Breaking Free: Efficient Multi-Party Private Set Union Without Non-Collusion Assumptions

Minglang Dong[1,2,3][0009-0002-5323-7119], Yu Chen[1,2,3][0000-0003-2553-1281], Cong Zhang[4][0009-0000-5403-2866], and Yujie Bai[1,2,3][0009-0006-7500-3531]

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao 266237, China
`minglang_dong@mail.sdu.edu.cn`, `yuchen@sdu.edu.cn`, `baiyujie@mail.sdu.edu.cn`
[4] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
`zhangcong@mail.tsinghua.edu.cn`

Abstract. Multi-party private set union (MPSU) protocol enables m ($m > 2$) parties, each holding a set, to collectively compute the union of their sets without revealing any additional information to other parties. There are two main categories of MPSU protocols: The first builds on public-key techniques. All existing works in this category involve a super-linear number of public-key operations, resulting in poor practical efficiency. The second builds on oblivious transfer and symmetric-key techniques. The only existing work in this category is proposed by Liu and Gao (ASIACRYPT 2023), which features the best concrete performance among all existing protocols, despite its super-linear computation and communication. Unfortunately, it does not achieve the standard semi-honest security, as it inherently relies on a non-collusion assumption, which is unlikely to hold in practice. Therefore, the problem of constructing a practical MPSU protocol based on oblivious transfer and symmetric-key techniques in standard semi-honest model remains open. Furthermore, there is no MPSU protocol achieving both linear computation and linear communication complexity, which leaves another unresolved problem. In this work, we resolve these two open problems.

- We propose the first MPSU protocol based on oblivious transfer and symmetric-key techniques in the standard semi-honest model. This protocol is $4.9 - 9.3\times$ faster than Liu and Gao in the LAN setting. Concretely, our protocol requires only 3.6 seconds in online phase for 3 parties with sets of 2^{20} items each.
- We propose the first MPSU protocol achieving both linear computation and linear communication complexity, based on public-key operations. This protocol has the lowest overall communication costs and shows a factor of $3.0 - 36.5\times$ improvement in terms of overall communication compared to Liu and Gao.

We implement our protocols and conduct an extensive experiment to compare the performance of our protocols and the state-of-the-art. To

the best of our knowledge, our implementation is the first correct and secure implementation of MPSU that reports on large-size experiments.

1 Introduction

Over the last decade, there has been growing interest in private set operation (PSO), which consists of private set intersection (PSI), private set union (PSU), and private computing on set intersection (PCSI), etc. Among these functionalities, PSI, especially two-party PSI [PSZ14, KKRT16, PRTY19, CM20, PRTY20, RS21, RR22], has made tremendous progress and become highly practical with extremely fast and cryptographically secure implementations. Meanwhile, multi-party PSI [KMP+17, NTY21, CDG+21, BNOP22] is also well-studied. In contrast, the advancement of PSU has been sluggish until recently, several works proposed efficient two-party PSU protocols [KRTW19, GMR+21, JSZ+22, ZCL+23, CZZ+24a]. However, multi-party PSU has still not been extensively studied. In this work, we focus on PSU in the multi-party setting.

Multi-party private set union (MPSU) enables m ($m > 2$) mutually untrusted parties, each holding a private set of elements, to compute the union of their sets without revealing any additional information. MPSU and its variants have numerous applications, such as information security risk assessment [LV04], IP blacklist and vulnerability data aggregation [HLS+16], joint graph computation [BS05], distributed network monitoring [KS05], building block for private DB supporting full join [KRTW19], private ID [GMR+21] etc.

According to the underlying techniques, existing MPSU protocols can be mainly divided into two categories: The first category, denoted PK-MPSU, is primarily based on public-key techniques, and has been explored in a series of works [KS05, Fri07, VCE22, GNT23]. The drawbacks of these works are that each party has to perform a substantial number of public-key operations, leading to super-linear computation complexity and poor practical efficiency. The second category, denoted SK-MPSU, is primarily based on symmetric-key techniques, and has only one existing work [LG23] to date. This work exhibits much better performance than all prior works. However, it fails to achieve the standard semi-honest security due to its inherent reliance on a non-collusion assumption, assuming the party who obtains the union (we call it leader hereafter) not to collude with other parties. Furthermore, it has super-linear complexity as well. Motivated by the above, we raise the following two questions:

Can we construct a MPSU protocol based on oblivious transfer and symmetric-key operations, without any non-collusion assumptions? Can we construct a MPSU protocol with both linear computation and linear communication complexity[5]?

[5] In the context of MPSU, linear complexity means that the complexity for each party scales linearly with the total size of all parties' sets. In this paper, we consider the balanced setting where each party holds sets of equal size, thus linear complexity means that the complexity for each party scales linearly with both the number of

1.1 Our Contribution

In this work, we resolve the above two open problems by first presenting a new primitive for MPSU, called batch secret-shared private membership test (batch ssPMT), then employing the batch ssPMT to build a SK-MPSU in the standard semi-honest model, and a PK-MPSU with linear computation and communication complexity. Our contributions are summarized as follows:

Efficient Batch ssPMT. At the technical core of the state-of-the-art MPSU protocol [LG23] (hereafter referred to as LG protocol) is the multi-query secret-shared private membership test (mq-ssPMT), which dominates both computation and communication costs of LG protocol. In analogy of the relation between batched oblivious pseudorandom function (batch OPRF) [KKRT16] and multi-point oblivious pseudorandom function (multi-point OPRF) [PRTY19, CM20], we abstract a new functionality called batch secret-shared private membership test (batch ssPMT), which allows a fairly efficient construction and can be used to build an alternative to mq-ssPMT in the context of MPSU. Looking ahead, our batch ssPMT serves as a core building block in our two MPSU protocols, and significantly contributes to our speedup compared to LG protocol.

SK-MPSU in Standard Semi-Honest Model. We generalize random OT (ROT) into multi-party setting, which we call multi-party secret-shared random oblivious transfer (mss-ROT). Based on batch ssPMT and mss-ROT, we propose the first SK-MPSU protocol in the standard semi-honest model. Compared to LG protocol, our SK-MPSU has superior online performance with a $4.9 - 9.3\times$ improvement in the LAN setting.

PK-MPSU with Linear Complexity. Based on batch ssPMT and multi-key rerandomizable public-key encryption (MKR-PKE) [GNT23], we propose the first MPSU protocol with both linear computation and communication. Our PK-MPSU has the lowest overall communication costs with a factor of $3.0{-}36.5\times$ improvement compared to LG protocol. It also achieves a $1.8 - 5.4\times$ speedup in terms of overall running time in the WAN setting. Along the way, we find that the PK-MPSU protocol of Gao et al. [GNT23] is insecure against arbitrary collusion and give a practical attack to demonstrate that it necessitates the same non-collusion assumption as LG protocol as well. [6]

parties m and the set size n. Meanwhile, following current conventions in the area of PSO, linear complexity only considers the online phase.

[6] After we pointed out the security flaw of the protocol in [GNT23] with this concrete attack, the authors of [GNT23] contacted us and confirmed our attack. Subsequently, they updated their paper and revised their original protocol (c.f. Appendix B in the new version of [GNT23]) to a new one, which is similar to our PK-MPSU protocol. The only difference lies in that they instantiate our batch ssPMT by invoking multiple instances of ssPMT separately, which renders their new protocol still having superlinear complexities. See also the summary of the relationship in their paper.

Figure 1 depicts the technical overview of our new MPSU framework. We will elaborate the details in Section 2.

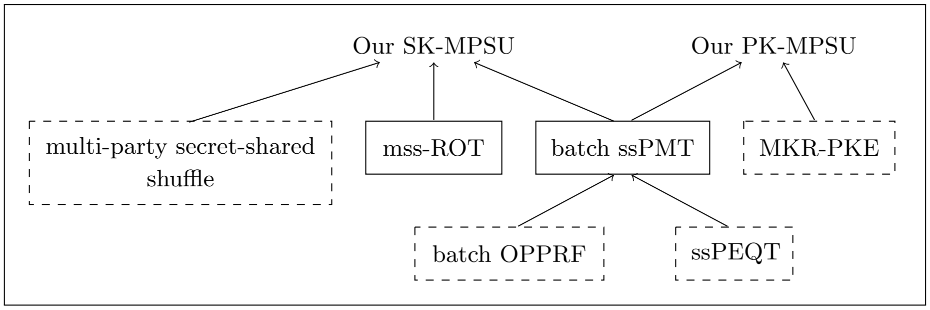

Fig. 1. Technical overview of our MPSU framework. The newly introduced primitives are marked with solid boxes. The existing primitives are marked with dashed boxes.

1.2 Related Works

We review the existing semi-honest MPSU protocols in the literature.

PK-MPSU. Kisser and Song [KS05] introduced the first MPSU protocol, based on polynomial representations and additively homomorphic encryption (AHE). This protocol requires a substantial number of AHE operations and high-degree polynomial calculations, so it is completely impractical.

Frikken [Fri07] improved [KS05] by decreasing the polynomial degree. However, the number of AHE operations remains quadratic in the set size due to the necessity of performing multi-point evaluations on the encrypted polynomials.

Vos et al. [VCE22] proposed a MPSU protocol based on the bit-vector representations. The parties collectively compute the union by performing the private OR operations on the bit-vectors, realized by ElGamal encryption. The original version of this protocol is merely applicable for small universes. Even after leveraging divide-and-conquer approach, the protocol requires quadratic computation and communication complexity in the number of parties for the leader and shows poor concrete efficiency reported by [LG23].

Recently, Gao et al. [GNT23] proposed a MPSU protocol based on three newly introduced cryptographic tools, including membership Oblivious Transfer (mOT), conditional Oblivious Pseudorandom Function (cOPRF) and MKR-PKE. This protocol achieves near-linear complexity in the set size and linear complexity in the number of parties, and is the most advanced MPSU in terms of theoretical complexity. Unfortunately, their protocol turns out to be insecure against arbitrary collusion. We propose a practical attack to show that it requires the same non-collusion assumption as LG protocol (see Appendix A for details).

SK-MPSU. Recently, Liu and Gao [LG23] proposed a practical MPSU protocol based on oblivious transfer and symmetric-key operations. This protocol is several orders of magnitude faster than the prior works. For instance, when computing on datasets of 2^{10} element, it is 109× faster than [VCE22]. However, their protocol is not secure in the standard semi-honest model.

Other Related Works. Blanton et al. [BA12] proposed a MPSU protocol based on oblivious sorting and generic MPC. Their work focuses on devising the circuit for MPSU. The heavy dependency on general MPC leads to inefficiency.

Table 1 provides a comprehensive theoretical comparison between existing MPSU protocols and our proposed protocols. Leader denotes the participant who obtains the union result. Client refers to the remaining participants.

Protocol	**Computation**		**Communication**		**Round**	**Security**
	Leader	Client	Leader	Client		
[KS05]	m^2n^3 pub	m^2n^3 pub	$\lambda m^3 n^2$	$\lambda m^3 n^2$	m	✓
[Fri07]	mn^2 pub	mn^2 pub	λmn	λmn	m	✓
[VCE22]	lm^2n pub	lmn pub	$\lambda lm^2 n$	λlmn	l	✓
[BA12]	$\sigma mn \log n + m^2$ sym		$\sigma^2 mn \log n + \sigma m^2$		$\log m$	✓
[GNT23]	$mn(\log n/\log\log n)$ pub		$(\gamma+\lambda)mn(\log n/\log\log n)$		$\log\gamma + m$	✗
[LG23]	$(T+l+m)mn$ sym	$(T+l)mn$ sym	$(T+l)mn + lm^2n$	$(T+l)mn$	$\log(l-\log n)+m$	✗
Our SK-MPSU	m^2n sym	m^2n sym	$\gamma mn + lm^2n$	$(\gamma+l+m)mn$	$\log\gamma + m$	✓
Our PK-MPSU	mn pub	mn pub	$(\gamma+\lambda)mn$	$(\gamma+\lambda)mn$	$\log\gamma + m$	✓

Table 1. Asymptotic communication (bits) and computation costs of MPSU protocols in the semi-honest setting. For the sake of comparison, we omit the Big O notations and simplify the complexity by retaining only the dominant terms. We use ✓ to denote protocols in the standard semi-honest model and ✗ to denote protocols requiring non-collusion assumption. pub: public-key operations; sym: symmetric cryptographic operations. We ignore the offline phase cost in all SK-MPSU protocols and the symmetric-key operations in all PK-MPSU protocols. [KS05] and [Fri07] use Paillier while [VCE22], [GNT23] and our PK-MPSU use ElGamal. n is the set size. m is the number of participants. λ and σ are computational and statistical security parameter respectively. T is the number of AND gate in a SKE decryption circuit in [LG23]. l is the bit length of input elements. γ is the output length of OPPRF. In the typical setting, $n \le 2^{24}$, $m \le 32$, $\lambda = 128$, $\sigma = 40$, $T \approx 600$, $l \le 128$, $\gamma \le 64$.

2 Technical Overview

2.1 LG Protocol Revisit

We start by recalling the high-level idea of LG protocol. For the sake of simplicity, we focus here on the case of three parties P_1, P_2, P_3, whose inputs are X_1, X_2, X_3

respectively. We designate P_1 as the leader. Since P_1 already holds X_1, it needs to obtain the set difference $Y_1 = (X_1 \cup X_2 \cup X_3) \setminus X_1$ from P_2 and P_3.

Roughly speaking, their protocol enables P_2 to somehow secret-share $Y_2 = X_2 \setminus X_1$ and P_3 to somehow secret-share $Y_3 = X_3 \setminus (X_1 \cup X_2)$ among all parties. Since $\{Y_2, Y_3\}$ is a partition of Y_1, each party holds a share of Y_1 in the order of Y_2, Y_3 eventually, and any two parties cannot collude to obtain information of the last one's inputs.

A naive approach to reconstructing Y_1 to P_1 is to let P_2 and P_3 send their shares to P_1 straightforwardly. However, in this way, P_1 can determine the party that each obtained element $x \in Y_1$ belongs. Their solution is to let the parties invoke multi-party secret-shared shuffle to randomly permute and re-share all shares. Multi-party secret-shared shuffle guarantees that any two parties have no knowledge of the permutation and all shares are refreshed after the invocation, hence the adversary is unaware of the correspondence between shares and the individual difference sets Y_2, Y_3. Then P_2 and P_3 can send their shuffled shares to P_1, who reconstructs Y_1 and obtains the union by appending elements of X_1.

LG protocol utilizes two main ingredients: (1) The secret-shared private membership test (ssPMT) [CO18, ZMS+21], where the sender $\mathcal{S}$ inputs a set X, and the receiver $\mathcal{R}$ inputs an element y. If $y \in X$, $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of 1, otherwise they receive secret shares of 0. Liu and Gao proposed a multi-query ssPMT (mq-ssPMT), which supports the receiver querying multiple elements' memberships of the sender's set simultaneously. Namely, $\mathcal{S}$ inputs X, and $\mathcal{R}$ inputs $y_1, \cdots, y_n$. $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of a bit vector of size n, where if $y_i \in X$, the ith bit is 1, otherwise 0. (2) A two-choice-bit version of random oblivious transfer (ROT), where the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ each holds a choice bit e_0, e_1. $\mathcal{S}$ receives two random messages r_0, r_1. If $e_0 \oplus e_1 = 0$, $\mathcal{R}$ receives r_0, otherwise r_1[7]. The following is to elaborate that how to utilize these ingredients to realize the secret-sharing processes.

The process for P_2 to secret-share Y_2 is as follows: P_2 acts as $\mathcal{R}$ and executes the mq-ssPMT with P_1. For each item $x \in X_2$, P_2 and P_1 receive shares $e_{2,1}$ and $e_{1,2}$. If $x \in X_1$, $e_{2,1} \oplus e_{1,2} = 1$, otherwise $e_{2,1} \oplus e_{1,2} = 0$. Then P_2 acts as $\mathcal{S}$ and executes the two-choice-bit ROT with P_1. P_2 and P_1 each inputs $e_{2,1}, e_{1,2}$. P_2 receives $r^0_{2,1}, r^1_{2,1}$. If $e_{2,1} \oplus e_{1,2} = 0$, P_1 receives $r^0_{2,1}$, otherwise P_1 receives $r^1_{2,1}$. P_1 sets the output as its share $s_{2,1}$. P_2 sets its share $s_{2,2}$ to be $r^0_{2,1} \oplus x \| \mathsf{H}(x)$[8]. P_3 sets $s_{2,3}$ to 0. If $x \notin X_1$, $e_{2,1} \oplus e_{1,2} = 0$, $s_{2,1} \oplus s_{2,2} \oplus s_{2,3} = r^0_{2,1} \oplus (r^0_{2,1} \oplus x \| \mathsf{H}(x)) =$

[7] This special ROT is identical to the standard 1-out-of-2 ROT, where e_0 is determined by $\mathcal{S}$ to indicate whether it would swap the order of r_0 and r_1. We use the standard ROT to formally describe our protocol specifications in the later sections. Here, we boil down a number of steps to the two-choice-bit ROT for facilitating illustration.

[8] Suppose a lack of specific structure for set elements, then P_1 cannot distinguish a set element x and a random value r when it reconstructs the difference set. To address this, all parties append the hash value when secret sharing an element, i.e., sharing $x \| \mathsf{H}(x)$. It is provable that if the output length of H is sufficiently long (according to [LG23], at least $\sigma + \log(m-1) + \log n$ bits), the probability of existing x that satisfies $x \| \mathsf{H}(x) = r$ is negligible. Therefore, P_1 can distinguish set elements from random values with overwhelming probability.

$x\|\mathsf{H}(x)$. Otherwise, $e_{2,1} \oplus e_{1,2} = 1$, $s_{2,1} \oplus s_{2,2} \oplus s_{2,3} = r^1_{2,1} \oplus r^0_{2,1} \oplus x\|\mathsf{H}(x)$ is uniformly random. That is, Y_2 is secret-shared among all parties[9], and the other elements in X_2 are masked by random values before being secret-share.

We proceed to the process for P_3 to secret-share Y_3: P_3 acts as $\mathcal{R}$ and executes the mq-ssPMT with P_1 and P_2 separately. For each $x \in X_3$, P_3 and P_1 receive shares $e_{3,1}$ and $e_{1,3}$, while P_3 and P_2 receive shares $e_{3,2}$ and $e_{2,3}$. Then P_3 acts as $\mathcal{S}$ and executes the two-choice-bit ROT with P_1 and P_2. In the ROT between P_3 and P_1, P_3 inputs $e_{3,1}$ and P_1 inputs $e_{1,3}$. P_3 receives $r^0_{3,1}, r^1_{3,1}$. If $e_{3,1} \oplus e_{1,3} = 0$, P_1 receives $r^0_{3,1}$, otherwise P_1 receives $r^1_{3,1}$. P_1 sets the output as its share $s_{3,1}$. In the ROT between P_3 and P_2, P_3 inputs $e_{3,2}$ and P_2 inputs $e_{2,3}$. P_3 receives $r^0_{3,2}, r^1_{3,2}$. If $e_{3,2} \oplus e_{2,3} = 0$, P_2 receives $r^0_{3,2}$, otherwise P_2 receives $r^1_{3,2}$. P_2 sets the output as its share $s_{3,2}$. P_3 sets its share $s_{3,3}$ to be $r^0_{3,1} \oplus r^0_{3,2} \oplus x\|\mathsf{H}(x)$. If $x \notin X_1$ and $x \notin X_2$, $e_{3,1} \oplus e_{1,3} = 0$ and $e_{3,2} \oplus e_{2,3} = 0$, $s_{3,1} \oplus s_{3,2} \oplus s_{3,3} = r^0_{3,1} \oplus r^0_{3,2} \oplus (r^0_{3,1} \oplus r^0_{3,2} \oplus x\|\mathsf{H}(x)) = x\|\mathsf{H}(x)$. Otherwise, there is at least one random value r ($r^1_{3,1}$ or $r^1_{3,2}$) cannot be canceled out from the summation.

The LG protocol described above has two main drawbacks: First, as the heart of LG protocol, mq-ssPMT is given a heavy instantiation which is fed much computation task through expensive general MPC machinery and renders it the bottleneck of the entire protocol. Second, LG protocol fails to achieve security against arbitrary collusion. In the following two sections, we are devoted to improving the efficiency of mq-ssPMT and enhancing LG protocol into standard semi-honest security, respectively.

2.2 Efficient Batch ssPMT

To improve the efficiency of mq-ssPMT, we abstract a new functionality called batch ssPMT, which is essentially the batched version of single-query ssPMT. In the batch ssPMT functionality, the sender $\mathcal{S}$ inputs n disjoint sets $X_1, \cdots, X_n$, and the receiver $\mathcal{R}$ inputs n elements $y_1, \cdots, y_n$. $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of a bit vector of size n, whose ith bit is 1 if $y_i \in X_i$, otherwise 0. The relationship between batch ssPMT and mq-ssPMT is two-fold: In terms of functionality, batch ssPMT is to mq-ssPMT what batch OPRF is to multi-point OPRF, testing a batch of elements' memberships across distinct sets rather than a common set; In terms of efficiency, batch ssPMT admits a more efficient construction, providing a superior alternative to mq-ssPMT in the context of MPSU.

