Scalable Multi-party Private Set Union from Multi-Query Secret-Shared Private Membership Test

Xiang Liu[1] and Ying Gao[1,2(✉)][0000−0001−8992−651X]

[1] School of Cyber Science and Technology, Beihang University, Beijing, China
{lx1234, gaoying}@buaa.edu.cn
[2] Zhongguancun Laboratory, Beijing, China

Abstract. Multi-party private set union (MPSU) allows $k(k \geq 3)$ parties, each holding a dataset of known size, to compute the union of their sets without revealing any additional information. Although two-party PSU has made rapid progress in recent years, applying its effective techniques to the multi-party setting would render information leakage and thus cannot be directly extended. Existing MPSU protocols heavily rely on computationally expensive public-key operations or generic secure multi-party computation techniques, which are not scalable.

In this work, we present a new efficient framework of MPSU from multi-party secret-shared shuffle and a newly introduced protocol called multi-query secret-shared private membership test (mq-ssPMT). Our MPSU is mainly based on symmetric-key operations and is secure against any semi-honest adversary that does not corrupt the leader and clients simultaneously. We also propose new frameworks for computing other multi-party private set operations (MPSO), such as the intersection, and the cardinality of the union and the intersection, meeting the same security requirements.

We demonstrate the scalability of our MPSU protocol with an implementation and a comparison with the state-of-the-art MPSU. Experiments show that when computing on datasets of 2^{10} elements, our protocol is 109× faster than the state-of-the-art MPSU, and the improvement becomes more significant as the set size increases. To the best of our knowledge, ours is the first protocol that reports on large-size experiments. For 7 parties with datasets of 2^{20} elements each, our protocol requires only 46 seconds.

Keywords: Multi-query secret-shared private membership test · Private set union · Multi-party secret-shared shuffle.

1 Introduction

Private set union (PSU) allows a group of mutually untrusted parties to compute the union of their sets without revealing any additional information. PSU has various applications, including cyber risk assessment and management [26, 34], privacy-preserving data aggregation [6, 8], and computing private DB full join

[33]. For example, to assess system risks and deploy corresponding defenses, security practitioners usually want to obtain a joint list of their IP blacklists. However, they are increasingly concerned with the privacy of sensitive data, which may experience disclosure without appropriate computation techniques. One crucial way this can be enhanced is PSU, which can be used to protect the privacy of each organization while computing the union correctly.

In addition, the combination of PSU with other set operations is also greatly in use. For instance, a social service organization needs to determine the cancer patients who are entitled to social welfare, and they would require the patient data from several hospitals to obtain the set of all cancer patients, and then identify those who are eligible for social welfare [30,33]. Using PSU, the organization can securely compute the union of all cancer patients from the hospitals without revealing any raw data to other parties. Subsequently, the organization can obtain the final result while preserving privacy by performing a private set intersection (PSI) calculation between the obtained set and those eligible for social welfare.

PSI and PSU can be classified into two-party and multi-party settings based on the number of participants. Over the last decade, two-party PSI has received considerable research attention [2,7,11,13,14,17,19,22,29,31,38–41,43,44]. The most efficient two-party PSI protocol to date [43] achieves performance comparable to the insecure naive hashing PSI. Multi-party PSI (MPSI) has also benefited from the research on two-party PSI, leading to the development of many efficient constructions [4,9,32,36] suitable for large sets with millions of elements. As for PSU, Kisser and Song initially studied the two-party PSU [30]. Although some subsequent works [5,16,20,25] have been proposed, their constructions rely heavily on additively homomorphic encryption (AHE) or complex circuits, resulting in low efficiency. In 2019, Kolesnikov et al. [33] proposed the first two-party PSU protocol suitable for large sets. Their construction is mainly based on symmetric-key operations combined with oblivious transfer (OT), achieving a three-orders-of-magnitude improvement in speed compared to [16]. In the following years, Garimella et al. [21] and Jia et al. [28] further reduced the communication and computation overhead using oblivious switching. Very recently, [12,24,49] realize linear computation and communication complexity and are more efficient.

1.1 Motivation

Despite growing interest in PSU, there has been no scalable multi-party PSU (MPSU) protocol. Most of the previous protocols [20,23,27,30,47] are not feasible on large datasets, due to non-constant AHE operations that are proportional to the size of the sets. [46] constructed a constant-round protocol from reversed Laurent Series and secret sharing but has a high computational and communication complexity. The protocol proposed in [5] sorts and merges the sets of the participants using general MPC techniques, but it suffers from the significant overhead caused by complex circuits. [48] requires public-key operations proportional to the input domain size of the sets. Although they have optimized the

protocol using a divide-and-conquer approach, the number of public-key operations still grows linearly with the size of the set and the number of participants. Currently, most work on MPSU is still in the theoretical stage. Only [5,48] have implemented and tested their protocols, but their performance is unsatisfactory. These above-mentioned drawbacks limit the applications of MSPU. Therefore, it is sensible to pose the problem:

Can we construct a truly scalable MPSU protocol?

1.2 Contribution

In this paper, we answer this problem affirmatively in the semi-honest setting. In detail, our contributions can be summarized as follows:

1. We analyze the differences between MPSU and MPSI, then discuss the performance gap between them, and point out the difficulties in extending two-party PSU protocols to multi-party settings (cf. Sect. 2.1).
2. We propose a new protocol called multi-query secret-shared private set membership test (mq-ssPMT) to cater for the multi-party setting, and provide an efficient construction of mq-ssPMT based on the multi-query reverse private set membership test (mq-RPMT) proposed in [49]. mq-ssPMT can easily realize mq-RPMT and can be directly used for computing two-party PSI and PSU. Specifically, when constructing a two-party PSU protocol, our mq-ssPMT reduces one round of communication and n bits of communication cost, where n is the size of the set, compared to mq-RPMT in [49] while keeping the same computation cost.
3. We present new frameworks for computing multi-party private set operations (MPSO) based on mq-ssPMT and multi-party secret-shared shuffle. Useful functions include:
   - Computing the union, i.e., MPSU
   - Computing only the cardinality of the union
   - Computing the intersection, i.e., MPSI
   - Computing only the cardinality of the intersection

   Furthermore, we prove that our frameworks are secure against any semi-honest adversary that does not corrupt the leader and clients simultaneously.
4. We demonstrate the scalability of our MPSU protocol with an implementation. As a result, our MPSU protocol is 109× faster in terms of running time on sets of 2^{10} elements than the state-of-the-art MPSU protocol. Moreover, for 7 parties each holding a million-element dataset, our MPSU protocol requires only 4 minutes on WAN and 46s on a LAN. Our implementation is released on Github: `https://github.com/lx-1234/MPSU`.

1.3 Related Work

We review previous MPSU protocols in the semi-honest setting and provide a theoretical comparison among them.

AHE-based MPSU. Kisser and Song [30] proposed the first MPSU protocol based on polynomial representations and threshold AHE. The core idea is that each participant $\mathcal{P}_i$ represent his set X_i as a polynomial f_i whose roots are the set elements, so the polynomial $\prod_{i=1}^{k} f_i$ represents the union $\bigcup_{i=1}^{k} X_i$. Their protocol requires a large number of AHE operations and high-degree polynomial calculations, which results in inefficiency.

Frikken [20] also uses polynomial representation and threshold AHE. Each $\mathcal{P}_i$ represents his set X_i as a polynomial f_i. $\mathcal{P}_1$ first encrypts f_1 using AHE, and sends to $\mathcal{P}_2$. $\mathcal{P}_2$ computes $(x \cdot \mathsf{Enc}(f_1), \mathsf{Enc}(f_1))$ for each $x \in X_2$. Note that if $x \notin X_1$, then $f_1(x) \neq 0$, and all participants can jointly decrypt the ciphertext and recover x by computing the inverse. Otherwise, both ciphertexts decrypt to 0. Therefore, they can compute the difference set $X_2 \setminus X_1$. Similarly, they compute $X_2 \setminus X_1, \cdots, X_k \setminus (X_1 \cup \cdots \cup X_{k-1})$ separately, which can be merged to get the union. Although the polynomial degree in [20] is lower than [30], the complexity is still of quadratic order in the size of the set due to the need to perform multi-point evaluations on the encrypted polynomials.

Gong et al. [23] proposed a constant-round MPSU protocol based on threshold AHE and Bloom filters (BF). They observed that if a BF has no collisions, then for each element stored in it, at least one of the positions is mapped only by itself. Exploiting this property, they first construct a BF storing the union and then check whether each position in the BF was mapped only by one element. If so, they could figure out that element. Since the length of the BF is related to the statistical security parameter and the union size, their protocol requires a large amount of AHE operations. So the computational overhead is unacceptable.

All these three works use a threshold AHE and when the threshold is set to k, they can resist arbitrary collusion.

Other MPSU. Seo et al. [46] proposed a constant-round MPSU protocol based on secret sharing and reversed Laurent series. Their core idea is that if two sets X and Y are represented by polynomials f_X and f_Y respectively, then the union $X \cup Y$ can be represented by the least common multiple of f_X and f_Y, denoted as $\mathrm{lcm}(f_X, f_Y)$. Note that $\frac{1}{f_X} + \frac{1}{f_Y} = \frac{q(x)}{\mathrm{lcm}(f_X, f_Y)}$, so it suffices to calculate $\frac{1}{f_X} + \frac{1}{f_Y}$. Although this protocol achieves constant-round communication, the operations on high-degree polynomials result in high computational and communication complexity. Additionally, their protocol relies on the honest majority assumption.

Blanton et al. [5] proposed a more efficient MPSU protocol based on oblivious sorting and generic MPC techniques in the honest majority setting. At a high level, they first merge all sets into a large set, then sort it, and remove duplicate elements by comparing adjacent elements to obtain the union. They focused on constructing corresponding circuits and implemented them using generic MPC techniques. Their experimental results show that in the three-party for 32-bit sized elements, computing the union of 2^{10} elements set takes 11.8 seconds.

Vos et al. [48] convert sets to bit-sets, i.e., a vector of bits is assigned to each dataset X_i in which the ith element of this bit-vector (bit-set) is equal to 1 if the ith element of an ordered universe $\mathcal{U}$ of elements belongs to X_i, and 0

otherwise. They obtain the union by performing a secure OR on the bit-sets. Their secure OR is based on totally public-key operation, and the number of secure OR is linearly related to the size of $\mathcal{U}$, which makes the protocol unsuitable for a large $\mathcal{U}$ such as $|\mathcal{U}| = 2^{32}$. To address this issue, they use a divide-and-conquer approach where each participant divides their bit-set into D parts and uses secure OR to check if each part contains any elements. If so, the part is further divided; otherwise, it is discarded. Nevertheless, each participant in the optimized protocol still needs to perform $O(kn \log |\mathcal{U}|)$ public-key operations, which makes it non-scalable for large sets or a large $\mathcal{U}$.

Other related work contains an MPSU protocol with an untrusted third party's help [47] and an MPSU protocol focus on multiset setting [27], which both rely heavily on AHE and are out of the scope of our consideration.

Table1 summarizes and compares the theoretical complexity and the ability to resist collusion of existing MPSU protocols and our protocol. Leader refers to the participant who obtains the union result or starts the computation. Client refers to the remaining participants. [5,20,46] can achieve malicious security, but we only compare with their semi-honest protocols here.

It should be noted that [23, 30, 48] will reveal the union to all participants unavoidably, while in our protocol only the leader gets the result. Moreover, our protocol can be extended to that case in the semi-honest setting, i.e., the leader just broadcasts the output once he receives it. So we could say we achieve a stronger MPSU function in the semi-honest setting.

Table 1. Asymptotic communication (bits) and computation costs of MPSU protocols in the semi-honest setting. Pub: public-key operations; sym: symmetric-key operations. n is the size of input set. k is the number of participants. N is the size of union. $\mathcal{U}$ is the universe of input elements. σ is the bit length of input elements. t is the number of AND gates in the SKE decryption circuit. λ is statistical security parameter. κ is computational security parameter. Generally, $\lambda = 40$. In our protocol, $\kappa = 128$ while in other works κ is the public key length. We ignore the offline phase cost in our protocol. * means that in our protocol the adversary does not corrupt the leader and clients simultaneously.

Protocol	Comm.			Comp.(#Ops sym/pub)		Corruption
	Leader	Client	Rounds	Leader	Client	
[30]	$O(\kappa k^3 n^2)$		$O(k)$	$O(k^2 n^3)$ pub		$< k$
[20]	$O(\kappa kn)$		$O(k)$	$O(kn^2)$ pub		$< k$
[46]	$O(\sigma k^3 n^2)$		$O(1)$	$O(k^4 n^2)$ sym		$< \lfloor (k+1)/2 \rfloor$
[5]	$O(\sigma(\sigma kn \log n + k^2))$		$O(\log k)$	$O(\sigma kn \log n + k^2)$ sym		$< \lfloor (k+1)/2 \rfloor$
[23]	$O(\kappa \lambda kN)$	$O(\kappa \lambda N)$	$O(1)$	$O(\lambda kN)$ pub	$O(\lambda N)$ pub	$< k$
[48]	$O(\kappa k^2 n \log \lvert\mathcal{U}\rvert)$	$O(\kappa kn \log \lvert\mathcal{U}\rvert)$	$O(\log \lvert\mathcal{U}\rvert)$	$O(k^2 n \log \lvert\mathcal{U}\rvert)$ pub	$O(kn \log \lvert\mathcal{U}\rvert)$ pub	$< k$
Ours	$O((t + \kappa + (\sigma + \lambda + \log(kn))k)kn)$	$O((t + \lambda + \kappa)kn)$	$O(\log(\sigma - \log n) + k)$	$O(tkn)$ sym		$< k^*$

2 Overview of Our Techniques

In this section, we provide a high-level technical overview of our MPSU protocol. We first analyze the difficulties in constructing MPSU protocols and then show how to address these issues using our new techniques. The ideal functionality of MPSU is given in Fig.1.

PARAMETERS: k parties: $\mathcal{P}_1, \cdots, \mathcal{P}_k$; Set size n; The bit length of set elements σ.

