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 multi-party private set union (MPSU) protocols: The first category builds on public-key techniques, where existing works require a super-linear number of public-key operations, resulting in their poor practical efficiency. The second category builds on oblivious transfer and symmetric-key techniques. The only work in this category, proposed by Liu and Gao (ASIACRYPT 2023), features the best concrete performance among all existing protocols, but still has super-linear computation and communication. Moreover, 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.

There remain two significant open problems so far: no MPSU protocol achieves semi-honest security based on oblivious transfer and symmetric-key techniques, and no MPSU protocol achieves both linear computation and linear communication complexity. In this work, we resolve both of them.

– We propose the first MPSU protocol based on oblivious transfer and symmetric-key techniques in the standard semi-honest model. This protocol is $3.9 - 10.0\times$ faster than Liu and Gao in the LAN setting. Concretely, our protocol requires only 4.4 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 code 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, JSZG24]. 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[5] can be divided into two categories:

- **PK-MPSU**: This category is primarily based on public-key techniques, and has been explored in a series of works [KS05, Fri07, VCE22, GNT23]. A common drawback of these works is that each party has to perform a substantial number of public-key operations, leading to unsatisfactory practical efficiency.
- **SK-MPSU**: This category is primarily based on symmetric-key techniques, and includes only one existing work [LG23] to date. This work exhibits much better performance than all prior works but fails to achieve 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.

Both categories share one common limitation: neither of them includes a protocol achieving linear computation and communication complexity[6]. Motivated by the context, we raise the following two questions:

[5] We only consider the special-purpose solutions for MPSU, excluding those based on circuit-based generic techniques of secure computation, due to their unacceptable performance.

[6] In the context of MPSU, linear complexity means that the complexity per party scales linearly with the total size of all parties' sets. In this paper, we consider the

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

1.1 Our Contribution

In this work, we answer the above two questions affirmatively. Our contributions are summarized as follows:

Efficient batch ssPMT. We present a new primitive called batch secret-shared private membership test (batch ssPMT). Compared to multi-query secret-shared private membership test (mq-ssPMT), which is the technical core of the state-of-the-art MPSU protocol [LG23] (hereafter referred to as LG), batch ssPMT admits a much more efficient construction. Combined with hashing to bins, batch ssPMT can replace 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 to LG.
SK-MPSU in standard semi-honest model. We generalize random oblivious transfer (ROT) into multi-party setting and present a new primitive called multi-party secret-shared random oblivious transfer (mss-ROT). Based on batch ssPMT and mss-ROT, we propose the first SK-MPSU in the standard semi-honest model. In addition to enhanced security, our SK-MPSU has superior online / total performance with a $3.9 - 10.0\times$ / $1.2 - 7.8\times$ 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. Along the way, we find that the MPSU protocol of Gao et al. [GNT23] is insecure against arbitrary collusion and give a practical attack to demonstrate that it necessitates non-collusion assumption as well. [7]

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

balanced setting where each party holds sets of equal size, thus linear complexity denotes the complexity per party to scale linearly with both the number of parties m and the set size n. Meanwhile, following current conventions, linear complexity only considers the online phase.

[7] After we pointed out the security flaw of the GNT protocol [GNT23] with a 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. The main difference lies in that they implement our batch ssPMT by invoking multiple instances of ssPMT separately, which renders their new protocol still has superlinear complexities. See also the summary of the relationship in their updated paper.

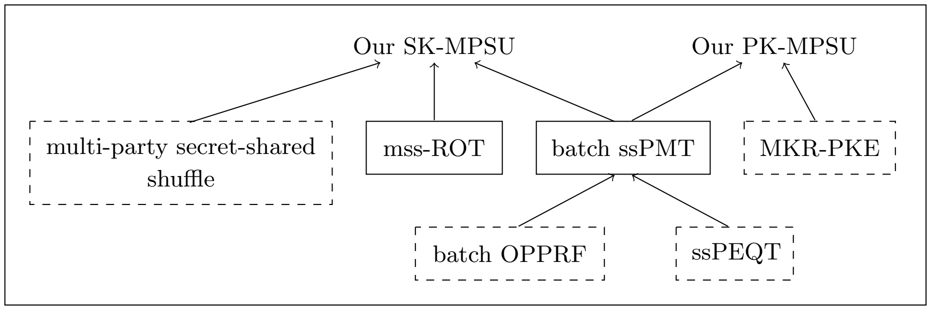

Fig. 1. Technical overview of our MPSU framework. The newly introduced primitives are marked with solid boxes. The existing primitives are marked with dashed boxes.

1.2 Related Works

We review the existing semi-honest MPSU protocols below.

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 an MPSU protocol based on the bit-vector representations, where the parties collectively compute the union by performing the private OR operations on the bit-vectors, using ElGamal encryption. It shows poor concrete efficiency reported by [LG23] and the leader requires quadratic computation and communication in the number of parties.

Recently, Gao et al. [GNT23] proposed an MPSU protocol, representing the most advanced MPSU in terms of asymptotic 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 (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 $109\times$ faster than [VCE22]. However, their protocol is not secure in the standard semi-honest model.

Other Related Works. Blanton et al. [BA12] proposed an MPSU protocol based on oblivious sorting and generic multi-party computation (MPC). 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	λlm^2n	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. n is the set size. m is the number of parties. λ and σ are computational and statistical security parameter. 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, $n \leq 2^{24}$, $m \leq 32$, $\lambda = 128$, $\sigma = 40$, $T \approx 600$, $l \leq 128$, $\gamma \leq 64$.

2 Technical Overview

2.1 LG Revisit

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

LG can be summarized as a secret-sharing based MPSU framework with two phases: The first phase involves two secret-sharing processes, where P_2 somehow secret-shares $Y_2 = X_2 \setminus X_1$ and P_3 somehow secret-shares $Y_3 = X_3 \setminus (X_1 \cup X_2)$ among all parties. Since $\{Y_2, Y_3\}$ is a partition of Y_1, as a result, the parties secret-share Y_1 in the order of Y_2, Y_3. The second phase is to reconstruct Y_1 to P_1. In this case, a straightforward approach is insufficient because P_1 can identify which party each element $x \in Y_1$ originates from. The solution is to let the parties invoke multi-party secret-shared shuffle to randomly permute and re-share all shares. This shuffle ensures that any two parties have no knowledge of the permutation, thereby concealing the correspondence between the secret shares and individual difference sets Y_2, Y_3. Afterwards, P_2 and P_3 can send their shuffled shares to P_1, who then reconstructs Y_1.

LG utilizes two ingredients to realize the secret-sharing processes of the framework: (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 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[8]. The following is to elaborate the concrete constructions of the two processes.

The process for P_2 to secret-share Y_2: 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 $r^0_{2,1} \oplus x \| \mathsf{H}(x)$[9]. P_3 sets $s_{2,3}$ to 0. If $x \notin X_1$, $e_{2,1} \oplus e_{1,2} = 0$, $s_{2,1} \oplus s_{2,2} \oplus s_{2,3} = x \| \mathsf{H}(x)$. 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[10], and the elements in $X_1 \cap X_2$ are masked by random values.

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 $e_{3,2} \oplus e_{2,3} = 0$, $s_{3,1} \oplus s_{3,2} \oplus s_{3,3} = x \| \mathsf{H}(x)$. Otherwise, there is at least one random value ($r^1_{3,1}$ or $r^1_{3,2}$) cannot be canceled out from the summation that makes it random.

The above outlines how LG works. The protocol has two main drawbacks: First, the core component, mq-ssPMT heavily relies on expensive general MPC machinery, which becomes the bottleneck of the entire protocol. Second, LG fails to achieve security against arbitrary collusion. The next two sections are devoted to addressing these two drawbacks.

8 This special ROT is identical to the standard 1-out-of-2 ROT, where e_0 is determined by $\mathcal{S}$ to indicate whether to swap the order of r_0 and r_1.

9 Suppose a lack of specific structure for set elements, then P_1 cannot distinguish a set element x and a random value r in the reconstruction. To address this, the parties append the hash value when secret sharing an element.

10 After the re-sharing of multi-party secret-shared shuffle.

2.2 Efficient Batch ssPMT

To improve the efficiency of mq-ssPMT, we introduce a new functionality called batch ssPMT: 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, where the ith bit is 1 if $y_i \in X_i$, and 0 otherwise.

Compared to mq-ssPMT, batch ssPMT enables the testing of element membership across multiple distinct sets rather than within a single common set. Their relationship is analogous to that between batched oblivious pseudorandom function (batch OPRF) [KKRT16] and multi-point oblivious pseudorandom function (multi-point OPRF) [PRTY19, CM20]. Thanks to its batching nature, batch ssPMT admits a much more efficient construction, lending it a superior alternative to mq-ssPMT in the context of MPSU, by coupling with hashing to bins.