We adopt the following two-step approach to build the batch ssPMT protocol: (1) $\mathcal{S}$ and $\mathcal{R}$ invoke batched oblivious programmable pseudorandom function (batch OPPRF) [KMP+17, PSTY19]. To elaborate, for the ith ssPMT instance ($i = 1, \cdots, n$), $\mathcal{S}$ choose a random s_i, and creates a set E_i comprising key-value pairs with elements from X_i as keys and s_i as all elements' corresponding values. Then $\mathcal{S}$ and $\mathcal{R}$ invoke batch OPPRF of size n, where in the ith instance of OPPRF, $\mathcal{S}$ inputs E_i, while $\mathcal{R}$ inputs y_i and receives t_i. According to the definition of OPPRF (cf. Section 3.5), if $y_i \in X_i$, t_i equals s_i, otherwise t_i is

[9] Since $X_2 \setminus X_1$ merely contains the information of X_1 and X_2, it is sufficient to secret-share $X_2 \setminus X_1$ among P_1 and P_2. Other difference sets are shared similarly.

pseudorandom to s_i. If $y_i \notin X_i$, the probability of $t_i = s_i$ is $2^{-\gamma}$, where γ is the output length of OPPRF, i.e., the bit length of s_i and t_i. To ensure that the probability of any $t_i \neq s_i$ occurring is less than $2^{-\sigma}$ (σ is the statistical security parameter), we set $\gamma \geq \sigma + \log n$ so that $n \cdot 2^{-\gamma} \leq 2^{-\sigma}$. (2) $\mathcal{S}$ and $\mathcal{R}$ invoke n instances of secret-shared private equality test (ssPEQT) [PSTY19, CGS22], where in the ith instance of ssPEQT, $\mathcal{S}$ inputs s_i and $\mathcal{R}$ inputs t_i. If $s_i = t_i$, they receive secret shares of 1, otherwise 0. It's easy to verify that if $y_i \in X_i$, $\mathcal{S}$ and $\mathcal{R}$ receive shares of 1, otherwise they receive shares of 0 with overwhelming probability.

Now we start to build an alternative to mq-ssPMT using our batch ssPMT. Recall that in mq-ssPMT, the sender $\mathcal{S}$'s input is a single set X. First, $\mathcal{S}$ and $\mathcal{R}$ preprocess their inputs through hashing to bin technique: $\mathcal{R}$ uses hash functions h_1, h_2, h_3 to assign its input elements $y_1, \cdots, y_n$ to B bins via Cuckoo hashing [PR04], which ensures that each bin accommodates at most one element. At the same time, $\mathcal{S}$ assigns each $x \in X$ to all bins determined by $h_1(x), h_2(x), h_3(x)$. Then the parties invoke B instances of ssPMT, where in the jth instance, $\mathcal{S}$ inputs the subset $X_j \in X$ containing all elements mapped into its bin j and $\mathcal{R}$ inputs the sole element mapped into its bin j. If some y_i is mapped to bin j, and $y_i \in X$, then $\mathcal{S}$ certainly maps y_i to bin j (and other bins) as well, i.e., $y_i \in X_j$. Therefore, for each bin j of $\mathcal{R}$, if the inside element y_i is in X, we have $y_i \in X_j$, then $\mathcal{S}$ and $\mathcal{R}$ receive shares of 1 from the ith instance of batch ssPMT. Otherwise, we have $y_i \notin X_j$, then $\mathcal{S}$ and $\mathcal{R}$ receive shares of 0.

The above process in fact realizes a different functionality with mq-ssPMT. In the above construction, the query sequence of $\mathcal{R}$ is determined by the Cuckoo hash positions of its input elements. Since the Cuckoo hash positions depends on the whole input set of $\mathcal{R}$, and all shares is arranged in the order of the parties' Cuckoo hash positions, the reconstruction stage may leak some information about the parties' input sets to P_1 and hence cannot be simulated. Therefore, it is crucial to eliminate the dependence of shares' order on the parties' Cuckoo hash positions before the reconstruction, while fortunately, it has been realized by the execution of multi-party secret-shared shuffle as we discussed before. For the difference in efficiency, refer to Section 7.1.

2.3 SK-MPSU from Batch ssPMT and mss-ROT

By replacing mq-ssPMT with the construction using batch ssPMT plus hash to bin technique as previous, we can improve the performance of LG protocol significantly. However, the improved protocol still relies on the non-collusion assumption. So it remains to show that how to eliminate the non-collusion assumption to obtain a SK-MPSU protocol in the standard semi-honest model.

First, let us figure out that why LG protocol needs the non-collusion assumption. We still use the aforementioned three-party PSU protocol as illustration:

After the secret sharing process of P_3, all parties hold the secret shares of

$$\begin{cases} x\|\mathsf{H}(x) & x \in Y_3 \\ r^0_{3,1} \oplus r^1_{3,1} \oplus x\|\mathsf{H}(x) & x \in Y_1 \\ r^0_{3,2} \oplus r^1_{3,2} \oplus x\|\mathsf{H}(x) & x \in Y_2 \\ r^1_{3,1} \oplus r^1_{3,2} \oplus r^0_{3,1} \oplus r^0_{3,2} \oplus x\|\mathsf{H}(x) & x \in X_2 \cap X_1 \end{cases}$$

In the final stage, if $x \in Y_3$, P_1 reconstructs the secret $x\|\mathsf{H}(x)$. Otherwise, the secret is uniformly at random in the P_1's view so that P_1 gains no information about x. Nevertheless, this is only true when P_1 does not collude with P_3. Since there is no randomness in P_3's view (In the two invocations of ROT, P_3 receives $r^0_{3,1}, r^1_{3,1}$ and $r^0_{3,2}, r^1_{3,2}$), for each $x \in X_3$, the coalition of P_1 and P_3 can easily check the reconstructed values, and distinguish the cases that $x \in Y_1$, $x \in Y_2$, and $x \in X_2 \cap X_1$, which reveals the information of P_2's inputs.

To resolve the above security problem, we generalize ROT into multi-party setting, which we call multi-party secret-shared random oblivious transfer (mss-ROT). Suppose there are $d < m$ involved parties and two of them, denoted P_{ch_0} and P_{ch_1}, possessing choice bits b_0 and b_1 respectively. The mss-ROT functionality gives each involved party P_i $(i \in \{1, \cdots, d\})$ two outputs r_i and Δ_i, such that if $b_0 \oplus b_1 = 0$, $r_1 \oplus \cdots \oplus r_d = 0$. Otherwise, $r_1 \oplus \cdots \oplus r_d = \Delta_1 \oplus \cdots \oplus \Delta_d$. Note that if $b_0 \oplus b_1 = 1$, the parties share a value which is uniformly random in the view of a coalition of any $d - 1$ parties.

Equipped with this new primitive, the above protocol can be amended to achieve security under arbitrary collusion: After P_3 and P_1 receiving shares $e_{3,1}$ and $e_{1,3}$, while P_3 and P_2 receiving shares $e_{3,2}$ and $e_{2,3}$, P_1, P_2, P_3 invoke mss-ROT twice. In the first invocation, P_3 and P_1 act as P_{ch_0} and P_{ch_1} holding as choice bits $e_{3,1}$ and $e_{1,3}$. P_1, P_2, P_3 receives $r_{1,31}, r_{2,31}, r_{3,31}$ and $\Delta_{1,31}, \Delta_{2,31}, \Delta_{3,31}$ separately. If $e_{3,1} \oplus e_{1,3} = 0$, $r_{1,31} \oplus r_{2,31} \oplus r_{3,31} = 0$, otherwise $r_{1,31} \oplus r_{2,31} \oplus r_{3,31} = \Delta_{1,31} \oplus \Delta_{2,31} \oplus \Delta_{3,31}$. Likewise, in the second invocation, P_3 and P_2 hold as choice bits $e_{3,2}$ and $e_{2,3}$. P_1, P_2, P_3 receives $r_{1,32}, r_{2,32}, r_{3,32}$ and $\Delta_{1,32}, \Delta_{2,32}, \Delta_{3,32}$ separately. If $e_{3,2} \oplus e_{2,3} = 0$, $r_{1,32} \oplus r_{2,32} \oplus r_{3,32} = 0$, otherwise $r_{1,32} \oplus r_{2,32} \oplus r_{3,32} = \Delta_{1,32} \oplus \Delta_{2,32} \oplus \Delta_{3,32}$. Finally, P_1 sets its share sh_1 to be $r^0_{1,31} \oplus r^0_{1,32}$. P_2 sets its share sh_2 to be $r^0_{2,31} \oplus r^0_{2,32}$. P_3 sets its share sh_3 to be $r^0_{3,31} \oplus r^0_{3,32} \oplus x\|\mathsf{H}(x)$.

We conclude that all parties hold the secret shares of

$$\begin{cases} x\|\mathsf{H}(x) & x \in Y_3 \\ \Delta_{1,31} \oplus \Delta_{2,31} \oplus \Delta_{3,31} \oplus x\|\mathsf{H}(x) & x \in Y_1 \\ \Delta_{1,32} \oplus \Delta_{2,32} \oplus \Delta_{3,32} \oplus x\|\mathsf{H}(x) & x \in Y_2 \\ \Delta_{1,31} \oplus \Delta_{2,31} \oplus \Delta_{3,31} \oplus \Delta_{1,32} \oplus \Delta_{2,32} \oplus \Delta_{3,32} \oplus x\|\mathsf{H}(x) & x \in X_2 \cap X_1 \end{cases}$$

It is immediate that if $x \notin Y_3$, the secret-shared value is uniform and independent of x in the view of a coalition of any two parties. Hence, the reconstruction reveals no additional information to P_1 even if it colludes with P_2 or P_3.

2.4 PK-MPSU from Batch ssPMT and MKR-PKE

As previously mentioned, there are no existing MPSU works that achieve both linear computation and communication complexity. Thereinto, LG protocol and our SK-MPSU fail to achieve linear complexity because in the secret-sharing based framework, each party holds shares of $(m-1)n$ elements and in the final reconstruction stage, all parties except P_1 send their shares to P_1, then P_1 reconstruct $(m-1)n$ secrets to pick up all elements comprising the union. It is clear that even in this single stage, the computation and communication complexity of P_1 deviate from the linear criteria. So we have to turn to another approach.

We notice that the most advanced work in terms of asymptotic complexity is [GNT23], achieving near-linear complexity. We start by analyzing [GNT23]: The first phase enables P_1 to somehow acquire encrypted $X_i \setminus (X_1 \cup \cdots \cup X_{i-1})$ for $2 \leq i \leq m$, interspersed with encrypted dummy messages filling positions of the duplicate elements. Note that if P_1 decrypts these ciphertexts by itself, then it can establish a mapping associating each element $x \in X_1 \cup \cdots \cup X_m \setminus X_1$ with the party to whom x belongs. To address this problem, they introduce a PKE variant, MKR-PKE, with several tailored properties, enabling all parties to conduct a collaborative decryption and shuffle procedure.

The second phase, which is the aforemetioned collaborative decryption and shuffle procedure, works as follows: P_1 sends the ciphertexts to P_2 after permuting them using a random permutation π_1. P_2 performs partial decryption on the received ciphertexts before rerandomization, and permutes them using a random permutation π_2. Then it forwards the permuted partially decrypted ciphertexts to P_3. This iterative process continues until the last party, P_m, who returns its permuted partially decrypted ciphertexts to P_1. P_1 decrypts the ultimate ciphertexts, filters out the dummy elements, retains the desired set $X_1 \cup \cdots \cup X_m \setminus X_1$, and appends the elements in X_1, to obtain the union.

We identify that their non-linear complexity stems from their construction of the first phase. Despite that they also use hashing to bin technique, their construction does not support amortizing the batching cost. In the case $B = O(n)$ bins are considered, each party has to execute mOT and cOPRF of size $O(\log \log n)$ per bin, resulting in overall $O(n \log \log n)$ computation and communication complexity. Besides, their insecurity against arbitrary collusion roots in the construction of the first phase (cf. Appendix A). In contrast, the second phase already has linear computation and communication complexity, as well as security against arbitrary collusion.[10] Therefore, the task of devising linear-complexity MPSU reduces to devising a linear-complexity and secure construction against arbitrary collusion for the first phase.

[10] As we have said, the second phase addresses the same problem solved by the multi-party secret-shared shuffle, meanwhile there currently exists no construction for multi-party secret-shared shuffle that achieves total linear complexity. This is the root why our PK-MPSU protocol can achieve total linear complexity while our SK-MPSU protocol cannot. The key distinction lies in our PK-MPSU protocol's departure from the secret-shared paradigm, opting instead for a PKE-based approach to replace the non-linear component.

Our first attempt is as follows: For $2 \leq i \leq m$, each P_i and P_1 execute batch ssPMT (after preprocessing their inputs using hashing to bin, we ignore this for the sake of simplicity in the later explanation) and receive secret shares as the choice bits for the subsequent execution of two-choice-bit OT[11], where for each $x \in X_i$, P_i acts as $\mathcal{S}$ and inputs the encrypted x and an encrypted dummy message. As a result, if x belongs to the difference set $X_i \setminus X_1$, P_1 receives the encrypted x; otherwise, it receives the encrypted dummy message. Nevertheless, the goal of the first phase is to obtain the encrypted difference set $X_i \setminus (X_1 \cup \cdots \cup X_{i-1})$, which is the subset of $X_i \setminus X_1$.

In order to attain the above goal, our next strategy is to let each P_i engage in a procedure with each P_j $(1 < j < i)$ before executing the batch ssPMT and two-choice-bit OT with P_1, so that P_j can "obliviously" replace the encrypted items in the intersection $X_j \cap X_i$ with encrypted dummy messages. The procedure unfolds as follows.

P_i plays the role of $\mathcal{R}$ and executes the batch ssPMT with P_j. For each $x \in X_i$, P_i and P_j receive shares $e_{i,j}$ and $e_{j,i}$. P_i initializes a message m_0 to the encrypted x and m_1 to an encrypted dummy message, then executes two-choice-bit OT with P_j. P_i and P_j input $e_{i,j}$ and $e_{j,i}$ as their choice bits. P_i acts as $\mathcal{S}$ and inputs m_0 and m_1. P_j acts as $\mathcal{R}$ and receives $\mathsf{ct} = m_{e_{i,j} \oplus e_{j,i}}$. If $x \notin X_j$, ct is the ciphertext of x; otherwise, the ciphertext of dummy message. P_j rerandomizes ct to ct' and returns ct' to P_i. P_i rerandomizes ct', then updates m_0 to ct' and m_1 to the rerandomized m_1 before repeating the above procedure with the next party P_{j+1}. The final values of m_0 and m_1 shall be used during the subsequent invocation of two-choice-bit OT with P_1.

3 Preliminaries

3.1 Notation

Let m denote the number of participants. We write $[m]$ to denote a set $\{1, \cdots, m\}$. We use P_i $(i \in [m])$ to denote participants, X_i to represent the sets they hold, where each set has n l-bit elements. $x\|y$ denotes the concatenation of two strings. We use λ, σ as the computational and statistical security parameters respectively, and use $\overset{s}{\approx}$ (resp.$\overset{c}{\approx}$) to denote that two distributions are statistically (resp. computationally) indistinguishable. We denote vectors by letters with a right arrow above and a_i denotes the i-th component of $\vec{a}$. $\vec{a} \oplus \vec{b} = (a_1 \oplus b_1, \cdots, a_n \oplus b_n)$. $\pi(\vec{a}) = (a_{\pi(1)}, \cdots, a_{\pi(n)})$, where π is a permutation over n items. $\pi = \pi_1 \circ \cdots \circ \pi_n$ represents that applying the permutation π is equivalent to applying the permutations $\pi_1, \cdots, \pi_n$ in sequence. $x[i]$ denotes the i-th bit of element x, and $X(i)$ denotes the i-th element of set X. When we refer to an additive secret shared value x among m parties, we mean that $x = x_1 \oplus \cdots \oplus x_m$, where $\oplus$ is bit-wise XOR and x_i is held by $P_i, i \in [m]$.

[11] The difference of functionality between two-choice-bit OT and two-choice-bit ROT is that $\mathcal{S}$ inputs two messages m_0, m_1 instead of obtaining two uniform messages.

3.2 Multi-party Private Set Union

The ideal functionality of MPSU is formalized in Figure 2.

> **Parameters.** m parties $P_1, \cdots, P_m$, where P_{ld} is the leader, $\text{ld} \in [m]$. Size n of input sets. The bit length l of set elements.
> **Functionality.** On input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$ from P_i, give $\bigcup_{i=1}^m X_i$ output to P_{ld}.

Fig. 2. Multi-party Private Set Union Functionality $\mathcal{F}_{\mathsf{mpsu}}$

3.3 Batch Oblivious Pseudorandom Function

Oblivious pseudorandom function (OPRF) [FIPR05] is a central primitive in the area of PSO. Kolesnikov et al. [KKRT16] introduced batched OPRF, which provides a batch of OPRF instances in the following way. In the ith instance, the receiver $\mathcal{R}$ has an input x_i; the sender $\mathcal{S}$ learns a PRF key k_i and $\mathcal{R}$ learns $\mathsf{PRF}(k_i, x_i)$. Note that $\mathcal{R}$ evaluates the PRF on only one point per key. The batch OPRF functionality is described formally in Figure 3.

> **Parameters.** Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of set elements. Some PRF family $\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^l \to \{0,1\}^\gamma$.
> **Functionality.** On input $\vec{x} \subseteq (\{0,1\}^l)^B$ from $\mathcal{R}$,
>
> - For each $i \in [B]$, choose uniform PRF key k_i.
> - For each $i \in [B]$, define $f_i = \mathsf{PRF}(k_i, x_i)$.
> - Give vector $\vec{k} = (k_1, \cdots, k_B)$ to $\mathcal{S}$ and vector $\vec{f} = (f_1, \cdots, f_B)$ to $\mathcal{R}$.

Fig. 3. Batch OPRF Functionality $\mathcal{F}_{\mathsf{bOPRF}}$

3.4 Oblivious Key-Value Stores

A key-value store [PRTY20, GPR+21, RR22, BPSY23] is a data structure that compactly represents a desired mapping from a set of keys to corresponding values.

Definition 1. *A key-value store is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H, and consists of two algorithms:*

– $\mathsf{Encode_H}$ *takes as input a set of key-value pairs* $\{(k_i, v_i)|i \in [n]\}$ *and outputs an object* D *(or, with statistically small probability, an error indicator* $\perp$*).*
– $\mathsf{Decode_H}$ *takes as input an object* D*, a key* k*, and outputs a value* v*.*

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

$$(k, v) \in A \;\; and \;\; \perp \neq D \leftarrow \mathsf{Encode_H}(A) \Longrightarrow \mathsf{Decode_H}(D, k) = v$$

Obliviousness. For all distinct $\{k_1^0, \cdots, k_n^0\}$ and all distinct $\{k_1^1, \cdots, k_n^1\}$, if $\mathsf{Encode_H}$ does not output $\perp$ for $\{k_1^0, \cdots, k_n^0\}$ or $\{k_1^1, \cdots, k_n^1\}$, then the distribution of $D_0 \leftarrow \mathsf{Encode_H}(\{(k_1^0, v_1), \cdots, (k_n^0, v_n)\})$ is computationally indistinguishable to $D_1 \leftarrow \mathsf{Encode_H}(\{(k_1^1, v_1), \cdots, (k_n^1, v_n)\})$, where $v_i \leftarrow \mathcal{V}$ for $i \in [n]$.

A KVS is an Oblivious KVS (OKVS) if it satisfies the obliviousness property. we also require an additional randomness property [ZCL+23] from the OKVS.
Randomness. For any $A = \{(k_1, v_1), \cdots, (k_n, v_n)\}$ and $k^* \notin \{k_1, \cdots, k_n\}$, the output of $\mathsf{Decode_H}(D, k^*)$ is statistically indistinguishable to that of uniform distribution over $\mathcal{V}$, where $D \leftarrow \mathsf{Encode_H}(A)$.

3.5 Batch Oblivious Programmable Pseudorandom Function

Oblivious programmable pseudorandom function (OPPRF) [KMP+17, PSTY19, CGS22, RS21, RR22] is an extension of OPRF, which lets $\mathcal{S}$ program a PRF F so that it has specific random outputs for some specific inputs and pseudorandom outputs for all other inputs. This kind of PRF with the additional property that on a certain programmed set of inputs the function outputs programmed values is called programmable PRF (PPRF) [PSTY19]. $\mathcal{R}$ evaluates the OPPRF with no knowledge of whether it learns a programmed output of F or just a pseudorandom value. The batch OPPRF functionality is given in Figure 4.