FUNCTIONALITY:

- Wait for input $X_i = \{x_i^1, \cdots, x_i^n\} \subset \mathbb{F}_{2^\sigma}$ from $\mathcal{P}_i$.
- Give output $\bigcup_{i=1}^{k} X_i$ to $\mathcal{P}_1$.

Fig. 1. Multi-party Private Set Union Functionality $\mathcal{F}_{\mathsf{mpsu}}$

2.1 Difficulties in MPSU

The Difference Between MPSU and MPSI. Like MPSI, the security of MPSU requires that the receiver cannot obtain any information except for the union. However, unlike MPSI, the intersection must be a subset of the receiver's set, so we only need to consider the elements in the receiver's set. On the other hand, the union output by MPSU contains all the sets of the participants, so the protocol must involve the transmission of elements from all the participants' sets. And it is necessary to prevent an adversary from distinguishing which participant an element belongs to. Moreover, if different participants have the same element, the duplicates must be removed during the protocol execution to ensure that each element appears only once in the union; otherwise, the adversary will know how many participants have that element. Therefore, the MPSU protocol is more complex in its design than MPSI and has some efficiency gaps.

Difficulties in Extending from Two-party PSU. In two-party PSI, there is a function called private set membership test (PMT). In the PMT, the sender inputs a set X, and the receiver inputs an element y. The receiver can determine whether y belongs to X, while the sender cannot obtain any information.

However, PMT cannot be applied to two-party PSU. To compute the union, [33] proposed reverse PMT (RPMT), in which the sender inputs an element x, and the receiver inputs a set Y, then the receiver determines whether $x \in Y$. They combine RPMT and oblivious transfer (OT) to construct a PSU protocol. For each element $x \in X$, the sender and the receiver run the OT protocol with input $(x, \perp)$ and the boolean value of the expression $x \in Y$ (i.e., 1 if $x \in Y$ otherwise 0), respectively, where $\perp$ is a special symbol. Note that the receiver can obtain x if and only if $x \notin Y$, so the receiver can obtain $X \setminus Y$, and finally output $(X \setminus Y) \cup Y = X \cup Y$. Fig.2 illustrates this idea.

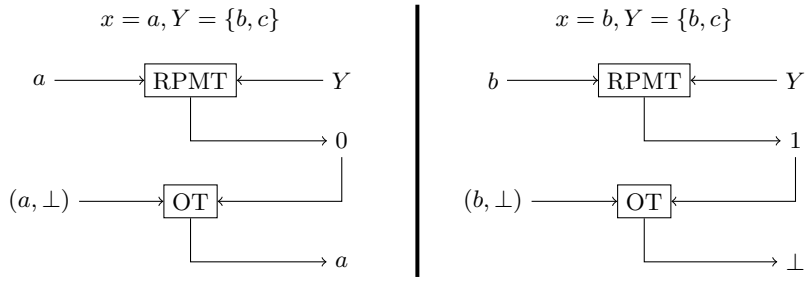

Fig. 2. Illustration of how to use RPMT and OT to perform PSU. The sender's set is $\{a, b\}$ and the receivers set is $\{b, c\}$. The left-hand side illustrates that the sender computes for his element a, which does not belong to Y. The right-hand side shows that the sender computes for his element $b \in Y$ that belongs to Y.

The core of the existing efficient two-party PSU protocols [12,21,24,28,33,49] is the efficient construction of RPMT. However, if we want to extend this idea to compute $X_1 \cup X_2 \cup \cdots \cup X_k$, which can be split into $X_1 \cup (X_2 \setminus X_1) \cup \cdots \cup (X_k \setminus (X_1 \cup \cdots \cup X_{k-1}))$, two problems arise:

1. Since the receiver in RPMT knows whether the sender's element belongs to the receiver's set, RPMT leaks the size of the cardinality of the difference set $|X \setminus Y|$ to the receiver, which is not allowed in the multi-party setting.
2. RPMT can only compute the difference between two sets but do not work in the multi-party setting, i.e., computing $X_k \setminus (X_1 \cup \cdots \cup X_{k-1})$ where $k \geq 3$.

Therefore, it is difficult to directly apply the techniques (RPMT) of two-party PSU to the multi-party setting. The construction of efficient MPSU protocols requires stronger functions.

2.2 Multi-Query Secret-Shared Private Membership Test

Based on the analysis in the previous section, the root cause of the first problem is that, in RPMT, the result is directly output to the receiver, which leads to information leakage. If we output the result of RPMT in the form of secret sharing, with each party holding a share of the output, then neither party can obtain any information. This function is called secret-shared RPMT. In this case, the roles of the two parties are completely symmetric, so secret-shared RPMT and secret-shared PMT (ssPMT) is the same function. We will use ssPMT to refer to this function in the following text.

As early as 2018, Ciampi et al. [14] combined PSI with secure two-party computation (2PC) to construct ssPMT. In 2021, Zhao et al. [50] formally defined ssPMT and built it based on the secure comparison protocol in [15]. However, their constructions have high communication overhead. Moreover, since only one element of the sender can be tested each time, multiple repetitions of ssPMT are needed to query all elements, which results in significant overhead.

In this work, we propose multi-query ssPMT (mq-ssPMT), which supports querying multiple elements of the sender simultaneously, thereby reducing the average cost per element. The ideal functionality of mq-ssPMT $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$ is given in Fig.3. mq-ssPMT can implement PMT(RPMT) simply by having the sender (receiver) send its share to the receiver (sender) to recover the result. Therefore, mq-ssPMT realizes a stronger function compared to PMT and RPMT. On the other hand, mq-ssPMT can also be used to construct two-party PSI or PSU protocols. For a PSI protocol, we only need to implement PMT. For a PSU protocol, we do not need to implement RPMT and then calculate the difference set using OT. We can directly construct a special input of OT: suppose the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ of mq-ssPMT each holds $e_0, e_1 \in \{0,1\}$, where $e_0 \oplus e_1$ is the boolean value of the expression $y \in X$. As the receiver of OT, $\mathcal{S}$ inputs the selection bit e_0, and as the sender of OT, $\mathcal{R}$ inputs a pair of messages $m_{e_1} = y$ and $m_{e_1 \oplus 1} = \perp$. Then, $\mathcal{S}$ can obtain the element y if and only if $e_0 = e_1$, that is, $y \notin X$, thus completing the computation of the union. Fig.4 illustrates the main idea behind this.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$; Set size n; The bit length of set elements σ.

FUNCTIONALITY:

- Wait an input $Y = \{y_1, \cdots, y_n\} \subset \mathbb{F}_{2^\sigma}$ from $\mathcal{R}$.
- Wait an input $X = \{x_1, \cdots, x_n\} \subset \mathbb{F}_{2^\sigma}$ from $\mathcal{S}$.
- For $i \in [n]$: Set $b_i = 1$ if $y_i \in X$, otherwise set $b_i = 0$. Sample $\boldsymbol{e}_0 \leftarrow \{0,1\}^n$ and compute $\boldsymbol{e}_1 = \boldsymbol{e}_0 \oplus \boldsymbol{b}$. Give $\boldsymbol{e}_0$ to $\mathcal{S}$ and give $\boldsymbol{e}_1$ to $\mathcal{R}$.

Fig. 3. Multi-Query Secret-Shared Private Membership Test Functionality $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$

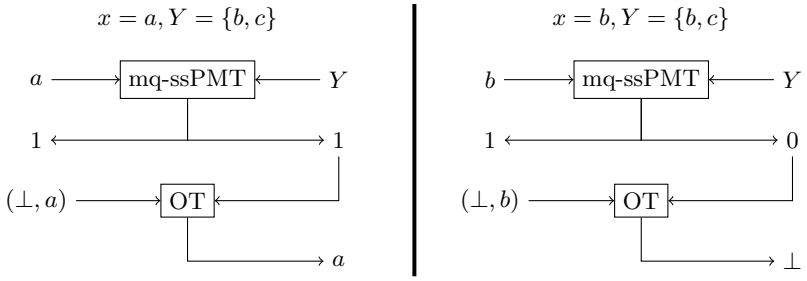

Fig. 4. Illustration of how to use mq-ssPMT and OT to perform PSU. The sender's set is $\{a, b\}$ and the receivers set is $\{b, c\}$. The left-hand side illustrates that the sender computes for his element a, which does not belong to Y. The right-hand side shows that the sender computes for his element $b \in Y$ that belongs to Y. We remark that the sender can query a, b simultaneously, and we separate a and b only for illustration.

2.3 MPSU Based on mq-ssPMT

We have now solved the first problem discussed in Sect. 2.1 using the new function mq-ssPMT. How can we solve the second problem, namely, how to compute $X_k \setminus (X_1 \cup \cdots \cup X_{k-1})$? Note that in MPSU, the adversary is not allowed to know any intermediate results $X_i \setminus (X_1 \cup \cdots \cup X_{i-1})(2 \leq i \leq k)$, so we cannot directly compute and output them. They must exist in some form of ciphertexts or secret sharings.

Let's consider a simplified setting where three parties $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ want to compute $X_3 \setminus (X_1 \cup X_2)$. In Sect. 2.2, we showed how to compute the difference set between two parties directly using mq-ssPMT. If we split $X_3 \setminus (X_1 \cup X_2)$ into $(X_3 \setminus X_1) \cap (X_3 \setminus X_2)$, and send not the element itself but the share of the element in the OT phase, then $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ can recover the element x if and only if $x \in X_3 \setminus X_1 \wedge x \in X_3 \setminus X_2$, i.e., $x \in X_3 \setminus (X_1 \cup X_2)$. Specifically, $\mathcal{P}_3$ acts as the receiver and executes the mq-ssPMT separately with $\mathcal{P}_1$ and $\mathcal{P}_2$. For any $x \in X_3$, $\mathcal{P}_3$ and $\mathcal{P}_1$ each hold secret shares $e_{31}^1, e_{31}^0 \in \{0,1\}$, indicating whether $x \in X_1$, and $\mathcal{P}_3$ and $\mathcal{P}_2$ each hold secret shares $e_{32}^1, e_{32}^0 \in \{0,1\}$, indicating whether $x \in X_2$. $\mathcal{P}_3$ uses additive secret sharing to split x into $x = x_1 \oplus x_2 \oplus x_3$, and then acts as the sender to execute the OT protocol separately with $\mathcal{P}_1$ and $\mathcal{P}_2$. Taking the OT with $\mathcal{P}_1$ as an example, $\mathcal{P}_3$ inputs $m_{e_{31}^1} = x_1$ and $m_{e_{31}^1 \oplus 1} = r_1$, where r_1 is a random value, and $\mathcal{P}_1$ inputs e_{31}^0. $\mathcal{P}_1$ can obtain the share x_1 if and only if $x \notin X_1$. Similarly, $\mathcal{P}_2$ can obtain the share x_2 if and only if $x \notin X_2$. Therefore, only when $x \notin (X_1 \cup X_2)$, that is, $x \in X_3 \setminus (X_1 \cup X_2)$, can $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ recover x, otherwise they can only get a random value.

In the same way, we can compute all $X_i \setminus (X_1 \cup \cdots \cup X_{i-1})(2 \leq i \leq k)$. Since any party always holds a share of his own set, any number of other participants can not collude to obtain his input information. However, if we want to merge all difference sets and output the union to $\mathcal{P}_1$, there are two remaining problems:

1. $\mathcal{P}_1$ knows the correspondence between the shares and the difference sets, so the shares cannot be directly sent to $\mathcal{P}_1$ for recovery.
2. Since $\mathcal{P}_1$ has no knowledge of the elements in the other parties' sets, and these elements do not necessarily have a specific structure. So $\mathcal{P}_1$ cannot distinguish between set elements and random values, i.e., $\mathcal{P}_1$ don't know which element he should add to the union.

To solve the first problem, we use a multi-party secret-shared shuffle (cf. Sect. 3.4) to randomly permute and re-share all the shares held by the parties. Since any $k-1$ parties don't know the information about the permutation and all shares are refreshed, the adversary can not find the correspondence between the difference sets and the shares. To solve the second problem, all parties append the hash value of the element to the end of it when performing secret sharing, i.e., sharing $x \| \mathsf{H}(x)$. When the output length of the hash function H is long enough, the probability that there exists an s satisfying $r = s \| \mathsf{H}(s)$ is negligible. Therefore, $\mathcal{P}_1$ can distinguish the set elements from random values with overwhelming probability.

Note that both mq-ssPMT and multi-party secret-shared shuffle have an efficient online phase, so our MPSU protocol is also efficient in the online phase.

3 Preliminaries

3.1 Notation

We use $\mathcal{P}_i$ to denote participants, X_i to represent the sets they hold, where each set has n σ-bit elements. k denotes the number of participants. We use λ, κ as the statistical and computational security parameters, respectively. $[n]$ denotes the set $1, 2, \cdots, n$. $\mathbb{F}_{2^\sigma}$ denotes the finite field composed of all σ-bit strings. We use $x \| y$ to denote the concatenation of two strings. We denote vectors with bold fonts and individual elements with indices. For example, $\boldsymbol{a}$ is a vector of n elements where each individual element is denoted as a_i. $\boldsymbol{a} \oplus \boldsymbol{b}$ represents $(a_1 \oplus b_1, \cdots, a_n \oplus b_n)$. $\pi(\boldsymbol{a})$ represents $(a_{\pi(1)}, \cdots, a_{\pi(n)})$, where π is a permutation on n items. We use $\coloneqq$ to denote assignment. For some set S, the notation $s \leftarrow S$ means that s is assigned a uniformly random element from S. By $\mathsf{negl}(\lambda)$ we denote a negligible function, i.e., a function f such that $f(\lambda) < \frac{1}{p(\lambda)}$ holds for any polynomial $p(\cdot)$ and sufficient large λ. We use the abbreviation PPT to denote probabilistic polynomial-time.

3.2 Symmetric-key Encryption

Our construction of mq-ssPMT is based on the mq-RPMT in [49], which uses symmetric-key encryption (SKE). We use the standard definition of SKE. To ensure the security of our mq-ssPMT, we require a security notion called *multi-message multi-ciphertext pseudorandomness* like the mq-RPMT in [49]. We give these definitions in Appendix A.