The alternative to mq-ssPMT works as follows: First, $\mathcal{S}$ and $\mathcal{R}$ preprocess inputs through hashing to bins technique. $\mathcal{R}$ uses hash functions h_1, h_2, h_3 to assign $y_1, \cdots, y_n$ to B bins via Cuckoo hashing [PR04], ensuring 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 they 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, while $\mathcal{R}$ 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, ensuring $y_i \in X_j$. Therefore, for each bin j of $\mathcal{R}$, if the inside element y_i is in X, $y_i \in X_j$, $\mathcal{S}$ and $\mathcal{R}$ receive shares of 1 from the ith instance of batch ssPMT. Otherwise, $y_i \notin X_j$, $\mathcal{S}$ and $\mathcal{R}$ receive shares of 0.

The functionality realized by the above construction differs slightly from mq-ssPMT, as 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 entire input set of $\mathcal{R}$, and all shares is arranged according to the parties' Cuckoo hash positions, a straightforward reconstruction may leak information about the parties' input sets to P_1. Therefore, we have to eliminate the dependence of shares' order on the parties' Cuckoo hash positions during the reconstruction phase. Our crucial insight is that the multi-party secret-shared shuffle used in the reconstruction phase eliminates not only the correspondence between secret shares and difference sets, but also this dependence on the hash positions. As a result, our construction can plug in the secret-sharing based framework seamlessly, without introducing any information leakage or additional overhead.

Unlike the mq-ssPMT, which heavily relies on general 2PC, our batch ssPMT is built on two highly efficient primitives, batched oblivious programmable pseudorandom function (batch OPPRF) [KMP$^+$17, PSTY19] and secret-shared private equality test (ssPEQT) [PSTY19, CGS22], where the former has an extremely fast specialized instantiation through vector oblivious linear evaluation (VOLE), while the later is composed of a small circuit, allowing for efficient implementation with general 2PC. Therefore, our batch ssPMT greatly reduces dependency on general 2PC, leading to a substantial performance improvement. For more details, refer to Appendix E.2.

2.3 SK-MPSU from Batch ssPMT and mss-ROT

By replacing mq-ssPMT with the construction in the previous section, we improve LG's performance significantly. However, the resulting protocol still relies on the non-collusion assumption. We proceed to show how to eliminate it.

First, we explain why LG requires the non-collusion assumption using the three-party example: After the secret-sharing process of P_3, all parties hold the secret shares of

$$\begin{cases} x\|\mathsf{H}(x) & x \in X_3 \setminus (X_1 \cup X_2) \\ r^0_{3,1} \oplus r^1_{3,1} \oplus x\|\mathsf{H}(x) & x \in (X_2 \cup X_3) \setminus X_1 \\ r^0_{3,2} \oplus r^1_{3,2} \oplus x\|\mathsf{H}(x) & x \in (X_1 \cup X_3) \setminus X_2 \\ r^1_{3,1} \oplus r^1_{3,2} \oplus r^0_{3,1} \oplus r^0_{3,2} \oplus x\|\mathsf{H}(x) & x \in X_1 \cap X_2 \cap X_3 \end{cases}$$

If $x \in Y_3$, P_1 reconstructs the secret $x\|\mathsf{H}(x)$. If $x \notin Y_3$, the secret is random so that P_1 gains no information about x. Nevertheless, this is only guaranteed when P_1 does not collude with P_3. Recall that P_3 receives $r^0_{3,1}, r^1_{3,1}$ and $r^0_{3,2}, r^1_{3,2}$ from the ROT execution with P_1 and P_2 respectively, so the coalition of P_1 and P_3 can easily check the secret, and distinguish the distinct cases, which reveals information of P_2's inputs.

To fix this security issue, we generalize ROT into multi-party setting, which we call multi-party secret-shared random oblivious transfer (mss-ROT). Consider d involved parties, with two parties, denoted as P_{ch_0} and P_{ch_1}, possessing choice bits b_0 and b_1 respectively. There is a subset J of $\{1, \cdots, d\}$ so that each party P_j $(j \in J)$ holds Δ_j as input. The mss-ROT functionality outputs a random r_i to each party P_i, s.t. if $b_0 \oplus b_1 = 0$, $\bigoplus_{i=1}^d r_i = 0$. Otherwise, $\bigoplus_{i=1}^d r_i = \bigoplus_{j \in J} \Delta_j$.

Equipped with this new primitive, the secret-sharing process of P_3 is revised as follows: After P_3 and P_1 receiving shares $e_{3,1}$ and $e_{1,3}$, $\{P_1, P_2, P_3\}$ invoke mss-ROT, where P_3 and P_1 act as P_{ch_0} and P_{ch_1} holding $e_{3,1}$ and $e_{1,3}$, while P_2 and P_3 samples $\Delta_{2,31}, \Delta_{3,31}$ uniformly as inputs. P_1, P_2, P_3 receives $r_{1,31}, r_{2,31}, r_{3,31}$ separately. If $e_{3,1} \oplus e_{1,3} = 0$, $r_{1,31} \oplus r_{2,31} \oplus r_{3,31} = 0$, otherwise $r_{1,31} \oplus r_{2,31} \oplus r_{3,31} = \Delta_{2,31} \oplus \Delta_{3,31}$. After P_3 and P_2 receiving shares $e_{3,2}$ and $e_{2,3}$, $\{P_2, P_3\}$ invoke mss-ROT, where P_3 and P_2 hold choice bits $e_{3,2}$ and $e_{2,3}$, while P_2 and P_3 samples $\Delta_{2,32}, \Delta_{3,32}$ uniformly as inputs. P_2 and P_3 receives $r_{2,32}$ and $r_{3,32}$ separately. If $e_{3,2} \oplus e_{2,3} = 0$, $r_{2,32} \oplus r_{3,32} = 0$, otherwise $r_{2,32} \oplus r_{3,32} = \Delta_{2,32} \oplus \Delta_{3,32}$. Finally, P_1 sets $s_{3,1} = r_{1,31}$. P_2 sets $s_{3,2} = r_{2,31} \oplus r_{2,32}$. P_3 sets $s_{3,3} = r_{3,31} \oplus r_{3,32} \oplus x\|\mathsf{H}(x)$.

We conclude that all parties hold the secret shares of

$$\begin{cases} x\|\mathsf{H}(x) & x \in X_3 \setminus (X_1 \cup X_2) \\ \Delta_{2,31} \oplus \Delta_{3,31} \oplus x\|\mathsf{H}(x) & x \in (X_2 \cup X_3) \setminus X_1 \\ \Delta_{2,32} \oplus \Delta_{3,32} \oplus x\|\mathsf{H}(x) & x \in (X_1 \cup X_3) \setminus X_2 \\ \Delta_{2,31} \oplus \Delta_{3,31} \oplus \Delta_{2,32} \oplus \Delta_{3,32} \oplus x\|\mathsf{H}(x) & x \in X_1 \cap X_2 \cap X_3 \end{cases}$$

It is immediate that if $x \notin Y_3$, the secret is random in the view of the coalition of $\{P_1, P_2\}$ or $\{P_1, P_3\}$. Since the coalition of $\{P_2, P_3\}$ cannot know the secret, the protocol achieves security under arbitrary collusion.

2.4 PK-MPSU from Batch ssPMT and MKR-PKE

There are no existing MPSU protocols with linear computation and communication complexity. Notably, it is impossible to achieve this goal within the secret-sharing based framework, given that P_1 must reconstruct $(m-1)n$ secrets with each secret comprising m shares. Therefore, we have to seek for another technical route.

Our start point is the existing work [GNT23] with the best asymptotic complexity $mn(\log n/\log\log n)$. We distill their protocol into a framework with two phases: The first phase allows P_1 to somehow obtain encrypted $X_i \setminus (X_1 \cup \cdots \cup X_{i-1})$ for $2 \leq i \leq m$, interspersed with encrypted dummy messages filling the positions of duplicate elements. However, if P_1 decrypts ciphertexts by itself, then it could associate each element in $X_1 \cup \cdots \cup X_m \setminus X_1$ with the specific party it originates from. This problem is addressed by the second phase, which is the collaborative decryption and shuffle procedure, making use of a PKE variant MKR-PKE: P_1 sends the ciphertexts to P_2 after permuting them using a random permutation π_1. P_2 performs a partial decryption on the received ciphertexts, rerandomizes them, permutes them with a random permutation π_2, and forwards the ciphertexts to P_3. This iterative process continues until the last party, P_m, returns its permuted partially decrypted ciphertexts to P_1. Finally, P_1 fully decrypts the ciphertexts, filters out the dummy elements, retains the set $X_1 \cup \cdots \cup X_m \setminus X_1$, and appends X_1 to compute the union.

We identify that both the non-linear complexities and insecurity against arbitrary collusion of [GNT23] arise from their construction in the first phase. Therefore, the task of building a linear-complexity MPSU reduces to devising a linear-complexity and secure construction for this phase.