3.6 Hashing to Bin

The hashing to bin technique was introduced by Pinkas et al. [PSZ14, PSSZ15], which is originally applied to construct two-party PSI protocol. At the high level, the receiver $\mathcal{R}$ uses hash functions h_1, h_2, h_3 to assign its items to B bins via Cuckoo hashing [PR04], so that each bin has at most one item. The hashing process uses eviction and the choice of which of the bins is used depends on the entire set. Using the same hash functions and simple hashing, sender $\mathcal{S}$ assigns each of its items x to all of the bins $h_1(x), h_2(x), h_3(x)$, so that if the item x is also in $\mathcal{R}$'s set and is mapped into the bin $b \in \{h_1(x), h_2(x), h_3(x)\}$ by Cuckoo hashing, then the bin b of $\mathcal{S}$'s simple hash table certainly contains x.

We write simple hashing with the following notation:

$$\mathcal{T}^1, \cdots, \mathcal{T}^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X)$$

This expression means to hash the items of X into B bins using simple hashing with hash functions $h_1, h_2, h_3 : \{0, 1\}^* \rightarrow [B]$. The output is the simple hash

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of keys. The bit length γ of values. An OKVS scheme ($\mathsf{Encode_H}, \mathsf{Decode_H}$).
Sender's inputs. $\mathcal{S}$ inputs B sets of key-value pairs including:

- Disjoint key sets $K_1, \cdots, K_B$.
- The value sets $V_1, \cdots, V_B$, where $|K_i| = |V_i|$ for every $i \in [B]$, and $V_i(j) \in \{0,1\}^\gamma$ for every $i \in [B]$ and $1 \leq j \leq |K_i|$.

Receiver's inputs. $\mathcal{R}$ inputs B queries $\vec{x} \subseteq (\{0,1\}^l)^B$.
Functionality: On input $(K_1, \cdots, K_B)$ and $(V_1, \cdots, V_B)$ from $\mathcal{S}$ and $\vec{x} \subseteq (\{0,1\}^l)^B$ from $\mathcal{R}$,

- Choose uniformly random and independent PPRF key k_i, for each $i \in [B]$;
- Sample a PPRF $F : \{0,1\}^* \times \{0,1\}^l \to \{0,1\}^\gamma$ such that $F(k_i, K_i(j)) = V_i(j)$ for $i \in [B], 1 \leq j \leq |K_i|$;
- Define $f_i = F(k_i, x_i)$, for each $i \in [B]$;
- Give vector $\vec{f} = (f_1, \cdots, f_B)$ to $\mathcal{R}$.

Fig. 4. Batch OPPRF Functionality $\mathcal{F}_{\mathsf{bOPPRF}}$

table denoted by $\mathcal{T}^1, \cdots, \mathcal{T}^B$, where for each $x \in X$ we have $\mathcal{T}^{h_i(x)} = \{x\|i | i = 1,2,3\}$.[12]

We write Cuckoo hashing with the following notation:

$$\mathcal{C}^1, \cdots, \mathcal{C}^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X)$$

This expression means to hash the items of X into B bins using Cuckoo hashing on hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [B]$. The output is the Cuckoo hash table denoted by $\mathcal{C}^1, \cdots, \mathcal{C}^B$, where for each $x \in X$ there is some $i \in \{1,2,3\}$ such that $\mathcal{C}^{h_i(x)} = x\|i$. Some Cuckoo hash positions do not matter, corresponding to empty bins. We use these symbols throughout the subsequent sections.

3.7 Secret-Shared Private Equality Test

Secret-shared private equality test protocol (ssPEQT) [PSTY19, CGS22] can be viewed as an extreme case of ssPMT when the sender $\mathcal{S}$'s input set has only one item. In Figure 5 we formally define this functionality.

3.8 Random Oblivious Transfer

Oblivious transfer (OT) [Rab05] is a foundational primitive in MPC, the functionality of 1-out-of-2 random OT (ROT) is given in Figure 6.

[12] Appending the index of the hash function is helpful for dealing with edge cases like $h_1(x) = h_2(x) = i$, which happen with non-negligible probability. Without

Parameters. Two parties P_1, P_2. The bit length γ of inputs.
Functionality. On input x from P_1 and input y from P_2, sample two random bits a, b under the constraint that $a \oplus b = \bigwedge_{i=1}^{\gamma}(1 \oplus x[i] \oplus y[i])$. Namely, if $x = y$, $a \oplus b = 1$. Otherwise $a \oplus b = 0$. Give a to P_1 and b to P_2.

Fig. 5. Secret-Shared Private Equality Test Functionality $\mathcal{F}_{\mathsf{ssPEQT}}$

Parameters. Sender $\mathcal{S}$, Receiver $\mathcal{R}$. The message length l.
Functionality. On input $b \in \{0,1\}$ from $\mathcal{R}$, sample $r_0, r_1 \leftarrow \{0,1\}^l$. Give (r_0, r_1) to $\mathcal{S}$ and give r_b to $\mathcal{R}$.

Fig. 6. 1-out-of-2 Random OT Functionality $\mathcal{F}_{\mathsf{rot}}$

3.9 Multi-Party Secret-Shared Shuffle

Multi-party secret-shared shuffle has the capability to permute the share vectors of all parties in a random permutation unknown to any coalition of $m-1$ parties. Then it refreshs all shares. The functionality is given in Figure 7.

Eskandarian et al. [EB22] propose an online-efficient protocol by extending [CGP20] to the multi-party setting. 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 the online protocol in Appendix C.

Parameters. m parties $P_1, \cdots P_m$. The dimension of vector n. The item length l.
Functionality. On input $\vec{x}_i = (x_i^1, \cdots, x_i^n)$ from each P_i, sample a random permutation $\pi : [n] \to [n]$. For $1 \leq i \leq m$, sample $\vec{x_i'} \leftarrow (\{0,1\}^l)^n$ satisfying $\bigoplus_{i=1}^m \vec{x_i'} = \pi(\bigoplus_{i=1}^m \vec{x}_i)$. Give $\vec{x_i'}$ to P_i.

Fig. 7. Multi-Party Secret-Shared Shuffle Functionality $\mathcal{F}_{\mathsf{ms}}$

appending i, the OKVS would contain the identical value $F(k_i, x)$, which leaks the fact that a collision $h_1(x) = h_2(x)$ occurred. Such an event is input-dependent so cannot be simulated.

3.10 Multi-Key Rerandomizable Public-Key Encryption

Gao et al. [GNT23] introduce the concept of multi-key rerandomizable public-key encryption (MKR-PKE), which is a variant of PKE with several additional properties. Let $\mathcal{SK}$ denote the space of secret keys which forms an abelian group under the operation $+$, $\mathcal{PK}$ the space of public keys which forms an abelian group under the operation $\cdot$, $\mathcal{M}$ the space of plaintexts, and $\mathcal{C}$ the space of ciphertexts. MKR-PKE is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$ such that:

- The key-generation algorithm Gen takes as input a security parameter 1^λ and outputs a pair of keys $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$.
- The encryption algorithm Enc takes as input a public key $pk \in \mathcal{PK}$ and a plaintext message $x \in \mathcal{M}$, and outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$.
- The partial decryption algorithm ParDec takes as input a secret key share $sk \in \mathcal{SK}$ and a ciphertext $\mathsf{ct} \in \mathcal{C}$, and outputs another ciphertext $\mathsf{ct}' \in \mathcal{C}$.
- The decryption algorithm Dec takes as input a secret key $sk \in \mathcal{SK}$ and a ciphertext $\mathsf{ct} \in \mathcal{C}$, outputs a message $x \in \mathcal{M}$ or an error symbol $\perp$.
- The rerandomization algorithm ReRand takes as input a public key $pk \in \mathcal{PK}$ and a ciphertext $\mathsf{ct} \in \mathcal{C}$, outputs another ciphertext $\mathsf{ct}' \in \mathcal{C}$.

MKR-PKE requires the following additional properties besides those of PKE with indistinguishable multiple encryptions. (cf. Appendix B):

Partially Decryptable For any two pairs of keys $(sk_1, pk_1) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(sk_2, pk_2) \leftarrow \mathsf{Gen}(1^\lambda)$ and any $x \in \mathcal{M}$, it holds that

$$\mathsf{ParDec}(sk_1, \mathsf{Enc}(pk_1 \cdot pk_2, x)) = \mathsf{Enc}(pk_2, x)$$

Rerandomizable For any $pk \in \mathcal{PK}$ and any $x \in \mathcal{M}$, it holds that

$$\mathsf{ReRand}(pk, \mathsf{Enc}(pk, x)) \overset{s}{\approx} \mathsf{Enc}(pk, x)$$

[GNT23] uses elliptic curve based ElGamal encryption [ElG85] to instantiate MKR-PKE. The concrete EC MKR-PKE is described in Appendix D.

Remark 1. We note that only a few of elliptic curves support efficient encoding from bit-strings to EC points. Therefore, the plaintext space of EC MKR-PKE is generally restricted to EC points to guarantee rerandomizable property.

4 Batch Secret-Shared Private Membership Test

The batch secret-shared private membership test (batch ssPMT) is a central building block in both our SK-MPSU and PK-MPSU protocols. In this section, we formally introduce this functionality and provide a construction built by the aforementioned primitives.

The batch ssPMT is a two-party protocol implementing a batch of ssPMT instances between a sender and a receiver. Assuming a batch size of B, the sender $\mathcal{S}$ inputs B sets $X_1, \cdots, X_B$, while the receiver $\mathcal{R}$ inputs B elements $x_1, \cdots, x_B$. Consequently, $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of a bit vector of size B, whose ith bit is 1 if $x_i \in X_i$, otherwise 0. The batch ssPMT functionality is presented in Figure 8 and the construction is given in Figure 9.

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of set elements. The output length γ of OPPRF.
Inputs. $\mathcal{S}$ inputs B disjoint sets $X_1, \cdots, X_B$ and $\mathcal{R}$ inputs $\vec{x} \subseteq (\{0,1\}^l)^B$.
Functionality. On inputs $X_1, \cdots, X_B$ from $\mathcal{S}$ and input $\vec{x}$ from $\mathcal{R}$, for each $i \in [B]$, sample two random bits e_S^i, e_R^i under the constraint that if $x_i \in X_i, e_S^i \oplus e_R^i = 1$, otherwise $e_S^i \oplus e_R^i = 0$. Give $\vec{e}_S = (e_S^1, \cdots, e_S^B)$ to $\mathcal{S}$ and $\vec{e}_R = (e_R^1, \cdots, e_R^B)$ to $\mathcal{R}$.

Fig. 8. Batch ssPMT Functionality $\mathcal{F}_{\mathsf{bssPMT}}$

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of set elements. The output length γ of OPPRF.
Inputs. $\mathcal{S}$ inputs B disjoint sets $X_1, \cdots, X_B$ and $\mathcal{R}$ inputs $\vec{x} \subseteq (\{0,1\}^l)^B$.
Protocol.

1. For each $i \in [B]$, $\mathcal{S}$ chooses random $s_i \leftarrow \{0,1\}^\gamma$ and computes a multiset S_i comprising $|X_i|$ repeated elements that all equal to s_i.
2. The parties invoke $\mathcal{F}_{\mathsf{bOPPRF}}$ of batch size B. $\mathcal{S}$ acts as sender and inputs $X_1, \cdots, X_B$ as key sets and $S_1, \cdots, S_B$ as value sets. $\mathcal{R}$ acts as receiver with input $\vec{x}$, and receives a vector $\vec{t} = (t_1, \cdots, t_B)$.
3. The parties invoke B instances of $\mathcal{F}_{\mathsf{ssPEQT}}$, where in the ith instance $\mathcal{S}$ inputs s_i and $\mathcal{R}$ inputs t_i. In the end, $\mathcal{S}$ receives $e_S^i \in \{0,1\}$ and $\mathcal{R}$ receives $e_R^i \in \{0,1\}$.

Fig. 9. Batch ssPMT Π_{bssPMT}

Correctness. According to the functionality of batch OPPRF, if $x_i \in X_i$, then $\mathcal{R}$ receives $t_i = s_i$. Subsequently, in the ith instance of ssPEQT, they input s_i and t_i which are equal, hence their outputs satisfy $e_S^i \oplus e_R^i = 1$. Conversely, if $x_i \notin X_i$, then $\mathcal{R}$ receives a pseudorandom value t_i. The probability of $t_i = s_i$ equals $2^{-\gamma}$ and any $t_i = s_i, i \in [B]$ occurs with a probability of $B \cdot 2^{-\gamma}$. As we set $\gamma \geq \sigma + \log B$, so $B \cdot 2^{-\gamma} \leq 2^{-\sigma}$, which means the probability of any $t_i = s_i$

occurring is negligible[13]. After the invocation of ssPEQT, we conclude that if $x_i \notin X_i$, e_S^i and e_R^i satisfy $e_S^i \oplus e_R^i = 0$ with overwhelming probability.

Theorem 1. *Protocol Π_{bssPMT} securely realizes $\mathcal{F}_{\mathsf{bssPMT}}$ in the $(\mathcal{F}_{\mathsf{bOPRF}}, \mathcal{F}_{\mathsf{ssPEQT}})$-hybrid model.*

The security of the protocol follows immediately from the security of the batch OPPRF and the ssPEQT functionalities.

Complexity Analysis. We set $B = O(n)$ to be consistent with our MPSU protocols. Our construction in Figure 9 achieves linear complexities mainly profits from our instantiation of batch OPPRF with linear computation complexity and communication complexity, which can be clarified from two folds: First, we follow the paradigm in [PSTY19] to construct batch OPPRF from batch OPRF and OKVS and ultilize their technique to amortize communication so that the total communication of computing all $O(n)$ instances of batch OPPRF is the same as the total number of items, which is $O(n)$ rather that $O(n \log n / \log \log n)$. Second, we employ subfield vector oblivious linear evaluation (subfield-VOLE) [BCG+19a, BCG+19b, RRT23] to instantiate the batch OPRF and the OKVS construction in [RR22] so that the computation complexity of batch OPPRF of size $O(n)$ also scales linearly with n. A comprehensive complexity analysis is in Appendix F.1. We summarize the total costs as below:

- **Offline phase.** The computation complexity of each party is $O(\gamma n \log n)$. The communication complexity of each party is $O(t\lambda \log(\gamma n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(n)$. The communication complexity of each party is $O(\gamma n)$. The round complexity is $O(\log \gamma)$.

5 MPSU from Symmetric-Key Techniques

In this section, we introduce a new primitive called multi-party secret-shared random oblivious transfer (mss-ROT), then we utilize its two-choice-bit version to construct a SK-MPSU based on oblivious transfer and symmetric-key operations in the standard semi-honest model.

5.1 Multi-Party Secret-Shared Random Oblivious Transfer

The abstraction of the general version of mss-ROT is inspired by the generation of Beaver Triples in multi-party setting. In the two-party GMW protocol, a Beaver Triple is produced by two role-switching executions of 1-bit

[13] In fact, the lower bound of γ is relevant to the total number of the batch OPPRF invocations. In our MPSU protocols, the batch OPPRF is invoked more than once, thus γ should be more larger than this lower bound. Refer to F.1 for more details.

ROT [ALSZ13]. As a result, P_1 holds a_1, b_1, c_1 and P_2 holds a_2, b_2, c_2, such that $(a_1 \oplus a_2) \cdot (b_1 \oplus b_2) = c_1 \oplus c_2$. In the multi-party setting, the functionality of Beaver Triple is extended to let each party P_i hold a_i, b_i, c_i with the correlation $(\bigoplus_{i=1}^{m} a_i) \cdot (\bigoplus_{i=1}^{m} b_i) = \bigoplus_{i=1}^{m} c_i$. Given that $(\bigoplus_{i=1}^{m} a_i) \cdot (\bigoplus_{i=1}^{m} b_i)$ can be written as $(2-m)(\bigoplus_{i=1}^{m} a_i \cdot b_i) + \bigoplus_{1 \le i \le j \le m} (a_i + a_j) \cdot (b_i + b_j)$, multi-party Beaver Triple reduces to pairwise two-party Beaver Triples. In consideration of the simple fact that the functionality of ROT can be interpreted as $r_0 \oplus r_b = b \cdot \Delta$, where $\Delta = r_0 \oplus r_1$, we follow the approach to producing multi-party Beaver Triples to extend ROT into multi-party setting, which is to let each party P_i to individually input a choice bit b_i and receive the output shares r_i and Δ_i such that $r_1 \oplus \cdots \oplus r_m = (b_1 \oplus \cdots \oplus b_m) \cdot (\Delta_1 \oplus \cdots \oplus \Delta_m)$.

In the context of MPSU, we only need two-choice-bit version of mss-ROT, which allows a more efficient construction with less pairwise invocations of two-party ROT compared to the general version. To elaborate, the two-choice-bit version of mss-ROT (for simplicity, we call it mss-ROT for short hereafter) allows two parties P_{ch_0} and P_{ch_1} to hold their choice bits b_0 and b_1 s.t. $r_1 \oplus \cdots \oplus r_m = (b_0 \oplus b_1) \cdot (\Delta_1 \oplus \cdots \oplus \Delta_m)$. We give the formal functionality in Figure 10 and the detailed construction in Figure 11.

Theorem 2. *Protocol $\Pi_{\mathsf{mss\text{-}rot}}$ securely implements $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ in the presence of any semi-honest adversary corrupting $t < m$ parties in the $\mathcal{F}_{\mathsf{rot}}$-hybrid model.*

It is easy to see that our construction essentially boils down to performing ROT pairwise. As one of the benefits, we can utilize the derandomization technique [Bea91] to bring most tasks forward to the offline phase. And the correctness and security of the mss-ROT protocol stems from the correctness and security of ROT. For the complete proof, refer to Appendix E.1.

5.2 Construction of Our SK-MPSU

We now turn our attention to construct a SK-MPSU. The construction of our SK-MPSU follows the high-level ideas we introduced in the technical overview and is formally presented in Figure 12.

Theorem 3. *Protocol $\Pi_{\mathsf{SK\text{-}MPSU}}$ securely implements $\mathcal{F}_{\mathsf{mpsu}}$ against any semi-honest adversary corrupting $t < m$ parties in the $(\mathcal{F}_{\mathsf{bssPMT}}, \mathcal{F}_{\mathsf{mss\text{-}rot}}, \mathcal{F}_{\mathsf{ms}})$-hybrid model, where $P_1 = P_{\mathrm{ld}}$.*

The security proof of Theorem 3 is deferred to Appendix E.2.

Complexity Analysis. We provide a comprehensive complexity analysis for our SK-MPSU protocol in Appendix F.2. The total costs are summarized below:

- **Offline phase.** The offline computation complexity per party is $O(\gamma m n \log n + m^2 n(\log m + \log n))$. The offline communication complexity per party is $O(t\lambda m \log(\gamma n/t) + t\lambda m^2 \log(n/t) + \lambda m^2 n(\log m + \log n))$. The offline round complexity is $O(1)$.

– **Online phase.** The online computation complexity per party is $O(m^2 n)$. The online communication complexity of P_1 is $O(\gamma mn + (l+\kappa)m^2 n)$. The online communication complexity of P_j is $O(\gamma mn + m^2 n + (l+\kappa)mn)$. The online round complexity is $O(\log \gamma + m)$.

Parameters. m parties $P_1, \cdots, P_m$, where P_{ch_0} and P_{ch_1} provide inputs as shares of the choice bit, $\mathrm{ch}_0, \mathrm{ch}_1 \in [m]$. The message length l.
Functionality. On input $b_0 \in \{0,1\}$ from P_{ch_0} and $b_1 \in \{0,1\}$ from P_{ch_1},

– Sample $r_2, \cdots r_m, \Delta_1, \Delta_2, \cdots, \Delta_m \leftarrow \{0,1\}^l$ and give (r_j, Δ_j) to P_j for $2 \leq j \leq m$.
– If $b_0 \oplus b_1 = 0$, compute $r_1 = \bigoplus_{j=2}^m r_j$, else compute $r_1 = \Delta_1 \oplus (\bigoplus_{j=2}^m (r_j \oplus \Delta_j))$. Give r_1 to P_1.

Fig. 10. Multi-Party Secret-Shared Random OT Functionality $\mathcal{F}_{\mathsf{mss\text{-}rot}}$

6 MPSU from Public-Key Techniques

In this section, we describe how to construct a PK-MPSU achieving both linear computation and linear communication complexity. The construction of our PK-MPSU is formally presented in Figure 13.

Theorem 4. *Protocol $\Pi_{\mathsf{PK\text{-}MPSU}}$ securely implements $\mathcal{F}_{\mathsf{mpsu}}$ against any semi-honest adversary corrupting $t < m$ parties in the $(\mathcal{F}_{\mathsf{bssPMT}}, \mathcal{F}_{\mathsf{rot}})$-hybrid model, assuming the security of MKR-PKE scheme, where $P_1 = P_{\mathrm{ld}}$.*

The security of the above protocol is based on security of the underlying building blocks, batch ssPMT and ROT, along with the rerandomizable property and indistinguishable multiple encryptions of MKR-PKE. For the complete proof, refer to Appendix E.3.