3.3 Oblivious Transfer

OT [42] is a foundational primitive in MPC, the functionality of 1-out-of-2 random OT (ROT) is given in Fig.5.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$; The bit length of message σ.

FUNCTIONALITY:

- Wait an input $b \in \{0, 1\}$ from $\mathcal{R}$.
- Sample $m_0, m_1 \leftarrow \mathbb{F}_{2^\sigma}$. Give (m_0, m_1) to S and give m_b to $\mathcal{R}$.

Fig. 5. 1-out-of-2 Random OT Functionality $\mathcal{F}_{\mathsf{rot}}$

3.4 Multi-party Secret-Shared Shuffle

Multi-party secret-shared shuffle can permute the share vectors of all parties randomly and refresh all shares, and the functionality is given in Fig.6. Early works [10, 35] focused on the construction for the two-party setting, and later, Eskandarian et al. [18] extended the protocol of [10] to the multi-party setting. Their protocol consists of an offline phase and an online phase. In the offline phase, each party generates a random permutation and a set of correlated vectors called share correlation. In the online phase, each party permutes and refreshes the share vectors efficiently using share correlation. We give the functionality of share correlation and details of their protocol in Appendix B.

PARAMETERS: k parties: $\mathcal{P}_1, \cdots, \mathcal{P}_k$; The dimension of vector n; The bit length of individual element σ.

FUNCTIONALITY:

- Wait an input $\boldsymbol{x}_i = (x_i^1, \cdots, x_i^n) \in \mathbb{F}_{2^\sigma}^n$ from each $\mathcal{P}_i (1 \leq i \leq k)$.
- Sample a random permutation $\pi : [n] \to [n]$. For $1 \leq i \leq k$, sample $\boldsymbol{x}'_i \leftarrow \mathbb{F}_{2^\sigma}^n$ satisfying

$$\bigoplus_{i=1}^{k} \boldsymbol{x}'_i = \pi\left(\bigoplus_{i=1}^{k} \boldsymbol{x}_i\right)$$

and give $\boldsymbol{x}'_i$ to $\mathcal{P}_i$.

Fig. 6. Multi-party Secret-Shared Shuffle Functionality $\mathcal{F}_{\mathsf{ms}}$

3.5 Oblivious Key-Value Stores

A key-value store [22, 38] is a data structure that stores a map from keys to corresponding values. The definition is as follows:

Definition 1. *A key-value store (KVS) is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a random value $r \in \{0,1\}^\kappa$, and consists of two algorithms:*

- $\mathsf{Encode}(\{(k_1, v_1), \cdots, (k_n, v_n)\}, r)$*: takes as input a set of $\{(k_i, v_i)\}_{i \in [n]} \subseteq \mathcal{K} \times \mathcal{V}$ and outputs an object D (or, with statistically small probability, an error indicator $\perp$).*
- Decode*: takes as input an object S, a key k, and outputs a value v.*

Correctness. For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(k, v) \in A \text{ and } \perp \neq D \leftarrow \mathsf{Encode}(A, r) \Rightarrow \mathsf{Decode}(D, k, r) = v$$

Obliviousness. For all distinct $\{k_1^0, \cdots, k_n^0\}$ and all distince $\{k_1^1, \cdots, k_n^1\}$, if Encode does not output $\perp$ for $\{k_1^0, \cdots, k_n^0\}$ or $\{k_1^1, \cdots, k_n^1\}$, then the output $\{D|v_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}(\{(k_1^0, v_1), \cdots, (k_n^0, v_n)\}, r)\}$ is computationally indistinguishable to $\{D|v_i \leftarrow \mathcal{V}, i \in [n], \mathsf{Encode}(\{(k_1^1, v_1), \cdots, (k_n^1, v_n)\}, r)\}$.

A KVS is an oblivious KVS (OKVS) if it satisfies the obliviousness property. In addition to the obliviousness, [49] also proposed the randomness property to prove the security of their mq-RPMT protocol. Our mq-ssPMT protocol similarly requires this property.

Randomness. For any $A = \{(k_1, v_1), \cdots, (k_n, v_n)\}$ and $k^* \notin \{k_1, \cdots, k_n\}$, the output of $\mathsf{Decode}(D, k^*, r)$ is statistically indistinguishable to that of uniform distribution over $\mathcal{V}$, where $D \leftarrow \mathsf{Encode}(A, r)$.

Garbled cuckoo table (GCT) is the most efficient construction of KVS, incluing 2H-GCT in [38] and 3H-GCT in [22]. And 3H-GCT satisfies both obliviousness and randomness si. Recently, Raghuraman et al. conducted a thorough theoretical and experimental analysis of the Encode algorithm in GCT, and presented the most efficient Encode algorithm to date within the commonly used parameters range. Due to space limitation, the formal description of their algorithm is given in deffer to Appendix C.

3.6 Security Model

In this work, we consider only the semi-honest model, where adversaries strictly follow the protocol specification but try to learn more than allowed by inspecting the protocol transcript. Furthermore, since our protocol involves multiple parties, we also consider collusion, which means an adversary can corrupt multiple parties and combine their views to infer more information. We adopt the standard definition of semi-honest security as defined in [37].

Definition 2. *Let $f : (\{0,1\}^*)^k \to (\{0,1\}^*)^k$ be an k-ary functionality, where $f_i(x_1, \cdots, x_k)$ denotes the ith element of $f(x_1, \cdots, x_k)$. For $I = \{i_1, \cdots, i_t\} \subseteq [k]$, let $f_I(x_1, \cdots, x_k)$ denote the subsequence $f_{i_1}(x_1, \cdots, x_k), \cdots, f_{i_t}(x_1, \cdots, x_k)$. Let Π be a k-party protocol for computing f. The view of the ith party Π during an execution of Π on $\bar{x} = (x_1, \cdots, x_k)$ is denoted by $\mathrm{VIEW}_i^{\Pi}(\bar{x})$. For any $I = \{i_1, \cdots, i_t\}$, we let $\mathrm{VIEW}_I^{\Pi}(\bar{x}) \stackrel{\text{def}}{=} (I, \mathrm{VIEW}_{i_1}^{\Pi}(\bar{x}), \cdots, \mathrm{VIEW}_{i_t}^{\Pi}(\bar{x}))$. We say Π privately computes f against semi-honest adversaries if there exists a PPT algorithm, denoted Sim, such that for every $I \subseteq [k]$, it holds that*

$$\{\mathsf{Sim}(I, (x_{i_1}, \cdots, x_{i_t}), f_I(\bar{x}))\}_{\bar{x} \in (\{0,1\}^*)^k} \stackrel{c}{\equiv} \{\mathrm{VIEW}_I^{\Pi}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^k}$$

4 Multi-Query Secret-Shared Private Membership Test

In this section, we describe the details of our efficient mq-ssPMT, which securely computes the functionality in Fig.3 in the presence of semi-honest adversaries. We first revisit the mq-RPMT in [49], then show how to build an mq-ssPMT protocol based on it.

4.1 Revisit mq-RPMT in ZCLZL23

Zhang et al. [49] observe that the reason why the RPMT proposed in [33] cannot support multiple queries is that they use the same indication string s for every element belonging to the receiver $\mathcal{R}$. Specifically, if an element x belongs to $\mathcal{R}$'s set $Y = \{y_1, \cdots, y_n\}$, then S will get s when queries x. Allowing $\mathcal{S}$ to make multiple queries will result in information leakage if S gets the same string when queries distinct elements, which means they belong to the intersection with overwhelming probability. A natural idea to address this issue is that $\mathcal{R}$ uses different indication strings for different elements, but this will allow $\mathcal{R}$ to know the specific elements from $\mathcal{S}$ based on the correspondence between the elements and the indication strings.

To tackle this challenge, in [49], $\mathcal{R}$ uses a randomized encryption scheme to encrypt an indication string s n times to get n different ciphertexts $s_1, \cdots, s_n$. Then $\mathcal{R}$ uses an OKVS to map each element to a s_i, i.e., compute $D \coloneqq \mathsf{Encode}(\{(y_1, s_1), \cdots, (y_n, s_n)\}, r)$, and sends D to $\mathcal{S}$. $\mathcal{S}$ queries each element $x_i (1 \leq i \leq n)$ in his set X to get $s_i^* \coloneqq \mathsf{Decode}(D, x_i, r)$. Then they use a newly introduced function called vector oblivious decryption-then-matching (VODM), where $\mathcal{S}$ inputs s^* and $\mathcal{R}$ inputs an encryption key and s, and then $\mathcal{R}$ knows whether $\mathsf{Dec}(k, s^*)$ equals s. Note that if $x_i \in Y$, s_i^* belongs to$\{s_1, \cdots, s_n\}$. If not, s_i^* is a random ciphertext due to the randomness property of OKVS. So their construction realizes mq-RPMT correctly. The security relies on the security of VODM and the randomness of OKVS.

[49] proposed VODM constructions for both public-key encryption (PKE) and SKE. For PKE, they use a re-randomizable PKE. $\mathcal{S}$ directly sends the re-randomized ciphertext to $\mathcal{R}$ for decryption. For SKE, they use GMW protocol to compute the decryption circuit of SKE. Then $\mathcal{S}$ sends the output shares to $\mathcal{R}$, who recovers the final result. They notice that randomized SKE causes ciphertext expansion. To avoid this problem, they proposed new construction for deterministic SKE: $\mathcal{R}$ uses a deterministic SKE to encrypt $0, 1, \cdots, n-1$ to obtain n different ciphertexts $s_1, \cdots, s_n$, and adds a comparison circuit at the end of the decryption circuit in VODM to check whether the decryption result is less than n. If it is, the element belongs to Y. Otherwise, it does not.

4.2 Construction of mq-ssPMT

It is worth noting that in [49], the output of mq-RPMT is exactly the output of VODM. If we realize secret-shared VODM (ssVODM), we can achieve mq-ssPMT. The ideal function of ssVODM is given in Fig.7. In the SKE-based mq-RPMT in [49], GMW is used to implement VODM. However, the output of GMW is already in the form of secret sharing. If we omit the last step of recovering the secret, this construction is actually ssVODM. Therefore, we can directly obtain an efficient mq-ssPMT from the mq-RPMT in [49]. According to the method of constructing the two-party PSU using mq-ssPMT in Sect. 2.2, the SKE-based PSU in [49] can reduce one round of communication and n bits of communication.

PARAMETERS: Sender $\mathcal{S}$, Receiver $\mathcal{R}$; Set size n; An encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

FUNCTIONALITY:

- Wait an input k, S from $\mathcal{R}$.
- Wait an input $\{s_1^*, \cdots, s_n^*\} \subset \{0,1\}^*$ from $\mathcal{S}$.
- For $i \in [n]$, compute $s_i' = \mathsf{Dec}(k, s_i^*)$. Set $b_i = 1$ if $s_i' \in S$, otherwise set $b_i = 0$. Sample $\boldsymbol{e}_0 \leftarrow \{0,1\}^n$, then compute $\boldsymbol{e}_1 = \boldsymbol{e}_0 \oplus \boldsymbol{b}$. Give $\boldsymbol{e}_0$ to $\mathcal{S}$. Give $\boldsymbol{e}_1$ to $\mathcal{R}$.

Fig. 7. Secret-Shared VODM Functionality $\mathcal{F}_{\mathsf{ssvodm}}$

Furthermore, the OKVS scheme used in [49] is 3H-GCT in [22]. We use an optimized 3H-GCT in [43] to reduce computation and communication costs. The details of our mq-ssPMT are shown in Fig.8.

PARAMETERS:

- Sender $\mathcal{S}$, Receiver $\mathcal{R}$.
- A SKE $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies multi-message multi-ciphertext pseudorandomness.
- An OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ and its random value r.
- Ideal functionality $\mathcal{F}_{\mathsf{ssvodm}}$ in Fig.7.

INPUT OF $\mathcal{S}$: $X = \{x_1, \cdots, x_n\} \subset \mathbb{F}_{2^\sigma}$.
INPUT OF $\mathcal{R}$: $Y = \{y_1, \cdots, y_n\} \subset \mathbb{F}_{2^\sigma}$.

PROTOCOL:

1. $\mathcal{S}$ runs $pp \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{KeyGen}(pp)$ to get a key k. For $i \in [n]$, computes $s_i = \mathsf{Enc}(k, i-1)$.
2. $\mathcal{S}$ computes an OKVS $D \coloneqq \mathsf{Encode}\big((x_1, s_1), \cdots, (x_n, s_n), r\big)$ and sends D to $\mathcal{R}$.
3. $\mathcal{R}$ computes $s_i^* \coloneqq \mathsf{Decode}(D, y_i, r)$ for $i \in [n]$.
4. S and $\mathcal{R}$ invokes $\mathcal{F}_{\mathsf{ssvodm}}$. $\mathcal{R}$ acts as sender with input $(s_1^*, \cdots, s_n^*)$. $\mathcal{S}$ acts as receiver with input $\{0, 1, \cdots, n-1\}, k$. $\mathcal{S}$ and $\mathcal{R}$ receive $\boldsymbol{e}_0, \boldsymbol{e}_1 \in \{0,1\}^n$, respectively.

Fig. 8. mq-ssPMT Protocol $\Pi_{\mathsf{mq\text{-}sspmt}}$

Security. Regarding the security of $\Pi_{\mathsf{mq\text{-}sspmt}}$, we have the following theorem.

Theorem 1. *Assume the SKE scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies multi-message multi-ciphertext pseudorandomness. The protocol in Fig.8 securely computes $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$ against semi-honest adversaries in the $\mathcal{F}_{\mathsf{ssvodm}}$-hybrid model.*

Since $\Pi_{\mathsf{mq\text{-}sspmt}}$ is essentially the same as mq-RPMT in [49] except for omitting the last step of secret reconstruction, the security proof of $\Pi_{\mathsf{mq\text{-}sspmt}}$ is similar to that of mq-RPMT and is not repeated here. We recommend interested readers to refer to the proof in [49].