A rough idea is as follows: Let P_i hold a set of encrypted elements in X_i. For $2 \leq i \leq m$, each P_i engage in a procedure with each P_j $(1 < j < i)$ such that P_j can "obliviously" replace the encrypted elements in $X_i \cap X_j$ with encrypted dummy messages. After the procedure, P_i holds a set of encrypted elements in $X_i \setminus (X_2 \cup \cdots \cup X_{i-1})$. The replacement of encrypted elements in X_1 is handled at the end.

The replacing procedure between each P_i and P_j: (1) P_i acts as $\mathcal{R}$ and executes batch ssPMT (after preprocessing their inputs using hashing to bins, which we omit here for simplicity) with P_j. For each $x \in X_i$, P_i and P_j receive shares $e_{i,j}$ and $e_{j,i}$. (2) If $j = 2$, P_i initializes m_0 as the encrypted x and m_1 as an encrypted dummy message. (3) P_i acts as $\mathcal{S}$ and executes two-choice-bit OT[11] with P_j, where P_i and P_j input $e_{i,j}$ and $e_{j,i}$ as choice bits. P_i inputs m_0, m_1 and P_j 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. (4) P_j rerandomizes ct to ct' and returns ct' to P_i. (5) P_i rerandomizes ct', updates m_0 to ct', and rerandomizes m_1 before repeating the procedure with the next party P_{j+1}. Note that only if all conditions of $x \notin X_j$ are satisfied, the final m_0 is an encrypted message of x.

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

For $2 \leq i \leq m$, each P_i and P_1 execute batch ssPMT and receive secret shares as choice bits for the subsequent execution of two-choice-bit OT. For each $x \in X_i$, P_i acts as $\mathcal{S}$ and inputs the final m_0 of the above procedure and an encrypted dummy message. As a result, if $x \notin X_1$, P_1 receives the final m_0; otherwise, it receives the encrypted dummy message. This excludes all elements in X_1 from the encrypted set $X_i \setminus (X_2 \cup \cdots \cup X_{i-1})$ and lets P_1 obtain the results.

The linear complexities of our construction are attributed to two key facts: First, this framework ensures that each interaction between P_i and P_j only involves the input sets of themselves. Second, our batch ssPMT protocol has linear computation and communication complexity (cf. Section 4).

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 ith 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 applying the permutations $\pi_1, \cdots, \pi_n$ in sequence. $x[i]$ denotes the ith bit of element x, and $X(i)$ denotes the ith element of set X.

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_1 is the leader. 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 ($i \in [m]$), output $\bigcup_{i=1}^m X_i$ to P_1.

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

3.3 Batch Oblivious Programmable 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 ith instance, the sender $\mathcal{S}$ learns a PRF key k_i, while the receiver $\mathcal{R}$ inputs x_i and learns $\mathsf{PRF}(k_i, x_i)$.

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 uniform outputs for some specific inputs and pseudorandom outputs for all other inputs. This kind of PRF with the additional property that the function outputs programmed values on a certain programmed set of inputs 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 3.

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of keys. The bit length γ of values.
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 $|K_i| = |V_i|$, $i \in [B]$.

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 a uniform PPRF key k_i, for $i \in [B]$;
- 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 $i \in [B], 1 \le j \le |K_i|$;
- Define $f_i = F(k_i, x_i)$, for $i \in [B]$;
- Give vector $\vec{f} = (f_1, \cdots, f_B)$ to $\mathcal{R}$.

Fig. 3. Batch OPPRF Functionality $\mathcal{F}_{\mathsf{bOPPRF}}$

3.4 Hashing to Bins

The hashing to bins technique was introduced by Pinkas et al. [PSZ14, PSSZ15] to construct two-party PSI. At a 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 Cuckoo hashing process uses eviction and the choice of bins for each item depends on the entire set. On the other hand, the sender $\mathcal{S}$ assigns each of its items x to all bins $h_1(x), h_2(x), h_3(x)$ via simple hashing. This guarantees that for each item x of $\mathcal{R}$, if x is mapped into the bth bin of Cuckoo hash table ($b \in \{h_1(x), h_2(x), h_3(x)\}$), and x is in $\mathcal{S}$'s set, then the bth of simple hash table certainly contains x.

We denote 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 represents hashing the items of X into B bins using simple hashing with hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [B]$. The output is a simple hash table denoted by $\mathcal{T}^1, \cdots, \mathcal{T}^B$, where for each $x \in X$ we have $\mathcal{T}^{h_i(x)} \supseteq \{x \| i | i = 1,2,3\}$.[12]

We denote 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 represents hashing the items of X into B bins using Cuckoo hashing with hash functions $h_1, h_2, h_3 : \{0,1\}^* \to [B]$. The output is a 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 are irrelevant, corresponding to empty bins. We use these symbols throughout the subsequent sections.

3.5 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 size is 1. Figure 4 defines its functionality.

Parameters. Two parties P_1, P_2. The input bit length γ.
Functionality. On input x from P_1 and input y from P_2, sample two random bits a, b s.t. if $x = y$, $a \oplus b = 1$. Otherwise $a \oplus b = 0$. Give a to P_1 and b to P_2.

Fig. 4. Secret-Shared Private Equality Test Functionality $\mathcal{F}_{\mathsf{ssPEQT}}$

3.6 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 5.

3.7 Multi-Party Secret-Shared Shuffle

Multi-party secret-shared shuffle functionality works by randomly permuting the share vectors of all parties and then refreshing all shares, ensuring that the permutation remains unknown to any coalition of $m - 1$ parties. The formal functionality 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.

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

Eskandarian et al. [EB22] proposed 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 correlations. In the online phase, each party efficiently permutes and refreshes the share vectors using these share correlations.

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 $\pi : [n] \rightarrow [n]$. 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. 6. Multi-Party Secret-Shared Shuffle Functionality $\mathcal{F}_{\mathsf{ms}}$

Gao et al. [GNT23] introduced the concept of multi-key rerandomizable public-key encryption (MKR-PKE), 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 $+$, and $\mathcal{PK}$ denote the space of public keys, which forms an abelian group under the operation $\cdot$. $\mathcal{M}$ denotes the space of plaintexts, and $\mathcal{C}$ denotes 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 is an IND-CPA secure PKE scheme that requires the following additional properties:

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

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

Rerandomizable. For any $pk \in \mathcal{PK}$ and any $x \in \mathcal{M}$,

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

Gao et al. [GNT23] make use of elliptic curve (EC) based ElGamal encryption [ElG85] to instantiate MKR-PKE. The concrete EC MKR-PKE is described in Appendix C.

Remark 1. To the best of our knowledge, there is few elliptic curves that support efficient encoding and decoding between bit-strings and 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 our SK-MPSU and PK-MPSU protocols. It is a two-party protocol that implements multiple instances of ssPMT between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$. Given a batch size of B, $\mathcal{S}$ inputs B sets $X_1, \cdots, X_B$, while $\mathcal{R}$ inputs B elements $x_1, \cdots, x_B$. As a result, $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of a bit vector of size B, where the ith bit is 1 if $x_i \in X_i$, 0 otherwise. The batch ssPMT functionality is presented in Figure 7 and the construction is given in Figure 8, built with batch OPPRF and ssPEQT.

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 s.t. 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. 7. Batch ssPMT Functionality $\mathcal{F}_{\mathsf{bssPMT}}$

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

Correctness. According to the batch OPPRF functionality, if $x_i \in X_i$, $\mathcal{R}$ receives $t_i = s_i$. Then in the ith instance of ssPEQT, since $s_i = t_i$, $e_S^i \oplus e_R^i = 1$.

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, $\mathcal{S}$ and $\mathcal{R}$ receives $e_S^i, e_R^i \in \{0,1\}$ respectively.

Fig. 8. Batch ssPMT Π_{bssPMT}

Conversely, if $x_i \notin X_i$, $\mathcal{R}$ receives a pseudorandom value t_i. The probability that any $t_i = s_i$ for $i \in [B]$ is $B \cdot 2^{-\gamma}$. By setting $\gamma \geq \sigma + \log B$, we have $B \cdot 2^{-\gamma} \leq 2^{-\sigma}$, meaning the probability of any such collision is negligible[13]. After the invocation of ssPEQT, we have that if $x_i \notin X_i$, $e_S^i \oplus e_R^i = 0$ with overwhelming probability.
Security. The security of the protocol follows immediately from the security of batch OPPRF and ssPEQT functionalities.

In our MPSU protocols, the batch ssPMT is always combined with the hashing to bins technique. The combined construction has good efficiency, together with linear computation and communication in term of n, which mainly benefits from the following technical advances: First, we follow the paradigm in [PSTY19] to construct batch OPPRF from batch OPRF and oblivious key-value store (OKVS) [PRTY20, GPR$^+$21, RR22, BPSY23]. By leveraging the technique to amortize communication, the total communication of computing $B = O(n)$ instances of OPPRF is equal to the total number of items $3n$. Second, we utilize subfield vector oblivious linear evaluation (subfield-VOLE) [BCG$^+$19a, BCG$^+$19b, RRT23] to instantiate batch OPRF and the construction in [RR22] to instantiate OKVS. This ensures the computation complexity of batch OPPRF of size $O(n)$ to scale linearly with n. A comprehensive analysis is in Appendix E.2.