Complexity Analysis. We provide a comprehensive complexity analysis for our PK-MPSU protocol in Appendix F.3. The total costs are summarized below:

– **Offline phase.** The offline computation complexity per party is $O(\gamma mn \log n)$. The offline communication complexity per party is $O(t\lambda m \log(\gamma n/t))$. The offline round complexity is $O(1)$.
– **Online phase.** The online computation complexity per party is $O(mn)$ symmetric-key operations and $O(mn)$ public-key operations. The onlne communication complexity per party is $O((\gamma + \lambda)mn)$. The online round complexity is $O(\log \gamma + m)$.

Parameters. m parties $P_1, \cdots, P_m$. where P_{ch_0} and P_{ch_1} provide inputs as shares of the choice bit, $\text{ch}_0, \text{ch}_1 \in [m]$. We use J to denote the set of indices for the remaining parties who do not provide a choice-bit input, i.e., $J = [m] \setminus \{\text{ch}_0, \text{ch}_1\}$. The message length l.
Inputs. P_{ch_0} has input $b_0 \in \{0,1\}$ and P_{ch_1} has input $b_1 \in \{0,1\}$.
Protocol.

1. P_{ch_0} and P_{ch_1} invoke $\mathcal{F}_{\mathsf{rot}}$ twice: First, P_{ch_0} acts as receiver with input b_0 and P_{ch_1} acts as sender. P_{ch_0} receives $r^{b_0}_{\text{ch}_1,\text{ch}_0} \in \{0,1\}^l$. P_{ch_1} receives $r^0_{\text{ch}_1,\text{ch}_0}, r^1_{\text{ch}_1,\text{ch}_0} \in \{0,1\}^l$; Second, P_{ch_1} acts as receiver with input b_1 and P_{ch_0} acts as sender. P_{ch_1} receives $r^{b_1}_{\text{ch}_0,\text{ch}_1} \in \{0,1\}^l$. P_{ch_0} receives $r^0_{\text{ch}_0,\text{ch}_1}, r^1_{\text{ch}_0,\text{ch}_1} \in \{0,1\}^l$.
2. For $j \in J$: P_{ch_0} and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_{ch_0} acts as receiver with input b_0 and P_j as sender without input. P_{ch_0} receives $r^{b_0}_{j,\text{ch}_0} \in \{0,1\}^l$. P_j receives $r^0_{j,\text{ch}_0}, r^1_{j,\text{ch}_0} \in \{0,1\}^l$.
3. For $j \in J$: P_{ch_1} and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_{ch_1} acts as receiver with input b_1 and P_j as sender. P_{ch_1} receives $r^{b_1}_{j,\text{ch}_1} \in \{0,1\}^l$. P_j receives $r^0_{j,\text{ch}_1}, r^1_{j,\text{ch}_1} \in \{0,1\}^l$.
4. For $j \in J$: P_j samples $\Delta_j \leftarrow \{0,1\}^l$ and computes $r_j = r^0_{j,\text{ch}_0} \oplus r^0_{j,\text{ch}_1}$. P_j sends $u_{j,\text{ch}_0} = \Delta_j \oplus r^0_{j,\text{ch}_0} \oplus r^1_{j,\text{ch}_0}$ to P_{ch_0}, and $u_{j,\text{ch}_1} = \Delta_j \oplus r^0_{j,\text{ch}_1} \oplus r^1_{j,\text{ch}_1}$ to P_{ch_1}, then outputs (r_j, Δ_j).
5. P_{ch_0} computes $\Delta_{\text{ch}_0} = r^0_{\text{ch}_0,\text{ch}_1} \oplus r^1_{\text{ch}_0,\text{ch}_1}$ and $r_{\text{ch}_0} = \bigoplus_{j\in J}(r^{b_0}_{j,\text{ch}_0} \oplus b_0 \cdot u_{j,\text{ch}_0}) \oplus r^0_{\text{ch}_0,\text{ch}_1} \oplus r^{b_0}_{\text{ch}_1,\text{ch}_0} \oplus b_0 \cdot \Delta_{\text{ch}_0}$ ($\cdot$ denotes bitwise-AND between the repetition code of b_0 and u_{j,ch_0}, which are both strings of length l. Similarly hereinafter), then outputs $(r_{\text{ch}_0}, \Delta_{\text{ch}_0})$. P_{ch_1} computes $\Delta_{\text{ch}_1} = r^0_{\text{ch}_1,\text{ch}_0} \oplus r^1_{\text{ch}_1,\text{ch}_0}$ and $r_{\text{ch}_1} = \bigoplus_{j\in J}(r^{b_1}_{j,\text{ch}_1} \oplus b_1 \cdot u_{j,\text{ch}_1}) \oplus r^0_{\text{ch}_1,\text{ch}_0} \oplus r^{b_1}_{\text{ch}_0,\text{ch}_1} \oplus b_1 \cdot \Delta_{\text{ch}_1}$, then outputs $(r_{\text{ch}_1}, \Delta_{\text{ch}_1})$.

Fig. 11. Multi-Party Secret-Shared Random OT $\Pi_{\mathsf{mss\text{-}rot}}$

Parameters. m parties $P_1, \cdots, P_m$. Size n of input sets. The bit length l of set elements. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and number of bins B. An OKVS scheme $(\mathsf{Encode_H}, \mathsf{Decode_H})$. A collision-resisitant hash function $\mathsf{H}(x) : \{0,1\}^l \to \{0,1\}^\kappa$.
Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.
Protocol.

1. **Hashing to bin.** P_1 does $\mathcal{T}_1^1, \cdots, \mathcal{T}_1^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_1)$. For $1 < j \leq m$, P_j does $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_j)$ and $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_j)$.
2. **Batch secret-shared private membership test.** For $1 \leq i < j \leq m$: P_i and P_j invoke $\mathcal{F}_{\mathsf{bssPMT}}$ of batch size B, where P_i acts as sender with inputs $\mathcal{T}_i^1, \cdots, \mathcal{T}_i^B$ and P_j acts as receiver with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$. For the instance $b \in [B]$, P_i receives $e^b_{i,j} \in \{0,1\}$, and P_j receives $e^b_{j,i} \in \{0,1\}$.
3. **Multi-party secret-shared random oblivious transfers.** For $1 \leq i < j \leq m, 1 \leq b \leq B$: $P_1, \cdots, P_j$ invoke $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ where P_i acts as P_{ch_0} with input $e^b_{i,j}$ and P_j acts as P_{ch_1} with input $e^b_{j,i}$. For $1 \leq d < j$, P_d receives $r^b_{d,ji}, \Delta^b_{d,ji} \in \{0,1\}^{l+\kappa}$ and computes $u^b_{d,j} = \bigoplus_{i=1}^{j-1} r^b_{d,ji}$. P_j receives $r^b_{j,ji}, \Delta^b_{j,ji} \in \{0,1\}^{l+\kappa}$ and computes $u^b_{j,j} = \bigoplus_{i=1}^{j-1} r^b_{j,ji} \oplus (\mathsf{Elem}(\mathcal{C}_j^b) \| \mathsf{H}(\mathsf{Elem}(\mathcal{C}_j^b)))$ if $\mathcal{C}_j^b$ is not corresponding to an empty bin, otherwise chooses $u^b_{j,j}$ at random. $\mathsf{Elem}(\mathcal{C}_j^b)$ denotes the element in $\mathcal{C}_j^b$.
4. **Multi-party secret-shared shuffle.**
   (a) For $1 \leq i \leq m$, each party P_i computes $\vec{sh}_i \in (\{0,1\}^{l+\kappa})^{(m-1)B}$ as follows: For $max(2,i) \leq j \leq m, 1 \leq b \leq B$, $sh_{i,(j-2)B+b} = u^b_{i,j}$. Set all other positions to 0.
   (b) For $1 \leq i \leq m$, all parties P_i invoke $\mathcal{F}_{\mathsf{ms}}$ with input $\vec{sh}_i$. P_i receives $\vec{sh}_i'$.
5. **Output reconstruction.** For $2 \leq j \leq m$, P_j sends $\vec{sh}_j'$ to P_1. P_1 recovers $\vec{v} = \bigoplus_{i=1}^m \vec{sh}_i'$ and sets $Y = \emptyset$. For $1 \leq i \leq (m-1)B$, if $v_i' = x \| \mathsf{H}(x)$ holds for some $x \in \{0,1\}^l$, adds x to Y. Outputs $X_1 \cup Y$.

Fig. 12. Our SK-MPSU $\Pi_{\mathsf{SK\text{-}MPSU}}$

Parameters. m parties $P_1, \cdots, P_m$. Size n of input sets. The bit length l of set elements. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and number of bins B. An OKVS scheme $(\mathsf{Encode_H}, \mathsf{Decode_H})$. A MKR-PKE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$.
Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.
Protocol. Each party P_i runs $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$, and distributes its public key pk_i to other parties. Define $sk = sk_1 + \cdots + sk_m$ and each party can compute its associated public key $pk = \prod_{i=1}^m pk_i$.

1. **Hashing to bin.** P_1 does $\mathcal{T}_1^1, \cdots, \mathcal{T}_1^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_1)$. For $1 < j \leq m$, P_j does $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_j)$ and $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_j)$.
2. **Batch secret-shared private membership test.** For $1 \leq i < j \leq m$: P_i and P_j invoke $\mathcal{F}_{\mathsf{bssPMT}}$ of batch size B, where P_i acts as sender with inputs $\mathcal{T}_i^1, \cdots, \mathcal{T}_i^B$ and P_j acts as receiver with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$. For the instance $b \in [B]$, P_i receives $e_{i,j}^b \in \{0,1\}$, and P_j receives $e_{j,i}^b \in \{0,1\}$.
3. **Random oblivious transfers and messages rerandomization.**
   (a) For $2 \leq j \leq m, 1 \leq b \leq B$: P_j defines a vector $\vec{c}_j$ and sets $c_j^b = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}_j^b))$. $\mathsf{Elem}(\mathcal{C}_j^b)$ denotes the element in $\mathcal{C}_j^b$.
   - For $2 \leq i < j$: P_i and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_i acts as receiver with input $e_{i,j}^b$ and P_j acts as sender. P_i receives $r^b_{j,i,e^b_{i,j}} \in \{0,1\}^\lambda$. P_j receives $r^b_{j,i,0}, r^b_{j,i,1} \in \{0,1\}^\lambda$. P_j computes $u^b_{j,i,e^b_{j,i}} = r^b_{j,i,e^b_{j,i}} \oplus c_j^b$, $u^b_{j,i,e^b_{j,i}\oplus 1} = r^b_{j,e^b_{j,i}\oplus 1} \oplus \mathsf{Enc}(pk, \perp)$, then sends $u^b_{j,i,0}, u^b_{j,i,1}$ to P_i.
   - P_i defines $v^b_{j,i} = u^b_{j,i,e^b_{i,j}} \oplus r^b_{i,j}$ and sends $v'^b_{j,i} = \mathsf{ReRand}(pk, v^b_{j,i})$ to P_j. P_j updates $c_j^b = \mathsf{ReRand}(pk, v'^b_{j,i})$.

   (b) For $2 \leq j \leq m, 1 \leq b \leq B$:
   - P_1 and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_1 acts as receiver with input $e^b_{1,j}$ and P_j acts as sender. P_1 receives $r^b_{1,j} = r^b_{j,1,e^b_{1,j}} \in \{0,1\}^\lambda$. P_j receives $r^b_{j,1,0}, r^b_{j,1,1} \in \{0,1\}^\lambda$. P_j computes $u^b_{j,1,e^b_{j,1}} = r^b_{j,1,e^b_{j,1}} \oplus c_j^b$, $u^b_{j,1,e^b_{j,1}\oplus 1} = r^b_{j,1,e^b_{j,1}\oplus 1} \oplus \mathsf{Enc}(pk, \perp)$, then sends $u^b_{j,1,0}, u^b_{j,1,1}$ to P_1.
   - P_1 defines $\vec{\mathsf{ct}}'_1 \in (\{0,1\}^\lambda)^{(m-1)B}$, and sets $\mathsf{ct}_1'^{(j-2)B+b} = \mathsf{ReRand}(pk, u^b_{j,1,e^b_{1,j}} \oplus r^b_{1,j})$.
4. **Messages decryptions and shufflings.** P_1 samples $\pi_1 : [(m-1)B] \to [(m-1)B]$ and computes $\vec{\mathsf{ct}}''_1 = \pi_1(\vec{\mathsf{ct}}'_1)$. P_1 sends $\vec{\mathsf{ct}}''_1$ to P_2.
   (a) For $2 \leq j \leq m, 1 \leq i \leq (m-1)B$: P_j computes $\mathsf{ct}_j^i = \mathsf{ParDec}(sk_j, \mathsf{ct}''^i_{j-1})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $\mathsf{ct}'^i_j = \mathsf{ReRand}(pk_{A_j}, \mathsf{ct}_j^i)$. Then it samples $\pi_j : [(m-1)B] \to [(m-1)B]$ and computes $\vec{\mathsf{ct}}''_j = \pi_j(\vec{\mathsf{ct}}'_j)$. If $j \neq m$, P_j sends $\vec{\mathsf{ct}}''_j$ to P_{j+1}; else, P_m sends $\vec{\mathsf{ct}}''_m$ to P_1.
   (b) For $1 \leq i \leq (m-1)B$: P_1 computes $\mathsf{pt}_i = \mathsf{Dec}(sk_1, \mathsf{ct}''^i_m)$.
5. **Output reconstruction.** P_1 sets $Y = \emptyset$. For $1 \leq i \leq (m-1)B$, if $\mathsf{pt}_i \neq \perp$, it updates $Y = Y \cup \{\mathsf{pt}_i\}$. P_1 outputs Y.

Fig. 13. Our PK-MPSU $\Pi_{\mathsf{PK\text{-}MPSU}}$

Remark 2. When instantiating our PK-MPSU framework with EC MKR-PKE, the element space has to be set as EC points accordingly, which may limit its usage in a wide range of applications. We argue that the resulting PK-MPSU protocol is still useful in scenarios that the element space is exactly EC points. We demonstrate this by building the first multi-party private ID protocol (as described in Appendix G), in which $\Pi_{\mathsf{PK\text{-}MPSU}}$ for EC point elements is used as a core building block.

7 Theoretical Comparison

In this section, we compare the construction of mq-ssPMT in [LG23] (mq-ssPMT for short) and our alternative construction built on batch ssPMT (batch ssPMT for short) theoretically. Then we compare our two protocols with the recent works [LG23] (which represents the MPSU protocol with the best concrete performance) and [GNT23] (which represents the MPSU protocol with the best asymptotic complexity), respectively. We emphasize that both of the two protocols are not in the standard semi-honest model.

7.1 Comparison Between mq-ssPMT and Batch ssPMT

Liu and Gao adapt the SKE-based mq-RPMT in [ZCL$^+$23] to construct mq-ssPMT. The most expensive part of their construction is the secret-shared oblivious decryption-then-matching (ssVODM) [ZCL$^+$23] protocol, which is to implement a decryption circuit and a comparison circuit by the GMW protocol [GMW87]. In total, the ssVODM protocol requires $(T + l - \log n - 1)n$ AND gates, where l is the bit length of set elements, and T is the number of AND gates in the SKE decryption circuit and is set to be considerably large (≈ 600) according to their paper.

The costs of the batch ssPMT of size $B = 1.27n$[14] consist of the costs of batch OPPRF of size $1.27n$ and the costs of $1.27n$ instances of ssPEQT, where the batch OPPRF can be instantiated by extremely fast specialized protocol and the ssPEQT is implemented by the GMW protocol. Moreover, the state-of-the-art batch OPPRF construction [CGS22, RS21, RR22] can achieve linear computation and communication complexity with respect to n . In the ssPEQT, the parties engage in $1.27(\gamma - 1)n$ AND gates, where γ is the output length of OPPRF. Note that in the typical setting where $n \leq 2^{24}, l \leq 128, \gamma \leq 64$, we have $(T + l - \log n - 1)n \gg 1.27(\gamma - 1)n$, which means that the number of AND gates desired involved in batch ssPMT is far smaller than mq-ssPMT. Therefore, the construction built on batch ssPMT greatly reduces the dependency on general 2PC and significantly decreases both computation and communication complexity by a considerable factor, compared to the mq-ssPMT (For a finer-grained asymptotic analysis of mq-ssPMT and batch ssPMT, refer to Appendix F.4 and Appendix F.1).

[14] We use stash-less Cuckoo hashing [PSTY19] with 3 hash functions, where $B = 1.27n$.

7.2 Comparison Between LG Protocol and Our SK-MPSU

We provide a comprehensive complexity analysis for the state-of-the-art LG protocol in Appendix F.4. In Table 2, we present a comparison of the theoretical computation and communication complexity for each party in both offline and online phases between LG protocol and our SK-MPSU.

	Computation			**Communication**		
	Offline	**Online**		**Offline**	**Online**	
		Leader	Client		Leader	Client
[LG23]	$(T+l+m)mn\log n$	$(T+l+m)mn$	$(T+l)mn$	$\lambda m^2 n\log n$	$(T+l)mn+lm^2n$	$(T+l)mn$
Ours	$(\gamma+m)mn\log n$	m^2n		$\lambda m^2 n\log n$	$\gamma mn+lm^2n$	$(\gamma+l+m)mn$

Table 2. Asymptotic communication (bits) and computation costs of LG protocol and our SK-MPSU in the offline and online phases. n is the set size. m is the number of participants. λ and σ are computational and statistical security parameter respectively and $\lambda=128$, $\sigma=40$. T is the number of AND gate in an SKE decryption circuit, $T\approx 600$. l is the bit length of input elements. γ is the output length of OPPRF. t is the noise weight in dual LPN, $t\approx 128$.

As depicted in Table 2, our offline communication complexity is comparable to theirs, our offline computation complexity and online computation and communication complexity of the leader are superior to theirs. Nevertheless, our online computation and communication complexity of client seem to be surpassed by theirs, as ours are quadratic in the number of parties m, while their is linear. We argue that it is because that in order to achieve security against arbitrary collusion, our protocol introduces extra overhead (from mss-ROT). Even so, for most applications when m is moderate, and our online computation and communication costs still outperform theirs. This shall give the credit to our speedup by replacing their mq-ssPMT with batch ssPMT.

7.3 Comparison Between [GNT23] and Our PK-MPSU

In Table 3, we present a comparison of the theoretical computation and communication complexity for each party in both offline and online phases between [GNT23] and our PK-MPSU.

We conclude that the complexity of our PK-MPSU surpasses theirs in all respects, including the computation and communication complexity of the leader and clients in the offline and online phases, as depicted in the Table 3.

Remark 3. The reason why our PK-MSU protocol achieves linear complexities whereas the protocol in [GNT23] does not, lies in that the combination of our batch ssPMT and ROT extension essentially realizes a batched version of their membership Oblivious Transfer (mOT) with $O(1)$ amortized costs for each instance (which mainly profits from the batch OPPRF instantiation achieving linear computation complexity and communication complexity, see Section 4).

	Computation		Communication	
	Offline	**Online**	**Offline**	**Online**
[GNT23]	$\gamma mn \log n(\log n / \log\log n)$	$mn(\log n / \log\log n)$	$t\lambda m \log n(\log n / \log\log n)$	$(\gamma+\lambda)mn(\log n / \log\log n)$
Ours	$\gamma mn \log n$	mn	$t\lambda m \log n$	$(\gamma+\lambda)mn$

Table 3. Asymptotic communication (bits) and computation costs of [GNT23] and our PK-MPSU protocol in the offline and online phases. In the offline phase, the computation is composed of symmetric-key operations; In the online phase, the computation is composed of public-key operations since we ignore symmetric-key operations. n is the set size. m is the number of participants. λ and σ are computational and statistical security parameter respectively and $\lambda = 128$, $\sigma = 40$. l is the bit length of input elements. γ is the output length of OPPRF. t is the noise weight in dual LPN, $t \approx 128$.

In contrast, their protocol requires the invocation of conditional oblivious pseudorandom function (cOPRF) besides mOT, and the instantiations of cOPRF and mOT incur $O(\log n/\log\log n)$ amortized costs for each instance.

8 Performance Evaluation

In this section, we provide implementation details and experimental results for our SK-MPSU and PK-MPSU protocols. Most previous works [KS05, Fri07, BA12, GNT23] lack open-source codes. The work of [VCE22] shows fairly poor performance in comparison with the state-of-the-art LG protocol. Therefore, we only compare our works with LG protocol whose complete implementation is available on https://github.com/lx-1234/MPSU.