5 Protocol Overviews and Details

In this section, we give the details of our MPSU protocol. We also construct multi-party private set union cardinality (MPSU-CA), MPSI, and multi-party private set intersection cardinality (MPSI-CA) protocols based on mq-ssPMT and multi-party secret-shared shuffle.

5.1 MPSU

Our MPSU protocol follows the approach outlined in Sect. 2. Each participant $\mathcal{P}_i$ $(2 \leq i \leq k)$ acts as the receiver and performs the mq-ssPMT protocol with all $\mathcal{P}_j$ $(1 \leq j < i)$. For each element in the set of $\mathcal{P}_i$, $\mathcal{P}_i$ splits it into i shares using addtively secret sharing and shares with all $\mathcal{P}_j$. Since each share is a random value, we use random OT to generate each share, and $\mathcal{P}_i$ computes his own share locally to reduce the communication overhead caused by OT. Next, all parties combine their shares of all the difference sets (if there is no corresponding share, set it to 0) to generate a $(k-1) \times n$ dimensional vector and set it as the input to the multi-party secret-shared shuffle protocol. Then, $\mathcal{P}_2, \cdots, \mathcal{P}_k$ send their new shares to $\mathcal{P}_1$, who recovers $\bigcup_{i=2}^{k} X_i \setminus X_1$. Finally, $\mathcal{P}_1$ outputs $\bigcup_{i=1}^{k} X_i = X_1 \cup (\bigcup_{i=2}^{k} X_i \setminus X_1)$. The protocol details are given in Fig.9.

Correctness. We first prove $\bigcup_{i=1}^{k} X_i \subseteq X_1 \cup Y$. For any $x \in \bigcup_{i=1}^{k} X_i$, if $x \in X_1$, then $x \in X_1 \cup Y$. Otherwise, there exists a unique j that satisfies $x \in X_j \setminus (X_1 \cup \cdots \cup X_{j-1})$. Therefore, when $\mathcal{P}_j$ shares $x \| \mathsf{H}(x)$ to all $\mathcal{P}_i (1 \leq i < j)$ using ROT, all $\mathcal{P}_i$ will choose the share instead of the random value. Because the multi-party secret-shared shuffle does not affect the correctness of the recovery, $\mathcal{P}_1$ will always obtain $x \| \mathsf{H}(x)$ and add x to the output.

Then, we prove $(X_1 \cup Y) \subseteq \bigcup_{i=1}^{k} X_i$, which is equivalent to proving $Y \subset \bigcup_{i=1}^{k} X_i$. Since Y is a subset of $\{z_1, \cdots, z_{(k-1)n}\}$, we only need to consider each individual z_i. If there is no $x \in \bigcup_{i=1}^{k} X_i$ that satisfies $z_i \neq x \| \mathsf{H}(x)$, then z_i must be a random value due to the randomness of ROT's output. Therefore, $\Pr[z_i = s \| \mathsf{H}(s) \text{ for some } s] = 2^{-\ell}$. By a union bound, we have:

$$\Pr\left[X_1 \cup Y \not\subseteq \bigcup_{i=1}^{k} X_i\right] \leq (k-1) \cdot n \cdot 2^{-\ell} = 2^{\log(k-1) + \log n - \ell}$$

When $\ell \geq \lambda + \log(k-1) + \log n$, the probability is negligible.

Security. Now we prove the security of Π_{mpsu} in Fig.9.

Theorem 2. *Π_{mpsu} in Fig.9 securely computes $\mathcal{F}_{\mathsf{mpsu}}$ against any semi-honest adversary that does not corrupt $\mathcal{P}_1$ and any subset of $\{\mathcal{P}_2, \cdots, \mathcal{P}_k\}$ simultaneously in the $(\mathcal{F}_{\mathsf{mq\text{-}sspmt}}, \mathcal{F}_{\mathsf{rot}}, \mathcal{F}_{\mathsf{ms}})$-hybrid model.*

PARAMETERS:

- k parties: $\mathcal{P}_1, \cdots, \mathcal{P}_k$.
- Ideal functionalities $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$ in Fig.3, $\mathcal{F}_{\mathsf{rot}}$ in Fig.5, $\mathcal{F}_{\mathsf{ms}}$ in Fig.6.
- A collision-resistant hash function $\mathsf{H}(x) : \{0,1\}^* \to \{0,1\}^\ell$.

INPUT OF $\mathcal{P}_i$: $X_i = \{x_i^1, \cdots, x_i^n\} \subset \mathbb{F}_{2^\sigma}$.

PROTOCOL:

1. For $1 \le i < j \le k$, $\mathcal{P}_i$ and $\mathcal{P}_j$ invoke $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$. $\mathcal{P}_i$ acts as sender with input X_i. $\mathcal{P}_j$ acts receiver with input X_j. $\mathcal{P}_j, \mathcal{P}_i$ receive $\boldsymbol{e}_{ji}^0, \boldsymbol{e}_{ji}^1 \in \{0,1\}^n$, respectively.
2. For $1 \le i < j \le k, 1 \le t \le n$:
   - $\mathcal{P}_i$ and $\mathcal{P}_j$ invoke $\mathcal{F}_{\mathsf{rot}}$.
   - $\mathcal{P}_j$ acts as sender with no input.
   - $\mathcal{P}_i$ acts as receiver with input $e_{ji,t}^1$.
   - $\mathcal{P}_j$ receives $r_{ji,t}^0, r_{ji,t}^1 \in \{0,1\}^{\sigma+\ell}$. $\mathcal{P}_i$ receives $r_{ji,t} \coloneqq r_{ji,t}^{e_{ji,t}^1}$.
3. For $2 \le j \le k$, $\mathcal{P}_j$ computes $r_{jj,t} \coloneqq \left(x_j^t \| \mathsf{H}(x_j^t)\right) \oplus \bigoplus_{i=1}^{j-1} r_{ji,t}^{e_{ji,t}^0}$ for $t \in [n]$.
4. Each $\mathcal{P}_i$ computes $\boldsymbol{sh}_i \in \mathbb{F}_{2^{\sigma+\ell}}^{(k-1)n}$ as follows: for $\max(2,i) \le j \le k, 1 \le t \le n$, $sh_{i,(j-2)n+t} \coloneqq r_{ji,t}$. Set all other positions to 0.
5. All $\mathcal{P}_i$ invoke $\mathcal{F}_{\mathsf{ms}}$ with input $\boldsymbol{sh}_i$. $\mathcal{P}_i$ receives $\boldsymbol{sh}_i'$.
6. For $2 \le i \le k$, $\mathcal{P}_i$ sends $\boldsymbol{sh}_i'$ to $\mathcal{P}_1$. $\mathcal{P}_1$ recovers $\boldsymbol{z} \coloneqq \bigoplus_{i=1}^k \boldsymbol{sh}_i'$. Set $Y \coloneqq \emptyset$. For $1 \le i \le (k-1)n$, if $z_i = s \| \mathsf{H}(s)$ holds for some $s \in \mathbb{F}_{2^\sigma}$, $\mathcal{P}_1$ computes $Y = Y \cup \{z_i\}$. Outputs $X_1 \cup Y$.

Fig. 9. Multi-party Private Set Union Protocol Π_{mpsu}

Proof. Let C and H be a coalition of corrupt and honest parties, respectively. $|C| = \eta$. To show how to simulate C's view in the ideal model, we consider two cases based on whether $\mathcal{P}_1$ is corrupted.

$\mathcal{P}_1$ is Honest. In this case, $\mathcal{P}_1 \notin C$. $\mathsf{Sim}_C(X_{i_1}, \cdots, X_{i_\eta})$ runs as follows:

1. For all $\mathcal{P}_i \in C$, Sim_C samples $\boldsymbol{e}_{iu}'^0 \leftarrow \{0,1\}^n$ for $1 \le u < i$ and $\boldsymbol{e}_{vi}'^1 \leftarrow \{0,1\}^n$ for $i < v \le k$, which satisfy for all $\mathcal{P}_i, \mathcal{P}_j \in C (i < j), 1 \le t \le n$:
   - $e_{ji,t}'^0 \oplus e_{ji,t}'^1 = 1$, if $x_j^t \in X_i$
   - $e_{ji,t}'^0 \oplus e_{ji,t}'^1 = 0$, if $x_j^t \notin X_i$

   Then Sim_C appends all $\boldsymbol{e}_{iu}'^0, \boldsymbol{e}_{vi}'^1$ to the view.
2. For all $P_i \in C$, Sim_C samples $\boldsymbol{r}_{iu}'^0, \boldsymbol{r}_{iu}'^1 \leftarrow \mathbb{F}_{2^{\sigma+\ell}}^n$ for $1 \le u < i$ and $\boldsymbol{r}_{vi}' \leftarrow \mathbb{F}_{2^{\sigma+\ell}}^n$ for $i < v \le k$, which satisfy for all $\mathcal{P}_i, \mathcal{P}_j \in C (i < j)$ and all $1 \le t \le n$:

$$\forall 1 \le t \le n, r_{ji,t}' = r_{ji,t}'^{e_{ji,t}'^1}$$

   Then Sim_C appends all $\boldsymbol{r}_{iu}'^0, \boldsymbol{r}_{iu}'^1, \boldsymbol{r}_{vi}'$ to the view.
3. For all $\mathcal{P}_i \in C$, Sim_C samples $\boldsymbol{sh}_i'' \leftarrow \mathbb{F}_{2^{\sigma+\ell}}^{(k-1)n}$ and appends is to the view.
4. For all $\mathcal{P}_i \in C$, Sim_C invokes mq-ssPMT simulator $\mathsf{Sim}_{\mathsf{mq\text{-}sspmt}}^{\mathcal{R}}(X_i, \boldsymbol{e}_{iu}'^0)$ for $1 \le u < i$ and $\mathsf{Sim}_{\mathsf{mq\text{-}sspmt}}^{\mathcal{S}}(X_i, \boldsymbol{e}_{vi}'^1)$ for $i < v \le k$. Then appends the output to the view.

5. For all $\mathcal{P}_i \in C$, Sim_C invokes ROT simulator $\mathsf{Sim}^{\mathcal{S}}_{\mathsf{rot}}(\boldsymbol{r}'^0_{iu}, \boldsymbol{r}'^1_{iu})$ for $1 \leq u < i$ and $\mathsf{Sim}^{\mathcal{R}}_{\mathsf{rot}}(\boldsymbol{e}'^1_{vi}, \boldsymbol{r}'_{vi})$ for $i < v \leq k$. Then appends the output to the view.
6. For all $\mathcal{P}_i \in C$, Sim_C creates $\boldsymbol{sh}_i$ as Step 3 and Step 4 of Π_{mpsu}. Then invokes multi-party secret-shared shuffle simulator $\mathsf{Sim}^{\mathcal{P}_i}_{\mathsf{ms}}(\boldsymbol{sh}_i, \boldsymbol{sh}''_i)$ and appends the output to the view.

Now we argue that the view output by Sim_C is indistinguishable from the real one. In the real world, the output $\boldsymbol{e}^0_{ji}, \boldsymbol{e}^1_{ji}$ of mq-ssPMT, the output $\boldsymbol{r}^0_{ji}, \boldsymbol{r}^1_{ji}$ of ROT, and the output $\boldsymbol{sh}'_i$ of multi-party secret-shared shuffle are uniformly random from the perspective of a single corrupted party. Even if all the parties in C combine their views, the outputs of the mq-ssPMT and ROT protocols run with the honest parties, i.e., $\boldsymbol{e}^0_{ji}, \boldsymbol{e}^1_{ji}, \boldsymbol{r}^0_{ji}, \boldsymbol{r}^1_{ji} (\mathcal{P}_i \in H$ or $\mathcal{P}_j \in H)$, are uniformly random and mutually independent. Moreover, the outputs of the protocols run with the corrupted parties, i.e., $\boldsymbol{e}^0_{ji}, \boldsymbol{e}^1_{ji}, \boldsymbol{r}^0_{ji}, \boldsymbol{r}^1_{ji} (\mathcal{P}_i, \mathcal{P}_j \in C)$, are still uniformly random but constrained by the correctness of mq-ssPMT and ROT.

As for the output of multi-party secret-shared shuffle protocol, at least one share $\boldsymbol{sh}'_h$ is unknown because the number of corrupted parties is always less than k. So from the perspective of C, the output $\boldsymbol{sh}'_i (\mathcal{P}_i \in C)$ are uniformly random and independent of each other.

Notice that in the view output by Sim_C, all messages are uniformly random and satisfy the correctness constraints, which is exactly the same as that of real world. So the simulated view is computationally indistinguishable from the real.

$\mathcal{P}_1$ *is Corrupted.* In this case, $C = \{\mathcal{P}_1\}$. So the simulator $\mathsf{Sim}_C(X_1, \bigcup_{i=1}^k X_i)$ needs to simulate $\mathcal{P}_1$'s view. Sim_C runs as follows:

1. Sim_C samples $\boldsymbol{e}'^1_{i1} \leftarrow \{0,1\}^n$ for $2 \leq i \leq k$ and appends them to the view.
2. Sim_C samples $\boldsymbol{r}'_{i1} \leftarrow \mathbb{F}^n_{2^{\sigma+\ell}}$ for $2 \leq i \leq k$ and appends them to the view.
3. Sim_C computes $Y' \coloneqq (\bigcup_{i=1}^k X_i) \setminus X_1$, and constructs $\boldsymbol{z}' \in \mathbb{F}^{(k-1)n}_{2^{\sigma+\ell}}$ as follows:
   - for $\forall y_i \in Y'$, $z'_i \coloneqq y_i \| \mathsf{H}(y_i)$
   - for $|Y'| < i \leq (k-1)n$, samples $z'_i \leftarrow \mathbb{F}_{2^{\sigma+\ell}}$

   Then, Sim_C samples a random permutation $\pi : [(k-1)n] \to [(k-1)n]$, and computes $\boldsymbol{z}'' \coloneqq \pi(\boldsymbol{z}')$.
4. For $1 \leq i \leq k$, Sim_C samples $\boldsymbol{sh}''_i \leftarrow \mathbb{F}^{(k-1)n}_{2^{\sigma+\ell}}$, which satisfies $\bigoplus_{i=1}^k \boldsymbol{sh}''_i = \boldsymbol{z}''$. Then Sim_C appends all $\boldsymbol{sh}''_i$ to the view.
5. Sim_C invokes mq-ssPMT simulator $\mathsf{Sim}^{\mathcal{S}}_{\mathsf{mq\text{-}sspmt}}(X_1, \boldsymbol{e}'^1_{i1})$ for $2 \leq i \leq k$. Then appends the output to the view.
6. Sim_C invokes ROT simulator $\mathsf{Sim}^{\mathcal{R}}_{\mathsf{rot}}(\boldsymbol{e}'^1_{i1}, \boldsymbol{r}'_{i1})$ for $2 \leq i \leq k$. Then appends the output to the view.
7. Sim_C constructs $\boldsymbol{sh}_1$ as Step 3 and Step 4 of Π_{mpsu}. Then invokes multi-party secret-shared shuffle simulator $\mathsf{Sim}^{\mathcal{P}_1}_{\mathsf{ms}}(\boldsymbol{sh}_1, \boldsymbol{sh}''_1)$ and appends the output to the view.

