

# Multi-Party Private Set Operations from Predicative Zero-Sharing

Minglang Dong<sup>1</sup>, Yu Chen<sup>1</sup>, Cong Zhang<sup>2</sup>, Yujie Bai<sup>1</sup>, and Yang Cao<sup>1</sup>

<sup>1</sup> School of Cyber Science and Technology, Shandong University, Qingdao 266237, China

<sup>2</sup> Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China  
{minglang\_dong,baiyujie}@mail.sdu.edu.cn, yuchen@sdu.edu.cn,  
zhangcong@mail.tsinghua.edu.cn

**Abstract.** Typical protocols in the multi-party private set operations (MPSO) setting enable  $m > 2$  parties to perform certain secure computation on the intersection or union of their private sets, realizing a very limited range of MPSO functionalities. Most works in this field focus on just one or two specific functionalities, resulting in a large variety of isolated schemes and a lack of a unified framework in MPSO research. In this work, we present an MPSO framework, which allows  $m$  parties, each holding a set, to securely compute any set formulas (arbitrary compositions of a finite number of binary set operations, including intersection, union and difference) on their private sets. Our framework is highly versatile and can be instantiated to accommodate a broad spectrum of MPSO functionalities. To the best of our knowledge, this is the first framework to achieve such a level of flexibility and generality in MPSO, without relying on generic secure multi-party computation (MPC) techniques. Our framework exhibits favorable theoretical and practical performance. The computation and communication complexity scale linearly with the set size  $n$ , and it achieves optimal complexity that is on par with the naive solution for widely used functionalities, such as multi-party private set intersection (MPSI), MPSI with cardinality output (MPSI-card), and MPSI with cardinality and sum (MPSI-card-sum), in the standard semi-honest model. Furthermore, the instantiations of our framework mainly from symmetric-key techniques yield efficient protocols for MPSI, MPSI-card, MPSI-card-sum, and multi-party private set union (MPSU), with online performance surpassing or matching the state of the art. At the technical core of our framework is a newly introduced primitive called predicative zero-sharing. This primitive captures the universality of a number of MPC protocols and is composable. We believe it may be of independent interest.

## 1 Introduction

In the setting of multi-party private set operations (MPSO), a set of  $m$  ( $m > 2$ ) parties, each holding a private set of items, wish to perform secure computation on their private sets without revealing any additional information. In the end,

only one of the parties (denoted as the leader) learns the resulting set and other parties (denoted as clients) learn nothing. MPSO is an expansive research field with a variety of rich functionalities. The typical functionalities that have been studied in the MPSO literature can be divided into two categories:

- Multi-party private set intersection (MPSI) [22, 36, 54, 39, 53, 10, 32, 38, 34, 27, 26, 41, 15, 30, 8, 57, 6, 60, 58], which is to compute intersection, and its variants — MPSI with cardinality output (MPSI-card) [36, 10, 18, 25, 28], which is to compute intersection cardinality, MPSI with cardinality and sum (MPSI-card-sum) [18, 28], which is to compute intersection cardinality and sum (of the associated payloads), and circuit-MPSI [36, 10, 15, 60, 56], which allows parties to learn secret shares of the indicator vector for intersection with respect to leader’s elements, that can be further fed into generic MPC (with leader’s elements) to compute arbitrary function on intersection;
- Multi-party private set union (MPSU) [36, 23, 10, 55, 57, 40, 24, 19], which is to compute union, and its variants — MPSU with cardinality output (MPSU-card) [10], which is to compute union cardinality, and circuit-MPSU [10], which allows parties to learn secret shares of elements in union, that can be further fed into generic MPC to compute arbitrary function on union.

There are several major problems in the field of MPSO:

- **Unrealistic security assumptions.** Despite the vast body of existing works in the MPSO literature, many rely on assumptions of unconditional trust, which is fraught with security risks. For example, some works [40, 60, 25, 56] assume non-collusion among particular parties, which is unlikely to hold in practice. Therefore, an important theme in the MPSO research is to achieve security against arbitrary collusion.
- **Unsatisfied application needs.** Many real-world applications want to compute more than intersection and union (or partial/aggregate information on them), however, the existing protocols cannot meet these demands, both in terms of functionality and efficiency. For instance, a social services organization intends to determine the list of people on welfare with cancer [36]. To fulfill this task, all hospitals should collectively calculate the union of lists of cancer patients, meanwhile keeping the union privacy, then an intersection operation between the unrevealed union of cancer patients and the welfare rolls is performed. This problem is apparently beyond the above two categories and still lacks practical solutions to date.
- **Fragmented landscape of protocols.** Most existing works focus on only one or two specific functionalities, resulting in a large variety of isolated schemes and a lack of a unified framework in MPSO.

Given the numerous possible compositions of a finite number of binary set operations (including intersection, union and difference) on  $m$  sets, ideally, MPSO should enable  $m$  parties to securely compute arbitrary set formulas on their private sets. All the aforementioned functionalities are special cases of this generic functionality (hereafter, we use MPSO to refer particularly to this generic functionality). The seminal work of Kissner and Song [36] has explored the MPSO

functionality. Unfortunately, they failed to fully realize it. The set formulas being computed in their protocol only allow to include union and intersection set operations, excluding the difference operation. Namely, their protocol only realizes a restricted MPSO functionality. For instance, computing  $X_1 \setminus (X_2 \cap X_3)$  is not feasible in their protocol. Furthermore, their protocol relies heavily on additively homomorphic encryption (AHE) and high-degree polynomial calculations, leading to prohibitively large computational costs, hence is totally impractical.

A follow-up work by Blanton and Aguiar [10] redesigns the circuits for computing intersection, union and difference as oblivious sorting and adjacent comparisons in a sorted set, and implements these circuits using generic MPC protocols. Thanks to the composability of generic MPC, their protocols are also composable to compute arbitrary compositions of binary set operations, thereby fully realizing the MPSO functionality. However, the protocol’s heavy reliance on generic MPC incurs substantial computational costs, and even the simplest cases — MPSI and MPSU — exhibit poor practical efficiency. For instances, their experiments report a runtime of 24.8 seconds for both MPSI and MPSU with 3 parties, each holding  $2^{11}$  items of 32 bits, which is the largest experiment in terms of the number of parties and set size reported in their paper. Moreover, their protocols are only secure in the honest majority setting.