In the implementation of LG protocol, they designate one party to generate share correlations in the offline phase and store them as local files, so that other parties can read these files and consume these share correlations in the online phase[15]. This implementation is not faithful to the protocol specifications, which is to let the parties pairwise execute share translation protocols [CGP20]. It has two pitfalls: (1) Their code does not support distributed execution. (2) It would lead to serious information leakage. Concretely, if it is the leader who generates share correlations, then it can learn the party to whom each element in the union belongs (which is the same security problem addressed by the execution of multi-party secret shuffle as we mentioned in the technical overview). If it is a client who generates share correlations, then it can learn the union. Moreover, their implementation gives incorrect results in several test cases when the set size n is quite large (cf. Table 4). As a result, their code cannot be considered as a correct or secure implementation for MPSU.

To conduct a fair comparison, we replace their implementation of share correlation generation with our correct one.

[15] Refer to the function `ShareCorrelation::generate()` in ShareCorrelationGen.cpp of the repository https://github.com/lx-1234/MPSU.git

8.1 Experimental Setup

We run LG and our two protocols on Ubuntu 22.04 with a single Intel i7-13700 2.10 GHz CPU (16 physical cores) and 64 GB RAM. We emulate the two network connections using Linux `tc` command. In the LAN setting, the bandwidth is set to be 10 Gbps with 0.1 ms RTT latency. In the WAN setting, the bandwidth is set to be 400 Mbps with 80 ms RTT latency. We record the running time as the maximal time from protocol begin to end, including messages transmission time. We compute the communication costs of the leader as the sum of the data it sent and received. For a fair comparison, we stick to the following setting for all protocols being evaluated:

– We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$.
– We test the balanced scenario by setting all m input sets to be of equal size. In LG protocol and our SK-MPSU, each party holds n 64-bit strings. In our PK-MPSU, we assume that each party holds n elements encoded as elliptic curve points (We provide two versions of the implementation of our PK-MPSU, including encoding elements in uncompressed form and in compressed form respectively).
– For each party, we use $m - 1$ threads to interact with all other parties simultaneously and 4 threads to perform share correlation generation (in LG protocol and our SK-MPSU), Beaver Triple generation (in all three), parallel SKE encryption (in LG protocol), ciphertext rerandomization and partial decryption (in our PK-MPSU).

8.2 Implementation Details

Our protocols are written in C++, and we use the following libraries in our implementation.

– VOLE: We use VOLE implemented in libOTe[16], and instantiate the code family with Expand-Convolute codes in [RRT23].
– OKVS and GMW: We use the optimized OKVS in [RR22] as our OKVS instantiation[17], and re-use the implementation of OKVS and GMW by the authors of in [RR22][18].
– ROT: We use SoftSpokenOT [Roy22] implemented in libOTe, and set field bits to 5 to balance computation and communication costs.
– Share Correlation: We re-use the implementation of Permute+Share [MS13, CGP20] by the authors of in [JSZ+22][19] to build the generation of share correlations for our SK-MPSU and LG.

[16] https://github.com/osu-crypto/libOTe.git
[17] Since the existence of suitable parameters for the new OKVS construction of the recent work [BPSY23] is unclear when the set size is less than 2^{10}, we choose to use the OKVS construction of [RR22].
[18] https://github.com/Visa-Research/volepsi.git
[19] https://github.com/dujiajun/PSU.git

– MKR-PKE: We implement MKR-PKE on top of the curve `NIST P-256` (also known as `secp256r1` and `prime256v1`) implementation from openssl[20].
– Additionally, we use the cryptoTools[21] library to compute hash functions and PRNG calls, and we adopt Coproto[22] to realize network communication.

8.3 Experimental Results

We conduct an extensive experiment for the numbers of parties $\{3, 4, 5, 7, 9, 10\}$ and a wide range of set sizes $\{2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$ in the LAN and WAN settings. We compare the performance of the protocols from four dimensions: online and total running time, and online and total communication costs. The results of online and total running time are depicted in Table 4. The results of online and total communication costs are depicted in Table 5.

As we can see in the tables, our protocols outperform LG protocol in almost all the case studies. Especially, our SK-MPSU protocol performs best in terms of online running time in LAN setting, online communication costs for a relatively small number of parties, and total running time for medium and large sets in LAN setting among the three protocols. We observe a $4.9 - 9.3\times$ improvement in online running time and a $1.7 - 8.5\times$ improvement in total running time in the LAN setting. It is worth mentioning that our SK-MPSU takes only 3.6 seconds for 3 parties with sets of 2^{20} items each, and 4.9 seconds for 4 parties with sets of 2^{20} items each in online phase, while the implementation of LG protocol fails to run correctly in these cases.

On the other hand, our PK-MPSU protocol performs best in terms of total running time in WAN setting and total communication costs among the three protocols. It shows a $3.0 - 36.5\times$ improvement in overall communication, and a $1.8 - 5.4\times$ improvement in overall running time in the WAN setting, compared to LG protocol.

For the cases where both the number of parties and the set size are relatively large and the network environment is bandwidth-constrained, LG protocol marginally surpasses ours in terms of online running time, but its speedup is only up to $1.4\times$. This result aligns with our theoretical analysis in Section 7.2. Nevertheless, when each party's set size increases to sufficiently large (2^{20}), their implementation gives wrong results.

Acknowledgements. We would like to thank Jiahui Gao for the clarification of their work.

References

ALSZ13. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, 2013*, pages 535–548. ACM, 2013.

[20] https://github.com/openssl/openssl.git
[21] https://github.com/ladnir/cryptoTools.git
[22] https://github.com/Visa-Research/coproto.git

Sett.	m	Protocol	Set size n: Online time (seconds)								Set size n: Total time (seconds)							
			2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
LAN	3	LG23	0.030	0.038	0.056	0.106	0.286	1.024	3.899	×	0.631	0.660	0.822	1.188	4.709	22.62	95.98	×
		Our SK	0.004	0.005	0.006	0.012	0.033	0.138	0.749	3.641	0.232	0.254	0.282	0.344	0.824	3.042	14.32	119.2
		Our PK$^\blacklozenge$	0.061	0.134	0.406	1.466	5.452	22.29	87.23	357.3	0.086	0.163	0.440	1.543	5.753	23.55	92.74	389.6
		Our PK$^\blacktriangledown$	0.083	0.200	0.562	2.039	8.030	31.53	127.4	514.3	0.108	0.229	0.596	2.116	8.331	32.79	133.0	546.6
	4	LG23	0.050	0.055	0.063	0.117	0.336	1.274	5.572	×	0.948	0.970	1.098	1.744	6.939	32.12	135.4	×
		Our SK	0.006	0.007	0.008	0.017	0.049	0.202	1.043	4.905	0.442	0.458	0.489	0.665	1.700	7.167	36.16	344.71
		Our PK$^\blacklozenge$	0.125	0.243	0.734	2.642	10.35	40.83	166.7	–	0.161	0.280	0.784	2.760	11.00	43.51	178.3	–
		Our PK$^\blacktriangledown$	0.160	0.346	1.033	3.936	15.50	61.19	246.3	–	0.196	0.383	1.083	4.054	16.16	63.87	257.9	–
	5	LG23	0.064	0.066	0.077	0.133	0.383	1.509	7.613	–	1.325	1.355	1.419	2.680	10.82	48.95	213.6	–
		Our SK	0.009	0.010	0.013	0.021	0.061	0.290	1.478	–	0.677	0.708	0.760	1.114	3.260	13.10	62.35	–
		Our PK$^\blacklozenge$	0.170	0.401	1.229	4.191	16.41	64.84	262.5	–	0.211	0.443	1.273	4.348	17.13	67.89	276.3	–
		Our PK$^\blacktriangledown$	0.192	0.509	1.611	6.226	24.62	98.04	400.0	–	0.233	0.551	1.655	6.383	25.33	101.1	413.8	–
	7	LG23	0.099	0.111	0.118	0.233	0.676	3.684	14.45	–	2.356	2.515	2.752	5.247	23.77	103.6	452.1	–
		Our SK	0.014	0.017	0.021	0.040	0.109	0.545	2.701	–	1.312	1.357	1.510	2.336	8.368	37.17	146.0	–
		Our PK$^\blacklozenge$	0.273	0.649	2.225	8.325	31.62	125.7	521.2	–	0.338	0.715	2.314	8.695	33.02	131.7	552.7	–
		Our PK$^\blacktriangledown$	0.339	1.031	3.230	12.25	48.71	195.2	798.1	–	0.404	1.097	3.319	12.62	50.11	201.3	829.6	–
	9	LG23	0.136	0.142	0.18	0.49	1.584	6.121	–	–	4.169	4.251	4.716	9.418	44.19	191.6	–	–
		Our SK	0.025	0.029	0.034	0.058	0.186	0.972	–	–	2.183	2.272	2.557	5.085	18.93	80.01	–	–
		Our PK$^\blacklozenge$	0.429	1.064	3.553	13.23	51.74	206.8	–	–	0.495	1.133	3.660	13.79	54.37	217.9	–	–
		Our PK$^\blacktriangledown$	0.567	1.558	5.346	22.65	80.03	322.5	–	–	0.633	1.627	5.453	23.21	82.66	333.6	–	–
	10	LG23	0.161	0.167	0.215	0.531	2.123	8.301	–	–	4.943	5.178	5.604	12.82	59.17	252.5	–	–
		Our SK	0.03	0.033	0.042	0.073	0.234	1.219	–	–	2.745	2.883	3.290	6.733	25.12	110.7	–	–
		Our PK$^\blacklozenge$	0.498	1.338	4.348	16.35	63.54	260.6	–	–	0.581	1.435	4.492	17.03	66.77	275.4	–	–
		Our PK$^\blacktriangledown$	0.677	1.913	6.552	25.11	98.94	404.3	–	–	0.760	2.010	6.696	25.79	102.2	419.1	–	–
WAN	3	LG23	4.510	4.516	4.521	4.979	6.295	11.87	17.98	×	11.87	13.04	15.00	18.33	28.49	85.85	247.6	×
		Our SK	2.413	2.570	2.580	3.402	4.129	5.241	10.91	31.83	8.591	9.731	12.81	16.63	25.56	56.39	192.6	826.3
		Our PK$^\blacklozenge$	4.559	4.952	5.787	7.481	13.91	37.62	131.90	505.93	5.300	5.698	6.694	8.420	15.22	39.86	138.5	538.4
		Our PK$^\blacktriangledown$	4.571	4.662	5.643	7.796	14.82	42.08	150.5	593.6	5.312	5.408	6.550	8.735	16.13	44.33	157.1	626.1
	4	LG23	5.789	5.876	6.293	6.956	7.587	13.58	23.32	×	17.10	19.64	26.64	32.29	49.43	139.9	490.9	×
		Our SK	3.376	3.536	3.553	4.408	5.919	8.448	20.65	64.01	11.81	14.41	20.42	27.29	48.65	132.1	499.2	2283
		Our PK$^\blacklozenge$	6.231	7.181	8.962	11.88	23.57	69.76	251.4	–	7.374	8.327	10.27	13.22	25.31	73.35	264.1	–
		Our PK$^\blacktriangledown$	6.264	6.436	8.256	12.14	25.92	81.50	294.2	–	7.407	7.582	9.565	13.48	27.66	85.08	306.9	–
	5	LG23	7.556	7.794	7.902	9.102	10.63	15.18	34.82	–	23.26	26.46	35.49	47.53	83.94	250.6	969.6	–
		Our SK	4.501	4.583	5.217	5.955	7.936	12.21	32.14	–	18.11	21.85	30.93	45.68	91.47	278.9	1112	–
		Our PK$^\blacklozenge$	8.050	9.745	11.31	16.08	35.07	108.2	388.9	–	9.593	11.31	13.03	17.84	37.24	112.5	404.4	–
		Our PK$^\blacktriangledown$	8.072	8.869	10.91	10.84	42.83	129.1	468.9	–	9.615	10.43	12.63	18.85	45.00	133.3	484.4	–
	7	LG23	9.722	10.44	11.25	11.67	12.71	22.73	68.51	–	34.35	45.68	65.99	95.61	203.1	707.2	2881	–
		Our SK	7.241	7.414	8.121	9.808	14.52	22.93	69.19	–	35.65	46.73	62.48	99.11	234.0	817.7	3399	–
		Our PK$^\blacklozenge$	11.44	14.77	17.55	25.72	61.13	201.6	745.7	–	13.79	17.14	20.10	28.30	64.28	209.3	779.2	–
		Our PK$^\blacktriangledown$	11.54	14.11	15.89	29.37	70.84	249.3	921.8	–	13.89	16.48	18.43	31.96	73.98	257.0	955.3	–
	9	LG23	11.07	11.41	13.94	14.91	15.86	34.82	–	–	55.36	74.52	103.4	166.2	413.7	1531	–	–
		Our SK	10.62	10.81	11.75	13.88	21.07	39.73	–	–	57.91	88.27	113.7	189.5	505.3	1881	–	–
		Our PK$^\blacklozenge$	16.30	19.45	24.87	37.99	93.69	327.8	–	–	19.45	22.64	28.21	41.33	97.85	341.3	–	–
		Our PK$^\blacktriangledown$	15.65	19.58	22.33	42.38	109.7	396.3	–	–	18.80	22.76	25.68	45.73	113.8	409.9	–	–
	10	LG23	12.81	12.99	16.29	16.83	17.50	45.20	–	–	66.79	91.10	137.2	126.4	579.0	2205	–	–
		Our SK	12.56	12.76	13.60	16.27	25.20	49.94	–	–	70.02	113.8	140.6	250.8	693.0	2644	–	–
		Our PK$^\blacklozenge$	19.48	22.71	28.51	43.45	112.5	392.0	–	–	23.06	26.32	32.28	47.28	117.2	408.5	–	–
		Our PK$^\blacktriangledown$	18.28	21.67	26.72	50.03	134.4	487.3	–	–	21.87	25.28	30.49	53.86	139.1	503.8	–	–

Table 4. Online and total running time of LG protocol and our protocols in LAN and WAN settings. m is the number of parties. LG denotes LG protocol. Our SK denotes our SK-MPSU protocol. Our PK denotes our PK-MPSU protocol and we denote the version that does not use or uses point compression technique with $\blacklozenge$ and $\blacktriangledown$ respectively. Cells with $\times$ denotes trials that obtain wrong results. Cells with $-$ denotes trials that ran out of memory. The best protocol within a setting is marked in blue.

m	Protocol	Set Size n															
		Online communication (MB)								Total communication (MB)							
		2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
3	LG23	0.157	0.284	0.962	3.662	14.43	57.58	229.8	×	6.311	7.904	13.37	32.58	114.6	474.3	2054	×
	Our SK	0.031	0.111	0.426	1.687	6.778	27.83	112.5	455.0	2.541	3.582	8.529	30.90	132.7	588.7	2615	11533
	Our PK⧫	1.854	2.147	3.293	7.984	26.69	102.4	406.8	1634	2.123	2.489	3.711	8.474	27.30	103.0	408.8	1640
	Our PK▼	1.815	1.983	2.655	5.418	16.36	60.78	239.3	959.5	2.084	2.325	3.073	5.908	16.92	61.41	241.3	966.1
4	LG23	0.242	0.449	1.536	5.868	23.15	92.38	368.6	×	9.591	12.44	23.89	67.25	253.7	1074	4677	×
	Our SK	0.060	0.218	0.848	3.376	13.57	55.44	223.9	904.4	3.943	6.399	17.01	67.12	294.7	1315	5851	25819
	Our PK⧫	2.781	3.212	4.939	11.98	40.04	153.6	610.2	–	3.185	3.725	5.566	12.71	40.88	154.5	613.2	–
	Our PK▼	2.723	2.975	3.983	8.130	24.54	91.20	359.0	–	3.127	3.488	4.610	8.127	25.38	92.12	362.0	–
5	LG23	0.330	0.630	2.173	8.325	32.86	131.2	523.5	–	13.12	17.75	36.48	110.7	435.2	1869	8172	–
	Our SK	0.097	0.361	1.411	5.628	22.63	92.18	372.0	–	5.511	10.22	30.59	125.3	551.2	2455	10875	–
	Our PK⧫	3.708	4.283	6.585	15.97	53.38	204.8	813.6	–	4.247	4.966	7.421	16.95	54.51	206.0	817.5	–
	Our PK▼	3.630	3.967	5.311	10.84	32.72	121.6	478.7	–	4.169	4.650	6.147	11.82	33.84	122.8	482.6	–
7	LG23	0.519	1.038	3.635	13.99	55.29	220.8	881.3	–	19.92	29.90	71.59	243.2	997.9	4327	18930	–
	Our SK	0.198	0.750	2.956	11.82	47.55	193.1	778.6	–	8.853	18.92	65.23	277.2	1233	5505	24402	–
	Our PK⧫	5.563	6.425	9.878	23.95	80.07	307.1	1220	–	6.371	7.450	11.13	25.42	81.76	309.0	1226	–
	Our PK▼	5.445	5.950	7.966	16.25	49.07	182.3	718.0	–	6.253	6.975	9.220	17.72	50.76	184.2	723.9	–
9	LG23	0.723	1.509	5.347	20.65	81.72	326.3	–	–	28.15	44.74	115.0	415.3	1742	7609	–	–
	Our SK	0.333	1.278	5.060	20.27	81.54	330.5	–	–	13.90	33.27	120.9	520.0	2310	10268	–	–
	Our PK⧫	7.417	8.566	13.17	31.94	106.8	409.5	–	–	8.495	9.933	14.84	33.90	109.0	412.0	–	–
	Our PK▼	7.26	7.933	10.62	21.67	65.43	243.1	–	–	8.338	9.300	12.29	23.63	67.68	245.6	–	–
10	LG23	0.831	1.768	6.296	24.36	96.43	385.1	–	–	32.32	54.34	148.7	549.4	2314	10081	–	–
	Our SK	0.413	1.594	6.322	25.34	101.9	412.9	–	–	16.12	40.87	151.2	656.4	2921	12989	–	–
	Our PK⧫	8.344	9.637	14.82	35.93	120.1	460.7	–	–	9.556	11.18	16.70	38.13	122.6	463.5	–	–
	Our PK▼	8.168	8.925	11.95	24.38	73.61	273.5	–	–	9.380	10.46	13.83	26.59	76.14	276.3	–	–

Table 5. Online and total communication costs of LG protocol and our protocols. m is the number of parties. LG denotes LG protocol. Our SK denotes our SK-MPSU protocol. Our PK denotes our PK-MPSU protocol and we denote the version that does not use or uses point compression technique with ⧫ and ▼ respectively. Cells with × denotes trials that obtain wrong results. Cells with − denotes trials that ran out of memory. The best protocol within a setting is marked in blue.

BA12. Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS 2012*, pages 40–41. ACM, 2012.

BC23. Dung Bui and Geoffroy Couteau. Improved private set intersection for sets with small entries. In *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 13941 of *Lecture Notes in Computer Science*, pages 190–220. Springer, 2023.

BCG+19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*, pages 291–308. ACM, 2019.

BCG+19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology - CRYPTO 2019*. Springer, 2019.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

BKM+20. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. 2020. https://eprint.iacr.org/2020/599.

BNOP22. Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. Psimple: Practical multiparty maliciously-secure private set intersection. In *ASIA CCS '22*, pages 1098–1112. ACM, 2022.

BPSY23. Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-Optimal oblivious Key-Value stores for efficient PSI, PSU and Volume-Hiding Multi-Maps. In *USENIX Security 2023*, pages 301–318, 2023.

BS05. Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security*, volume 3788 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2005.

CDG+21. Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In *CCS '21*, pages 1182–1204. ACM, 2021.

CGP20. Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *Advances in Cryptology - ASIACRYPT 2020*, volume 12493 of *Lecture Notes in Computer Science*, pages 342–372. Springer, 2020.

CGS22. Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch OPPRF. *Proc. Priv. Enhancing Technol.*, 2022(1):353–372, 2022.

CM20. Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

CO18. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.

CZZ+24a. Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In *Public-Key Cryptography - PKC 2024*. Springer, 2024.

CZZ+24b. Yu Chen, Min Zhang, Cong Zhang, Minglang Dong, and Weiran Liu. Private set operations from multi-query reverse private membership test. In *PKC 2024*, 2024. https://eprint.iacr.org/2022/652.

EB22. Saba Eskandarian and Dan Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society, 2022.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

FIPR05. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

Fri07. Keith B. Frikken. Privacy-preserving set union. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007.

GMR+21. Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *Public-Key Cryptography - PKC 2021*, volume 12711 of *Lecture Notes in Computer Science*, pages 591–617. Springer, 2021.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987*, pages 218–229. ACM, 1987.

GNT23. Jiahui Gao, Son Nguyen, and Ni Trieu. Toward a practical multi-party private set union. Cryptology ePrint Archive, Paper 2023/1930, 2023. https://eprint.iacr.org/2023/1930.

GPR+21. Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Advances in Cryptology - CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.

HLS+16. Kyle Hogan, Noah Luther, Nabil Schear, Emily Shen, David Stott, Sophia Yakoubov, and Arkady Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE Cybersecurity Development, 2016*, pages 75–76. IEEE Computer Society, 2016.

JSZ+22. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *USENIX 2022*, 2022.