Now we argue that the view output by Sim_C is indistinguishable from the real one. Specifically, we need to prove $\boldsymbol{sh}'_i$ from each $\mathcal{P}_i$ does not leak any other information except for the union. For all $2 \leq j \leq k$, consider an element $x^t_j \in X_j$.

If there exists some $X_i (1 \leq i < j)$ that $x_j^t \in X_i$, then $e_{ji,t}^0 \oplus e_{ji,t}^1 = 1, r_{ji,t} = r_{ji,t}^{e_{ji,t}^1} \neq r_{ji,t}^{e_{ji,t}^0}$. So $\bigoplus_{i=1}^k sh_{i,(j-2)n+t} = \bigoplus_{i=1}^j r_{ji,t} = r \oplus r_{ji,t}^{e_{ji,t}^0} \oplus r_{ji,t}^{e_{ji,t}^1}$ is uniformly random from the perspective of $\mathcal{P}_1$, where r is the sum of remaining terms. So in the real world, the individual elements of $\bigoplus_{i=1}^k \boldsymbol{sh}_i'$ are all uniformly random values except for $|\bigcup_{i=2}^k X_i \setminus X_1|$ elements, which is the same as simulated view. So the simulated view is computationally indistinguishable from the real. □

An Efficiency Optimization. In our MPSU protocol, each $\mathcal{P}_i (1 \leq i \leq k-2)$ acts as sender to execute mq-ssPMT with $\mathcal{P}_{i+1}, \cdots, \mathcal{P}_k$. And in each mq-ssPMT, $\mathcal{P}_i$ needs to encrypt n messages and compute a corresponding OKVS. We observe that $\mathcal{P}_i$ can use the same OKVS in all mq-ssPMT, which could avoid additional computation costs caused by multiple encryptions and Encode without compromising the security of the protocol. The only change required is the length of ciphertexts of SKE, which should be increased from $\lambda + 2\log n$ to $\lambda + 2\log n + 2\log(k-i)$. So we can guarantee that the probability of collisions between the random results of Decode and the ciphertexts of indication strings is negligible. Therefore, we can ensure the correctness of the protocol.

Insecurity Against Arbitrary Collusion. We now illustrate why we need the assumption that $\mathcal{P}_1$ does not collude with others. In our protocol, $\mathcal{P}_1$ reconstructs the vector $\boldsymbol{z}$ in Step 6, which is a permutation of $\bigoplus_{i=1}^k \boldsymbol{sh}_i$. So each individual element of $\boldsymbol{z}$ is the form of $\bigoplus_{i=1}^j r_{ji,t}$. However, all $r_{ji,t} (1 \leq i < j, 1 \leq t \leq n)$ is known by $\mathcal{P}_j$, which is not uniformly random from the perspective of colluding $\mathcal{P}_1, \mathcal{P}_j$. More specifically, they can recover the output of $\mathcal{P}_i$ in ROT. So they could get the output of $\mathcal{P}_i$ in mq-ssPMT with $\mathcal{P}_j$, which will reveal the information of X_i. We argue that although we require this special assumption, we achieve security against any number of semi-honest clients and significant improvement in efficiency.

MPSU-CA. MPSU-CA is a variant of MPSU, where the receiver is only allowed to know the cardinality of the union. To fill the gap between MPSU and MPSU-CA, we only need to make minor modifications to our MPSU protocol. Each party no longer shares his own elements, but instead, an indication string s agreed on in advance. Finally, $\mathcal{P}_i$ could count the number of s to get the cardinality of the union.

Another difference is that since each party agrees on an indication string, the problem of distinguishing random values and set elements in MPSU does not arise here. So we do not need to append a hash at the end of each element, which could reduce communication costs. The details of our MPSU-CA protocol are given in the full version.

5.2 MPSI

We now discuss MPSI. We first give the functionality in Fig.10. Since the intersection must be a subset of X_1, we only need to share X_1. So $\mathcal{P}_1$ acts as sender

and executes mq-ssPMT with all other $\mathcal{P}_i (2 \leq i < k)$. On the other hand, unlike computing the union, the intersection $\bigcap_{i=1}^{k} X_i$ can be decomposed into $\bigcap_{i=1}^{k} (X_1 \cap X_i)$. Therefore, each $\mathcal{P}_i$ should obtain the share of the intersection $X_1 \cap X_i$ instead of the share of the difference set $X_1 \setminus X_i$. Therefore, $\mathcal{P}_i$ cannot directly use the output of mq-ssPMT as the selection bit in ROT as in the MPSU protocol but should choose the other message. The detailed description is given in Fig.11.

PARAMETERS: k parties $\mathcal{P}_1, \cdots, \mathcal{P}_k$; Set size n; The bit length of set elements σ.

FUNCTIONALITY:

- Wait for input $X_i = \{x_i^1, \cdots, x_i^n\} \subset \mathbb{F}_{2^\sigma}$ from $\mathcal{P}_i$.
- Give output $\bigcap_{i=1}^{k} X_i$ to $\mathcal{P}_1$.

Fig. 10. Multi-party Private Set Intersection Functionality $\mathcal{F}_{\mathsf{mpsi}}$

PARAMETERS:

- k parties: $\mathcal{P}_1, \cdots, \mathcal{P}_k$.
- Ideal functionalities $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$ in Fig.3, $\mathcal{F}_{\mathsf{rot}}$ in Fig.5, $\mathcal{F}_{\mathsf{ms}}$ in Fig.6.
- The bit length of indication string ℓ.

INPUT OF $\mathcal{P}_i$: $X_i = \{x_i^1, \cdots, x_i^n\} \subset \mathbb{F}_{2^\sigma}$.

PROTOCOL:

1. For $2 \leq i \leq k$, $\mathcal{P}_1$ and $\mathcal{P}_i$ invoke $\mathcal{F}_{\mathsf{mq\text{-}sspmt}}$. $\mathcal{P}_i$ acts as sender with input X_i, $\mathcal{P}_1$ acts as receiver with input X_1. $\mathcal{P}_1, \mathcal{P}_i$ receive $\boldsymbol{e}_{1i}^0, \boldsymbol{e}_{1i}^1 \in \{0,1\}^n$, respectively.
2. For $2 \leq i \leq k, 1 \leq t \leq n$:
   - $\mathcal{P}_1$ and $\mathcal{P}_i$ invoke $\mathcal{F}_{\mathsf{rot}}$.
   - $\mathcal{P}_i$ acts as sender with no input.
   - $\mathcal{P}_1$ acts as receiver with input $e_{1i,t}^0 \oplus 1$.
   - $\mathcal{P}_i$ receives $r_{1i,t}^0, r_{1i,t}^1 \in \{0,1\}^{\sigma+\ell}$. $\mathcal{P}_1$ receives $r_{1i,t} \coloneqq r_{1i,t}^{e_{1i,t}^0 \oplus 1}$.
3. For $1 \leq t \leq n$, $\mathcal{P}_1$ computes $r_{11,t} \coloneqq \left(x_1^t \| \mathsf{H}(x_1^t)\right) \oplus \bigoplus_{i=2}^{k} r_{1i,t}$.
4. Each $\mathcal{P}_i (2 \leq i \leq k)$ computes $\boldsymbol{sh}_i \in \mathbb{F}_{2^{\sigma+\ell}}^n$ as follows: for $1 \leq t \leq n$, $sh_t \coloneqq r_{1i,t}^{e_{1i,t}^1}$. $\mathcal{P}_1$ computes $\boldsymbol{sh}_1 \in \mathbb{F}_{2^{\sigma+\ell}}^n$ as follows: for $1 \leq t \leq n$, $sh_t \coloneqq r_{1i,t}$.
5. All $\mathcal{P}_i (1 \leq i \leq k)$ invoke $\mathcal{F}_{\mathsf{ms}}$ with input $\boldsymbol{sh}_i$. $\mathcal{P}_i$ receives $\boldsymbol{sh}_i'$.
6. For $2 \leq i \leq k$, $\mathcal{P}_i$ sends $\boldsymbol{sh}_i'$ to P_1. P_1 recovers $\boldsymbol{z} \coloneqq \bigoplus_{i=1}^{k} \boldsymbol{sh}_i'$. Set $Y \coloneqq \emptyset$. For $1 \leq i \leq n$, if $z_i = x \| \mathsf{H}(x)$ holds for some $x \in X_1$, computes $Y = Y \cup \{z_i\}$. Outputs Y.

Fig. 11. Multi-party Private Set Intersection Protocol Π_{mpsi}

Correctness. Similar to the analysis of MPSU protocol in Sect.5.1. Since z_i needs to satisfy $z_i = x \| \mathsf{H}(x)$ and $x \in X_1$, the probability of Π_{mpsi} outputting a wrong result does not exceed $n \cdot 2^{-\sigma} \cdot 2^{-\ell} = 2^{\log n - \sigma - \ell}$. If $\sigma - \log n \geq \lambda$, it's not necessary to append $\mathsf{H}(x)$ to the end of x to ensure that the error probability is less than $2^{-\lambda}$. If $\sigma - \log n < \lambda$, we need $\ell \geq \lambda + \log n - \sigma$.

Security. We now prove the security of Π_{mpsi} in Fig.11.

Theorem 3. *Π_{mpsi} in Fig.11 securely computes $\mathcal{F}_{\mathsf{mpsi}}$ against any semi-honest adversary that does not corrupt $\mathcal{P}_1$ and any subset of $\{\mathcal{P}_2, \cdots, \mathcal{P}_k\}$ simultaneously in the $(\mathcal{F}_{\mathsf{mq\text{-}sspmt}}, \mathcal{F}_{\mathsf{rot}}, \mathcal{F}_{\mathsf{ms}})$-hybrid model.*

Proof. Since the proof is similar to the proof of theorem 2, we leave it to Appendix D.

MPSI-CA. Like our MPSU-CA protocol, we could similarly build MPSI-CA based on our MPSI protocol. In contrast to MPSI, $\mathcal{P}_1$ shares an indication string s instead of his own elements, just like MPSU-CA protocol. The details are given in the full version.

Remark 1. These approaches for MPSI and MPSI-CA are not competitive with the state-of-the-art special-purpose protocols for MPSI and MPSI-CA. In particular, mq-ssPMT and multi-party secret-shared shuffle is unnecessary for them. We include these two protocols merely for illustrative purposes.

6 Complexity Analysis

In this section, we analyze the computation and communication complexity of our four protocols.

6.1 mq-ssPMT

We first analyze the complexity of our mq-ssPMT protocol. The costs of mq-ssPMT can be divided into three parts: the cost of SKE encryption, the cost of OKVS, and the cost of ssVODM. Similar to [49], we use LowMC [1] to initialize SKE and implement ssVODM using 2PC. In addition to the decryption circuit of LowMC, we also need a comparison circuit. In this work, we only consider the case where $n = 2^q$, so we can use the method in [49]: to compare whether a σ-bit string is less than n, we only need to check if one of its first $\sigma - \log n$ bits is 1, so we need a total of $\sigma - \log n - 1$ AND gates. Therefore, the ssVODM protocol requires a total of $n(t + \sigma - \log n - 1) = O(tn)$ AND gates, where t is the number of AND gates in the SKE decryption circuit. We now calculate the cost of each part.

- SKE encryption: The computation complexity of encrypting $0, 1, \cdots, n-1$ is $O(n)$.

- OKVS: We use the optimized 3H-GCT algorithm in [43]. We employ it with a cluster size of 2^{14}, weight $w = 3$. These result in the size of OKVS is $1.28n|c|$ bits, where $|c|$ is the size of a ciphertext of SKE. The computation complexities of $\mathsf{Encode}, \mathsf{Decode}$ are both $O(n)$. Sending an OKVS to other parties needs one round communication.
- ssVODM: We use GMW to employ 2PC. Notice that we do not need to share the input, i.e., the sender of ssVODM could use 0 as the share of $s_1^*, \cdots, s_n^*$ and the receiver could use 0 as the share of the decryption key k. Therefore, the costs contain only the evaluation of the AND gates. Using Beaver triple [3], each AND gate requires 4 bits communication and $O(1)$ computation. Therefore, the communication costs are $4n(t + \sigma - \log n - 1)$ bits and the computation complexity is $O(n(t + \sigma - \log n))$. Since the round complexity of GMW depends on the depth of AND gates, which is one in each round of LowMC decryption and $\log_2(\sigma - \log n)$ in string comparison, the round complexity is $O(\log(\sigma - \log n))$.

According to the analysis in Sect.5.1, the length of a ciphertext is no more than $\lambda + 2\log(kn)$. In the scenario we are studying, where $\sigma \leq 128, n \leq 2^{24}, k \leq 20$, we have t is greater than σ and $\log(kn)$. Therefore, we can approximate $O(t + \sigma + \log(kn))$ as $O(t)$. Thus, in mq-ssPMT, the computation complexity is $O(nt)$, the communication complexity is $O(n(t + \lambda))$ bits, and the round complexity is $O(\log(\sigma - \log n))$.

