the final result from the $\mathsf{Shuffle\&Decrypt}$ 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_1 and P_i on input related to $y_{b,i} = F(k_i, C^i_{b,i-1})$ gives P_1 the $y_{b,i}$ and $e_{b,i} = \mathsf{Enc}(\mathsf{pk}, x_{i,k})$. The PRF value $y_{b,i}$ is added to the set $S^1_{b,i}$, and the $x_{i,k}$ will appear in the final union output.

Assume that $x_{t,j} = x_{i,k} \in X_j$ was mapped into the b-th Cuckoo bin. Since $x_{t,j} \in X_i$, the cOPRF and mOT with P_i gives P_j an inactive/fake PRF value $C^t_{b,i}$ and $e_{t,b} = \mathsf{Enc}(\mathsf{pk}, 0)$. Thus, when executing mOT with P_t, the P_1 obtains $\mathsf{Enc}(\mathsf{pk}, 0)$. Hence, the $x_{t,j}$ does not appear in the final result.

- Case 3 ($x_{i,k} \notin \bigcup_{j=1, j\neq i}^{n} X_j$) – no party has $x_{i,k}$: The mOT execution between P_1 and P_i gives P_1 an $\mathsf{Enc}(\mathsf{pk}, x_{i,k})$. Thus, $x_{i,k}$ appears in the final result from the $\mathsf{Shuffle\&Decrypt}$ functionality.

Security. The security of our mPSU protocol is given as below.

Theorem 7. *Given the multi-key cryptosystem, mOT, OPRF, cOPRF and $\mathsf{Shuffle\&Decrypt}$ functionalities described in Section 2.6, and Figures 2&9&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: If H contains only one honest party P_i, then P_i has x. The corrupted parties C 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. If H has more than one honest party (say P_i and $P_{j>i}$). We consider two following cases:
   - 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 cOPRF and mOT between P_i and previous corrupt parties $P_{t<i} \in C$ reveals nothing to C, even when C contains same items, this leads to submitting the same cOPRF queries. Thus, the simulation is simple.

     If $P_1 \in C$, the corrupt P_1 obtains $\mathsf{Enc}(\mathsf{pk}, x)$ and the $F((k_i, \ldots, k_2), x)$ from P_i. Since the encryption is protected under the $\mathsf{Shuffle\&Decrypt}$ functionality until the P_1 learns the union sets which was permuted by the honest party P_i, the encryption reveals nothing to C. Similarly, the PRF value consists of the P_i's key k_i, which hides x from C.
   - Both P_i and P_j have x: If $i = 1$, then the honest leader P_1 receives encryptions of zeros $\mathsf{Enc}(\mathsf{pk}, 0)$ and "fake" PRF values when executing mOT with P_j. Thus, 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. Thanks to the CCA property of the encryption scheme and the permutation in $\mathsf{Shuffle\&Decrypt}$, 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 cOPRF (or OPRF) and mOT executions where a party in H acts as the sender and a party in C acts as the receiver, the corrupt set C receives

nothing related to x. 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 invokes the cOPRF (or OPRF) with an honest sender which does not have x, the C obtains the correct/active PRF values. When C interacts with an honest party that has x, the C obtains the active/correct PRF values (if $P_1 \in C$) and the inactive/fake PRF values. Since the PRF values contain the PRF key of the honest set H and their distribution is random. Thus, C learns nothing from cOPRF or OPRF executions. Similarly, the corrupt coalition's view is simulated from Step (3b, 3d) based on the functionality of mOT and encryption scheme. Moreover, the $\mathsf{Enc}(\mathsf{pk}, x)$ appears only once in the encryption set E, thus, C learns nothing about whether H has x.