Motivated by the above, we raise the following question:

*Can we fully realize the MPSO functionality with security against arbitrary collusion and acceptable performance in the semi-honest model?*

### 1.1 Our Contributions

In this work, we answer the above question affirmatively. Our technical route is: First, defining a predicate formula representation for any set formulas; Second, presenting a composable primitive — predicative zero-sharing — and its composition technique; Then, instantiating predicative zero-sharing as a primitive tailored for MPSO — membership zero-sharing — with lightweight building blocks; Finally, constructing a framework based on oblivious transfer (OT) and symmetric-key operations in the standard semi-honest model, which fully realizes not only the MPSO functionality, but also the extended MPSO-card and circuit-MPSO functionalities. Our contributions can be detailed as follows:

**Predicate Formula Representation.** The first challenge in realizing MPSO is to identify a suitable representation for any set formulas, which determines the generality and practicality of the resulting framework. The prior work [36] represents set formulas using intersection, union, and element reduction operations, whose arbitrary compositions can only express a limited subset of set formulas, thereby restricting its generality. The follow-up work [10] adopts the naive representation based on intersection, union, and difference operations to achieve full generality. However, to support composability, it relies heavily on generic MPC, which significantly hinders practicality. In this work, we introduce a new representation called canonical predicate formula (CPF), which is designed with

a particular structure to enable an MPSO framework achieving the best of both worlds: generality and practicality. Specifically, this representation is a subset of first-order set predicate formulas (which are first-order predicate formulas where each atomic proposition is a set membership predicate  $x \in X_i$ , connected by AND, OR and NOT operators), defined as a disjunction of several subformulas that are in a certain form, representing a partition of the desired set. We prove that any set formulas can be transformed into CPF representations, and the number of subformulas in CPF dominates the performance of protocols.

**Predicative Zero-Sharing and Relaxation.** The second challenge is to devise a composable primitive based on our predicate formula representation. We introduce a novel primitive called predicative zero-sharing, which is a family of protocols, each associated with a first-order predicate formula and encoding the truth-value of the formula on the parties' inputs into a secret-sharing over a finite field among the parties. Specifically, if the formula is true, the parties hold a secret-sharing of 0, otherwise a secret sharing of a random value. We put forward a simpler simulation-based security definition for predicative zero-sharing protocols, which is composed of three requirements: correctness, privacy and independence, and give a rigorous proof of its equivalence to the standard security definition for a broader class of MPC protocols (predicative zero-sharing is its subset). This simpler security definition simplifies the security proof of our predicative zero-sharing protocols. Moreover, under this simpler security definition, we can relax the security of predicative zero-sharing by removing the independence requirement. This relaxed version of predicative zero-sharing admits the abstraction of much more MPC protocols, such as random oblivious transfer (ROT), equality-conditional randomness generation (ECRG) [35],<sup>3</sup> and so on. We present a composition technique to compose several relaxed predicative zero-sharing protocols into a single relaxed predicative zero-sharing protocol based on AND and OR operators. We also present a transformation technique to transform any relaxed predicative zero-sharing protocol into a standard version. Combining these two techniques, we can construct predicative zero-sharing for any first-order predicate formulas, from relaxed predicative zero-sharing associated with all literals (atomic propositions or their negations) within the formula.

**Membership Zero-Sharing.** To enable the instantiation of predicative zero-sharing, we introduce membership zero-sharing, a particular class of predicative zero-sharing tailored for MPSO, by specifying the associated predicate formula as a first-order set predicate formula  $Q$ . In this setting, one party (denoted as  $P_{\text{pivot}}$ ) inputs an element and the other parties input sets. The output secret-sharing among the parties encodes whether  $P_{\text{pivot}}$ 's input element, together with all input sets, satisfy  $Q$ . For example, consider 3 parties where  $P_1$  inputs an element  $x$ ,  $P_2$  inputs a set  $X_2$ , and  $P_3$  inputs a set  $X_3$ . Suppose  $Q$  is in the form of  $x \in X_2 \wedge x \notin X_3$ , if  $x \in X_2 \setminus X_3$ ,  $P_1, P_2, P_3$  hold a secret-sharing of

<sup>3</sup> We found that the ECRG functionality satisfies the definition of predicative zero-sharing while the construction in [35] only achieves the security of relaxed predicative zero-sharing. This is because ECRG is a probabilistic functionality whereas [35] proved its security using the definition for deterministic functionalities.

0, otherwise they hold a secret-sharing of a random value. Given that any first-order set predicate formula  $Q$  is only composed of two types of literals — set membership predicates  $x \in Y$  and the negations  $x \notin Y$ , by instantiating relaxed membership zero-sharing associated with  $x \in Y$  and  $x \notin Y$  respectively, we can build membership zero-sharing protocols for any first-order set predicate formulas, by following the recipe for predicative zero-sharing. Our instantiations are both built on lightweight components, including oblivious programmable pseudo-random function (OPPRF), batch secret-shared private membership test (batch ssPMT), and ROT. This contributes to the good efficiency of our framework.

**MPSO, MPSO-card and Circuit-MPSO.** In analogy with MPSI (resp. MPSU) functionality to MPSI-card and circuit-MPSI (resp. MPSU-card and circuit-MPSU), we extend MPSO functionality into two new functionalities — MPSO-card and circuit-MPSO, where MPSO-card computes the resulting set’s cardinality and circuit-MPSO reveals secret shares of the resulting set, which can be further fed into generic MPC to compute arbitrary function on the resulting set. Based on the CPF representation for any set formulas and membership zero-sharing for any first-order set predicate formulas, we put forth a framework fully realizing MPSO, MPSO-card and circuit-MPSO functionalities. A high level of our framework proceeds as follows. We begin with the simplest case, where the desired set is a subset of the input set of the leader.<sup>4</sup> In this case, the leader acts as  $P_{\text{pivot}}$ , and for each element in its input set, the leader invokes the membership zero-sharing associated with the CPF representation of the desired set, with the other parties inputting their sets. As a result, for each elements in the leader’s set that belongs to the desired set, the parties hold a secret-sharing of 0. Since all these elements exactly compose the desired set, the parties can simply reconstruct all secret-sharings to the leader, who computes the resulting set by identifying all elements with corresponding secrets as 0. This construction can be optimized using the hashing to bins technique (see Figure). We extend this simplest case of our framework to achieve full MPSO functionality, by leveraging the structural properties of our CPF representation, which guarantee that the set represented by each subformula  $Q_i$  in the CPF is a subset of some party  $P_j$ ’s input set, and all these sets form a partition of the desired set. For each subformula  $Q_i$ , the parties invoke membership zero-sharing with  $P_j$  acting as  $P_{\text{pivot}}$ , and the invocation is similar to the simplest case. After the membership zero-sharing invocations for all subformulas, the union of all output secret-sharings encode a partition of the desired set. However, a straightforward reconstruction in this setting may reveal information through the order of secret-sharings, therefore, the parties have to invoke a multi-party secret-shared shuffle protocol to randomly permute and re-share all secret-sharings. Finally, the shuffled secret-sharings are reconstructed to the leader. Since the resulting set remains being secret-shared before the last reconstruction step, this MPSO protocol is easy to be extended to MPSO-card and circuit-MPSO protocols.