6.2 MPSU and MPSU-CA

MPSU. The costs of our MPSU protocol can be split to four parts, including mq-ssPMT, ROT, multi-party secret-shared shuffle and secret reconstruction. We analyze the costs of these parts, respectively.

- mq-ssPMT: Since $\mathcal{P}_i$ executes mq-ssPMT $k - 1$ times, the communication complexity is $O((t + \lambda)kn)$, and the computation complexity is $O(tkn)$ for all parties. Moreover, $\mathcal{P}_i$ can run mq-ssPMT with others in parallel, so the round complexity is $O(\log(\sigma - \log n))$.
- ROT: $\mathcal{P}_i$ executes ROT $k - 1$ times, so the communication complexity is $O(\kappa kn)$, and the computation complexity is $O(kn)$ for all parties. Since we use a constant-round ROT and execute all ROT in parallel, the round complexity is $O(1)$.
- Multi-party secret-shared shuffle: We use the protocol in [18]. In its offline phase, each pair of parties need to run a share translation protocol in [10]. Omitting message sizes and log factors, the computation complexity and the communication complexity for each party are both $\tilde{O}(k^2 n)$. In the online phase, since we need to shuffle $(\sigma + \ell)$-bits elements, the computation complexity is $O(kn)$ for each party, and the communication complexity is $O((\sigma+\ell)k^2 n)$ for $\mathcal{P}_1$ and $O((\sigma+\ell)kn)$ for $\mathcal{P}_2, \cdots, \mathcal{P}_k$ according to the details described in Appendix B. The round complexity is $O(k)$.

– Secret reconstruction: The communication complexity and computation complexity of $\mathcal{P}_1$ is $O((\sigma+\ell)k^2n)$ and $O(n)$, respectively. The communication complexity of other parties is $O((\sigma+\ell)kn)$. And it needs one round communication.

Therefore, omitting the costs in the offline phase and taking $\ell = \lambda + \log n + \log(k-1)$. The computation complexity of all parties is $O(tkn)$. The communication complexity of $\mathcal{P}_1$ is $O((t+\kappa+(\sigma+\lambda+\log(kn))k)kn)$. The communication complexity of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is $O((t+\lambda+\kappa)kn)$. The round complexity is $O(\log(\sigma - \log n) + k)$.

MPSU-CA. Our MPSU-CA protocol differs from the MPSU protocol only in the elements being shared. In MPSU, the length of the secret being shared is $\sigma+\ell$, while in MPSU-CA it is ℓ. We take $\ell = \lambda + \log n + \log(k-1)$. The computation complexity of all parties is $O(tkn)$. The communication complexity of $\mathcal{P}_1$ is $O((t+\kappa+(\lambda+\log(kn))k)kn)$. The communication complexity of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is $O((t+\lambda+\kappa)kn)$. The round complexity is $O(\log(\sigma - \log n) + k)$.

6.3 MPSI and MPSI-CA

MPSI. Our MPSI protocol's costs can also be divided into mq-ssPMT, ROT, multi-party secret-shared shuffle, and secret reconstruction. We analyze the costs of these parts, respectively.

– mq-ssPMT: Since $\mathcal{P}_1$ needs to act as receiver to execute mq-ssPMT with all $\mathcal{P}_2, \cdots, \mathcal{P}_k$, the communication complexity is $O((t+\lambda)kn)$ and the computation complexity is $O(tkn)$ for $\mathcal{P}_1$. The communication complexity is $O((t+\lambda)n)$ and the computation complexity is $O(tn)$ for $\mathcal{P}_2, \cdots, \mathcal{P}_k$.
– ROT: Assume the average communication cost per ROT is $c_{ot} = O(\kappa)$ bits. Since $\mathcal{P}_1$ needs to run ROT with $\mathcal{P}_2, \cdots, \mathcal{P}_k$, the communication complexity and the computation complexity of $\mathcal{P}_1$ is $O(\kappa kn)$ and $O(kn)$, respectively. The communication complexity and the computation complexity of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is $O(\kappa n)$ and $O(n)$, respectively.
– Multi-party secret-shared shuffle: Same as the analysis in Sect.6.2, the computation of each party is $O(n)$. The communication complexity of $\mathcal{P}_1$ is $O((\sigma+\ell)kn)$.The communication complexity of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is $O((\sigma+\ell)n)$.
– Secret reconstruction: The communication complexity and computation complexity of $\mathcal{P}_1$ is $O((\sigma+\ell)kn)$ and $O(n)$, respectively. The communication complexity of other parties is $O((\sigma+\ell)n)$.

Omitting the costs in the offline phase, since $\sigma + \ell = \max\{\lambda + \log n, \sigma\}$, the computation complexity and communication complexity of $\mathcal{P}_1$ is $O(tkn)$ and $O((t+\kappa+\lambda)kn)$, respectively. The computation complexity and communication complexity of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is $O(tn)$ and $O((t+\kappa+\lambda)n)$, respectively. Similarly, the round complexity is $O(\log(\sigma - \log n) + k)$.

MPSI-CA. The bit length of the secret shared in our MPSI protocol is $\sigma + \ell = \max\{\lambda + \log n, \sigma\}$, which is the same as that of MPSI-CA. Therefore, they have the same complexity.

We conclude the complexity of our four protocols in Table2.

Table 2. The conclusion and comparison of our MPSU, MPSU-CA, MPSI, and MPSI-CA protocols. n is the set size. k is the number of parties. σ is the bit length of set elements. t is the number of AND gates in an SKE decryption circuit. λ is statistical security parameter. κ is computational security parameter. The computation complexity refers to the number of symmetric-key operations.

<table>
<tr><th rowspan="2">Protocol</th><th colspan="3">Comm.</th><th colspan="2">Comp.</th></tr>
<tr><th>Leader</th><th>Client</th><th>Rounds</th><th>Leader</th><th>Client</th></tr>
<tr><td>MPSU</td><td>$O((t+\kappa+(\sigma+\lambda+\log(kn))k)kn)$</td><td rowspan="2">$O((t+\lambda+\kappa)kn)$</td><td rowspan="4">$O(\log(\sigma-\log n)+k)$</td><td colspan="2" rowspan="2">$O(tkn)$</td></tr>
<tr><td>MPSU-CA</td><td>$O((t+\kappa+(\lambda+\log(kn))k)kn)$</td></tr>
<tr><td>MPSI</td><td rowspan="2">$O((t+\lambda+\kappa)kn)$</td><td rowspan="2">$O((t+\lambda+\kappa)n)$</td><td>$O(tkn)$</td><td>$O(tn)$</td></tr>
<tr><td>MPSI-CA</td></tr>
</table>

7 Implementation

In this section, we provide experimental details and test results for MPSU, and compare our results with previous work. We ignore the costs of the offline phase, including the generation of base OTs, share correlations, and Beaver triples. All experimental data are the average of 10 trials under the same environment. We compute the communication costs of a party as the sum of the data he sent and received.

7.1 Experimental Setup

We run all protocols on a single Intel Ice Lake processor at 3.2GHz with 256 GB RAM. We emulate the two network connections using Linux `tc` command. For the LAN setting, we set network latency to 0.02 ms and bandwidth of 10 Gbps and for the WAN setting the latency is set to 40 ms and bandwidth 400 Mbps.

7.2 Implementation Details

For concrete analysis we set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. Our protocol is written in C++ and we use the following libraries in our implementation.

– OKVS and GMW: We use the optimized 3H-GCT in [43] as our OKVS instantiation, and re-use the implementation of 3H-GCT and GMW by the authors of [43][1].

[1] https://github.com/Visa-Research/volepsi.git

- LowMC: We set both block size and key length to 128 bits, and the number of Sbox to 10, so the number of rounds is 20. Therefore, the number of AND gates in decryption circuit is $t = 600$. The concrete parameters we use are from [49][2]. And we use the implementation of LowMC by the authors of [1][3].
- ROT: We use SoftSpokenOT [45] implemented in libOTe[4], and set field bits to 5 to balance computation and communication costs.
- Others: We utilize the implementations of circuit, PRNG, and hash function provided by cryptoTools[5]. In addition, we adopt Coproto[6] to realize network communication.

7.3 Comparison with Prior Work

Since only [5] and [48] have implemented their protocols so far, we only compare our work with these two works.

Blanton et al. [5]. Since their implementation is not available, we can only use the experimental results they published in their papers. They implemented the 3-party PSU protocol in C++, assuming an honest majority, and tested it in a LAN with a bandwidth of 1 Gbps on 2.4GHz AMD Opteron. They tested their protocol with 32-bit elements and set sizes of $2^4, 2^6, 2^8, 2^{10}$, and the running times were 0.13s, 0.52s, 2.41s, and 11.89s, respectively. In contrast, our protocol, which is secure under the assumption that $\mathcal{P}_1$ does not collude with other parties, runs in a 10 Gbps bandwidth environment in the same setting, taking 0.10s, 0.10s, 0.11s, and 0.14s, respectively.

Vos et al. [48]. Vos et al.'s protocol employs a divide-and-conquer approach, so the computational cost and communication overhead are related to the distribution of the input set. Precisely, if the input elements are concentrated in neighboring regions of the universe, then most branches can be pruned by the divide-and-conquer algorithm. If the input elements are more dispersed, then more secure OR operations are required. We test their open-source code on random datasets and set the divide-and-conquer parameter $D = 2$ to minimize the number of secure OR operations required. The (expected) communication overhead is calculated using the formula provided in their paper. Finally, since in their implementation parties transmit data directly through memory, which is not affected by bandwidth, we only test our protocol in a LAN and compare it with theirs. The results are shown in Table3.

We test our protocol for set sizes of $n = \{2^4, 2^6, 2^8, 2^{10}\}$ and different numbers of parties $k = \{3, 4, 5, 7, 10\}$. For each set size, we set the universe size to be

[2] https://github.com/alibaba-edu/mpc4j/blob/adee91f7966a3166f6e662f6b4a321ea36fcf39d/mpc4j-common-tool/src/main/resources/low_mc/lowmc_128_128_20.txt
[3] https://github.com/LowMC/lowmc.git
[4] https://github.com/osu-crypto/libOTe.git
[5] https://github.com/ladnir/cryptoTools.git
[6] https://github.com/Visa-Research/coproto.git

Table 3. The comparison of [48] and our MPSU protocol in running time (seconds) and communication cost (MB) in the LAN setting. The bit length of our elements is 64. The output length of H is $\ell = 64$. The size of universe in [48] is set to $|\mathcal{U}| = 2^{16}$. The parameter in divide-and-conquer is $D = 2$. The communication cost refers to Leader's communication cost. In our protocol, each party uses $k-1$ threads to interact with all other parties separately, and 4 threads are used to perform parallel SKE encryption. In their protocol, the leader uses $k-1$ threads to interact with all other parties simultaneously. Bold numbers indicate the best results under current conditions. Cells with - means there is almost no improvement.

	Number Parties k	Protocol	Set Size n 2^4	2^6	2^8	2^{10}
Time	3	[48]	0.56	1.71	4.84	15.36
		Ours	**0.10**	**0.10**	**0.11**	**0.14**
	4	[48]	0.76	2.36	7.64	20.84
		Ours	**0.15**	**0.16**	**0.17**	**0.19**
	5	[48]	1.08	3.50	10.73	26.43
		Ours	**0.22**	**0.22**	**0.23**	**0.24**
	7	[48]	1.84	4.49	15.29	52.82
		Ours	**0.36**	**0.36**	**0.37**	**0.39**
	10	[48]	3.15	9.12	29.65	75.58
		Ours	**0.58**	**0.62**	**0.63**	**0.68**
	Speedup		5×	12×	41×	109×
Comm.	3	[48]	0.16	0.56	1.82	5.68
		Ours	**0.15**	**0.16**	**0.28**	**0.96**
	4	[48]	0.25	0.84	2.74	8.52
		Ours	**0.22**	**0.24**	**0.45**	**1.54**
	5	[48]	0.33	1.11	3.65	11.36
		Ours	**0.30**	**0.33**	**0.63**	**2.17**
	7	[48]	0.49	1.67	5.47	17.03
		Ours	**0.45**	**0.52**	**1.04**	**3.63**
	10	[48]	0.74	2.51	8.21	25.55
		Ours	**0.69**	**0.83**	**1.77**	**6.30**
	Speedup		-	3×	4×	4×

$|\mathcal{U}| = 2^{16}$ to make [48] achieve higher efficiency. From the perspective of running time, our protocol shows significant efficiency improvements compared to [48]. Moreover, it gains greater improvements for larger set sizes. In particular, under the condition of a set size of 2^{10}, our protocol achieves a 109× speedup. If the set size is increased to 2^{20}, there will be a remarkable improvement. However, [48] already takes 75 seconds for 10 parties and a set size of 2^{10}, and testing larger sets will take several tens of minutes or even an hour and require a larger input domain, such as $|\mathcal{U}| = 2^{32}$, which will further decrease the efficiency of [48]. Therefore, we believe that testing for a set size of 2^{10} is sufficient to demonstrate that our protocol is more efficient than [48].

From the perspective of communication overhead, all communication costs in [48] come from secure OR operations, so we should minimize the number

of secure OR operations required. Therefore, we chose $D = 2$ as the optimal parameter for the expected communication overhead. Nevertheless, our protocol still achieves 3-4x improvements in communication overhead. Specifically, when $n = 2^{10}$ and $k = 10$, the communication overhead of the leader in their protocol is 25.55MB, while our protocol only requires 6.30MB. Moreover, as the universe size increases, the gap becomes even more significant.

7.4 Scalability

To demonstrate the scalability and practicality of our protocol, we test our protocol for larger set sizes of $n = 2^{14}, 2^{16}, 2^{18}, 2^{20}$ and different number of parties $k = 3, 4, 5, 7, 10$. For each party, we use $k - 1$ threads to interact with all other parties simultaneously and 4 threads to perform parallel SKE encryption.