KKRT16. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS 2016*, pages 818–829. ACM, 2016.

KMP+17. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1257–1272. ACM, 2017.

KRTW19. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *Advances in Cryptology - ASIACRYPT 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 636–666. Springer, 2019.

KS05. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

LG23. Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. In *Advances in Cryptology - ASIACRYPT 2023*. Springer, 2023.

LV04. Arjen K. Lenstra and Tim Voss. Information security risk assessment, aggregation, and mitigation. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004.*, volume 3108 of *Lecture Notes in Computer Science*, pages 391–401. Springer, 2004.

MS13. Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 557–574. Springer, 2013.

NTY21. Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multi-party private set intersection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021*, pages 1151–1165. ACM, 2021.

PR04. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

PRTY19. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology - CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

PRTY20. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *Advances in Cryptology - EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

PSSZ15. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

PSTY19. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

PSZ14. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, 2014*, pages 797–812. USENIX Association, 2014.

Rab05. Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 187, 2005.

Roy22. Lawrence Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *Advances in Cryptology - CRYPTO 2022*. Springer, 2022.

RR22. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM CCS 2022*, 2022.

RRT23. Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *Advances in Cryptology - CRYPTO 2023*, volume 14084 of *Lecture Notes in Computer Science*, pages 602–632. Springer, 2023.

RS21. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *Advances in Cryptology - EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

VCE22. Jelle Vos, Mauro Conti, and Zekeriya Erkin. Fast multi-party private set operations in the star topology from secure ands and ors. *IACR Cryptol. ePrint Arch.*, page 721, 2022.

ZCL+23. Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Optimal private set union from multi-query reverse private membership test. In *USENIX 2023*, 2023. `https://eprint.iacr.org/2022/358`.

ZMS+21. Shengnan Zhao, Ming Ma, Xiangfu Song, Han Jiang, Yunxue Yan, and Qiuliang Xu. Lightweight threshold private set intersection via oblivious transfer. In *Wireless Algorithms, Systems, and Applications - 16th International Conference, WASA 2021*, volume 12939 of *Lecture Notes in Computer Science*, pages 108–116. Springer, 2021.

A Leakage Analysis of [GNT23]

The MPSU protocol in [GNT23] is claimed to be secure in the presence of arbitrary colluding participants. However, our analysis suggests that the protocol fails to achieve this security, and also requires the non-collusion assumption as LG protocol. First, we give a brief review of the protocol.

Apart from MKR-PKE, their protocol utilizes three primary ingredients: 1) The OPRF [PRTY19, CM20, RS21, RR22], where the sender $\mathcal{S}$ inputs a PRF key k, and the receiver $\mathcal{R}$ inputs a elements x and receives the corresponding PRF evaluation $\{F_k(x)\}$. 2) The conditional oblivious pseudorandom function (cOPRF), an extension they develop on the OPRF, where the sender $\mathcal{S}$ additionally inputs a set Y. If $x \notin Y$, $\mathcal{R}$ receives $F_k(x)$, else $\mathcal{R}$ receives a random value sampled by $\mathcal{S}$.[23] 3) The membership Oblivious Transfer (mOT), where $\mathcal{S}$ inputs an element y and two messages u_0, u_1, while $\mathcal{R}$ inputs a set X and receives u, one of u_0, u_1. If $y \in X$, $u = u_0$, else $u = u_1$.

To illustrate the insecurity of their protocol, we consider a three-party case where P_1 and P_3 each possess a single item $X_1 = \{x_1\}$ and $X_3 = \{x_3\}$ respectively, while P_2 possesses a set X_2. We assume that $x_1 = x_3$. According to their protocol, in step 3.(a), P_1 and P_2 invoke the OPRF where P_1 acts as $\mathcal{R}$

[23] In the latest version of [GNT23], they have made a slight adjustment to the functionality of cOPRF. However, this adjustment does not change their insecurity. Their latest protocol is still vulnerable to our attack.

inputting x_1 and P_2 acts as $\mathcal{S}$ inputting its PRF key k_2. P_1 receives the PRF value $F_{k_2}(x_1)$. Meanwhile, in step 3.(c), P_2 and P_3 invoke the cOPRF where P_3 acts as $\mathcal{R}$ inputting x_3, and P_2 acts as $\mathcal{S}$ inputting its PRF key k_2 and the set X_2. P_3 receives the output w from the cOPRF. By the definition of cOPRF functionality, if $x_3 \notin X_2$, $w = F_{k_2}(x_3)$, otherwise w is a random value.

If P_1 and P_3 collude, they can distinguish the cases where $x_3 \in X_2$ and $x_3 \notin X_2$ by comparing P_1's output $F_{k_2}(x_1)$ from the OPRF and P_3's output w from the cOPRF for equality. To elaborate, we recall that $x_1 = x_3$, so $F_{k_2}(x_1) = F_{k_2}(x_3)$. If $F_{k_2}(x_1) = w$, it implies that P_3 receives $F_{k_2}(x_3)$ from the cOPRF, so the coalition learns that $x_3 \notin X_2$; On the contrary, if $F_{k_2}(x_1) \neq w$, it implies that the output of P_3 from the cOPRF is not $F_{k_2}(x_3)$, so it is a random value, then the coalition learns that $x_3 \in X_2$. More generally, as long as P_1 and P_3 collude, they can identify whether each element $x \in X_1 \cap X_3$ belongs to X_2 or not, by comparing the PRF value $F_k(x)$ from the OPRF between P_1 and P_2 and the cPRF value (whose condition depends on $x \in X_2$ or not) from the cOPRF between P_2 and P_3. This acquired knowledge is information leakage in MPSU. Therefore, their protocol also requires the non-collusion assumption.

B Public-Key Encryption

A public-key encryption (PKE) scheme is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- The key-generation algorithm Gen takes as input the security parameter 1^λ and outputs a pair of keys $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$.
- The encryption algorithm Enc takes as input a public key pk and a plaintext $x \in \mathcal{M}$, and outputs a ciphertext ct.
- The decryption algorithm Dec takes as input a secret key sk and a ciphertext ct, and outputs a message x or or an error symbol $\perp$.

Correctness. For any (pk, sk) outputed by $\mathsf{Gen}(1^\lambda)$, and any $x \in \mathcal{M}$, it holds that $\mathsf{Dec}(sk, (\mathsf{Enc}(pk, x))) = x$.

The IND-CPA security of PKE implies security for encryption of multiple messages whose definition is as follows:

Definition 2. *A public-key encryption scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable multiple encryptions if for all PPT adversaries* $\mathcal{A}$ *s.t. any tuples* $(m_1, \cdots, m_q)$ *and* $(m'_1, \cdots, m'_q)$ *chosen by* $\mathcal{A}$ *(where* q *is polynomial in* λ*):*

$$\{\mathsf{Enc}(pk, m_1), \cdots, \mathsf{Enc}(pk, m_q) : (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)\} \overset{c}{\approx}$$
$$\{\mathsf{Enc}(pk, m'_1), \cdots, \mathsf{Enc}(pk, m'_q) : (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)\}$$

C Construction of Multi-Party Secret-Shared Shuffle

The functionality of share correlation generated in the offline phase is depicted in Figure 14. The online details of multi-party secret-shared shuffle in [EB22] are given in Figure 15.

Parameters. m parties $P_1, \cdots P_m$. The dimension of vector n. The item length l.
Functionality. On input $\pi_i : [n] \to [n]$ from each P_i ($1 \leq i \leq m$), sample $\vec{a}'_i, \vec{b}_i \leftarrow (\{0,1\}^l)^n$ for $1 \leq i \leq m-1$, and $\vec{a}_i, \vec{\Delta}_m \leftarrow (\{0,1\}^l)^n$ for $2 \leq i \leq m$, which satisfy

$$\vec{\Delta}_m = \pi_m(\cdots(\pi_2(\pi_1(\bigoplus_{i=2}^{m} \vec{a}_i) \oplus \vec{a}'_1) \oplus \vec{a}'_2) \cdots \oplus \vec{a}'_{m-1}) \oplus \bigoplus_{i=1}^{m-1} \vec{b}_i$$

Give $\vec{a}'_1, \vec{b}_1$ to P_1, $\vec{a}'_i, \vec{a}_i, \vec{b}_i$ to P_i ($1 < i < m$), and $\vec{a}_m, \vec{\Delta}_m$ to P_m.

Fig. 14. Share Correlation Functionality $\mathcal{F}_{\mathsf{sc}}$

Parameters: m parties $P_1, \cdots, P_m$. Ideal functionality $\mathcal{F}_{\mathsf{sc}}$ in Figure 14. The dimension of vector n. The item length l.
Inputs: Each party P_i has input $\vec{x}_i = (x_i^1, \cdots, x_i^n)$.
Protocol:

1. Each party P_i samples a random permutation $\pi_i : [n] \to [n]$ and invokes $\mathcal{F}_{\mathsf{sc}}$ with input π_i. P_1 receives $\vec{a}'_1, \vec{b}_1 \in (\{0,1\}^l)^n$; For $2 \leq j < m$, P_j receives $\vec{a}'_j, \vec{a}_j, \vec{b}_j \in (\{0,1\}^l)^n$; P_m receives $\vec{a}_m, \vec{\Delta}_m \in (\{0,1\}^l)^n$.
2. For $2 \leq j \leq m$, P_j computes $\vec{c}_j = \vec{x}_j \oplus \vec{a}_j$ and sends $\vec{c}_j$ to P_1.
3. P_1 computes $\vec{c'}_1 = \pi_1(\bigoplus_{j=2}^m \vec{c}_j \oplus \vec{x}_1) \oplus \vec{a}'_1$ and send it to P_2. P_1 outputs $\vec{b}_1$.
4. For $2 \leq j < m$, P_j computes $\vec{c'}_j = \pi_j(\vec{c'}_{j-1}) \oplus \vec{a}'_j$ and send it to P_{j+1}. P_j outputs $\vec{b}_j$.
5. P_m outputs $\pi_m(\vec{c'}_{m-1}) \oplus \vec{\Delta}_m$.

Fig. 15. Multi-Party Secret-Shared Shuffle Protocol Π_{ms}

D Construction of MKR-PKE from ElGamal

The MKR-PKE is instantiated using ElGamal encryption as follows:

- The key-generation algorithm Gen takes as input the security parameter 1^λ and generates $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a cyclic group, g is the generator and q is the order. Outputs sk and $pk = g^{sk}$.
- The randomized encryption algorithm Enc takes as input a public key pk and a plaintext message $x \in \mathbb{G}$, samples $r \leftarrow \mathbb{Z}_q$, and outputs a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (g^r, x \cdot pk^r)$.
- The partial decryption algorithm ParDec takes as input a secret key share sk and a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, and outputs $\mathsf{ct}' = (\mathsf{ct}_1, \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-sk})$.
- The decryption algorithm Dec takes as input a secret key sk and a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, and outputs $x = \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-sk}$.
- The rerandomization algorithm ReRand takes as input pk and a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, samples $r \leftarrow \mathbb{Z}_q$, and outputs $\mathsf{ct}' = (\mathsf{ct}_1 \cdot g^{r'}, \mathsf{ct}_2 \cdot pk^{r'})$.

E Missing Security Proofs

E.1 The Proof of Theorem 2

In order to facilitate the proof, we assume that $\mathrm{ch}_0 = 1$ and $\mathrm{ch}_1 = m$, and P_1 and P_m provide inputs $b_1 \in \{0, 1\}$ and $b_m \in \{0, 1\}$ respectively. We turn to proving the correctness and security of the protocol in Figure 16. Note that when ch_0 and ch_1 are assigned different values, the proof is essentially the same.

Correctness. From the description of the protocol, we have the following equations:

$$r_1 \oplus (\bigoplus_{j=2}^{m-1} r_j) \oplus r_m = (\bigoplus_{j=2}^{m-1} (r_{j,1}^{b_1} \oplus b_1 \cdot u_{j,1}) \oplus r_{1,m}^0 \oplus r_{m,1}^{b_1} \oplus b_1 \cdot \Delta_1)$$

$$\oplus (r_{j1}^0 \oplus r_{j,m}^0) \oplus (\bigoplus_{j=2}^{m-1} (r_{j,m}^{b_m} \oplus b_m \cdot u_{j,m}) \oplus r_{m,1}^0 \oplus r_{m,1}^{b_m} \oplus b_m \cdot \Delta_m) \tag{1}$$

$$u_{j,1} = \Delta_j \oplus r_{j,1}^0 \oplus r_{j,1}^1, u_{j,m} = \Delta_j \oplus r_{j,m}^0 \oplus r_{j,m}^1 \tag{2}$$

From the definition of Random OT functionality (Figure 6), we have the following equations:

$$r_{j,1}^{b_1} = r_{j,1}^0 \oplus b_1 \cdot (r_{j,1}^0 \oplus r_{j,1}^1), r_{j,m}^{b_m} = r_{j,m}^0 \oplus b_m \cdot (r_{j,m}^0 \oplus r_{j,m}^1), \tag{3}$$

$$r_{m,1}^{b_1} = r_{m,1}^0 \oplus b_1 \cdot (r_{j,1}^0 \oplus r_{j,1}^1) = r_{m,1}^0 \oplus b_1 \cdot \Delta_1, \tag{4}$$

$$r_{1,m}^{b_m} = r_{1,m}^0 \oplus b_m \cdot (r_{1,m}^0 \oplus r_{1,m}^1) = r_{1,m}^0 \oplus b_m \cdot \Delta_m \tag{5}$$

Parameters: m parties $P_1, \cdots, P_m$. The message length l.
Inputs: P_1 has input $b_1 \in \{0,1\}$ and P_m has input $b_m \in \{0,1\}$.
Protocol:

1. P_1 and P_m invoke $\mathcal{F}_{\mathsf{rot}}$ twice: First, P_1 acts as receiver with input b_1 and P_m acts as sender. P_1 receives $r_{m,1}^{b_1} \in \{0,1\}^l$. P_m receives $r_{m,1}^0, r_{m,1}^1 \in \{0,1\}^l$; Second, P_m acts as receiver with input b_m and P_1 acts as sender. P_m receives $r_{1,m}^{b_m} \in \{0,1\}^l$. P_1 receives $r_{1,m}^0, r_{1,m}^1 \in \{0,1\}^l$.
2. For $1 < j < m$: P_1 and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_1 acts as receiver with input b_1 and P_j as sender without input. P_1 receives $r_{j,1}^{b_1} \in \{0,1\}^l$. P_j receives $r_{j,1}^0, r_{j,1}^1 \in \{0,1\}^l$.
3. For $1 < j < m$: P_m and P_j invoke $\mathcal{F}_{\mathsf{rot}}$ where P_m acts as receiver with input b_m and P_j as sender. P_m receives $r_{j,m}^{b_m} \in \{0,1\}^l$. P_j receives $r_{j,m}^0, r_{j,m}^1 \in \{0,1\}^l$.
4. For $1 < j < m$: P_j samples $\Delta_j \leftarrow \{0,1\}^l$ and computes $r_j = r_{j,1}^0 \oplus r_{j,m}^0$. P_j sends $u_{j,1} = \Delta_j \oplus r_{j,1}^0 \oplus r_{j,1}^1$ to P_1, and $u_{j,m} = \Delta_j \oplus r_{j,m}^0 \oplus r_{j,m}^1$ to P_m, then outputs (r_j, Δ_j).
5. P_1 computes $\Delta_1 = r_{1,m}^0 \oplus r_{1,m}^1$ and $r_1 = \bigoplus_{j=2}^{m-1}(r_{j,1}^{b_1} \oplus b_1 \cdot u_{j,1}) \oplus r_{1,m}^0 \oplus r_{m,1}^{b_1} \oplus b_1 \cdot \Delta_1$ ($\cdot$ denotes bitwise-AND between the repetition code of b_1 and $u_{j,1}$, which are both strings of length l. Similarly hereinafter), then outputs (r_1, Δ_1). P_m computes $\Delta_m = r_{m,1}^0 \oplus r_{m,1}^1$ and $r_m = \bigoplus_{j=2}^{m-1}(r_{j,m}^{b_m} \oplus b_m \cdot u_{j,m}) \oplus r_{m,1}^0 \oplus r_{1,m}^{b_m} \oplus b_m \cdot \Delta_m$, then outputs (r_m, Δ_m).

Fig. 16. Multi-Party Secret-Shared Random OT $\Pi_{\mathsf{mss\text{-}rot}}$

Substitute Equation 2, 3, 4, 5 into Equation 1 and cancel out the same terms, we obtain:

$$
\begin{aligned}
&r_1 \oplus (\bigoplus_{j=2}^{m-1} r_j) \oplus r_m \\
&= (b_1 \oplus b_m) \cdot \Delta_1 \oplus (\bigoplus_{j=2}^{m-1} (b_1 \oplus b_m) \cdot \Delta_j) \oplus (b_1 \oplus b_m) \cdot \Delta_m \\
&= \bigoplus_{i=1}^{m} (b_1 \oplus b_m) \cdot \Delta_i
\end{aligned}
$$

Then we can summarize that if $b_1 \oplus b_m = 0$, $r_1 = \bigoplus_{j=2}^{m} r_j$, else $r_1 = \Delta_1 \oplus (\bigoplus_{j=2}^{m} (r_j \oplus \Delta_j))$. This is exactly the functionality $\mathcal{F}_{\mathsf{mss\text{-}rot}}$.
Security. We now prove the security of the protocol.

Proof. Let $\mathcal{C}orr$ denote the set of all corrupted parties and $\mathcal{H}$ denote the set of all honest parties. $|\mathcal{C}orr| = t$.

Intuitively, the protocol is secure because all things the parties do are invoking $\mathcal{F}_{\mathsf{rot}}$ and receiving random messages. The simulator can easily simulate these outputs from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages by generating random values, which are independent of honest parties' inputs.

To elaborate, in the case that $P_1 \notin \mathcal{C}orr$ and $P_m \notin \mathcal{C}orr$, simulator receives all outputs (r_c, Δ_c) of $P_c \in \mathcal{C}orr$ and needs to emulate each P_c's view, including its private Δ_c, outputs $(r_{c,1}^0, r_{c,1}^1)$ and $(r_{c,m}^0, r_{c,m}^1)$ from $\mathcal{F}_{\mathsf{rot}}$. The simulator for corrupted P_c runs the protocol honestly except that it sets P_c's random tape to be the output Δ_c, and simulates uniform outputs from $\mathcal{F}_{\mathsf{rot}}$ under the constraint that $r_{c,1}^0 \oplus r_{c,m}^0 = r_c$. Clearly, the joint distribution of all outputs (r_c, Δ_c) of $P_c \in \mathcal{C}orr$, along with their view emulated by simulator, is indistinguishable from that in the real process.

In the case that $P_1 \in \mathcal{C}orr$ or $P_m \in \mathcal{C}orr$, since the protocol is symmetric with respect to the roles of P_1 and P_m, we focus on the case of corrupted P_1. The simulator receives b_1 in addition to all outputs (r_c, Δ_c) of $P_c \in \mathcal{C}orr$. For P_1, its view consists of its private Δ_1, outputs $r_{m,1}^{b_1}$, $(r_{1,m}^0, r_{1,m}^1)$, $\{r_{j,1}^{b_1}\}_{1<j<m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,1}\}_{1<j<m}$ from P_j. For each $P_c (c \neq 1)$, its view consists of its private Δ_c, outputs $(r_{c,1}^0, r_{c,1}^1)$ and $(r_{c,m}^0, r_{c,m}^1)$ from $\mathcal{F}_{\mathsf{rot}}$.

The simulator sets each corrupted party P_c's random tape to be its output Δ_c. Then for P_1's view, it runs the protocol honestly except that it simulates uniform outputs $r_{m,1}^{b_1}$, $(r_{1,m}^0, r_{1,m}^1)$, $r_{i,1}^{b_1}$ from $\mathcal{F}_{\mathsf{rot}}$ and uniformly random messages $u_{i,1}$ from P_i under the constraint $\bigoplus_{j=2}^{m-1} (r_{j,1}^{b_1} \oplus b_1 \cdot u_{j,1}) \oplus r_{1,m}^0 \oplus r_{m,1}^{b_1} \oplus b_1 \cdot \Delta_1 = r_1$, where $P_i \in \mathcal{H}$. For other corrupted parties' view, it runs the protocol honestly except that it sets $r_{c,m}^0 = r_{c,1}^0 \oplus r_c$ and simulates uniform output $r_{c,m}^1$ from $\mathcal{F}_{\mathsf{rot}}$.

In the real execution, P_1 receives $u_{i,1} = \Delta_i \oplus r_{i,1}^0 \oplus r_{i,1}^1$. From the definition of ROT functionality, $r_{i,1}^0$ (when $b_1 = 0$) or $r_{i,1}^1$(when $b_1 = 0$) is uniform and independent of P_1's view. Therefore, $u_{i,1}$ is uniformly at random from the perspective of P_1. Clearly, the joint distribution of b_1 and all outputs (r_c, Δ_c)

of $P_c \in \mathcal{C}orr$, along with their view emulated by simulator, is indistinguishable from that in the real process.