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 it to build SK-MPSU based on oblivious transfer and symmetric-key operations in the standard semi-honest model.

[13] In MPSU, the batch OPPRF is invoked multiple times, and the lower bound of γ is relevant to the total number of batch OPPRF invocations. Refer to E.2 for more details.

5.1 Multi-Party Secret-Shared Random Oblivious Transfer

In Figure 5, the ROT functionality can be interpreted as $r_0 \oplus r_b = b \cdot \Delta$, where $\Delta = r_0 \oplus r_1$. As the aforementioned two-choice-bit ROT, it can be consider to secret-share the choice bit b between two parties and interpreted as $r_0 \oplus r_b = (b_0 \oplus b_1) \cdot \Delta$. We further extend this secret-sharing idea to multiple parties, allowing not only any non-empty subset of the involved parties to secret-share b, but also any non-empty subset of the involved parties to secret-share Δ.

To build our SK-MPSU, we only need two-choice-bit version of mss-ROT. For simplicity, we will refer to this as mss-ROT hereafter. The formal functionality[14] is given in Figure 9 and the detailed construction is given in Figure 10.

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 D.1.

5.2 Construction of Our SK-MPSU

We now turn our attention to construct a SK-MPSU. The construction follows the high-level ideas we introduced in the technical overview and is formally presented in Figure 11.

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.*

The proof of Theorem 3 is in Appendix D.2. We provide a comprehensive complexity analysis for our SK-MPSU / LG and an exhaustive comparison between them in Appendix E.

6 MPSU from Public-Key Techniques

In this section, we describe how to construct a PK-MPSU achieving linear computation and linear communication complexity. The construction is formally presented in Figure 12.

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 rerandomizable property and indistinguishable multiple encryptions (cf. Appendix B) of MKR-PKE scheme.*

[14] We choose to let the parties input the shares of Δ instead of outputting them, as the former functionality simply implies the latter.

Parameters. m parties $P_1, \cdots, P_m$, where P_{ch_0} and P_{ch_1} provide inputs as shares of the choice bit, $\mathrm{ch}_0, \mathrm{ch}_1 \in [m]$. We use $J = \{j_1, \cdots, j_d\}$ to denote the set of indices for the parties who provide inputs as shares of Δ, where $d \leq m$. The message length l.
Functionality. On input $b_0 \in \{0,1\}$ from P_{ch_0}, $b_1 \in \{0,1\}$ from P_{ch_1}, and Δ_j from each P_j where $j \in J$,

- Sample $r_i \leftarrow \{0,1\}^l$ and give r_i to P_i for $2 \leq i \leq m$.
- If $b_0 \oplus b_1 = 0$, compute $r_1 = \bigoplus_{i=2}^{m} r_i$, else compute $r_1 = \bigoplus_{i=2}^{m} r_i \oplus (\bigoplus_{j \in J} \Delta_j)$. Give r_1 to P_1.

Fig. 9. Multi-Party Secret-Shared Random OT Functionality $\mathcal{F}_{\mathsf{mss\text{-}rot}}$

Parameters. m parties $P_1, \cdots, P_m$. where P_{ch_0} and P_{ch_1} provide inputs as shares of the choice bit, $\mathrm{ch}_0, \mathrm{ch}_1 \in [m]$. We use $J = \{j_1, \cdots, j_d\}$ to denote the set of indices for the parties who provide inputs as shares of Δ, where $d \leq m$. The message length l.
Inputs. P_{ch_0} has input $b_0 \in \{0,1\}$ and P_{ch_1} has input $b_1 \in \{0,1\}$. Each P_j for $j \in J$ has input Δ_j,
Protocol.

1. For $1 \leq i \leq m$, P_i initializes $r_i = 0$. If $ch_{j=0,1} \in J$, P_{ch_j} sets $r_{ch_j} = b_j \cdot \Delta_{ch_j}$ ($\cdot$ denotes bitwise-AND).
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. P_{ch_0} receives $u_{j,\mathrm{ch}_0}^{b_0} \in \{0,1\}^l$. P_j receives $u_{j,\mathrm{ch}_0}^{0}, u_{j,\mathrm{ch}_0}^{1} \in \{0,1\}^l$. P_j updates $r_j = r_j \oplus u_{j,\mathrm{ch}_0}^{0}$ and sends $\Delta'_{j,\mathrm{ch}_0} = u_{j,\mathrm{ch}_0}^{0} \oplus u_{j,\mathrm{ch}_0}^{1} \oplus \Delta_j$ to P_{ch_0}. P_{ch_0} updates $r_{\mathrm{ch}_0} = r_{\mathrm{ch}_0} \oplus u_{j,\mathrm{ch}_0}^{b_0} \oplus b_0 \cdot \Delta'_{j,\mathrm{ch}_0}$.
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 $u_{j,\mathrm{ch}_1}^{b_1} \in \{0,1\}^l$. P_j receives $u_{j,\mathrm{ch}_1}^{0}, u_{j,\mathrm{ch}_1}^{1} \in \{0,1\}^l$. P_j updates $r_j = r_j \oplus u_{j,\mathrm{ch}_1}^{0}$ and sends $\Delta'_{j,\mathrm{ch}_1} = u_{j,\mathrm{ch}_1}^{0} \oplus u_{j,\mathrm{ch}_1}^{1} \oplus \Delta_j$ to P_{ch_1}. P_{ch_1} updates $r_{\mathrm{ch}_1} = r_{\mathrm{ch}_1} \oplus u_{j,\mathrm{ch}_1}^{b_1} \oplus b_1 \cdot \Delta'_{j,\mathrm{ch}_1}$.
4. If $|I = \{ch_0, ch_1\} \cup J| < m$, each P_i ($i \in I$) samples $r'_{i,i'}$ and sends $r'_{i,i'}$ to each $P_{i'}$ ($i' \in [m] \setminus I$) and updates $r_i = r_i \oplus r'_{i,i'}$. $P_{i'}$ updates $r_{i'} = r_{i'} \oplus r'_{i,i'}$.
5. For $1 \leq i \leq m$, P_i outputs the final value of r_i.

Fig. 10. 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. A collision-resisitant hash function $\mathsf{H}(x) : \{0,1\}^l \to \{0,1\}^\kappa$.
Inputs. Each party P_i inputs $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.
Protocol.

1. **Hashing to bins.** 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 $1 \leq i < j \leq m, 1 \leq b \leq B$: $P_{min(2,i)}, \cdots, P_j$ 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 $1 < j' \leq j$, $P_{j'}$ samples $\Delta^b_{j',ji} \leftarrow \{0,1\}^{l+\kappa}$ and inputs $\Delta^b_{j',ji}$ as shares of Δ. For $min(2,i) \leq d < j$, P_d receives $r^b_{d,ji} \in \{0,1\}^{l+\kappa}$ and computes $u^b_{d,j} = \bigoplus_{i=1}^{j-1} r^b_{d,ji}$. P_j receives $r^b_{j,ji} \in \{0,1\}^{l+\kappa}$ 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. 11. Our SK-MPSU $\Pi_{\mathsf{SK\text{-}MPSU}}$

For the complete proof, refer to Appendix D.3. A comprehensive complexity analysis and an exhaustive comparison with [GNT23] can be found in Appendix E.

Remark 2. When instantiating our PK-MPSU framework with EC MKR-PKE, the element space is set as EC points accordingly, which might limit its usage in applications. We also identified the same issue in [GNT23]. However, we argue that this limitation is not inherent in our PK-MPSU (nor to the protocol in[GNT23]), as it can be addressed by finding an elliptic curve with efficient encoding and decoding algorithms or by exploring alternative instantiations of MKR-PKE.

Despite the limitation, our EC-element-space PK-MPSU remains useful in many scenarios. For instance, we present the first multi-party private ID protocol in Appendix F, where our EC-element-space PK-MPSU MPSU serves as a core subprotocol. Our multi-party private-ID takes advantages of our PK-MPSU and achieves linear computation and communication complexity and low communication costs as well.

7 Performance Evaluation

In this section, we provide the implementation details and experimental results for our works. For most previous works [KS05, Fri07, BA12, GNT23] do not provide open-source code, and the protocol in [VCE22] shows fairly inferior performance in comparison with the state-of-the-art LG, here we only compare our protocols with LG, whose implementation is available on https://github.com/lx-1234/MPSU.

During our experiments, we identify several issues with the LG implementation:

1. The code is neither a complete nor a secure implementation for MPSU, as it lacks the offline share correlation generation of the multi-party secret-shared shuffle protocol [EB22]. Instead of adhering to the protocol specifications, the implementation designates one party to generate and store some "fake" share correlations as local files during the offline phase, while the other parties read these files and consume the share correlations in the online phase[15]. This results in two serious consequences:
   - The distributed execution is not supported.
   - The security of multi-party secret-shared shuffle is compromised, leading to information leakage.