---

<sup>4</sup> MPSI is a typical example of this case, as the intersection is a subset of any input sets.

In addition to the above contributions, perhaps surprisingly, we also make independent contributions in the following sub-fields, by instantiating our framework to yield the aforementioned typical protocols:

**MPSI.** The MPSI protocol from our framework has the best computation and communication complexity among all MPSI protocols based on OT and symmetric-key operations in the standard semi-honest model. Particularly, this is the first MPSI construction to achieve the optimal complexity that is on par with the naive solution (the leader’s computation and communication complexity are both  $O(mn)$  and each client’s computation and communication complexity are both  $O(n)$ , where  $n$  is the set size and  $m$  is the number of parties) without extensive use of public-key operations, in standard semi-honest model. The previous MPSI protocol [38] with this optimal complexity is only secure in the weaker augmented semi-honest model. In this work, we close this gap. Our MPSI protocol is also highly efficient in the online phase, which requires only 8.9 seconds and 738 MB of communications for 10 parties with sets of  $2^{20}$  items each, regardless of the item length, while the state-of-the-art MPSI protocol [58] requires 32.9 seconds and 1921 MB of communications for their full protocol.

**MPSI-card, MPSI-card-sum and Circuit-MPSI.** The MPSI-card and MPSI-card-sum protocols from our framework are the first MPSI-card and MPSI-card-sum constructions with the optimal computation and communication complexity in the standard semi-honest model. Our MPSI-card has  $14.0 - 20.3\times$  lower communication than the state-of-the-art MPSI-card protocol [18]. We provide the first MPSI-card-sum implementation and it only doubles the computation and communication costs of our MPSI while realizing a richer functionality. Concretely, our MPSI-card requires only 9.2 seconds while our MPSI-card-sum requires 16.7 seconds in online phase for 10 parties with sets of  $2^{20}$  items each, regardless of the item length. Additionally, the circuit-MPSI protocol from our framework is the first circuit-MPSI construction in dishonest majority setting.

**MPSU.** The MPSU protocol from our framework has the best computation and communication complexity among all MPSU protocols based on OT and symmetric-key operations in the standard semi-honest model. It could be seen as an instance of the secret-sharing based MPSU paradigm, which abstracts all existing MPSU protocols relying only on symmetric-key primitives [40, 19]. Our protocol achieves the optimal complexity of this paradigm for the first time (with  $O(m^2n)$  computation and communication complexity of leader and  $O(mn)$  computation and communication complexity of clients). Our MPSU protocol is comparable with the state-of-the-art MPSU protocol [19]. Concretely, it requires only 13.2 seconds in online phase for 5 parties, each holding  $2^{20}$  items of 64 bits.

**MPSU-card and circuit-MPSU.** The MPSU-card and circuit-MPSU protocols from our framework are the only efficient constructions for MPSU-card and circuit-MPSU, with performance that is nearly the same as our MPSU protocol.

## 1.2 Related Work

Despite the immense amount of existing works on the typical functionalities in this field, many are insecure against arbitrary collusion [39, 10, 55, 15, 40,

60, 25, 56], or have non-negligible false positives [6, 57]. We only focus on works achieving semi-honest security against arbitrary collusion without non-negligible false positives. Distribution of research attention among these works is extremely imbalanced: MPSI has been extensively studied [22, 36, 54, 53, 32, 38, 34, 27, 26, 41, 30, 8, 58], while MPSU only receives relatively little attention [36, 23, 24, 19]. MPSI-card [36, 18, 28] and MPSI-card-sum [18, 28] are extremely understudied sub-fields, with only a couple of secure MPSI-card protocol [36, 18, 28] and MPSI-card-sum protocol [18, 28] against arbitrary collusion. Even worse, no prior work has realized circuit-MPSI, MPSU-card, circuit-MPSU in the dishonest majority setting. We provide more details on the classic and state-of-the-art protocols below. A comprehensive theoretical comparison between related protocols and ours is provided in Appendix A.

**MPSI.** Freedman et al. [22] introduced the first MPSI protocol based on oblivious polynomial evaluation (OPE), which is implemented using AHE. Kissner and Song [36] proposed an MPSI protocol using the OPE technique along with the polynomial representations. These two protocols both require quadratic computation complexity with respect to the set size  $n$  for each party, resulting in complete impracticality.

Kolesnikov et al. [38] proposed two MPSI protocols in the augmented semi-honest model and standard semi-honest model, respectively. The former achieves the optimal complexity of MPSI, while it is only secure in the augmented semi-honest model. The latter fails to achieve optimal complexity as it requires the clients' complexity to depend on the corruption threshold  $t$ . Garimella et al. [26] improved these protocols using oblivious key-value store (OKVS) [43, 26, 48, 9] and showed that the augmented semi-honest protocol actually enjoys malicious security. Following these works, Nevo et al. [41] proposed an efficient MPSI protocol in the malicious model, where the client's communication complexity depends only on  $n$  (while the computation complexity still depends on  $t$ ).

Inbar et al. [34] proposed two MPSI protocols in the augmented semi-honest and standard semi-honest model, based on OT and garbled Bloom filters. In these two protocols, each party's computation complexity is  $O(mn)$ . The Ben-Efraim et al. [8] extended the former to the malicious model.