In the case that $P_1 \in \mathcal{C}orr$ and $P_m \in \mathcal{C}orr$, the simulator receives b_1, b_m and all outputs (r_c, Δ_c) of $P_c \in \mathcal{C}orr$. For P_1, its view consists of its private Δ_1, outputs $r_{m,1}^{b_1}$, $(r_{1,m}^0, r_{1,m}^1)$, $\{r_{j,1}^{b_1}\}_{1<j<m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,1}\}_{1<j<m}$ from P_j. For each $P_c (c \neq 1, c \neq m)$, its view consists of its private Δ_c, outputs $(r_{c,1}^0, r_{c,1}^1)$ and $(r_{c,m}^0, r_{c,m}^1)$ from $\mathcal{F}_{\mathsf{rot}}$. For P_m, its view consists of its private Δ_m, outputs $r_{1,m}^{b_m}$, $(r_{m,1}^0, r_{m,1}^1)$, $\{r_{j,1}^{b_m}\}_{1<j<m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,m}\}_{1<j<m}$ from P_j.

The simulator sets each corrupted party P_c's random tape to be its output Δ_c. Then for P_1's view, it runs the protocol honestly except that it simulates uniform outputs $r_{i,1}^{b_1}$ from $\mathcal{F}_{\mathsf{rot}}$ and uniformly random messages $u_{i,1}$ from P_i under the constraint $\bigoplus_{j=2}^{m-1}(r_{j,1}^{b_1} \oplus b_1 \cdot u_{j,1}) \oplus r_{1,m}^0 \oplus r_{m,1}^{b_1} \oplus b_1 \cdot \Delta_1 = r_1$, where $P_i \in \mathcal{H}$. For the view of $P_c (c \neq 1, c \neq m)$, it runs the protocol honestly except that it sets $r_{c,m}^0 = r_{c,1}^0 \oplus r_c$ and simulates uniform output $r_{c,m}^1$ from $\mathcal{F}_{\mathsf{rot}}$. For P_m's view, it runs the protocol honestly with the following changes:

- It simulates uniform outputs $r_{i,m}^{b_m}$ from $\mathcal{F}_{\mathsf{rot}}$ and uniform messages $u_{i,m}$ from P_i under the constraint $\bigoplus_{j=2}^{m-1}(r_{j,m}^{b_m} \oplus b_m \cdot u_{j,m}) \oplus r_{m,1}^0 \oplus r_{1,m}^{b_m} \oplus b_m \cdot \Delta_m = r_m$, where $P_i \in \mathcal{H}$.
- It sets the output $r_{c,m}^{b_m}$ from $\mathcal{F}_{\mathsf{rot}}$ to be consistent with the partial view $(r_{c,m}^0, r_{c,m}^1)$ of each corrupted P_c in preceding simulation, where $c \neq 1$ and $c \neq m$.

Clearly, the joint distribution of b_1, b_m and all outputs (r_c, Δ_c) of $P_c \in \mathcal{C}orr$, along with their view emulated by simulator, is indistinguishable from that in the real process.

E.2 The Proof of Theorem 3

Proof. This proof is supposed to be divided into two cases in terms of whether $P_1 \in \mathcal{C}orr$, since this determines whether the adversary has knowledge of the output. Nevertheless, the simulation of these two cases merely differ in the output reconstruction stage, thus we combine them together for the sake of simplicity. Specifically, the simulator receives the input X_c of $P_c \in \mathcal{C}orr$ and the output $\bigcup_{i=1}^m X_i$ if $P_1 \in \mathcal{C}orr$.

For each P_c, its view consists of its input X_c, outputs from $\mathcal{F}_{\mathsf{bssPMT}}$, $\mathcal{F}_{\mathsf{mss\text{-}rot}}$, output $\vec{sh}_c'$ from $\mathcal{F}_{\mathsf{ms}}$ as its share, and $m-1$ sets of shares $\{\vec{sh}_i'\}_{1<i\leq m}$($P_i$'s output from $\mathcal{F}_{\mathsf{ms}}$) from P_i if $c=1$. The simulator emulates each P_c's view by running the protocol honestly with the following changes:

- In step 2, it simulates uniform outputs $\{e_{c,j}^b\}_{c<j\leq m}$ and $\{e_{c,i}^b\}_{1\leq i<c}$ from $\mathcal{F}_{\mathsf{bssPMT}}$, on condition that $P_i, P_j \in \mathcal{H}$.
- In step 3, it simulates uniform outputs $\{r_{c,ji}^b\}_{c<j\leq m, 1\leq i<j}, \{\Delta_{c,ji}^b\}_{c<j\leq m, 1\leq i<j}$ from $\mathcal{F}_{\mathsf{mss\text{-}rot}}$, on condition that $\exists 1 \leq d \leq j, P_d \in \mathcal{H}$.

– In step 4, it simulates uniformly output $\vec{sh}_c'$ from $\mathcal{F}_{\mathsf{ms}}$.

Now we discuss the case when $P_1 \in \mathcal{C}orr$. In step 4 and 5, it computes $Y = \bigcup_{i=1}^m X_i \setminus X_1$ and constructs $\vec{v} \in (\{0,1\}^{l+\kappa})^{(m-1)B}$ as follows:

– For $\forall x_i \in Y$, $v_i = x_i \| \mathsf{H}(x)$, $1 \le i \le |Y|$.
– For $|Y| < i \le (m-1)B$, samples $v_i \leftarrow \{0,1\}^{l+\kappa}$.

Then it samples a random permutation $\pi : [(m-1)B] \rightarrow [(m-1)B]$ and computes $\vec{v'} = \pi(\vec{v})$. For $1 \le i \le m$, it samples share $\vec{sh}_i' \leftarrow (\{0,1\}^{l+\kappa})^{(m-1)B}$, which satisfies $\bigoplus_{i=1}^m \vec{sh}_i' = \vec{v'}$ and is consistent with the previous sampled $\vec{sh}_c'$ for each corrupted P_c. Add all $\vec{sh}_i'$ to P_1's view and $\vec{sh}_{c'}'$ to each corrupted $P_{c'}$'s view($c' \neq 1$) as its output from $\mathcal{F}_{\mathsf{ms}}$, respectively.

The changes of outputs from $\mathcal{F}_{\mathsf{bssPMT}}$ and $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ have no impact on P_c's view, for the following reasons. By the definition of $\mathcal{F}_{\mathsf{bssPMT}}$, each output $e_{c,j}^b$ and $e_{c,i}^b$ from $\mathcal{F}_{\mathsf{bssPMT}}$ is uniformly distributed as a secret-share between P_c and P_j, or P_i and P_c, where $P_i, P_j \in \mathcal{H}$. By the definition of $\mathcal{F}_{\mathsf{mss\text{-}rot}}$, each output $r_{c,ji}^b$ from $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ is a secret-share of 0 among $P_1, \cdots, P_j$ if $e_{i,j}^b \oplus e_{j,i}^b = 0$, or a secret-share of $\bigoplus_{d=1}^j \Delta_{d,ji}^b$ if $e_{i,j}^b \oplus e_{j,i}^b = 1$. Therefore, even if P_c colludes with others, $r_{c,ji}^b$ is still uniformly random from the perspective of adversary, since there always exists a party $P_d \in \mathcal{H}(1 \le d \le j)$ holding one share. Besides, the outputs $\{\Delta_{c,ji}^b\}_{c<j\le m, 1\le i<j}$ from $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ are uniformly distributed.

It remains to demonstrate that the output $\vec{sh}_c'$ from $\mathcal{F}_{\mathsf{ms}}(P_1 \notin \mathcal{C}orr)$ or all outputs $\{\vec{sh}_i'\}_{1\le i\le m}$ from $\mathcal{F}_{\mathsf{ms}}(P_1 \in \mathcal{C}orr)$ does not leak any other information except for the union. The former case is easier to tackle with. The output $\vec{sh}_c'$ is distributed as a secret-share among all parties, so it is uniformly distributed from the perspective of adversary.

We now proceed to explain the latter case. For all $1 < j \le m$, consider an element $x \in X_j$ and x is placed in the bth bin by P_j. In the real protocol, if there is no $X_i(1 \le i < j)$ s.t. $x \in X_i$, then for all $1 \le i < j$, $e_{i,j}^y \oplus e_{j,i}^y = 0$. By the $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ functionality in Figure 10, each $r_{d,ji}^b$ is uniform in $\{0,1\}^{l+\kappa}$ conditioned on $\bigoplus_{d=1}^j r_{d,ji}^b = 0$. From the descriptions of the protocol, We derive that each $u_{d,j}^b$ is uniform in $\{0,1\}^{l+\kappa}$ conditioned on $\bigoplus_{d=1}^j u_{d,j}^b = x \| \mathsf{H}(x)$, namely, they are additive shares of $x \| \mathsf{H}(x)$ among all parties. This is exactly identical to the simulation.

If there exists some $X_i(1 \le i < j)$ s.t. $x \in X_i$, then $e_{i,j}^b \oplus e_{j,i}^b = 1$. By the $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ functionality in Figure 10, each $r_{d,ji}^b$ is uniform conditioned on $\bigoplus_{d=1}^j r_{d,ji}^b = \bigoplus_{d=1}^j \Delta_{d,ji}^b$, where each $\Delta_{d,ji}^b$ is uniformly held by P_d. From the descriptions of the protocol, We derive that each $u_{d,j}^b$ is uniform conditioned on $\bigoplus_{d=1}^j u_{d,j}^b = x \| \mathsf{H}(x) \oplus \bigoplus_{d=1}^j \Delta_{d,ji}^b \oplus r$, where r is the sum of remaining terms. Then, even if P_1 colludes with others, $\bigoplus_{d=1}^j u_{d,j}^b$ is still uniformly random from the perspective of adversary, since there always exists a party $P_d \in \mathcal{H}(1 \le d \le j)$ holding one uniform $\Delta_{d,ji}^b$ and independent of all honest parties' inputs. For all empty bins, $u^b_{d,j}$ is chosen uniformly random, so the corresponding $\bigoplus_{d=1}^{j} u^b_{d,j}$ is also uniformly at random, which is identical to the simulation. By the definition of $\mathcal{F}_{\mathsf{ms}}$, all parties additively share $\bigoplus_{d=1}^{j} u^b_{d,j}$ in a random permutation that maintains privacy against a coalition of arbitrary corrupted parties, and receive back $\{\vec{sh}'_i\}$, respectively. We conclude that all outputs $\{\vec{sh}'_i\}_{1\leq i\leq m}$ from $\mathcal{F}_{\mathsf{ms}}$ distribute identically between the real and ideal executions.

E.3 The Proof of Theorem 4

Proof. The simulator receives the input X_c of $P_c \in \mathcal{C}orr$ and the output $\bigcup_{i=1}^{m} X_i$ if $P_1 \in \mathcal{C}orr$.

For each P_c, its view consists of its input X_c, outputs from $\mathcal{F}_{\mathsf{bssPMT}}$ and $\mathcal{F}_{\mathsf{rot}}$, protocol messages $\{u^b_{j,c,0}\}_{c<j\leq m}, \{u^b_{j,c,1}\}_{c<j\leq m}$ from P_j, rerandomization messages $\{v'^b_{c,i}\}_{1<i<c}$ from P_i, π_c, permuted partial decryption messages $\vec{\mathsf{ct}}''_{c-1}$ from P_{c-1} if $c > 1$, or $\vec{\mathsf{ct}}''_m$ from P_m if $c = 1$. The simulator emulates each P_c's view by running the protocol honestly with the following changes:

- In step 2, it simulates uniform outputs $\{e^b_{c,j}\}_{c<j\leq m}$ and $\{e^b_{c,i}\}_{1\leq i<c}$ from $\mathcal{F}_{\mathsf{bssPMT}}$, on condition that $P_i, P_j \in \mathcal{H}$.
- In step 3, it simulates uniform outputs $\{r^b_{c,i,0}\}_{1\leq i<c}$, $\{r^b_{c,i,1}\}_{1\leq i<c}$ from $\mathcal{F}_{\mathsf{rot}}$, and $\{r^b_{j,c,e^b_{c,j}}\}_{c<j\leq m}$ from $\mathcal{F}_{\mathsf{rot}}$, on condition that $P_i, P_j \in \mathcal{H}$. For $c < j \leq m$, it computes $u^b_{j,c,e^b_{c,j}} = r^b_{j,c,e^b_{c,j}} \oplus \mathsf{Enc}(pk, \perp)$ and simulates $u^b_{j,c,e^b_{c,j}\oplus 1}$ uniformly at random, on condition that $P_j \in \mathcal{H}$. For $1 < i < c$, it simulates $v'^b_{c,i} = \mathsf{Enc}(pk, \perp)$, on condition that $P_i \in \mathcal{H}$.
- If $P_1 \notin \mathcal{C}orr$, in step 4, for $1 \leq i \leq (m-1)B$, it computes $\mathsf{ct}'^i_{c-1} = \mathsf{Enc}(pk, \perp$), and then simulates the vector $\vec{\mathsf{ct}}''_{c-1} = \pi(\vec{\mathsf{ct}}'_{c-1})$ from P_{c-1}, where π is sampled uniformly random and $P_{c-1} \in \mathcal{H}$.

Now we discuss the case when $P_1 \in \mathcal{C}orr$. In step 4, assume d is the largest number that $P_d \in \mathcal{H}$, namely, $P_{d+1}, \cdots, P_m \in \mathcal{C}orr$. The simulator emulates the partial decryption messages $\vec{\mathsf{ct}}''_d$ from P_d in the view of P_{d+1} as follows:

- For $\forall x_i \in Y = \bigcup_{j=1}^{m} X_j$, $\mathsf{ct}'^i_d = \mathsf{Enc}(pk_A, x_i)$, $1 \leq i \leq |Y|$.
- For $|Y| < i \leq (m-1)B$, sets $\mathsf{ct}'^i_d = \mathsf{Enc}(pk_A, \perp)$.

where $pk_{A_d} = pk_1 \cdot \prod_{j=d+1}^{m} pk_j$. Then it samples a random permutation $\pi : [(m-1)B] \to [(m-1)B]$ and computes $\vec{\mathsf{ct}}''_d = \pi(\vec{\mathsf{ct}}'_d)$.

For other corrupted $P_{d'+1} \in \{P_2, \cdots, P_{d-1}\}$, if $P_{d'} \in \mathcal{H}$, it simulates each partial decryption message $\mathsf{ct}'^i_{d'} = \mathsf{Enc}(pk, \perp)$ for $1 \leq i \leq (m-1)B$, and then computes the vector $\vec{\mathsf{ct}}''_{d'} = \pi(\vec{\mathsf{ct}}'_{d'})$ from $P_{d'}$, where $\pi_{d'}$ is sampled uniformly random. Append $\vec{\mathsf{ct}}''_{d'}$ to the view of $P_{d'+1}$.

The changes of outputs from $\mathcal{F}_{\mathsf{bssPMT}}$ and $\mathcal{F}_{\mathsf{rot}}$ have no impact on P_c's view, for similar reasons in Theorem 3.

Indeed, $u^b_{j,c,e^b_{c,j}\oplus 1}$ is uniform in the real process, as $r^b_{j,c,e^b_{c,j}\oplus 1}$ (which is one of P_j's output from $\mathcal{F}_{\mathsf{rot}}$ hidden from P_c, and is used to mask the encrypted message in $u^b_{j,c,e^b_{c,j}\oplus 1}$) is uniform and independent of $r^b_{j,c,e^b_{c,j}}$ from P_c's perspective.

It's evident from the descriptions of the protocol and the simulation that the simulated $u^b_{j,c,e^b_{c,j}}$ is identically distributed to that in the real process, conditioned on the event $e^b_{c,j} \oplus e^b_{j,c} = 1$. The analysis in the case of $e^b_{c,j} \oplus e^b_{j,c} = 0$ can be further divided into two subcases, $c \neq 1$ and $c = 1$. We first argue that when $c \neq 1$, $u^b_{j,c,e^b_{c,j}}$ emulated by simulator is indistinguishable from that in the real process.

In the real process, for $1 < c < j \leq m, 1 \leq b \leq B$, if $\mathsf{Elem}(\mathcal{C}^b_j) \in X_j \setminus (X_2 \cup \cdots \cup X_{c-1})$, $c^b_j = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}^b_j))$, $u^b_{j,c,e^b_{c,j}} = r^b_{j,c,e^b_{c,j}} \oplus \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}^b_j))$; else $c^b_j = \mathsf{Enc}(pk, \perp)$, $u^b_{j,c,e^b_{c,j}} = r^b_{j,c,e^b_{c,j}} \oplus \mathsf{Enc}(pk, \perp)$. In the real process, for $1 < c < j \leq m, 1 \leq b \leq B$, $u^b_{j,c,e^b_{c,j}} = r^b_{j,c,e^b_{c,j}} \oplus \mathsf{Enc}(pk, \perp)$. If there exists an algorithm that distinguishes these two process, it implies the existence of an algorithm that can distinguish two lists of encrypted messages, with no knowledge of sk (since sk is secret-shared among m parties, it is uniformly distributed for any coalition of $m-1$ parties). Consequently, this implies the existence of a adversary to break the indistinguishable multiple encryptions of $\mathcal{E}$ in Definition 2 (where $q = (m-1)B$).

When $c = 1$, $u^b_{j,e^b_{1,j}}$ emulated by simulator is indistinguishable from that in the real process for the similar reason as the above analysis when $c > 1$.

Next, we start demonstrating that all $v'^b_{c,i} = \mathsf{Enc}(pk, \perp)$ emulated by simulator are indistinguishable from the real ones via the sequences of hybrids:

- Hyb_0 The real interaction. For $1 < i < c, 1 \leq b \leq B$: If $\mathsf{Elem}(\mathcal{C}^b_c) \in X_c \setminus (X_1 \cup \cdots \cup X_i)$, $v^b_{c,i} = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}^b_c))$; else $v^b_{c,i} = \mathsf{Enc}(pk, \perp)$. $v'^b_{c,i} = \mathsf{ReRand}(pk, v^b_{c,i})$.
- Hyb_1 For $1 < i < c, 1 \leq b \leq B$: If $\mathsf{Elem}(\mathcal{C}^b_c) \in X_c \setminus (X_1 \cup \cdots \cup X_i)$, $v'^b_{c,i} = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}^b_c))$; else $v'^b_{c,i} = \mathsf{Enc}(pk, \perp)$. This change is indistinguishable by the rerandomizable property of $\mathcal{E}$.
- Hyb_2 For $1 < i < c, 1 \leq b \leq B$: $v'^b_{c,i} = \mathsf{Enc}(pk, \perp)$. This change is indistinguishable by the indistinguishable multiple encryptions of $\mathcal{E}$.

When $P_1 \in \mathcal{C}orr$, we prove that $\vec{\mathsf{ct}}''_d$ emulated by simulator is indistinguishable from that in the real process via the sequences of hybrids:

- Hyb_0 The real interaction. $\vec{\mathsf{ct}}''_1 = \pi_1(\vec{\mathsf{ct}}'_1)$. For $2 \leq j \leq d, 1 \leq i \leq (m-1)B$: $\mathsf{ct}^i_j = \mathsf{ParDec}(sk_j, \mathsf{ct}''^i_{j-1}), \mathsf{ct}'^i_j = \mathsf{ReRand}(pk_{A_j}, \mathsf{ct}^i_j)$, $\vec{\mathsf{ct}}''_j = \pi_j(\vec{\mathsf{ct}}'_j)$.
- Hyb_1 For $2 \leq j \leq d, 1 \leq i \leq (m-1)B$: $\mathsf{ct}^i_j = \mathsf{ParDec}(sk_j, \mathsf{ct}^i_{j-1})$. $\vec{\mathsf{ct}}'_d = \mathsf{ReRand}(pk_{A_d}, \vec{\mathsf{ct}}_d)$, $\vec{\mathsf{ct}}''_d = \pi(\vec{\mathsf{ct}}'_d)$, where $\pi = \pi_1 \circ \cdots \circ \pi_d$. It's easy to see that Hyb_1 is identical to Hyb_0.
- Hyb_2 $\vec{\mathsf{ct}}_1$ is replaced by the following:
  - For $\forall x_i \in Y = \bigcup_{j=1}^m X_j$, $\mathsf{ct}^i_1 = \mathsf{Enc}(pk, x_i)$, $1 \leq i \leq |Y|$.

  - For $|Y| < i \leq (m-1)B$, sets $\mathsf{ct}_1^i = \mathsf{Enc}(pk, \perp)$.

  Hyb_2 rearranges $\vec{\mathsf{ct}}_1$ and it is identical to Hyb_1 as the adversary is unaware of π_d s.t. the order of elements in $\vec{\mathsf{ct}}_1$ has no effect on the result of $\vec{\mathsf{ct}}_d''$.
- Hyb_3 $\vec{\mathsf{ct}}_d$ is replaced by the following:
  - For $\forall x_i \in Y = \bigcup_{j=1}^{m} X_j$, $\mathsf{ct}_d^i = \mathsf{Enc}(pk_{A_d}, x_i)$, $1 \leq i \leq |Y|$.
  - For $|Y| < i \leq (m-1)B$, sets $\mathsf{ct}_d^i = \mathsf{Enc}(pk_{A_d}, \perp)$.