   To conduct a fair and complete comparison, we integrated our correct version of the share correlation generation implementation into their code.
2. The code is not a correct implementation, as it produces incorrect results when the set size n increases beyond a certain threshold. We figure out that

[15] Refer to the function `ShareCorrelation::generate()` in ShareCorrelationGen.cpp

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. 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 bins.** 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 $\vec{c}_j$ 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 $r_{i,j}^b = r^b_{j,i,e^b_{i,j}} \in \{0,1\}^\lambda$. P_j receives $r^b_{j,i,0}, r^b_{j,i,1} \in \{0,1\}^\lambda$. P_j computes $u^b_{j,i,e^b_{j,i}} = r^b_{j,i,e^b_{j,i}} \oplus c_j^b$, $u^b_{j,i,e^b_{j,i}\oplus 1} = r^b_{j,i,e^b_{j,i}\oplus 1} \oplus \mathsf{Enc}(pk, \perp)$, 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.**
   (a) P_1 samples $\pi_1 : [(m-1)B] \rightarrow [(m-1)B]$ and computes $\vec{\mathsf{ct}}''_1 = \pi_1(\vec{\mathsf{ct}}'_1)$. P_1 sends $\vec{\mathsf{ct}}''_1$ to P_2.
   (b) 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 $\mathsf{ct}'^i_j = \mathsf{ReRand}(pk_{A_j}, \mathsf{ct}^i_j)$. Then it samples $\pi_j : [(m-1)B] \rightarrow [(m-1)B]$ 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.
   (c) 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. 12. Our PK-MPSU $\Pi_{\mathsf{PK\text{-}MPSU}}$

this issue arises from the use of a fixed parameter[16] for generating Beaver Triples in the offline phase, which leads to an insufficient number of Beaver Triples for the online phase when n is large. We modified their code to ensure correct execution and mark the cases where the original code fails with $*$.

7.1 Experimental Setup

We conduct our experiments on an Alibaba Cloud virtual machine with Intel(R) Xeon(R) 2.70GHz CPU (32 physical cores) and 128 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 measure the running time as the maximal time from protocol begin to end, including messages transmission time, and the communication costs as the total data that leader sent and received. For a fair comparison, we stick to the following settings for all protocols:

- 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 and our SK-MPSU, each party holds n 64-bit strings. In our PK-MPSU, each party holds n elements encoded as EC points in compressed form.
- Each party uses $m - 1$ threads to interact simultaneously with all other parties and 4 threads to perform share correlation generation (in LG and our SK-MPSU), Beaver Triple generation (in all three), parallel SKE encryption (in LG), ciphertext rerandomization and partial decryption (in our PK-MPSU).

7.2 Implementation Details

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

- VOLE: We use VOLE implemented in libOTe[17], instantiating the code family with Expand-Convolute codes [RRT23].
- OKVS and GMW: We use the optimized OKVS in [RR22] as our OKVS instantiation[18], and re-use the implementation of OKVS and GMW by the authors of in [RR22][19].

[16] The variable `batchsize` in line 30 of circuit/TripleGen.cpp

[17] https://github.com/osu-crypto/libOTe.git

[18] 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].

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

- 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 [JSZ+22][20] to build share correlation generation for our SK-MPSU and LG.
- MKR-PKE: We implement MKR-PKE on top of the curve `NIST P-256` (also known as `secp256r1` and `prime256v1`) implementation from openssl[21].
- Additionally, we use the cryptoTools[22] library to compute hash functions and PRNG calls, and we adopt Coproto[23] to realize network communication.

7.3 Experimental Results

We conduct extensive experiments across various 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. The performance of protocols is evaluated from four dimensions: online and total running time, and online and total communication costs. The results for running time / communication costs are depicted in Table 2. As we can see in the table, our protocols outperform LG in all the case studies, even with enhanced security.

Online computation improvement. Our SK-MPSU achieves a $3.9 - 10.0\times$ speedup compared to LG in the LAN setting, and a $1.1 - 2.1\times$ speedup compared to LG in the WAN setting. For example, in the online phase to compute the union of 2^{20}-size sets among 3 parties, our SK-MPSU runs in 4.4 seconds (LAN) and 33.1 seconds (WAN), which is $5.8\times$ and $1.64\times$ faster than LG that takes 25.2 seconds (LAN) and 54.1 seconds (WAN), respectively.
Total computation improvement. Our SK-MPSU achieves a $1.2 - 7.8\times$ speedup compared to LG in the LAN setting. Our PK-MPSU achieves a $1.2 - 4.0\times$ speedup compared to LG in the WAN setting. For example, to compute the union of 2^{20}-size sets among 3 parties, our SK-MPSU runs in 96.2 seconds overall (LAN), which is $6.4\times$ faster than LG that takes 610.8 seconds. To compute the union of 2^{18}-size sets among 9 parties, our PK-MPSU runs in 2304 seconds overall (WAN), which is $2.8\times$ faster than LG that takes 6381 seconds.
Online communication improvement. The online communication costs of our SK-MPSU is $1.2 - 4.9\times$ smaller than LG. For example, in the online phase to compute the union of 2^{20}-size sets among 3 parties, our SK-MPSU requires 455.6 MB communication, which is a $2.0\times$ improvement of LG that requires 917.0 MB communication.
Total communication improvement. The total communication costs of our PK-MPSU is $3.0 - 36.5\times$ smaller than LG. For example, to compute the union of 2^{18}-size sets among 9 parties, our PK-MPSU requires 960.1 MB communication, a $34.8\times$ improvement of LG that requires 33360 MB.

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

In conclusion, our protocols exhibit distinct advantages and are suitable for different scenarios:

- Our SK-MPSU: This protocol excels in online performance. Additionally, it also outperforms other protocols in nearly all cases in the LAN setting. Notably, the online phase takes only 4.4 seconds for 3 parties with sets of 2^{20} items each, and 7 seconds for 5 parties with sets of 2^{20} items each in the LAN setting. Therefore, our SK-MPSU is the optimal choice when online performance is the primary concern, or in high-bandwidth networks.
- Our PK-MPSU: This protocol has the best total performance in the WAN setting, so it is preferred when the total cost is crucial (e.g. in the small-sets setting), at the same time in low-bandwidth networks. Note that our PK-MPSU is the only protocol with the capability of running among 10 parties with sets of 2^{18} items each in our experiments, which sufficiently indicates the superiority of its linear complexities.

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

References

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.