Recently, Wu et al. [58] proposed two semi-honest MPSI protocols based on OPRF and OKVS. In these two protocols, the client's complexity depends on  $t$ . **MPSI-card and MPSI-card-sum.** Chen et al. [18] proposed the first MPSI-card and MPSI-card-sum protocols based on OT and symmetric-key operations, which are also the only practical MPSI-card and MPSI-card-sum protocols in the standard semi-honest model. In their protocols, the leader's complexity is  $O(mn + tn \log n)$  and the client's complexity is  $O(tn)$ .

**MPSU.** Kissner and Song [36] introduced the first MPSU protocol, based on polynomial representations and AHE. The substantial number of AHE operations and high-degree polynomial calculations incur unacceptable efficiency.

Recently, Gao et al. [24] proposed an MPSU protocol in the standard semi-honest model. This protocol relies on public-key operations and has super-linear computation and communication complexity for each party in term of  $n$ .



Recently, Dong et al. [19] proposed two MPSU protocols in the standard semi-honest model. The first protocol, based on OT and symmetric-key operations, eliminates the non-collusion assumption in [40], at the cost of increasing client's complexity to quadratic in terms of  $m$ . The second protocol achieves linear complexity. However, it relies on public-key operations with a lower efficiency.

## 2 Preliminaries

### 2.1 Notation

Let  $m$  denote the number of parties. We use  $P_i$  ( $1 \leq i \leq m$ ) to denote the parties,  $X_i$  to represent the sets they hold, where each set has  $n$   $l$ -bit elements.  $[x] = (x_1, \dots, x_m)$  denotes an additive secret-sharing among  $m$  parties, i.e., each  $P_i$  holds a share  $x_i$  such that  $x_1 + \dots + x_m = x$ .  $x \parallel y$  denotes the concatenation of two strings. We use  $\lambda, \sigma$  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. For a vector  $\mathbf{a}$ ,  $a_i$  denotes the  $i$ -th component,  $\text{HW}(\mathbf{a})$  denotes the hamming weight of  $\mathbf{a}$ ,  $\text{zero}(\mathbf{a})$  denotes the number of 0 in  $\mathbf{a}$ , and  $\pi(\mathbf{a}) = (a_{\pi(1)}, \dots, a_{\pi(n)})$ , where  $\pi$  is a permutation over  $n$  items. The notation  $\mathbf{a} \oplus \mathbf{b}$  denotes a component-wise XOR, i.e.,  $(a_1 \oplus b_1, \dots, a_n \oplus b_n)$ .

### 2.2 Security Model

In this work, we consider semi-honest and static adversaries  $\mathcal{A}$  with the capability to corrupt an arbitrary subset of parties. To capture the security of a protocol in the simulation-based model [29, 14], we use the following notations:

- Let  $f = (f_1, \dots, f_m)$  be a probabilistic polynomial-time  $m$ -ary functionality and let  $\Pi$  be a  $m$ -party protocol for computing  $f$ .
- The view of  $P_i$  ( $1 \leq i \leq m$ ) during an execution of  $\Pi$  on all parties' inputs  $\mathbf{x} = (x_1, \dots, x_m)$  is denoted by  $\text{View}_i^\Pi(\mathbf{x})$ , including the  $i$ -th party's input  $x_i$ , its internal random tape and all messages that it received.
- The output of  $P_i$  during an execution of  $\Pi$  on  $\mathbf{x}$  is denoted by  $\text{Output}_i^\Pi(\mathbf{x})$ . The joint output of parties is  $\text{Output}^\Pi(\mathbf{x}) = (\text{Output}_1^\Pi(\mathbf{x}), \dots, \text{Output}_m^\Pi(\mathbf{x}))$ .

**Definition 1.** We say that  $\Pi$  securely computes  $f$  in the presence of  $\mathcal{A}$  if there exists a PPT algorithm  $\text{Sim}$  s.t. for every  $\mathbf{P}_\mathcal{A} = \{P_{i_1}, \dots, P_{i_t}\} \subset \{P_1, \dots, P_m\}$ ,

$$\{\text{Sim}(\mathbf{P}_\mathcal{A}, \mathbf{x}_\mathcal{A}, f_\mathcal{A}(\mathbf{x})), f(\mathbf{x})\}_x \overset{c}{\approx} \{\text{View}_\mathcal{A}^\Pi(\mathbf{x}), \text{Output}^\Pi(\mathbf{x})\}_x,$$

where  $\mathbf{x}_\mathcal{A} = (x_{i_1}, \dots, x_{i_t})$ ,  $f_\mathcal{A} = (f_{i_1}, \dots, f_{i_t})$ ,  $\text{View}_\mathcal{A}^\Pi(\mathbf{x}) = (\text{View}_{i_1}^\Pi(\mathbf{x}), \dots, \text{View}_{i_t}^\Pi(\mathbf{x}))$ .

### 2.3 Multi-party Private Set Operations

MPSO is a special case of secure multi-party computation (MPC). Figure 1 formally defines the typical ideal functionalities computing the intersection, intersection cardinality, intersection sum with cardinality, union, and union cardinality over the parties' private sets.



**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_1$  is the leader. Size  $n$  of input sets. The bit length  $l$  of set elements. The mapping function  $\text{payload}_i()$  from  $P_i$ 's elements to the associated payloads.

**Functionality:** On input  $X_i = \{x_i^1, \dots, x_i^n\} \subseteq \{0, 1\}^l$  from  $P_i$ ,

- **MPSI.** give the intersection  $\bigcap_{i=1}^m X_i$  to  $P_1$ .
- **MPSI-card.** give the intersection cardinality  $|\bigcap_{i=1}^m X_i|$  to  $P_1$ .
- **MPSI-card-sum.** give the intersection cardinality  $|\bigcap_{i=1}^m X_i|$  to each  $P_i$  for  $1 \leq i \leq m$ , and give  $\sum_{x \in \bigcap_{i=1}^m X_i, 1 \leq j \leq m} \text{payload}_j(x)$  to  $P_1$ .
- **MPSU.** give the union  $\bigcup_{i=1}^m X_i$  to  $P_1$ .
- **MPSU-card.** give the union cardinality  $|\bigcup_{i=1}^m X_i|$  to  $P_1$ .