□

4.3 Complexity

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)	$n=3$	2.34	0.68	0.38	36.19	9.33	2.94	601.09	153.94	42.10	7206.93	1928.18	538.60
	$n=4$	4.41	1.24	0.62	69.64	17.53	5.02	1147.23	291.49	78.70	13499.71	3578.83	985.87
	$n=6$	10.62	2.90	1.20	167.80	42.46	11.79	2786.58	724.33	189.36	32298.18	8884.75	2334.88
	$n=8$	19.68	5.24	1.98	312.52	79.01	21.53	5176.58	1331.38	349.97	59807.53	16212.85	4290.46
WAN (s)	$n=3$	12.03	10.58	10.32	52.38	26.56	20.48	655.88	208.86	97.07	9920.07	2961.36	1219.62
	$n=4$	19.58	16.53	15.93	94.21	43.12	30.91	1214.65	370.88	161.04	18937.01	5444.82	2088.87
	$n=6$	36.72	29.04	27.36	210.05	85.91	55.57	2909.79	859.65	327.68	45732.48	12943.89	4434.85
	$n=8$	56.70	42.36	39.13	371.11	139.81	82.78	5356.24	1523.28	544.93	84535.25	23227.10	7576.88
Comm. Cost (MB)	$n=3$	1.51			21.01			333.27			5332.40		
	$n=4$	2.28			31.74			503.39			8054.17		
	$n=6$	3.82			53.24			844.55			13512.85		
	$n=8$	5.36			74.80			1186.52			18984.38		

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		LAN	WAN		LAN	WAN		LAN	WAN
$m = 2^8$	**Total**	4.56	4.41	19.58	4.53	4.41	19.58	4.56	4.41	19.58	4.59	4.41	19.58
	OPRF	0.11	2.89	4.93	0.02	0.43	1.11	0.04	0.96	1.64	0.06	1.50	2.18
	mOT	4.30	0.16	12.37	1.43	0.05	4.12	1.43	0.05	4.12	1.43	0.05	4.12
	cOPRF	-	-	-	2.93	1.08	9.69	2.93	1.08	9.69	2.93	1.08	9.69
	Shuffle&Decrypt	0.15	1.02	2.00	0.15	1.02	2.00	0.15	1.02	2.00	0.15	1.02	2.00
$m = 2^{12}$	**Total**	63.47	69.64	94.21	63.01	69.64	94.21	63.50	69.64	94.21	63.91	69.64	94.21
	OPRF	1.76	47.03	50.32	0.26	6.62	7.61	0.59	15.21	16.37	0.91	25.19	26.34
	mOT	59.33	0.91	19.98	19.78	0.30	6.66	19.78	0.30	6.66	19.78	0.30	6.66
	cOPRF	-	-	-	40.59	15.99	29.83	40.59	15.99	29.83	40.59	15.99	29.83
	Shuffle&Decrypt	2.38	16.05	19.32	2.38	16.05	19.32	2.38	16.05	19.32	2.38	16.05	19.32
$m = 2^{16}$	**Total**	1006.73	1147.23	1214.65	999.43	1147.23	1214.65	1007.21	1147.23	1214.65	1013.72	1147.23	1214.65
	OPRF	28.09	739.20	744.78	4.13	107.07	108.67	9.36	242.86	244.73	14.60	389.27	391.38
	mOT	940.54	12.38	83.61	313.51	4.13	27.87	313.51	4.13	27.87	313.51	4.13	27.87
	cOPRF	-	-	-	643.69	254.17	302.74	643.69	254.17	302.74	643.69	254.17	302.74
	Shuffle&Decrypt	38.10	303.31	309.82	38.10	303.31	309.82	38.10	303.31	309.82	38.10	303.31	309.82

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

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 [34, 37], 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 cOPRF with the remaining parties, 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 $O(n^2 m \log m/\log\log m)$.

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 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; a WAN setting with a 80ms round-trip latency, 400 Mbps network bandwidth.

Our mPSU protocol is built on ElGamal encryption scheme (multi-key cryptosystem), Diffie-Hellman OPRF (cOPRF), SS-PMT, and OT (mOT and cOPRF). We implement the exponentiation for OPRF and ElGamal encryption using the elliptic curve code (Curve25519) from Relic [40]. For the SS-PMT implementation which requires garbled circuit for two strings comparison, we use the EMP-toolkit library [45]. Finally, we use the OT-extension [22] provided in [33] to implement mOT. Our complete implementation will be 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 OPRF with mOT and cOPRF building blocks 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 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.