Sett.	m	Protocol	Set size n															
			Online								Total = Offline + Online							
			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}
Time. (s) LAN	3	LG	0.050	0.058	0.069	0.143	0.425	1.582	6.219	25.16*	0.982	1.016	1.098	1.767	5.650	32.42	153.8	610.8*
		Our SK	0.005	0.007	0.009	0.017	0.050	0.213	1.005	4.352	0.361	0.386	0.395	0.489	1.170	4.534	19.77	96.23
		Our PK	0.092	0.237	0.822	3.176	12.69	51.17	205.7	829.1	0.117	0.273	0.868	3.270	13.08	53.05	215.0	872.6
	4	LG	0.069	0.076	0.093	0.162	0.478	1.637	6.375	25.79*	1.292	1.358	1.514	2.432	8.716	45.37	197.2	762.9*
		Our SK	0.008	0.010	0.013	0.023	0.071	0.286	1.393	5.645	0.639	0.665	0.696	0.917	2.597	10.94	50.25	237.8
		Our PK	0.159	0.465	1.570	6.136	24.60	44.91	398.4	1604	0.196	0.505	1.643	6.299	25.45	48.95	417.5	1694
	5	LG	0.107	0.109	0.126	0.190	0.541	1.998	7.588	31.33*	1.939	1.993	2.202	3.408	12.54	65.04	289.9	1152*
		Our SK	0.012	0.013	0.017	0.030	0.087	0.368	1.714	7.003	0.914	0.964	1.023	1.558	4.551	18.71	85.79	373.4
		Our PK	0.232	0.693	2.487	9.858	39.33	158.1	637.5	2816	0.292	0.796	2.547	10.09	40.25	162.2	655.0	2979
	7	LG	0.154	0.165	0.174	0.281	0.795	2.894	10.92	–	3.484	3.583	3.921	5.84	23.42	111.2	484.5	–
		Our SK	0.019	0.021	0.028	0.048	0.156	0.607	2.817	–	2.051	2.079	2.214	3.470	12.92	56.50	245.6	–
		Our PK	0.463	1.365	5.030	19.43	77.22	310.0	1247	–	0.550	1.469	5.158	19.73	78.59	316.3	1276	–
	9	LG	0.222	0.226	0.234	0.417	1.216	4.449	17.29	–	5.501	5.612	6.249	10.27	41.69	182.7	793.3	–
		Our SK	0.027	0.031	0.039	0.075	0.230	0.970	4.293	–	3.363	3.402	3.895	7.161	30.09	126.3	678.8	–
		Our PK	0.764	2.271	8.074	31.64	126.7	507.0	2039	–	0.880	2.375	8.238	32.19	128.7	515.5	2080	–
	10	LG	0.228	0.243	0.276	0.514	1.479	5.467	–	–	6.736	6.846	8.335	13.56	55.58	238.3	–	–
		Our SK	0.031	0.036	0.043	0.088	0.286	1.183	–	–	3.965	4.232	4.821	9.334	41.97	175.9	–	–
		Our PK	0.865	2.800	9.944	39.09	155.9	622.7	2503	–	0.970	2.912	10.11	39.58	158.4	634.6	2556	–
Time. (s) WAN	3	LG	4.502	4.505	4.522	4.914	6.272	8.744	17.78	54.13*	12.46	13.59	15.76	19.52	30.73	78.74	282.8	1188*
		Our SK	2.165	2.166	2.332	3.157	3.734	4.444	9.705	33.10	8.403	9.710	12.62	16.30	25.02	56.88	194.7	801.2
		Our PK	4.419	4.555	5.553	7.984	18.21	59.77	226.3	900.3	5.168	5.306	6.472	8.968	19.65	62.73	237.3	946.1
	4	LG	5.696	5.880	6.540	7.094	7.323	11.36	23.13	78.81*	17.94	20.99	27.63	32.69	50.59	145.7	554.7	2167*
		Our SK	2.967	2.969	3.298	3.976	4.618	6.507	17.10	59.21	11.46	13.93	20.21	26.77	47.49	133.2	520.0	2141
		Our PK	5.622	5.899	7.929	12.17	32.40	113.8	440.9	1761	6.773	7.052	9.25	13.53	34.27	118.9	461.3	1857
	5	LG	7.385	7.708	8.621	9.198	9.687	14.62	32.83	126.3*	23.64	26.82	37.39	48.48	88.32	263.3	1053	4394*
		Our SK	3.768	3.733	4.471	4.800	5.521	8.938	25.95	95.40	17.53	21.10	30.28	44.30	88.15	278.9	1119	4820
		Our PK	6.849	7.928	10.50	17.08	49.75	179.8	703.3	2873	8.424	9.483	12.22	18.86	52.07	185.4	724.5	2980
	7	LG	9.312	9.833	10.55	11.35	12.14	21.29	66.93	–	34.92	45.69	66.26	94.89	203.4	705.8	2898	–
		Our SK	5.373	5.381	6.207	6.644	8.164	17.67	56.894	–	34.00	44.95	61.34	95.03	228.7	817.4	3506	–
		Our PK	9.504	12.32	15.14	29.73	92.66	348.5	1377.4	–	11.88	14.71	17.72	32.35	95.86	356.5	1409	–
	9	LG	11.41	12.21	13.34	14.41	15.09	33.84	115.5	–	56.65	75.24	104.1	169.1	406.0	1503	6387	–
		Our SK	6.977	7.068	7.830	8.387	12.24	29.20	105.7	–	58.81	84.80	107.6	182.0	502.5	1915	8137	–
		Our PK	13.67	18.84	22.96	45.54	148.2	570.9	–	–	16.87	22.04	26.31	48.98	152.3	581.5	2034	–
	10	LG	11.77	12.24	15.80	16.49	17.48	45.20	–	–	66.19	92.22	125.1	219.8	582.1	2179	–	–
		Our SK	7.780	8.032	8.635	9.203	14.54	38.31	–	–	71.58	109.2	132.7	242.7	684.0	2687	–	–
		Our PK	16.32	23.05	28.66	59.79	184.8	708.5	2792	–	19.90	26.66	32.44	63.66	189.3	720.9	2846	–
Comm. (MB)	3	LG	0.157	0.284	0.962	3.662	14.43	57.58	229.8	917.0*	6.311	7.904	13.37	32.58	114.6	474.3	2052	8973*
		Our SK	0.032	0.111	0.426	1.690	6.788	27.87	112.7	455.6	2.542	3.582	8.529	30.91	132.7	588.8	2614	11529
		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	240.0	960.3
	4	LG	0.242	0.449	1.536	5.868	23.15	92.38	368.6	1471*	9.591	12.44	23.89	67.25	253.7	1074	4674	20419*
		Our SK	0.058	0.204	0.791	3.145	12.81	51.69	208.8	843.7	3.941	6.385	16.96	66.89	293.9	1312	5834	25751
		Our PK	2.723	2.975	3.983	8.126	24.54	101.9	359.0	1439	3.127	3.488	4.610	8.861	25.38	102.8	360.1	1440
	5	LG	0.330	0.630	2.173	8.325	32.86	131.2	523.5	2090*	12.92	17.75	36.48	110.7	435.2	1868	8228	35737*
		Our SK	0.090	0.323	1.257	5.007	20.36	82.11	331.5	1339	5.504	10.18	30.44	124.7	548.9	2445	10832	47635
		Our PK	3.630	3.967	5.311	10.84	32.72	121.6	478.7	1919	4.169	4.650	6.147	11.82	33.84	122.8	480.1	1921
	7	LG	0.519	1.038	3.635	13.99	55.29	220.8	881.3	–	19.92	29.90	41.59	243.3	997.9	4326	18924	–
		Our SK	0.172	0.634	2.489	10.03	40.31	162.5	655.4	–	8.827	18.80	64.76	275.4	1226	5474	24275	–
		Our PK	5.445	5.950	7.966	16.25	49.07	182.3	718.0	–	6.253	6.98	9.220	17.72	50.76	184.2	720.1	–
	9	LG	0.723	1.509	5.347	20.65	81.72	326.3	1303	–	28.15	44.74	115.0	415.33	1742	7609	33360	–
		Our SK	0.279	1.046	4.122	16.60	66.76	269.0	1084	–	13.99	33.04	119.9	516.3	2295	10207	45079	–
		Our PK	7.260	7.933	10.62	21.67	65.43	243.1	957.3	–	8.338	9.300	12.29	23.63	67.68	245.6	960.1	–
	10	LG	0.831	1.768	6.296	24.36	96.43	385.1	–	–	32.32	54.34	148.7	549.4	2314	10081	–	–
		Our SK	0.341	1.288	5.086	20.48	82.39	332.0	–	–	16.21	40.56	149.9	651.6	2901	12908	–	–
		Our PK	8.168	8.925	11.95	24.38	73.61	273.5	1077	–	9.380	10.46	13.83	26.59	76.14	276.3	1080	–

Table 2. Online and total running time / communication costs of LG and our protocols in LAN and WAN settings. m is the number of parties. Our SK /PK denotes our SK-MPSU / PK-MPSU protocol. Cells with $*$ denotes trials that the original LG code will give wrong results. Cells with $-$ denotes trials that ran out of memory. The best protocol within a setting is marked in blue.

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. Version: 20240316:210303, https://eprint.iacr.org/archive/2023/1930/20240316:210303.

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.

JSZG24. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. Scalable private set union, with stronger security. In *33rd USENIX Security Symposium, USENIX Security 2024*. USENIX Association, 2024.

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 would suggest that the protocol fails to achieve this security, and also requires the non-collusion assumption as LG. First, we give a brief review of the protocol.

Apart from MKR-PKE, their protocol utilizes two new ingredients: 1) 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 $\mathcal{S}$. 2) 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 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 the protocol (cf. Figure 8 in their paper), in step 3.(a), P_1 and P_2 invoke the OPRF where P_1 acts as $\mathcal{R}$ 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 P_3's output 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, the non-collusion assumption is required.

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 1. *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 MKR-PKE from ElGamal

MKR-PKE is instantiated with ElGamal encryption below:

- 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 $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (g^r, x \cdot pk^r)$.
- 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 $\mathsf{ct}' = (\mathsf{ct}_1 \cdot g^{r'}, \mathsf{ct}_2 \cdot pk^{r'})$.

D Missing Security Proofs

D.1 The Proof of Theorem 2

We consider the case that $\mathrm{ch}_0 = 1$ and $\mathrm{ch}_1 = m$, and $J = \{1, \cdots, m\}$. Let P_1 and P_m input $b_1 \in \{0, 1\}$ and $b_m \in \{0, 1\}$, and P_i $(1 \leq i \leq m)$ input Δ_i respectively. We turn to proving the correctness and security of the protocol in Figure 10 in this particular case. Note that in the different cases, the proof is essentially the same.