Fig. 1. Typical Functionalities in MPSO

## 2.4 Random Oblivious Transfer

Oblivious transfer (OT) [47] is a foundational primitive in MPC, the functionality of 1-out-of-2 random OT (ROT) is given in Figure 2.

**Parameters.** Sender  $\mathcal{S}$ , Receiver  $\mathcal{R}$ . A field  $\mathbb{F}$ .

**Functionality.** On input  $e \in \{0, 1\}$  from  $\mathcal{R}$ , sample  $r_0, r_1 \leftarrow \mathbb{F}$ . Give  $(r_0, r_1)$  to  $\mathcal{S}$  and give  $r_e$  to  $\mathcal{R}$ .

Fig. 2. 1-out-of-2 Random OT Functionality  $\mathcal{F}_{\text{ROT}}$ 

## 2.5 Batch Oblivious Programmable Pseudorandom Function

Oblivious pseudorandom function (OPRF) [21, 17, 51] is a central primitive in the area of PSO. Kolesnikov et al. [37] introduced batched OPRF, which provides a batch of OPRF instances. In the  $i$ -th instance, the sender  $\mathcal{S}$  learns a PRF key  $k_i$ , while the receiver  $\mathcal{R}$  inputs  $x_i$  and learns  $\text{PRF}(k_i, x_i)$ .

Oblivious programmable pseudorandom function (OPPRF) [38, 45, 16, 51, 48] 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 that outputs programmed values on a certain programmed set of inputs is called programmable PRF (PPRF) [45].  $\mathcal{R}$  evaluates 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  $\gamma$  of values.

**Sender's inputs.**  $\mathcal{S}$  inputs  $B$  sets of key-value pairs including:

- Disjoint key sets  $K_1, \dots, K_B$ .
- The value sets  $V_1, \dots, V_B$ , where  $|K_i| = |V_i|$ ,  $i \in [B]$ .

**Receiver's inputs.**  $\mathcal{R}$  inputs  $B$  queries  $\mathbf{x} \subseteq (\{0, 1\}^l)^B$ .

**Functionality:** On input  $(K_1, \dots, K_B)$  and  $(V_1, \dots, V_B)$  from  $\mathcal{S}$  and  $\mathbf{x} \subseteq (\{0, 1\}^l)^B$  from  $\mathcal{R}$ ,

- Generate a uniform PPRF key  $k_i$  and an auxiliary information  $\text{hint}_i$  for  $i \in [B]$ ;
- Give vector  $\mathbf{k} = (k_1, \dots, k_B)$  and  $(\text{hint}_1, \dots, \text{hint}_B)$  to  $\mathcal{S}$ .
- Sample a PPRF  $F : \{0, 1\}^* \times \{0, 1\}^l \rightarrow \{0, 1\}^\gamma$  such that  $F(k_i, K_i(j)) = V_i(j)$  for  $i \in [B], 1 \leq j \leq |K_i|$ ;
- Define  $f_i = F(k_i, x_i)$ , for  $i \in [B]$ ;
- Give vector  $\mathbf{f} = (f_1, \dots, f_B)$  to  $\mathcal{R}$ .

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

## 2.6 Batch Secret-Shared Private Membership Test

Batch secret-shared private membership test (batch ssPMT) [19] is a two-party protocol that implements multiple instances of ssPMT [16, 40] between a sender  $\mathcal{S}$  and a receiver  $\mathcal{R}$ . Given a batch size of  $B$ ,  $\mathcal{S}$  inputs  $B$  sets  $X_1, \dots, X_B$ , while  $\mathcal{R}$  inputs  $B$  elements  $x_1, \dots, x_B$ . As a result,  $\mathcal{S}$  and  $\mathcal{R}$  receive secret shares of a bit vector of size  $B$ , where the  $i$ -th bit is 1 if  $x_i \in X_i$ , 0 otherwise. The batch ssPMT functionality is given in Figure 4. Dong et al. [19] proposed an efficient construction with linear complexities, based on batch OPPRF and secret-shared private equality test (ssPEQT) [45, 16].

**Parameters.** Sender  $\mathcal{S}$ . Receiver  $\mathcal{R}$ . Batch size  $B$ . The bit length  $l$  of set elements.

**Inputs.**  $\mathcal{S}$  inputs  $B$  disjoint sets  $X_1, \dots, X_B$  and  $\mathcal{R}$  inputs  $\mathbf{x} \subseteq (\{0, 1\}^l)^B$ .

**Functionality.** On inputs  $X_1, \dots, X_B$  from  $\mathcal{S}$  and input  $\mathbf{x}$  from  $\mathcal{R}$ , for  $1 \leq i \leq B$ , sample two random bits  $e_S^i, e_R^i$  under the constraint that if  $x_i \in X_i$ ,  $e_S^i \oplus e_R^i = 1$ , otherwise  $e_S^i \oplus e_R^i = 0$ . Give  $\mathbf{e}_S = (e_S^1, \dots, e_S^B)$  to  $\mathcal{S}$  and  $\mathbf{e}_R = (e_R^1, \dots, e_R^B)$  to  $\mathcal{R}$ .

Fig. 4. Batch ssPMT Functionality  $\mathcal{F}_{\text{bssPMT}}$ 

## 2.7 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 5. Eskandarian et al. [20] proposed an online-efficient protocol where the parties generate share correlations in the offline phase, so that the leader’s online complexity scales linearly with  $n$  and  $m$ , while the clients’ online complexity scales linearly with  $n$  and is independent of  $m$ .

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

**Fig. 5.** Multi-Party Secret-Shared Shuffle Functionality  $\mathcal{F}_{\text{shuffle}}$

## 2.8 Hashing to Bins

The hashing to bins technique was introduced by Pinkas et al. [46, 44] 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 [42], so that each bin has at most one item.<sup>5</sup> 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  $b$ -th 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  $b$ -th of simple hash table certainly contains  $x$ .