Running Time. The running time of our protocol in both LAN and WAN settings are shown in Table4. Our protocol demonstrates good scalability. In the LAN setting, the running time of the protocol increases linearly with the set size. Specifically, for the 3-party setting, computing MPSU for small sets ($n = 2^{14}$) takes less than one second; medium-sized sets ($n = 2^{16}, 2^{18}$) can be computed within 10s; and large sets ($n = 2^{20}$) only require 29.02s. Moreover, since our protocol can be well parallelized, it also shows good scalability as the number of parties increases. For example, for $n = 2^{18}$, when the number of parties increases from 3 to 10, the running time only increases by 2.8×, and when the number of parties increases from 3 to 7 for $n = 2^{20}$, the running time only increases by 1.6×. On the other hand, our protocol has reasonable communication overhead, making it also efficient in the WAN setting. For example, in the 10-party setting, computing $n = 2^{18}$ takes around 2 minutes, and in the 7-party setting, computing $n = 2^{20}$ takes around 4 minutes.

Table 4. Running time (seconds) of our protocol in LAN and WAN settings. Each party holds n 64-bit elements. The output length of H is $\ell = 64$. Cells with - denotes trials that ran out of memory.

Setting	Number Parties k	Set Size n			
		2^{14}	2^{16}	2^{18}	2^{20}
LAN	3	0.55	1.79	7.04	29.02
	4	0.60	1.88	7.46	30.28
	5	0.67	2.01	7.92	34.10
	7	0.88	2.71	10.77	45.68
	10	1.41	4.89	19.90	-
WAN	3	3.36	6.64	15.38	51.81
	4	4.14	8.63	20.28	72.61
	5	5.53	10.56	29.35	111.06
	7	6.91	17.21	60.17	227.75
	10	11.08	33.89	127.71	-

Communication Overhead. The communication overhead of the protocol is shown in Table5. $\mathcal{P}_1$ has the highest communication overhead, and the communication overhead of $\mathcal{P}_2, \cdots, \mathcal{P}_{k-1}$ is the same, and the communication overhead of $\mathcal{P}_k$ is slightly lower than that of $\mathcal{P}_i (2 \leq i \leq k-1)$. The communication overhead of all parties is linearly proportional to the set size. The communication overhead of $\mathcal{P}_2, \cdots, \mathcal{P}_k$ is linearly proportional to the number of parties. The communication overhead of $\mathcal{P}_1$ grows quadratically with the number of parties, as it need to reconstruct the secret.

Table 5. Communication (MB) of our protocol for different set sizes and different numbers of parties. Each party holds n 64-bit elements. The output length of H is $\ell = 64$. $\mathcal{P}_i$ denotes $\mathcal{P}_2, \cdots, \mathcal{P}_{k-1}$. Cells with - denotes trials that ran out of memory.

Number Parties	Set Size n											
	2^{14}			2^{16}			2^{18}			2^{20}		
k	$\mathcal{P}_1$	$\mathcal{P}_i$	$\mathcal{P}_k$	$\mathcal{P}_1$	$\mathcal{P}_i$	$\mathcal{P}_k$	$\mathcal{P}_1$	$\mathcal{P}_i$	$\mathcal{P}_k$	$\mathcal{P}_1$	$\mathcal{P}_i$	$\mathcal{P}_k$
3	14.43	13.93	13.43	57.58	55.58	53.58	229.76	221.76	213.76	917	885	853
4	23.14	20.89	20.14	92.38	83.38	80.38	368.64	332.64	320.64	1472	1328	1280
5	32.86	27.86	26.86	131.17	111.17	107.17	523.52	443.52	427.52	2090	1770	1706
7	55.29	41.79	40.29	220.75	166.75	160.75	881.28	665.28	641.28	3519	2655	2559
10	96.43	62.68	60.43	385.13	250.13	241.13	1537.92	997.92	961.92	-	-	-

8 Conclusion

In this work, we introduce a new protocol called mq-ssPMT, which is an effective technique in multi-party private set operations. We also give an efficient construction of mq-ssPMT, which is mainly based on symmetric-key operations. By combining with multi-party secret-shared shuffle and ROT, we propose a multi-party private set operation framework from mq-ssPMT, including MPSU, MPSU-CA, MPSI, and MPSI-CA. We stress that although our protocols require the assumption that the leader does not collude with others, which achieves weaker security, our MPSU protocol is the first protocol that reports on large-size experiments and is truly scalable. We leave the construction of efficient MPSU protocol which resists arbitrary collusion as a future work.

Acknowledgement

We thank all the anonymous reviewers for helpful feedback on the write-up. This work is supported by the National Key Research and Development Program of China (2022YFB2701600), National Natural Science Foundation of China (U21A20467, 61932011, 61972019), and Beijing Natural Science Foundation (M21033, M21031).

A Symmetric-key Encryption

A SKE scheme is a tuple of four algorithms:

- $\mathsf{Setup}(1^\kappa)$: on input the security parameter κ outputs public parameters pp, which include the description of the message and ciphertext space $\mathcal{M}, \mathcal{C}$.
- $\mathsf{KeyGen}(pp)$: on input public parameter pp outputs a key k.
- $\mathsf{Enc}(k, m)$: on input a key k and a plaintext $m \in \mathcal{M}$, outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec}(k, c)$: on input a key k and a ciphertext$c \in \mathcal{C}$, outputs a message $m \in \mathcal{M}$ or an error symbol$\perp$.

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

Security. To ensure the security of our mq-ssPMT, we require a security notion called *multi-message multi-ciphertext pseudorandomness* like the mq-RPMT in [49]. Formally, a SKE is multi-message multi-ciphertext pseudorandom if for any PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\mathsf{Adv}_{\mathcal{A}}(1^\kappa) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(pp); \\ (m_1, \cdots, m_n, state) \leftarrow \mathcal{A}_1(pp); \\ \beta \leftarrow \{0,1\}; \\ \text{for } i \in [n] : c_{i,0} \leftarrow \mathsf{Enc}(k, m_i), c_{i,1} \leftarrow \mathcal{C}; \\ \beta' \leftarrow \mathcal{A}_2(pp, state, \{c_{i,\beta}\}_{i\in[n]}) \end{array}\right] - \frac{1}{2}$$

is negligible is κ.

B Multi-party Secret-Shared Shuffle

The functionality of share correlation is given in Fig.12. The protocol details of multi-party secret-shared shuffle in [18] are given in Fig.13.

C Garbled Cuckoo Table

The formal description of GCT in [43] is given in Fig.14.

D Proof of Theorem 3

Below we give the details of the proof of Theorem 3.

Proof. Let C and H be a coalition of corrupt and honest parties, respectively. $|C| = \eta$. To show how to simulate C's view in the ideal model, we consider two cases based on whether $\mathcal{P}_1$ is corrupted.

PARAMETERS: k parties: $\mathcal{P}_1, \cdots, \mathcal{P}_k$; The dimension of vector n; The bit length of individual element σ.

FUNCTIONALITY:

- Wait an input $\pi_i : [n] \to [n]$ from each $\mathcal{P}_i (1 \leq i \leq k)$.
- Sample $\boldsymbol{a}'_i, \boldsymbol{b}_i \leftarrow \mathbb{F}^n_{2^\sigma}$ for $1 \leq i \leq k-1$ and $\boldsymbol{a}_i \leftarrow \mathbb{F}^n_{2^\sigma}, \boldsymbol{\Delta} \leftarrow \mathbb{F}^n_{2^\sigma}$ for $2 \leq i \leq k$, which satisfy:

$$\boldsymbol{\Delta} = \pi_k\left(\cdots\left(\pi_2\Big(\pi_1\big(\bigoplus_{i=2}^{k}\boldsymbol{a}_i\big)\oplus\boldsymbol{a}'_1\Big)\oplus\boldsymbol{a}'_2\right)\cdots\oplus\boldsymbol{a}'_{k-1}\right)\oplus\bigoplus_{i=1}^{k-1}\boldsymbol{b}_i$$

  Give $\boldsymbol{a}'_1, \boldsymbol{b}_1$ to $\mathcal{P}_1$. Give $\boldsymbol{a}'_i, \boldsymbol{a}_i, \boldsymbol{b}_i$ to $\mathcal{P}_i (2 \leq i \leq k-1)$. Give $\boldsymbol{a}_k, \boldsymbol{\Delta}_k$ to $\mathcal{P}_k$.

Fig. 12. Share Correlation Functionality $\mathcal{F}_{\mathsf{sc}}$

PARAMETERS:

- k parties $\mathcal{P}_1, \cdots, \mathcal{P}_k$.
- Ideal functionality $\mathcal{F}_{\mathsf{sc}}$ in Fig.12.
- The dimension of vector n and the bit length of individual element σ.

INPUT OF $\mathcal{P}_i$: $[\boldsymbol{x}]_i$, a share of $\boldsymbol{x} \in \mathbb{F}^n_{2^\sigma}$ based on addtively secret sharing.

PROTOCOL:

1. Each $\mathcal{P}_i$ sample a random permutation $\pi_i : [n] \to [n]$ and invokes $\mathcal{F}_{\mathsf{sc}}$ with input π_i. $\mathcal{P}_1$ receives $\boldsymbol{a}'_1, \boldsymbol{b}_1 \in \mathbb{F}^n_{2^\sigma}$. $\mathcal{P}_i (2 \leq i \leq k-1)$ receives $\boldsymbol{a}'_i, \boldsymbol{a}_i, \boldsymbol{b}_i \in \mathbb{F}^n_{2^\sigma}$. $\mathcal{P}_k$ receives $\boldsymbol{a}_k, \boldsymbol{\Delta}_k \in \mathbb{F}^n_{2^\sigma}$.
2. For $2 \leq i \leq k$, $\mathcal{P}_i$ compute $\boldsymbol{z}_i := [\boldsymbol{x}]_i \oplus \boldsymbol{a}_i$, and send $\boldsymbol{z}_i$ to $\mathcal{P}_1$.
3. $\mathcal{P}_1$ compute $\boldsymbol{z}'_1 := \pi_1\big(\bigoplus_{i=2}^{k}\boldsymbol{z}_i \oplus [\boldsymbol{x}]_1\big) \oplus \boldsymbol{a}'_1$ and send it to $\mathcal{P}_2$. $\mathcal{P}_1$ outputs $\boldsymbol{b}_1$.
4. For $2 \leq i \leq k-1 \mathcal{P}_i$ computes $\boldsymbol{z}'_i := \pi_i(\boldsymbol{z}'_{i-1}) \oplus \boldsymbol{a}'_i$ and sends it to $\mathcal{P}_{i+1}$. $\mathcal{P}_i$ output $\boldsymbol{b}_i$.
5. $\mathcal{P}_k$ outputs $\pi_k(\boldsymbol{z}'_{k-1}) \oplus \boldsymbol{\Delta}_k$.

Fig. 13. Multi-party Secret-Shared Shuffle Protocol Π_{ms}

$\mathcal{P}_1$ *is Honest.* In this case, $\mathcal{P}_1 \notin C$. $\mathsf{Sim}_C(X_{i_1}, \cdots, X_{i_\eta})$ runs as follows:

1. For all $\mathcal{P}_i \in C$, Sim_C samples $\boldsymbol{e}'^1_{1i} \leftarrow \{0,1\}^n, \boldsymbol{r}'^0_{1i}, \boldsymbol{r}'^1_{1i} \leftarrow \mathbb{F}^n_{2^{\sigma+\ell}}, \boldsymbol{sh}''_i \leftarrow \mathbb{F}^n_{2^{\sigma+\ell}}$ and appends them to the view.
2. For all $\mathcal{P}_i \in C$, Sim_C invokes mq-ssPMT simulator $\mathsf{Sim}^{\mathcal{S}}_{\mathsf{mq\text{-}sspmt}}(X_i, \boldsymbol{e}'^1_i)$ and appends the output to the view.
3. For all $\mathcal{P}_i \in C$, Sim_C invokes ROT simulator $\mathsf{Sim}^{\mathcal{S}}_{\mathsf{rot}}(\boldsymbol{r}'^0_{1i}, \boldsymbol{r}'^1_{1i})$ and appends the output to the view.
4. For all $\mathcal{P}_i \in C$, Sim_C creates $\boldsymbol{sh}_i$ as Step 4 of Π_{mpsi}. Then invokes multi-party secret-shared shuffle simulator $\mathsf{Sim}^{\mathcal{P}_i}_{\mathsf{ms}}(\boldsymbol{sh}_i, \boldsymbol{sh}''_i)$ and appends the output to the view.

Now we argue that the view output by Sim_C is indistinguishable from the real one. In the real world, the output $\boldsymbol{e}^1_{1i}$ of mq-ssPMT, the output $\boldsymbol{r}^0_{1i}, \boldsymbol{r}^1_{1i}$ of ROT, and the output $\boldsymbol{sh}'_i$ of multi-party secret-shared shuffle are uniformly

PARAMETERS:

- Statistical security parameter λ and computational security parameter κ.
- Finite field $\mathbb{F}$ and finite group $\mathbb{G}$.
- Output length $m = m' + \hat{m}$ where $m' = O(n), \hat{m} = O(\lambda)$.

Input: n key-value pairs $\{(k_1, v_1), \cdots, (k_n, v_n)\} \subseteq \mathcal{K} \times \mathcal{V}$.

$\mathsf{Encode}(\{(k_1, v_1), \cdots, (k_n, v_n)\}, r)$:

(1) **[Sample]** Let $\mathsf{row}' : \mathcal{K} \times \{0,1\}^\kappa \to S_w$ and $\mathsf{r\hat{o}w} : \mathcal{K} \times \{0,1\}^\kappa \to \mathbb{F}^{\hat{m}}$ be random functions where $S_w \subset \{0,1\}^{m'}$ is the set of all weight w strings. Let $\mathsf{row}(k, r) \coloneqq \mathsf{row}'(k, r) \| \mathsf{r\hat{o}w}(k, r)$ and define

$$H \coloneqq \begin{bmatrix} \mathsf{row}(k_1, r) \\ \cdots \\ \mathsf{row}(k_n, r) \end{bmatrix} \in \mathbb{F}^{n \times m}$$

(2) **[Triangulate]** Let $H' \coloneqq H, J \coloneqq \emptyset$. While H' has rows:
   (a) Select $j \in [m]$ such that the jth (sparse) column of H' has the minimum non-zero weight.
   (b) Append index j to the ordered list J. Remove all rows $i \in [n]$ from H' for which $H'_{i,j} \neq 0$.