The performance difference between WAN and LAN is primarily due to the latency instead of bandwidth 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 Figure 4. Specifically, we present the performance metrics of the mOT in Step (3,e) and the cOPRF in Step (3,d) in our protocol within the column designated for cOPRF in Figure 4. All reported running time values in Table 3 and Figure 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 [15, 4] and estimate the performance for protocol proposed in [18]. Table 5 presents the running time and communication cost of various mPSU protocols [15, 4, 18] which are secure in the dishonest majority [1] and semi-honest setting. We do not incorporate the results from [44] into our comparison, as their protocol only work for a small universe. Even in their largest setting, with a universe size of 2^{32}, it is considerably smaller than the general scenario involving 128-bit elements. According to [44, 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 [4], 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 [4]'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 [4]). Consequently, the secure computation takes place exclusively between these two parties. We implement [4] using EMP-toolkit library [45] which provides the 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 1.87 times faster for the large set size of 2^{20} in the LAN setting and $6.54 - 37.82\times$ faster than [4] in the WAN. Additionally, the cost for [4] is significantly $(162.43 - 389.85\times)$ 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 [18]. 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 [30] (as well as our [15]'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 [15, 18] is not practical even for small set sizes. Thus, we skip the evaluation of the [18, 15] in the WAN setting.

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, cOPRF 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. 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 achieve malicious MPSU, one can employ cryptographic commitment techniques at each step of the protocol, albeit with added costs.

[1] The recent mPSU protocol [29] provides a weak security guarantee wherein the leader does not collude with any parties.

	m	Ours	[18]	[4]	[15]
Running Time LAN (second)	2^8	4.41	155.63	**2.70**	6009.00
	2^{12}	69.64	2490.04	**57.725**	-
	2^{16}	**1147.23**	39840.65	1158.53	-
	2^{20}	**13499.71**	637450.40	25279.39	-
Running Time WAN (second)	2^8	**19.58**	-	128.05	-
	2^{12}	**94.21**	-	2387.70	-
	2^{16}	**1214.65**	-	45939.46	-
Comm. Cost (MB)	2^8	9.12	15.72	1481.34	**4.25**
	2^{12}	126.96	251.65	30116.50	**68.00**
	2^{16}	2013.56	4026.53	617047.00	**1088.00**
	2^{20}	32216.68	64424.48	12559743.00	**17408.00**

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.

References

[1] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[2] Alex Berke, Michiel Bakker, Praneeth Vepakomma, Kent Larson, and Alex 'Sandy' Pentland. Assessing disease exposure risk with location data: A proposal for cryptographic preservation of privacy, 2020.

[3] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-Optimal oblivious Key-Value stores for efficient PSI, PSU and Volume-Hiding Multi-Maps. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 301–318, Anaheim, CA, August 2023. USENIX Association.

[4] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In Heung Youl Youm and Yoojae Won, editors, *ASIACCS 12*, pages 40–41. ACM Press, May 2012.

[5] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012.

[6] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 236–252. Springer, Heidelberg, December 2005.

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

[8] Dung Bui and Geoffroy Couteau. Improved private set intersection for sets with small entries. PKC, 2023. https://eprint.iacr.org/2022/334.

[9] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving aggregation of Multi-Domain network events and statistics. In *19th USENIX Security Symposium (USENIX Security 10)*, Washington, DC, August 2010. USENIX Association.

[10] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1182–1204. ACM Press, November 2021.

[11] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: Scaling private contact discovery. Cryptology ePrint Archive, Paper 2018/579, 2018. https://eprint.iacr.org/2018/579.

[12] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[13] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[14] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography*, pages 303–324, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] Keith B. Frikken. Privacy-preserving set union. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 237–252. Springer, Heidelberg, June 2007.

[16] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 591–617, Cham, 2021. Springer International Publishing.

[17] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[18] Xuhui Gong, Qiang-Sheng Hua, and Hai Jin. Nearly optimal protocols for computing multi-party private set union. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2022.