We denote simple hashing with the following notation:

$$\mathcal{T}^1, \dots, \mathcal{T}^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X)$$

This expression represents hashing the items of  $X$  into  $B$  bins using simple hashing with hash functions  $h_1, h_2, h_3 : \{0, 1\}^* \rightarrow [B]$ . The output is a hash table denoted by  $\mathcal{T}^1, \dots, \mathcal{T}^B$ , where for each  $x \in X$ ,  $\mathcal{T}^{h_i(x)} \supseteq \{x \parallel i \mid i = 1, 2, 3\}$ .<sup>6</sup>

We denote Cuckoo hashing with the following notation:

$$\mathcal{C}^1, \dots, \mathcal{C}^B \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^B(X)$$

This expression represents hashing the items of  $X$  into  $B$  bins using Cuckoo hashing with hash functions  $h_1, h_2, h_3 : \{0, 1\}^* \rightarrow [B]$ . The output is a Cuckoo hash table denoted by  $\mathcal{C}^1, \dots, \mathcal{C}^B$ , where for each  $x \in X$  there is some  $i \in \{1, 2, 3\}$  such that  $\mathcal{C}^{h_i(x)} = \{x \parallel i\}$ . Some Cuckoo hash positions are irrelevant, corresponding to empty bins. We use these symbols throughout subsequent sections.

<sup>5</sup> The Cuckoo hashing process uses eviction and the choice of bins for each item depends on the entire set.

<sup>6</sup> 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.

### 3 Predicate Formula Representation of Set Formulas

In this section, we formally introduce our predicate formula representation for any set formulas. We define several notions to facilitate the subsequent discussion of our work and present several theorems.

#### 3.1 Constructible Set

We formalize the notion of the resulting sets that can be derived from any set formulas being computed over the parties' private sets in the context of MPSO. We refer to these resulting sets as constructible sets.

**Definition 2.** Let  $X_1, \dots, X_m$  be  $m$  sets. A set  $Y$  is called a *constructible set* (over  $X_1, \dots, X_m$ ) if it can be derived from  $X_1, \dots, X_m$  through a finite number of set operations, including intersection, union, and difference.

In particular, if a constructible set  $Y$  satisfies  $Y \subseteq X_i$  for some  $1 \leq i \leq m$ , we call it an  $X_i$ -constructible set (over  $X_1, \dots, X_m$ ).

**Definition 3.** Let  $\varphi(x, X_1, \dots, X_m)$  be a first-order predicate formula. If  $\varphi$  is composed of atomic propositions of the form  $M(x, X_i) : x \in X_i$ , we call it a (first-order) *set predicate formula*.

Any constructible set can be represented by a set predicate formula. This corresponding relationship is formalized in the following theorem.

**Theorem 1.** Let  $X_1, \dots, X_m$  be  $m$  sets and  $Y$  is a constructible set. There exists a set predicate formula  $\varphi(x, X_1, \dots, X_m)$ , s.t. for any element  $x$ ,

$$x \in Y \iff \varphi(x, X_1, \dots, X_m) = 1.$$

We prove this theorem in Appendix B.

#### 3.2 Canonical Predicate Formula Representation

**Definition 4.** A set predicate formula  $\varphi(x, X_1, \dots, X_m)$  is called *set-separable* with respect to  $X_i$  for some  $1 \leq i \leq m$  if it can be written in the form:

$$\varphi(x, X_1, \dots, X_m) = (x \in X_i) \wedge \psi(x, X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_m),$$

where  $\psi(x, X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_m)$  is a set predicate formula not involving  $X_i$ , which we call the *separation formula* of  $\varphi(x, X_1, \dots, X_m)$  with respect to  $X_i$ .

**Corollary 1.** If a constructible set  $Y$  corresponds to a set predicate formula which is set-separable with respect to  $X_i$ , then  $Y$  is an  $X_i$ -constructible set.

**Definition 5.** Let set predicate formula  $\psi(x, X_1, \dots, X_m)$  is a disjunction of one or more subformulas,<sup>7</sup> denoted as  $\psi = Q_1 \vee \dots \vee Q_s (s \geq 1)$ . Let  $Y_i$  be the corresponding set represented by  $Q_i$ , then if each subformula  $Q_i$  is set-separable with respect to some  $X_j$  ( $1 \leq j \leq m$ ), and the set of  $\{Y_1, \dots, Y_s\}$  forms a partition of  $Y$ , we call  $\psi(x, X_1, \dots, X_m)$  a canonical predicate formula (CPF) representation (over  $X_1, \dots, X_m$ ).

**Theorem 2.** Let  $X_1, \dots, X_m$  be  $m$  sets and  $Y$  is a constructible set. There exists a CPF representation  $\psi(x, X_1, \dots, X_m)$  s.t. for any element  $x$ ,

$$x \in Y \iff \psi(x, X_1, \dots, X_m) = 1$$

We prove this theorem by showing how to construct  $\psi$  in Appendix C.

In order to illustrate Theorem 2, consider three constructible sets in the three-party setting: the intersection  $Y = X_1 \cap X_2 \cap X_3$ , the union  $Y = X_1 \cup X_2 \cup X_3$  and a complex set formula  $Y = ((X_1 \cap X_2) \cup (X_1 \cap X_3)) \setminus (X_1 \cap X_2 \cap X_3)$ . We provide the respective CPF representation  $\psi(x, X_1, X_2, X_3)$  for each case below.

**Intersection.**  $\psi(x, X_1, X_2, X_3) = (x \in X_1) \wedge (x \in X_2) \wedge (x \in X_3)$ . In this case,  $\psi$  is a disjunction of one subformula  $Q_1 = \psi$ , corresponding to the set  $Y_1 = Y$ .  $Q_1$  is set-separable with respect to  $X_1, X_2$  and  $X_3$ .  $Y_1$  itself is a partition of  $Y$ .

**Union.**  $\psi(x, X_1, X_2, X_3) = (x \in X_1) \vee ((x \notin X_1) \wedge (x \in X_2)) \vee ((x \notin X_1) \wedge (x \notin X_2) \wedge (x \in X_3))$ .  $\psi$  is a disjunction of three subformulas  $Q_1, Q_2, Q_3$ , where each  $Q_i = (x \notin X_1) \wedge \dots \wedge (x \notin X_{i-1}) \wedge (x \in X_i)$  represents  $Y_i = X_i \setminus (X_1 \cup \dots \cup X_{i-1})$ .  $Q_i$  is set-separable with respect to  $X_i$ .  $\{Y_1, Y_2, Y_3\}$  is a partition of  $Y$ .