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

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

$$r_i = r_{i,1}^0 \oplus r_{i,m}^0, 1 < i < m, \tag{2}$$

$$r_m = \bigoplus_{j=1}^{m-1}(r_{j,m}^{b_m} \oplus b_m \cdot u_{j,m}) \oplus r_{m,1}^0 \oplus b_m \cdot \Delta_m, \tag{3}$$

$$u_{j,1} = \Delta_j \oplus r_{j,1}^0 \oplus r_{j,1}^1, u_{j,m} = \Delta_j \oplus r_{j,m}^0 \oplus r_{j,m}^1 \tag{4}$$

From the definition of Random OT functionality (Figure 5), we have the following equations:

$$r_{j,1}^{b_1} = r_{j,1}^0 \oplus b_1 \cdot (r_{j,1}^0 \oplus r_{j,1}^1), \tag{5}$$

$$r_{j,m}^{b_m} = r_{j,m}^0 \oplus b_m \cdot (r_{j,m}^0 \oplus r_{j,m}^1), \tag{6}$$

Substitute Equation 4, 5, 6 into Equation 1, 2, 2 and cancel out the same terms, we obtain:

$$r_1 = \bigoplus_{j=2}^{m}(r_{j,1}^{b_1} \oplus b_1 \cdot \Delta_j) \oplus r_{1,m}^0 \oplus b_1 \cdot \Delta_1, \tag{7}$$

$$r_m = \bigoplus_{j=1}^{m-1}(r_{j,m}^{b_m} \oplus b_m \cdot \Delta_j) \oplus r_{m,1}^0 \oplus b_m \cdot \Delta_m, \tag{8}$$

Substitute Equation 2, 7, 8 into $r_1 \oplus (\bigoplus_{j=2}^{m-1} r_j) \oplus r_m$, we have $r_1 \oplus (\bigoplus_{j=2}^{m-1} r_j) \oplus r_m = \bigoplus_{i=1}^{m}(b_1 \oplus b_m) \cdot \Delta_i$. 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 of $P_c \in \mathcal{C}orr$ and needs to emulate each P_c's view, including its private input 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 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 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 all outputs r_c of $P_c \in \mathcal{C}orr$. For P_1, its view consists of the choice bit b_1, its private input Δ_1, outputs $(r^0_{1,m}, r^1_{1,m})$, $\{r^{b_1}_{j,1}\}_{1<j\leq m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,1}\}_{1<j\leq m}$ from P_j. For each $P_c(c \neq 1)$, its view consists of its private input Δ_c, outputs $(r^0_{c,1}, r^1_{c,1})$ and $(r^0_{c,m}, r^1_{c,m})$ from $\mathcal{F}_{\mathsf{rot}}$.

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

In the real execution, P_1 receives $u_{i,1} = \Delta_i \oplus r^0_{i,1} \oplus r^1_{i,1}$. From the definition of ROT functionality, $r^0_{i,1}$ (when $b_1 = 0$) or $r^1_{i,1}$(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 all outputs r_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 all outputs r_c of $P_c \in \mathcal{C}orr$. For P_1, its view consists of the choice bit b_1, its private input Δ_1, outputs $(r^0_{1,m}, r^1_{1,m})$, $\{r^{b_1}_{j,1}\}_{1<j\leq m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,1}\}_{1<j\leq m}$ from P_j. For each $P_c(c \neq 1, c \neq m)$, its view consists of its private input Δ_c, outputs $(r^0_{c,1}, r^1_{c,1})$ and $(r^0_{c,m}, r^1_{c,m})$ from $\mathcal{F}_{\mathsf{rot}}$. For P_m, its view consists of its private input Δ_m, outputs $(r^0_{m,1}, r^1_{m,1})$, $\{r^{b_m}_{j,m}\}_{1\leq j<m}$ from $\mathcal{F}_{\mathsf{rot}}$ and protocol messages $\{u_{j,m}\}_{1\leq j<m}$ from P_j.

For P_1's view, the simulator runs the protocol honestly except that it simulates uniform outputs $r^{b_1}_{i,1}$ from $\mathcal{F}_{\mathsf{rot}}$ and uniformly random messages $u_{i,1}$ from P_i under the constraint $\bigoplus_{j=2}^{m}(r^{b_1}_{j,1} \oplus b_1 \cdot u_{j,1}) \oplus r^0_{1,m} \oplus b_1 \cdot \Delta_1 = r_1$, 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 $r^0_{c,m} = r^0_{c,1} \oplus r_c$ and simulates uniform output $r^1_{c,m}$ from $\mathcal{F}_{\mathsf{rot}}$. For P_m's view, it runs the protocol honestly with the following changes:

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

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

D.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, sampled values as shares of Δ for $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ 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 $\{r_{c,ji}^b\}_{c<j\leq m, 1\leq i<j}$ from $\mathcal{F}_{\mathsf{mss\text{-}rot}}$, on condition that $\exists min(2,i) \leq d \leq j, P_d \in \mathcal{H}$. If $c \neq 1$, it simulates uniform $\{\Delta_{c,ji}^b\}_{c<j\leq m, 1\leq i<j}$ as P_c's random tapes.
- 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 $\forall x_i \in Y$, $v_i = x_i \| \mathsf{H}(x)$, $1 \leq i \leq |Y|$.
- For $|Y| < i \leq (m-1)B$, samples $v_i \leftarrow \{0,1\}^{l+\kappa}$.

Then it samples a random permutation $\pi : [(m-1)B] \to [(m-1)B]$ and computes $\vec{v'} = \pi(\vec{v})$. For $1 \leq i \leq m$, 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 $P_{min(2,i)}, \cdots, P_j$ if $e_{i,j}^b \oplus e_{j,i}^b = 0$, or a secret-share of $\bigoplus_{d=2}^{j} \Delta_{d,ji}^b$ 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 $P_d \in \mathcal{H}(min(2,i) \leq d \leq j)$ holding one share.

It remains to demonstrate that the output $\vec{sh}_c'$ from $\mathcal{F}_{\mathsf{ms}}(P_1 \notin \mathcal{C}orr)$ or all outputs $\{\vec{sh}_i'\}_{1\leq i\leq m}$ 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 $1 < j \leq m$, 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 $X_i (1 \leq i < j)$ s.t. $x \in X_i$, then for all $1 \leq i < j$, $e^y_{i,j} \oplus e^y_{j,i} = 0$. By the $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ functionality in Figure 9, each $r^b_{d,ji}$ is uniform in $\{0,1\}^{l+\kappa}$ conditioned on $\bigoplus^j_{d=min(2,i)} r^b_{d,ji} = 0$. From the descriptions of the protocol, We derive that each $u^b_{d,j}$ is uniform in $\{0,1\}^{l+\kappa}$ conditioned on $\bigoplus^j_{d=min(2,i)} u^b_{d,j} = x \| \mathsf{H}(x)$, namely, they are additive shares of $x\|\mathsf{H}(x)$ among all parties. This is exactly identical to the simulation.

If there exists some $X_i (1 \leq i < j)$ s.t. $x \in X_i$, then $e^b_{i,j} \oplus e^b_{j,i} = 1$. By the $\mathcal{F}_{\mathsf{mss\text{-}rot}}$ functionality in Figure 9, each $r^b_{d,ji}$ is uniform conditioned on $\bigoplus^j_{d=min(2,i)} r^b_{d,ji} = \bigoplus^j_{j'=2} \Delta^b_{j',ji}$, where each $\Delta^b_{j',ji}$ is uniformly held by $P_{j'}$ $(1 < j' \leq j)$. From the descriptions of the protocol, We derive that each $u^b_{d,j}$ is uniform conditioned on $\bigoplus^j_{d=min(2,i)} u^b_{d,j} = x\|\mathsf{H}(x) \oplus \bigoplus^j_{j'=2} \Delta^b_{j',ji} \oplus r$, where r is the sum of remaining terms. Then, even if P_1 colludes with others, $\bigoplus^j_{d=min(2,i)} u^b_{d,j}$ is still uniformly random from the perspective of adversary, since there always exists a party $P_{j'} \in \mathcal{H}(1 < j' \leq j)$ holding one uniform $\Delta^b_{j',ji}$ and independent of all honest parties' inputs. For all empty bins, $u^b_{d,j}$ is chosen uniformly random, so the corresponding $\bigoplus^j_{d=1} u^b_{d,j}$ is also uniformly at random, which is identical to the simulation. By the definition of $\mathcal{F}_{\mathsf{ms}}$, all parties additively share $\bigoplus^j_{d=1} u^b_{d,j}$ 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.

D.3 The Proof of Theorem 4

Proof. The simulator receives the input X_c of $P_c \in \mathcal{C}orr$ and the output $\bigcup^m_{i=1} X_i$ 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 $\pi : [(m-1)B] \rightarrow [(m-1)B]$ 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 1 (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 $\mathsf{Elem}(\mathcal{C}_c^b) \in X_c \setminus (X_1 \cup \cdots \cup X_i)$, $v_{c,i}^b = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}_c^b))$; else $v_{c,i}^b = \mathsf{Enc}(pk, \perp)$. $v_{c,i}'^b = \mathsf{ReRand}(pk, v_{c,i}^b)$.
- Hyb_1 For $1 < i < c, 1 \leq b \leq B$: If $\mathsf{Elem}(\mathcal{C}_c^b) \in X_c \setminus (X_1 \cup \cdots \cup X_i)$, $v_{c,i}'^b = \mathsf{Enc}(pk, \mathsf{Elem}(\mathcal{C}_c^b))$; else $v_{c,i}'^b = \mathsf{Enc}(pk, \perp)$. This change is indistinguishable by the rerandomizable property of $\mathcal{E}$.
- Hyb_2 For $1 < i < c, 1 \leq b \leq B$: $v_{c,i}'^b = \mathsf{Enc}(pk, \perp)$. 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 $2 \leq j \leq d, 1 \leq i \leq (m-1)B$: $\mathsf{ct}_j^i = \mathsf{ParDec}(sk_j, \mathsf{ct}_{j-1}''^i), \mathsf{ct}_j'^i = \mathsf{ReRand}(pk_{A_j}, \mathsf{ct}_j^i)$, $\vec{\mathsf{ct}}_j'' = \pi_j(\vec{\mathsf{ct}}_j')$.
- Hyb_1 For $2 \leq j \leq d, 1 \leq i \leq (m-1)B$: $\mathsf{ct}_j^i = \mathsf{ParDec}(sk_j, \mathsf{ct}_{j-1}^i)$. $\vec{\mathsf{ct}}_d' = \mathsf{ReRand}(pk_{A_d}, \vec{\mathsf{ct}}_d)$, $\vec{\mathsf{ct}}_d'' = \pi(\vec{\mathsf{ct}}_d')$, 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 $\forall x_i \in Y = \bigcup_{j=1}^m X_j$, $\mathsf{ct}_1^i = \mathsf{Enc}(pk, x_i)$, $1 \leq i \leq |Y|$.
  - 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.