[19] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. Contact discovery in mobile messengers: Low-cost attacks, quantitative analyses, and efficient mitigations. *ACM Trans. Priv. Secur.*, 26(1), nov 2022.

[20] Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 75–76, 2016.

[21] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 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*, pages 370–389. IEEE, 2020.

[22] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[23] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2947–2964, Boston, MA, August 2022. USENIX Association.

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

[25] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

[26] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

[27] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 636–666. Springer, Heidelberg, December 2019.

[28] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from PIR with default. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 605–634. Springer, Heidelberg, December 2021.

[29] Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. Cryptology ePrint Archive, Paper 2023/1413, 2023. https://eprint.iacr.org/2023/1413.

[30] Huanyu Ma, Shuai Han, and Hao Lei. Optimized paillier's cryptosystem with fast encryption and decryption. In *Annual Computer Security Applications Conference*, ACSAC '21, page 106–118, New York, NY, USA, 2021. Association for Computing Machinery.

[31] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1151–1165. ACM, 2021.

[32] Duong Tung Nguyen and Ni Trieu. Mpccache: Privacy-preserving multi-party cooperative cache sharing at the edge. Cryptology ePrint Archive, Report 2021/317, 2021. https://eprint.iacr.org/2021/317.

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

[34] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

[35] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

[36] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157. Springer, Heidelberg, April / May 2018.

[37] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), jan 2018.

[38] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 89–104. Springer, Heidelberg, August 1998.

[39] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517. ACM Press, November 2022.

[40] RELIC. A modern research-oriented cryptographic meta-toolkit with emphasis on efficiency and flexibility. https://github.com/relic-toolkit.

[41] Jae Hong Seo, Jung Cheon, and Jonathan Katz. Constant-round multi-party private set union using reversed laurent series. volume 7293, pages 398–412, 05 2012.

[42] Katsunari Shishido and Atsuko Miyaji. Efficient and quasi-accurate multiparty private set union. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 309–314, 2018.

[43] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *CoRR*, abs/2004.13293, 2020.

[44] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Fast multi-party private set operations in the star topology from secure ands and ors. Cryptology ePrint Archive, Paper 2022/721, 2022. https://eprint.iacr.org/2022/721.

[45] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. Emp-toolkit: Efficient multiparty computation toolkit, 2016.

[46] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from Multi-Query reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 337–354, Anaheim, CA, August 2023. USENIX Association.

PARAMETERS: A PRF F, and a bound m on the number of queries.
FUNCTIONALITY:

- Wait for input $(q_1, \ldots, q_m)$ from the receiver where $q_i \in \{0,1\}^\kappa$.
- Sample a random PRF key k and give it to the sender.
- Give $\{F(k, q_1), \ldots, F(k, q_m)\}$ to the receiver.

Figure 9: OPRF Ideal Functionality

PARAMETERS: Two parties: Sender and Receiver
FUNCTIONALITY:

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

Figure 10: Oblivious Transfer (OT) Ideal Functionality.

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

- Wait for input a set of items $X = \{x_1, \ldots, x_n\} \subset (\{0,1\}^*)^n$ 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 11: Secret-shared Private Membership Test (SS-PMT) Ideal Functionality.

A Security Proof

A.1 Security Proof of Theorem 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 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.

□

A.2 Security Proof of Theorem 4

Proof. The security follows from the security of the mOT functionality and the fact the value $v_i = H(q||i)^\alpha$ and $y = w^{1/\alpha}$ is distributed uniformly.

More precisely, the corrupt sender $\mathcal{S}$ learns nothing from the mOT execution as v_0 and v_1 are in the same distribution. The value v_1 reveals nothing about the receiver's input q due to the secret α under the Diffie–Hellman assumption.

The corrupt receiver obtains $w = v^k$ from the honest sender. Due to the secret PRF key k, the receiver learns nothing from v. Thus, simulation is trivial, as the parties' views in the protocol are exactly the cOPRF output. □

A.3 Security Proof of Theorem 6

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). □