**Complex set formula.** There are two CPF representations for this case:

- $\psi(x, X_1, X_2, X_3) = ((x \in X_1) \wedge (x \in X_2) \wedge (x \notin X_3)) \vee ((x \in X_1) \wedge (x \in X_3) \wedge (x \notin X_2))$ .  $\psi$  is a disjunction of two subformulas  $Q_1, Q_2$  with the corresponding sets  $Y_1 = X_1 \cap X_2 \setminus X_3$  and  $Y_2 = X_1 \cap X_3 \setminus X_2$ .  $Q_1$  is set-separable with respect to  $X_1$  and  $X_2$ , while  $Q_2$  is set-separable with respect to  $X_1$  and  $X_3$ .  $\{Y_1, Y_2\}$  is a partition of  $Y$ .
- $\psi(x, X_1, X_2, X_3) = (x \in X_1) \wedge [((x \in X_2) \wedge (x \notin X_3)) \vee ((x \in X_3) \wedge (x \notin X_2))]$ .  $\psi$  is set-separable with respect to  $X_1$ , so it is a disjunction of one subformula  $Q_1 = \psi$ , which obviously satisfies the definition of CPF representation.

The third example demonstrates that the CPF representation for a given constructible set is not unique. Different CPF representations can impact our protocols' efficiency. A key principle is to minimize the number of subformulas in the CPF representation to optimize performance.

## 4 Predicative Zero-Sharing

In this section, we introduce a new notion called predicative zero-sharing. By zero-sharing, we refer to a “redundant” secret-sharing that distributes one bit

<sup>7</sup> A disjunction of one subformulas is itself.

into secret shares over a finite field, where this bit is 0 only if some condition holds (e.g. the truth-value of a first-order predicate formula is true). Predicative zero-sharing is a family of protocols, each associated with a first-order predicate formula, encoding the truth-value of the formula on the parties' inputs into a zero-sharing among the parties. This class of protocols can be composed based on AND and OR operators.

#### 4.1 Definitions

A predicative zero-sharing protocol allows a set of  $m$  ( $m \geq 2$ ) parties with private inputs to receive secret shares of 0, on condition that the truth-value of the associated first-order predicate formula  $Q$  in terms of their inputs is true, otherwise receive secret shares of a uniformly random value. The formal definition of predicative zero-sharing functionality is given in Figure 6.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$  with inputs  $\mathbf{x} = (x_1, \dots, x_m)$ . A field  $\mathbb{F}$ . A first-order predicate formula  $Q$ .

**Functionality:** On input  $x_i$  from each  $P_i$ , sample  $s_i \leftarrow \mathbb{F}$  s.t. if  $Q(\mathbf{x}) = 1$ ,  $s_1 + \dots + s_m = 0$ . Give  $s_i$  to  $P_i$  for  $1 \leq i \leq m$ .

**Fig. 6.** Ideal functionality for predicative zero-sharing  $\mathcal{F}_{\text{PZS}}^Q$

#### 4.2 Security

Given the probabilistic functionality, a protocol must meet Definition 1 to securely compute predicative zero-sharing. However, we observe that for predicative zero-sharing, a simpler security definition with three requirements, including correctness, privacy and independence, is equivalent. We demonstrate this equivalence through the following theorem. Note that we will use this simpler security definition to prove security of all predicative zero-sharing protocols in this work.

Consider a probabilistic  $m$ -ary functionality  $\mathcal{F}^f$ , which takes the parties' inputs  $\mathbf{x} = (x_1, \dots, x_m)$  and outputs secret shares of  $f(\mathbf{x})$  to the parties. Let  $\Pi$  be a  $m$ -party protocol for computing  $\mathcal{F}^f$ , and  $s_i$  and  $s_i^\Pi$  denote the output of  $P_i$  from  $\mathcal{F}^f$ , and that during the execution of  $\Pi$  on  $\mathbf{x}$ , respectively ( $1 \leq i \leq m$ ).

**Theorem 3.** *If  $f$  is a probabilistic functionality in terms of  $\mathbf{x}$ , and  $\Pi$  satisfies:*

- **Correctness.** *The outputs of  $\Pi$  are secret shares of  $f(\mathbf{x})$ , namely,*

$$\{s_1, \dots, s_m\}_{\mathbf{x}} \stackrel{s}{\approx} \{s_1^\Pi, \dots, s_m^\Pi\}_{\mathbf{x}}$$

- **Privacy.** *There exists a PPT algorithm  $\text{Sim}$  s.t. for every  $\mathbf{P}_A = \{P_{i_1}, \dots, P_{i_t}\}$ ,*

$$\{\text{Sim}(\mathbf{P}_A, \mathbf{x}_A, \mathbf{s}_A)\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_A^\Pi(\mathbf{x})\}_{\mathbf{x}}$$

- **Independence.** The randomness in  $f(\mathbf{x})$  is independent of  $\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x})$  for every  $\mathbf{P}_{\mathcal{A}} = \{P_{i_1}, \dots, P_{i_t}\}$  during an execution of  $\Pi$ .

Then, there exists a PPT algorithm  $\text{Sim}$  s.t. for every  $\mathbf{P}_{\mathcal{A}} = \{P_{i_1}, \dots, P_{i_t}\}$ ,

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), s_1, \dots, s_m\}_x \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), s_1^{\Pi}, \dots, s_m^{\Pi}\}_x$$

We prove this theorem in Appendix D. Note that predicative zero-sharing functionality  $\mathcal{F}_{\text{PZS}}^Q$  is a special case of  $\mathcal{F}^f$ , where

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } Q(\mathbf{x}) = 1 \\ s & \text{if } Q(\mathbf{x}) = 0 \end{cases}$$

and  $s$  is a uniform value (the randomness in  $f$ , denoted as  $s^{\Pi}$  in the real execution). The independence requirement in this case is instantiated as: if  $Q(\mathbf{x}) = 0$ , the distribution of the secret  $s^{\Pi} = s_1^{\Pi} + \dots + s_m^{\Pi}$  during an execution of  $\Pi$  is independent of  $\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x})$ , and the correctness and independence requirements ensure that if  $Q(\mathbf{x}) = 0$ ,  $s^{\Pi}$  is uniform and independent of the joint view of any  $t \leq m - 1$  parties in real execution.