  The indistinguishability between Hyb_3 and Hyb_2 is implied by the partially decryptable property of $\mathcal{E}$.
- Hyb_4 $\vec{\mathsf{ct}}_d'$ is replaced by the following:
  - For $\forall x_i \in Y = \bigcup_{j=1}^{m} X_j$, $\mathsf{ct}_d'^i = \mathsf{Enc}(pk_{A_d}, x_i)$, $1 \leq i \leq |Y|$.
  - For $|Y| < i \leq (m-1)B$, sets $\mathsf{ct}_d'^i = \mathsf{Enc}(pk_{A_d}, \perp)$.

  Hyb_4 is indistinguishable to Hyb_3 because of the rerandomizable property of $\mathcal{E}$.
- Hyb_5 The only change in Hyb_5 is that π are sampled uniformly by the simulator. Hyb_5 generates the same $\vec{\mathsf{ct}}_d''$ as in the simulation. Given that π_d is uniform in the adversary's perspective, the same holds for π, so Hyb_5 is identical to Hyb_4.

When $P_1 \notin \mathcal{C}orr$, the simulator is unaware of the final union, so it has to emulate the partial decryption messages $\vec{\mathsf{ct}}_{c-1}''$ as permuted $\mathsf{Enc}(pk, \perp)$ in the view of P_c. Compared to the above hybrid argument, we only need to add one additional hybrid after Hyb_4 to replace all rerandomized partial decryption messages $\vec{\mathsf{ct}}_{c-1}'$ with $\mathsf{Enc}(pk, \perp)$. This change is indistinguishable by the indistinguishable multiple encryptions of $\mathcal{E}$, as the adversary cannot distinguish two lists of messages with no knowledge of the partial secret key $sk_{A_{c-1}} = sk_1 + sk_c + \cdots + sk_m$ (it is unaware of sk_1).

The same applies for the simulation of the partial decryption messages $\vec{\mathsf{ct}}_{d'}''$ in the view of corrupted $P_{d'+1}$ ($d' \neq d$) when $P_1 \in \mathcal{C}orr$. $\vec{\mathsf{ct}}_{d'}''$ is also emulated by permuted $\mathsf{Enc}(pk, \perp)$. To avoid repetition, we omit the analysis here.

F Complete Complexity Analyses of Protocols

F.1 Theoretical Analysis of Protocol $\varPi_{\mathsf{bssPMT}}$

The costs of each stage in $\varPi_{\mathsf{bssPMT}}$ (Figure 9) are calculated as follows.

Batch OPPRF. The cost of batch OPPRF mainly consists of two parts:

- Batch OPRF: There are several options for instantiating batch OPRF functionality [KKRT16, BC23]. We opt for subfield vector oblivious linear evaluation (subfield-VOLE) [BCG+19a, BCG+19b, RRT23] to instantiate batch OPRF using the approach in [BC23].

  For $B = O(n)$ instances of OPRF, we require performing a subfield-VOLE of size B in the offline phase, and sending B derandomization messages of length l in the online phase. We resort to *Dual LPN with Fixed Weight*

and Regular Noise [BCG+19a, BCG+19b] to improve efficiency of subfield-VOLE, and instantiate the code family with Expand-Convolute codes [RRT23], which enables near-linear time syndrome computation. The computation complexity is $O(n \log n)$. The computation complexity is $O(t\lambda \log n/t)$, where t is the fixed noise weight[24]. The round complexity is $O(1)$.
We denote the output length of OPPRF as γ. The lower bound of γ is relevant to the total number of the batch OPPRF invocations. In our SK-MPSU and PK-MPSU protocols, for $1 \leq i < j \leq m$, P_i and P_j invoke batch OPPRF. Overall, there are $1 + 2 + \cdots + (m-1) = (m^2 - m)/2$ invocations of batch OPPRF. Considering all these invocations, we set $\gamma \geq \sigma + \log((m^2 - m)/2) + \log_2 B$, so that the probability of any $t_i \neq s_i$ occurring if $x \notin X_i$, which is $((m^2 - m)/2))B \cdot 2^{-\gamma}$, is less than or equal to $2^{-\sigma}$.
- OKVS: The size of key-value pairs encoded into OKVS is $|K_1| + \cdots + |K_B|$. Taking into account the invocation of batch ssPMT in our two MPSU protocols in advance, this size is $3n$. We use the OKVS construction of [RR22], and the computation complexities of $\mathsf{Encode_H}$ and $\mathsf{Decode_H}$ algorithms are both $O(n)$. As we employ their $w = 3$ scheme with a cluster size of 2^{14}, the size of OKVS is $1.28\gamma \cdot (3n)$ bits.

- **Offline phase.** The computation complexity of each party is $O(n \log n)$. The communication complexity of each party is $O(t\lambda \log(n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(n)$. The communication complexity of each party is $O(\gamma n)$. The round complexity is $O(1)$.

Secret-shared private equality tests. Like [PSTY19], we instantiate ssPEQT (Figure 5) using the generic MPC techniques. The circuit of $\mathcal{F}_{\mathsf{ssPEQT}}$ is composed of $\gamma - 1$ AND gates in GMW [GMW87], where the inputs are already in the form of secret-shaing. Executing $\gamma - 1$ AND gates in sequence would incur $\gamma - 1$ rounds. To reduce the round complexity, we leverage a divide-and-conquer strategy and recursively organize the AND gates within a binary tree structure, where each layer requires one round. This ensures that the number of rounds is directly related to the depth of the tree (i.e., $O(\log \gamma)$). To sum up, each party invoke B instances of ssPEQT, which amounts to $(\gamma - 1)B$ AND gates and takes $O(\log \gamma)$ rounds.

We use silent OT extension [BCG+19a] to generate Beaver triples in offline phase, then each AND gate only requires 4 bits communication and $O(1)$ computation in the online phase. As a result, an invocation of ssPEQT requires 4γ bits communication and $O(1)$ computation in the online phase[25].

[24] For instance, $t \approx 128$.

[25] When calculating computational complexity, one evaluation on PRG or one hash operation is usually considered as $O(1)$ operation. However, here, the computational unit is bitwise XOR operation, and it's evident that performing $O(\gamma)$ bitwise XOR operations is much faster than executing $O(1)$ PRG evaluation or hash operation. Therefore, we count the online computation complexity of ssPEQT as $O(1)$

– **Offline phase.** The computation complexity of each party is $O(\gamma n \log n)$. The communication complexity of each party is $O(t\lambda \log(\gamma n/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity of each party is $O(n)$. The communication complexity of each party is $O(\gamma n)$. The round complexity is $O(\log \gamma)$.

F.2 Theoretical Analysis of Protocol $\Pi_{\mathsf{SK\text{-}MPSU}}$

The costs of each stage in $\Pi_{\mathsf{SK\text{-}MPSU}}$ (Figure 12) are calculated as follows.

Batch ssPMT. To achieve linear communication of this stage, we use stash-less Cuckoo hashing [PSTY19]. To render the failure probability (failure is defined as the event where an item cannot be stored in the table and must be stored in the stash) less than 2^{-40}, we set $B = 1.27n = O(n)$ for 3-hash Cuckoo hashing. The cost of batch ssPMT follows the complexity analysis in Section 4. For $1 \leq i < j \leq m$, P_i and P_j invoke batch ssPMT. Overall, each party P_j engages in $m-1$ invocations of batch ssPMT, acting as $\mathcal{R}$ in the first $j-1$ invocations, and acting as $\mathcal{S}$ in the last $m-j$ invocations.

– **Offline phase.** The computation complexity of each party is $O(\gamma m n \log n)$. The communication complexity of each party is $O(t\lambda m \log(\gamma n/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity of each party is $O(mn)$. The communication complexity of each party is $O(\gamma m n)$. The round complexity is $O(\log \gamma)$.

Multi-party secret-shared random oblivious transfers. Each invocation of mss-ROT involves pairwise executions of two-party OT. If P_i and P_j hold the two choice bits, then P_i and P_j invoke two instances of OT and they separately invoke one instance of OT with the remaining parties.

Each OT execution consists of two parts: In the offline phase, the parties engages in a random-choice-bit ROT, then $\mathcal{S}$ sends one messages of length $l+\kappa$ ($\kappa = \sigma + \log(m-1) + \log n$) to $\mathcal{R}$[26]; In the online phase, $\mathcal{R}$ sends a 1-bit derandomization message to transform ROT into a chosen-input version.

For $1 \leq i < j \leq m$, $P_1, \cdots, P_j$ engage in B instances of mss-ROT, where P_i and P_j hold the two choice bits. Considering one instance of mss-ROT and fixing j, there are overall $j-1$ invocations of mss-ROT. For $1 \leq d < j$, each P_d has one chance to act as the choice-bit-holder (who invokes two instances of OT with another and one with $j-2$ remaining parties separately in one invocation of mss-ROT) and acts as a nomal party (who invokes one instances of OT with the two choice-bit-holders separately in one invocation of mss-ROT) in the rest

[26] The OT execution between two choice-bit-holder does not include the sending step. This does not affect the total complexity, so we ignore this.

$j-2$ invocations. P_j act as the choice-bit-holder in $j-1$ invocations (who invokes $2+(j-2)=j$ instances of OT).

Overall, each party $P_{j'}$ invokes $[j'(j'-1)+\sum_{j=j'+1}^{m}(j+2(j-2))]B = (\frac{3m^2-j'^2-5m+3j'}{2})B$ instances of two-party OT.

- **Offline phase.** The computation complexity of each party is $O(m^2 n \log n)$. The communication complexity of each party is $O(t\lambda m^2 \log(n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(m^2 n)$. The communication complexity of each party is $O(m^2 n)$. The round complexity is $O(1)$.

Multi-party secret-shared shuffle. We use the construction in [EB22]. In the offline phase, each pair of parties runs a Share Translation protocol [CGP20] of size $(m-1)B$ and length $l+\kappa$.

- **Offline phase.** The computation complexity of each party is $O(m^2 n \log(mn))$. The communication complexity of each party is $O(\lambda m^2 n \log(mn))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of P_1 is $O(m^2 n)$. The communication complexity of P_1 is $O((l+\kappa)m^2 n)$. The computation complexity of P_j is $O(mn)$. The communication complexity of P_j is $O((l+\kappa)mn)$. The round complexity is $O(m)$.

Output reconstruction. For $1 < j \le m$, P_j sends $\vec{sh}_j' \in (\{0,1\}^{l+\kappa})^{(m-1)B}$ to P_1. P_1 reconstructs $(m-1)B$ secrets, each having m shares.

- **Online phase.** The computation complexity of P_1 is $O(m^2 n)$. The communication complexity of P_1 is $O((l+\kappa)m^2 n)$. The communication complexity of P_j is $O((l+\kappa)mn)$. The round complexity is $O(1)$.

Total costs.

- **Offline phase.** The offline computation complexity per party is $O(\gamma mn \log n + m^2 n(\log m + \log n))$. The offline communication complexity per party is $O(t\lambda m \log(\gamma n/t) + t\lambda m^2 \log(n/t) + \lambda m^2 n(\log m + \log n))$. The offline round complexity is $O(1)$.
- **Online phase.** The online computation complexity per party is $O(\gamma mn + m^2 n)$. The online communication complexity of P_1 is $O(\gamma mn + (l+\kappa)m^2 n)$. The online communication complexity of P_j is $O(\gamma mn + m^2 n + (l+\kappa)mn)$. The online round complexity is $O(\log \gamma + m)$.

F.3 Theoretical Analysis of Protocol $\Pi_{\mathsf{PK\text{-}MPSU}}$

The costs of each stage in $\Pi_{\mathsf{PK\text{-}MPSU}}$ (Figure 13) are calculated as follows.

Batch ssPMT The cost of this stage is the same as that in $\Pi_{\mathsf{SK\text{-}MPSU}}$ (cf. Appendix F.3).

- **Offline phase.** The computation complexity of each party is $O(\gamma mn \log n)$. The communication complexity of each party is $O(t\lambda m \log(\gamma n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(mn)$. The communication complexity of each party is $O(\gamma mn)$. The round complexity is $O(\log \gamma)$.

Random oblivious transfers and messages rerandomization. We use EC ElGamal encryption to instantiate the MKR-PKE scheme. So each of encryption, partial decryption and rerandomization takes one point scalar operation. The length of ciphertext is 4λ.

For $1 \leq i < j \leq m$, P_i and P_j invoke B instances of silent ROT correlations with random inputs during the offline phase. In the online phase, P_i sends 1-bit derandomization message to transform each ROT into a chosen-input version. For each OT correlation, each P_j executes two encryptions and sends two 4λ-bit messages to P_i. P_i executes one rerandomization. If $i \neq 1$, P_i sends one ciphertext to P_j and P_j executes one rerandomization as well.

- **Offline phase.** The computation complexity of each party is $O(mn \log n)$. The communication complexity of each party is $O(t\lambda m \log(n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(mn)$ public-key operations. The communication complexity of each party is $O(\lambda mn)$. The round complexity is $O(1)$.

Messages decryptions and shufflings. Each party shuffles $(m-1)B$ ciphertexts and executes $(m-1)B$ partial decryptions before sending them to the next party.

- **Online phase.** The computation complexity of each party is $O(mn)$ public-key operations. The communication complexity of each party is $O(\lambda mn)$. The round complexity is $O(m)$.

Total costs.

- **Offline phase.** The computation complexity of each party is $O(\gamma mn \log n)$. The communication complexity of each party is $O(t\lambda m \log(\gamma n/t)$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(mn)$ symmetric-key operations and $O(mn)$ public-key operations. The communication complexity of each party is $O((\gamma+\lambda)mn)$. The round complexity is $O(\log \gamma + m)$.

F.4 The Complete Analysis of LG Protocol

The costs of each stage in LG protocol are calculated as follows.

mq-ssPMT. The cost of this stage mainly consists of three parts:

- SKE encryption: The computation complexity of encrypting $0, 1, \cdots, n-1$ is $O(n)$.
- OKVS: The computation complexities of Encode and Decode algorithms are both $O(n)$. The size of OKVS is $1.28\kappa n$ bits, where κ is the size of a SKE ciphertext with the same range as before.
- ssVODM: The ssVODM protocol requires a total of $(T+l-\log n-1)n$ AND gates, where T is the number of AND gates in the SKE decryption circuit.

For $1 \leq i < j \leq m$, P_i and P_j invoke mq-ssPMT of size B. Overall, each party P_j engages in $(m-1)$ instances of mq-ssPMT.

- **Offline phase.** The computation complexity of each party is $O((T+l-\log n)mn\log((T+l-\log n)n))$. The communication complexity of each party is $O(t\lambda m\log(((T+l-\log n)n)/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O((T+l-\log n)mn)$. The communication complexity of each party is $O((T+l+\kappa-\log n)mn)$. The round complexity is $O(\log(l-\log n))$.

Random oblivious transfers. For $1 \leq i < j \leq m$, P_i and P_j invoke ROT extension of size B. Overall, each party P_j engages in $(m-1)$ independent instances of ROT extension of size B.

- **Offline phase.** The computation complexity of each party is $O(mn\log n)$. The communication complexity of each party is $O(t\lambda m\log(n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of each party is $O(mn)$. The communication complexity of each party is $O(mn)$. The round complexity is $O(1)$.

The costs of the remaining two steps, **multi-party secret-shared shuffle** and **output reconstruction**, are exactly the same as those in our SK-MPSU protocol.(cf. Appendix F.2).

Total costs.

- **Offline phase.** The computation complexity of each party is $O((T+l-\log n)mn\log((T+l-\log n)n)+m^2n\log mn)$. The communication complexity of each party is $O(t\lambda m\log(((T+l-\log n)n)/t)+\lambda m^2n(\log m+\log n))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity of P_1 is $O((T+l-\log n)mn+m^2n)$. The communication complexity of P_1 is $O((T+l+\kappa-\log n)mn+(l+\kappa)m^2n)$. The computation complexity of P_j is $O((T+l-\log n)mn)$. The communication complexity of P_j is $O((T+l+\kappa-\log n)mn)$. The round complexity is $O(\log(l-\log n)+m)$.

G Multi-Party Private-ID

We generalize the two-party private-ID [BKM+20] to multi-party setting. Suppose there are m parties, each possessing a set of n elements. The multi-party private-ID functionality assigns a unique random identifier to each element across all input sets, ensuring that identical elements in different sets obtain the same identifier. Each party receives identifiers associated with its own input set, as well as identifiers associated with the union of all parties' input sets. With multi-party private-ID, the parties can sort their private sets based on a global set of identifiers and perform desired private computations item by item, ensuring alignment of identical elements across their sets. The formal definition of the multi-party private-ID is depicted in Figure 17. We build a concrete multi-party private-ID protocol based on the DDH assumption (described in Figure 18) by extending the "distributed OPRF+PSU" paradigm [CZZ+24b] to multi-party setting.

Parameters. m parties $P_1, \cdots P_m$. Size n of input sets. The bit length l of set elements. The range D of identifiers.
Functionality. On input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$ from P_i,

- For every element $x \in \bigcup_{i=1}^m X_i$, choose a random identifier $R(x) \leftarrow D$.
- Define $R^* = \{R(x) | x \in \bigcup_{i=1}^m X_i\}$ and $R_i = \{R(x) | x \in X_i\}$ for $i \in [m]$.
- Give output (R^*, R_i) to P_i.

Fig. 17. Multi-Party Private ID Functionality $\mathcal{F}_{\mathsf{MPID}}$

Correctness. The first two steps essentially realize a multi-party distributed OPRF protocol, where each party P_i inputs a set $\{x_i^1, \cdots, x_i^n\}$ and receives its own PRF key k_i and the PRF values computed on its input set using all parties' keys $k_1, \cdots, k_m$, denoted as $\{\mathsf{PRF}_{k_1,\cdots,k_m}(x_i^1), \cdots, \mathsf{PRF}_{k_1,\cdots,k_m}(x_i^n)\}$. In this case, $\mathsf{PRF}_{k_1,\cdots,k_m}(x) = \mathsf{H}(x)^{k_1 \cdots k_m}$. Note that even if $m-1$ parties collude, there exist one exponent private to the adversary, ensuring that the PRF values remain pseudorandom (The proof is analogous to that of the DH-based OPRF, so we omit it here).
Security. Assuming H is a random oracle, the security of protocol follows immediately from the DDH assumption and the security of our PK-MPSU protocol.

Parameters. m parties $P_1, \cdots P_m$. Size n of input sets. The bit length l of set elements. A cyclic group $\mathbb{G}$, where g is the generator and q is the order. The identifie range $D = \mathbb{G}$. Hash function $\mathsf{H}(x) : \{0,1\}^l \to \mathbb{G}$.
Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.
Protocol.

1. For $1 \leq i \leq m$, P_i samples $a_i \leftarrow \mathbb{Z}_q$ and $k_i \leftarrow \mathbb{Z}_q$, then sends $\{\mathsf{H}(x_i^1)^{a_i}, \cdots, \mathsf{H}(x_i^n)^{a_i}\}$ to $P_{(i+1) \bmod m}$. For $1 \leq j < m$, $P_{(i+j) \bmod m}$ receives $\{y_i^1, \cdots, y_i^n\}$ from $P_{(i+j-1) \bmod m}$. $P_{(i+j) \bmod m}$ computes $\{(y_i^1)^{k_{(i+j) \bmod m}}, \cdots, (y_i^n)^{k_{(i+j) \bmod m}}\}$ and sends to $P_{(i+j+1) \bmod m}$.
2. For $1 \leq i \leq m$, P_i receives $\mathsf{H}(x_i^j)^{a_i k_{i+1} \cdots k_m k_1 \cdots k_{i-1}}$ for $j \in [n]$, then it computes $(\mathsf{H}(x_i^j)^{a_i k_{i+1} \cdots k_m k_1 \cdots k_{i-1}})^{-a_i k_i} = \mathsf{H}(x_i^j)^{k_1 \cdots k_m}$. We denote the set $\{\mathsf{H}(x_i^1)^{k_1 \cdots k_m}, \mathsf{H}(x_i^n)^{k_1 \cdots k_m}\}$ as $R_i = \{r_i^1, \cdots, r_i^n\}$, where each $r_i^j \in \mathbb{G}$.
3. The parties invoke $\Pi_{\mathsf{PK\text{-}MPSU}}$ where P_i inputs $R_i = \{r_i^1, \cdots, r_i^n\}$. P_1 receives the union $R^* = \bigcup_{i=1}^m Y_i$, and sends it to other parties.
4. Each party P_i outputs (R^*, R_i).

Fig. 18. DH-based Multi-Party Private ID Π_{MPID}