E Complexity Analyses and Comparisons

E.1 Complete Analysis of LG

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

mq-ssPMT. The cost of this stage 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 per party is $O((T+l-\log n)mn\log((T+l-\log n)n))$. The communication complexity per party is $O(t\lambda m\log(((T+l-\log n)n)/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity per party is $O((T+l-\log n)mn)$. The communication complexity per 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 $(m-1)$ instances of ROT extension of size B.

– **Offline phase.** The computation complexity per party is $O(mn\log n)$. The communication complexity per party is $O(t\lambda m\log(n/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity per party is $O(mn)$. The communication complexity per 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.(cf. Appendix E.3).

Total costs.

- **Offline phase.** The computation complexity per party is $O((T+l-\log n)mn\log((T+l-\log n)n)+m^2n\log mn)$. The communication complexity per 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 $O(\log(l-\log n)+m)$.

E.2 Theoretical Analysis of Batch ssPMT

Complete Analysis The costs of each stage in Π_{bssPMT} (Figure 8) 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, 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.

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

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

Secret-shared private equality tests. Like [PSTY19], we instantiate ssPEQT (Figure 4) 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].

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

Comparison with mq-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 secret-shared oblivious decryption-then-matching (ssVODM) [ZCL+23], which is to implement a decryption circuit and a comparison circuit by the GMW protocol [GMW87]. In total, the ssVODM 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 batch ssPMT of size $B = 1.27n$[26] consist of the costs of batch OPPRF of size $1.27n$ and the costs of $1.27n$ instances of ssPEQT, where batch OPPRF can be instantiated by extremely fast specialized protocol and ssPEQT is implemented by the GMW protocol. Moreover, the state-of-the-art batch OPPRF construction [CGS22, RS21, RR22] can achieve linear computation and

[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 $O(1)$

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

communication with respect to n . In ssPEQT, the parties engage in $1.27(\gamma-1)n$ AND gates, where γ is the output length of OPPRF. 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 mq-ssPMT.

E.3 Theoretical Analysis of Our SK-MPSU

Complete Analysis The costs of each stage in $\Pi_{\mathsf{SK\text{-}MPSU}}$ (Figure 11) 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 per party is $O(\gamma mn \log n)$. The communication complexity per party is $O(t\lambda m \log(\gamma n/t))$. The round complexity is $O(1)$.
- **Online phase.** The computation complexity per party is $O(mn)$. The communication complexity per party is $O(\gamma mn)$. 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, where each OT execution consists of two parts: In the offline phase, the parties invoke random-choice-bit ROT, then $\mathcal{S}$ sends one messages of length $l + \kappa$ ($\kappa = \sigma + \log(m-1) + \log n$) to $\mathcal{R}$; In the online phase, $\mathcal{R}$ sends a 1-bit derandomization message to transform ROT into a chosen-input version. If P_i and P_j hold choice bits, then P_i and P_j invoke OT with the parties in $\{P_2, \cdots, P_j\}$ except itself.

For $1 \leq i < j \leq m$, $P_{min(2,i)}, \cdots, P_j$ engage in B instances of mss-ROT, where P_i and P_j hold the two choice bits. Considering the overall invocations of mss-ROT: P_1 invokes $[\sum_{j=2}^{m}(j-1)]B = \frac{m^2B - mB}{2}$ instances of two-party OT; For $1 < i \leq m$, each party P_i invokes $[(i-1) + \sum_{j=i+1}^{m} 3(j-2)]B = \frac{3m^2B - 3i^2B - 9mB + 11iB - 2B}{2}$ instances of two-party OT.

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

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

Comparison with LG We present a comparison of the theoretical computation and communication complexity for each party in both offline and online phases between LG and our SK-MPSU.

E.4 Theoretical Analysis of Our PK-MPSU

Complete Analysis The costs of each stage in $\Pi_{\textsf{PK-MPSU}}$ (Figure 12) are calculated as follows.

Batch ssPMT The cost of this stage is the same as that in $\Pi_{\textsf{SK-MPSU}}$ (cf. Appendix E.4).

	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 3. Asymptotic communication (bits) and computation costs of LG 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, $t \approx 128$.

	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 4. Asymptotic communication (bits) and computation costs of [GNT23] and our PK-MPSU 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, $t \approx 128$.

– **Offline phase.** The computation complexity per party is $O(\gamma mn\log n)$. The communication complexity per party is $O(t\lambda m\log(\gamma n/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity per party is $O(mn)$. The communication complexity per 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 $1 \le i < j \le m$, 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 per party is $O(mn\log n)$. The communication complexity per party is $O(t\lambda m\log(n/t))$. The round complexity is $O(1)$.
– **Online phase.** The computation complexity per party is $O(mn)$ public-key operations. The communication complexity per 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 per party is $O(mn)$ public-key operations. The communication complexity per party is $O(\lambda mn)$. The round complexity is $O(m)$.

Total costs.

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

Comparison with [GNT23] In Table 4, 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 4.

F Multi-Party Private-ID

It is natural to 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 identifiers. Each party receives identifiers associated with its own input elements, 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 13. We build a concrete multi-party private-ID protocol based on the DDH assumption (described in Figure 14) by extending the "distributed OPRF+PSU" paradigm [CZZ+24b] to multi-party setting.

Theorem 5. *Protocol Π_{MPID} securely implements $\mathcal{F}_{\mathsf{MPID}}$ against any semi-honest adversary corrupting $t < m$ parties in the $\mathcal{F}_{\mathsf{mpsu}}$-hybrid model, assuming the DDH assumption and H is a random oracle.*

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

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

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 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 $\mathsf{H}(x_i^j)^{a_i k_{i+1} \cdots k_m k_1 \cdots k_{i-1}}$, $j \in [n]$, and 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 $\{\mathsf{H}(x_i^1)^{k_1 \cdots k_m}, \mathsf{H}(x_i^n)^{k_1 \cdots k_m}\}$ as $R_i = \{r_i^1, \cdots, r_i^n\}$, where each $r_i^j \in \mathbb{G}$.
3. The parties invoke $\mathcal{F}_{\mathsf{mpsu}}$ 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. 14. DH-based Multi-Party Private ID Π_{MPID}

own PRF key k_i along with 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 Figure 14, each party P_i first masks the hash value of each element by raising it to the power of a_i. Then these masked values are then cycled among the parties, with each party raising them to the power of its key. Finally, P_i unmasks the values and raises them to the power of its won key k_i, resulting in the distributed PRF values $\mathsf{PRF}_{k_1,\cdots,k_m}(x) = \mathsf{H}(x)^{k_1 \cdots k_m}$. Even if $m-1$ parties collude, there still exists one exponent private to the adversary, ensuring that the PRF values remain pseudorandom according to the DDH assumption.

In the step 3, the parties invoke an MPSU protocol on their distributed PRF values, so than they can securely receive all PRF values on the union of input sets.

Security. Assuming H is a random oracle, the security of protocol follows immediately from the DDH assumption and the security of our PK-MPSU.

Modern DH-based cryptosystems often use elliptic curves for the underlying cyclic group, due to their compact size and better efficiency at a given level of security. In this context, the first two steps of the protocol have relatively low communication costs, meanwhile, the element space of the MPU protocol in step 3 is EC points, which aligns perfectly with the element space of our PK-MPSU. This alignment allows us to instantiate the MPSU functionality using our PK-MPSU, leveraging its lowest communication costs. As a result, the entire multi-party private-ID protocol achieves extremely low communication costs, making it well-suited for deployment in bandwidth-restricted environments.

Another benefit of instantiating the MPSU functionality with our PK-MPSU is to achieve linear computation and communication complexity. Since the multi-party distributed OPRF phase has linear computation and communication, by integrating our PK-MPSU, the entire multi-party private-ID protocol can maintain the linear complexities throughout.