   Define $\delta \coloneqq |J|$, the gap as $g \coloneqq n - \delta$, permutation matrices $\pi^r \in \{0,1\}^{n \times n}, \pi^c \in \{0,1\}^{m \times m}$ such that $\pi^c_{m-k, m-\delta-k} = 1$ for $k \in [0, \hat{m})$, $\pi^c_{J_i, m+1-i} = 1$ and $\pi^r_{n+1-i, i'} = 1$ for some i' where $H_{i', J_i} \neq 0$ and all $i \in [\delta]$. Let

$$T \coloneqq \pi^r \cdot H \cdot \pi^c = \begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix}$$

   where $F \in \{0,1\}^{\delta \times \delta}$ is lower triangular, $B \in \mathbb{F}^{g \times \hat{m}}, E \in \mathbb{F}^{\delta \times \hat{m}}$ are the dense columns.

(3) **[Zero-C]** Compute $T' \coloneqq \begin{bmatrix} I & -CF^{-1} \\ 0 & I \end{bmatrix} \cdot T = \begin{bmatrix} A' & B' & 0 \\ D & E & F \end{bmatrix}$.

(4) **[Solve-Dense]** If B' doesn't have full row rank, return $\perp$. Let $B^* \coloneqq QB'$ be the (lower) reduced row echelon form of B', and

$$T^* \coloneqq \begin{bmatrix} Q & 0 \\ 0 & I \end{bmatrix} \cdot T' = \begin{bmatrix} A^* & B^* & 0 \\ D & E & F \end{bmatrix}, v^* \coloneqq \begin{bmatrix} Q & -QCF^{-1} \\ 0 & I \end{bmatrix} \cdot \pi^r \cdot v$$

   where $v = (v_1, \cdots, v_n)^T \in \mathbb{F}^{n \times 1}$.

(5) **[Back-substitution]** Compute $\boldsymbol{P}^* \coloneqq T^{*-1} \cdot v^*$ via back-substitution and return $\boldsymbol{P} \coloneqq \boldsymbol{P}^* \cdot \pi^{c-1}$.

$\mathsf{Decode}(\boldsymbol{P}, k, r)$: return $\langle \mathsf{row}(k, r), \boldsymbol{P} \rangle$.

Fig. 14. GCT algorithm in [43]

random. Moreover, the outputs of mq-ssPMT and ROT of all parties in C are mutually independent.

As for the output of multi-party secret-shared shuffle protocol, the share $\boldsymbol{sh}'_1$ is unknown and uniformly random for corrupted parties because $\mathcal{P}_1$ is honest. So all $\boldsymbol{sh}'_i(\mathcal{P}_i \in C)$ are uniformly random and independent of each other from the perspective of C. Notice that in the simulated view, all messages are uniformly random and mutually independent, so the output view of Sim_C is computationally indistinguishable from real.

$\mathcal{P}_1$ is Corrupted. In this case, $C = \{\mathcal{P}_1\}$. So the simulator $\mathsf{Sim}_C(X_1, \bigcap_{i=1}^k X_i)$ needs to simulate $\mathcal{P}_1$'s view. Sim_C runs as follows:

1. For $2 \le i \le k$, Sim_C samples $\boldsymbol{e}'^0_{1i} \leftarrow \{0,1\}^n$ and $\boldsymbol{r}'_{1i} \leftarrow \mathbb{F}^n_{2^{\sigma+\ell}}$. Then appends all these vectors to the view.
2. Sim_C constructs $\boldsymbol{z}' \in \mathbb{F}^n_{2^{\sigma+\ell}}$ as follows:
   - Set $\boldsymbol{z}'$ uninitialized. For each $x \in \bigcap_{i=1}^k X_i$, Sim_C computes $x \| \mathsf{H}(x)$ and set a random uninitialized position of $\boldsymbol{z}'$ to this value.
   - For all uninitialized positions of $\boldsymbol{z}'$, set a random value from $\mathbb{F}_{2^{\sigma+\ell}}$.
3. Sim_C samples $\boldsymbol{sh}''_i \leftarrow \mathbb{F}^n_{2^{\sigma+\ell}}$ for all $1 \le i \le k$, which satisfies $\bigoplus_{i=1}^k \boldsymbol{sh}''_i = \boldsymbol{z}'$. Sim_C appends all $\boldsymbol{sh}''_i$ to the view.
4. Sim_C invokes mq-ssPMT simulator $\mathsf{Sim}^{\mathcal{R}}_{\mathsf{mq\text{-}sspmt}}(X_1, \boldsymbol{e}'^1_{1i})$ for $2 \le i \le k$. Then appends the output to the view.
5. Sim_C invokes ROT simulator $\mathsf{Sim}^{\mathcal{R}}_{\mathsf{rot}}(\boldsymbol{e}'^0_{1i} \oplus 1^n, \boldsymbol{r}'_{1i})$ for $2 \le i \le k$. Then appends the output to the view.
6. Sim_C constructs $\boldsymbol{sh}_1$ as Step 3 and Step 4 of Π_{mpsi}. Then invokes multi-party secret-shared shuffle simulator $\mathsf{Sim}^{\mathcal{P}_i}_{\mathsf{ms}}(\boldsymbol{sh}_1, \boldsymbol{sh}''_1)$ and appends the output to the view.

Now we argue that the view output by Sim_C is indistinguishable from the real one. In the real world, the output $\boldsymbol{e}^0_{1i}$ of mq-ssPMT, the output $\boldsymbol{r}_{1i}$ of ROT, and the output $\boldsymbol{sh}'_i$ of multi-party secret-shared shuffle are uniformly random. Moreover, for different i, they are independent of each other. We prove that $\boldsymbol{sh}'_i$ from each $\mathcal{P}_i$ does not leak any other information except for the intersection. For each element x^t_1 that belongs to X_1, if there exists $2 \le i \le k$ that $x^t_1 \notin X_1 \cap X_i$, we have $e'^0_{1i,t} \oplus e'^1_{1i,t} = 0, r_{1i,t} = r^{e^0_{1i,t} \oplus 1}_{1i,t} \ne r^{e^1_{ji,t}}_{ji,t}$. Therefore, $\bigoplus_{i=1}^k sh_{i,t} = \bigoplus_{i=1}^k r_{1i,t} = r \oplus r^{e^1_{1i,t}}_{1i,t} \oplus r^{e^0_{1i,t} \oplus 1}_{1i,t}$ is uniformly random from the perspective of $\mathcal{P}_1$, where r is the sum of remaining terms. So in the real world, $\bigoplus_{i=1}^k \boldsymbol{sh}'_i$ is uniformly random except for $|\bigcap_{i=1}^k X_i|$ positions. So the simulated view is computationally indistinguishable from the real. □

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 430–454. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

2. Aranha, D.F., Lin, C., Orlandi, C., Simkin, M.: Laconic private set-intersection from pairings. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 111124. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560642
3. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991). https://doi.org/10.1007/3-540-46766-1_34
4. Ben-Efraim, A., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A.: Psimple: Practical multiparty maliciously-secure private set intersection. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. p. 10981112. ASIA CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3488932.3523254
5. Blanton, M., Aguiar, E.: Private and oblivious set and multiset operations. Int. J. Inf. Sec. **15**(5), 493–518 (2016). https://doi.org/10.1007/s10207-015-0301-1
6. Brickell, J., Shmatikov, V.: Privacy-preserving graph algorithms in the semi-honest model. In: Roy, B. (ed.) Advances in Cryptology - ASIACRYPT 2005. pp. 236–252. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
7. Bui, D., Couteau, G.: Improved private set intersection forăsets withăsmall entries. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography – PKC 2023. pp. 190–220. Springer Nature Switzerland, Cham (2023)
8. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.A.: SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In: 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings. pp. 223–240. USENIX Association (2010)
9. Chandran, N., Dasgupta, N., Gupta, D., Obbattu, S.L.B., Sekar, S., Shah, A.: Efficient linear multiparty psi and extensions to circuit/quorum psi. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 11821204. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484591
10. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 342–372. Springer International Publishing, Cham (2020)
11. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious prf. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 34–63. Springer International Publishing, Cham (2020)
12. Chen, Y., Zhang, M., Zhang, C., Dong, M., Liu, W.: Private set operations from multi-query reverse private membership test. Cryptology ePrint Archive, Paper 2022/652 (2022), https://eprint.iacr.org/2022/652
13. Chongchitmate, W., Ishai, Y., Lu, S., Ostrovsky, R.: Psi from ring-ole. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 531545. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3559378
14. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Catalano, D., De Prisco, R. (eds.) Security and Cryptography for Networks. pp. 464–482. Springer International Publishing, Cham (2018)
15. Couteau, G.: New protocols for secure equality test and comparison. In: Preneel, B., Vercauteren, F. (eds.) Applied Cryptography and Network Security. pp. 303–320. Springer International Publishing, Cham (2018)

16. Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Pieprzyk, J., Suriadi, S. (eds.) Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10343, pp. 261–278. Springer (2017). https://doi.org/10.1007/978-3-319-59870-3_15
17. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: An efficient and scalable protocol. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. p. 789800. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516701
18. Eskandarian, S., Boneh, D.: Clarion: Anonymous communication from multiparty shuffling protocols. In: 29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022. The Internet Society (2022)
19. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) Advances in Cryptology - EUROCRYPT 2004. pp. 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
20. Frikken, K.: Privacy-preserving set union. In: Katz, J., Yung, M. (eds.) Applied Cryptography and Network Security. pp. 237–252. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
21. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J.A. (ed.) Public-Key Cryptography – PKC 2021. pp. 591–617. Springer International Publishing, Cham (2021)
22. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 395–425. Springer International Publishing, Cham (2021)
23. Gong, X., Hua, Q.S., Jin, H.: Nearly optimal protocols for computing multi-party private set union. In: 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS). pp. 1–10 (2022). https://doi.org/10.1109/IWQoS54832.2022.9812897
24. Gordon, D., Hazay, C., Le, P.H., Liang, M.: More efficient (reusable) private set union. Cryptology ePrint Archive, Paper 2022/713 (2022), https://eprint.iacr.org/2022/713
25. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. J. Cryptol. **25**(3), 383–433 (2012). https://doi.org/10.1007/s00145-011-9098-x
26. Hogan, K., Luther, N., Schear, N., Shen, E., Stott, D., Yakoubov, S., Yerukhimovich, A.: Secure multiparty computation for cooperative cyber risk assessment. In: IEEE Cybersecurity Development, SecDev 2016, Boston, MA, USA, November 3-4, 2016. pp. 75–76. IEEE Computer Society (2016). https://doi.org/10.1109/SecDev.2016.028
27. Hong, J., Kim, J.W., Kim, J., Park, K., Cheon, J.H.: Constant-round privacy preserving multiset union. Cryptology ePrint Archive, Paper 2011/138 (2011), https://eprint.iacr.org/2011/138
28. Jia, Y., Sun, S.F., Zhou, H.S., Du, J., Gu, D.: Shuffle-based private set union: Faster and more secure. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2947–2964. USENIX Association, Boston, MA (Aug 2022)
29. Kerschbaum, F., Blass, E., Mahdavi, R.A.: Faster secure comparisons with offline phase for efficient private set intersection. In: 30th Annual Network and Distributed

System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023. The Internet Society (2023)
30. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. pp. 241–257. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
31. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 818829. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978381
32. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multiparty private set intersection from symmetric-key techniques. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 12571272. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134065
33. Kolesnikov, V., Rosulek, M., Trieu, N., Wang, X.: Scalable private set union from symmetric-key techniques. In: Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II. pp. 636–666. Springer (2019)
34. Lenstra, A., Voss, T.: Information security risk assessment, aggregation, and mitigation. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy. pp. 391–401. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
35. Mohassel, P., Sadeghian, S.: How to hide circuits in mpc an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 557–574. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
36. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 11511165. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484772
37. Oded, G.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, USA, 1st edn. (2009)
38. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse ot extension. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 401–431. Springer International Publishing, Cham (2019)
39. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: Fast, malicious private set intersection. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 739–767. Springer International Publishing, Cham (2020)
40. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 797–812. USENIX Association, San Diego, CA (Aug 2014)
41. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Trans. Priv. Secur. **21**(2) (Jan 2018). https://doi.org/10.1145/3154794
42. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. p. 187 (2005), http://eprint.iacr.org/2005/187

43. Raghuraman, S., Rindal, P.: Blazing fast psi from improved okvs and subfield vole. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 25052517. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560658
44. Rindal, P., Schoppmann, P.: Vole-psi: Fast oprf and circuit-psi from vector-ole. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 901–930. Springer International Publishing, Cham (2021)
45. Roy, L.: Softspokenot: Communication–computation tradeoffs in ot extension. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 657–687. Springer Nature Switzerland, Cham (2022)
46. Seo, J.H., Cheon, J.H., Katz, J.: Constant-round multi-party private set union using reversed laurent series. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography – PKC 2012. pp. 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
47. Shishido, K., Miyaji, A.: Efficient and quasi-accurate multiparty private set union. In: 2018 IEEE International Conference on Smart Computing (SMARTCOMP). pp. 309–314 (2018). https://doi.org/10.1109/SMARTCOMP.2018.00021
48. Vos, J., Conti, M., Erkin, Z.: 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
49. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Linear private set union from multi-query reverse private membership test. In: 32st USENIX Security Symposium (USENIX Security 23) (2023), https://eprint.iacr.org/2022/358
50. Zhao, S., Ma, M., Song, X., Jiang, H., Yan, Y., Xu, Q.: Lightweight threshold private set intersection via oblivious transfer. In: Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part III 16. pp. 108–116. Springer (2021)