### 4.3 Relaxed Predicative Zero-Sharing

Predicative zero-sharing serves as an abstraction of many existing MPC protocols. Some protocols, like multi-party secret-shared ROT (mss-ROT) [19], rigidly conform to Theorem 3. In contrast, others realize functionality without meeting the independence requirement. We refer to this relaxed predicative zero-sharing functionality associated with  $Q$  as  $\mathcal{F}_{\text{rPZS}}^Q$ .

Relaxed predicative zero-sharing accommodates a broader range of existing protocols, such as ROT and equality-conditional randomness generation (ECRG) [35]. We demonstrate that ROT implies a relaxed predicative zero-sharing associated with a simple predicate to test whether the choice bit  $e = 1$ . Let  $\mathcal{S}$  set its share  $s_1 = -r_0$ , where  $r_0$  is the first message from ROT, and let  $\mathcal{R}$  set its share  $s_2 = r_e$ , the received message. Given that ROT functionality can be written as  $-r_0 + r_e = e \cdot (-r_0 + r_1)$ , if  $e = 0$ ,  $s_1 + s_2 = 0$ ; else,  $s_1 + s_2 = -r_0 + r_1$ , which is uniform but dependent on the output messages from ROT in  $\mathcal{S}$ 's view.

Using the standard simulation-based definition, it is hard to depict this security relaxation by defining merely a single functionality that considers all possible coalitions of  $t \leq m - 1$  parties in the multi-party setting ( $m > 2$ ). However, with our new security definition tailored for predicative zero-sharing, the relaxation is precisely formalized by removing the independence requirement.

### 4.4 From Relaxed to Standard Predicative Zero-Sharing

We give an efficient method for transforming relaxed predicative zero-sharing into standard predicative zero-sharing below.



Assuming that all parties obtain a secret-sharing  $[r] = (r_1, \dots, r_m)$  from a relaxed predicative zero-sharing protocol, with the goal to generate a new secret-sharing  $[s] = (s_1, \dots, s_m)$  meeting the standard predicative zero-sharing definition. All they need to do is to prepare a random secret-sharing  $[b]$  in the offline phase (by each  $P_i$  sampling a uniform share  $b_i$ ), and perform a secure multiplication  $[s] = [r] \cdot [b]$  in the online phase. We optimize the online phase of this secure multiplication through Beaver triples [7] in Appendix E.

**Correctness and independence.** We set the field size  $|\mathbb{F}| \geq 2^\sigma$ .

- If  $Q(\mathbf{x}) = 1$ ,  $r = 0$ , then  $s = 0$ .
- If  $Q(\mathbf{x}) = 0$ ,  $r$  is uniform. Let  $E$  be the event that  $s$  is uniform and independent of the joint view of any  $t \leq m - 1$  parties. Let  $E_0$  be the event  $r = 0$  and  $E_1$  be the event  $r \neq 0$ . Since  $b$  is uniform and independent, we have  $Pr[E] = Pr[E|E_0] \cdot Pr[E_0] + Pr[E|E_1] \cdot Pr[E_1] = 0 \cdot Pr[E_0] + 1 \cdot Pr[E_1] = Pr[E_1] = 1 - Pr[E_0] = 1 - \frac{1}{|\mathbb{F}|} \geq 1 - 2^{-\sigma}$ .

**Privacy.** The privacy follows immediately from the privacy of the relaxed predicative zero-sharing protocol and the secure multiplication.

#### 4.5 From Simple to Compound Predicative Zero-Sharing

According to the type of the associated first-order predicate formula  $Q$ , we divide predicative zero-sharing into two categories: If  $Q$  is a simple predicate, we call it a simple predicative zero-sharing; If  $Q$  is a compound predicate, we call it compound predicative zero-sharing. A compound predicate  $Q$  is formed from  $q$  literals ( $q > 1$ ) and logical connectives  $\wedge$  and  $\vee$ , where each literal  $Q_i$  corresponds to a simple predicate or its negation ( $1 \leq i \leq q$ ). We show that as long as we have a relaxed simple predicative zero-sharing protocol for each  $Q_i$ , we can build a compound predicative zero-sharing protocol for any compound predicate  $Q$ .

At a high level, a compound predicative zero-sharing protocol for  $Q$  proceeds in three phases: First, the parties execute the relaxed simple predicative zero-sharing protocol for each literal. For literals involving only a subset of the parties, the uninvolved parties set their missing secret share to 0; Second, they collectively manipulate the output secret-sharings by emulating the evaluation of  $Q$ , composing them into one output secret-sharing that meets the definition of relaxed compound predicative zero-sharing for  $Q$ , step by step. At the end of each step, they obtain a secret-sharing associated with the currently evaluated formula; Finally, the parties transform the relaxed compound predicative zero-sharing into the standard. The complete construction is described in Figure 7.

**Theorem 4.** *Protocol  $\Pi_{\text{PZS}}^Q$  securely realizes  $\mathcal{F}_{\text{PZS}}^Q$  against any semi-honest adversary corrupting  $t < m$  parties in the  $(\mathcal{F}_{\text{rPZS}}^{Q_1}, \dots, \mathcal{F}_{\text{rPZS}}^{Q_q})$ -hybrid model.*

**Correctness and independence.** In each step of the formula emulation stage,

- If  $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \wedge Q'_j(\mathbf{x})$ , the parties compute  $[r_i + r_j] = [r_i] + [r_j]$ . If  $Q'(\mathbf{x}) = 1$ , namely,  $Q'_i(\mathbf{x}) = 1 \wedge Q'_j(\mathbf{x}) = 1$ , by the functionalities of  $\mathcal{F}_{\text{rPZS}}^{Q'_i}$  and  $\mathcal{F}_{\text{rPZS}}^{Q'_j}$ ,  $r_i = 0 \wedge r_j = 0$ , hence we have  $r_i + r_j = 0$ ; otherwise,  $Q'_i(\mathbf{x}) = 0 \vee Q'_j(\mathbf{x}) = 0$ , which results that one of  $r_i$  and  $r_j$  is random, so  $r_i + r_j$  is random.
- If  $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \vee Q'_j(\mathbf{x})$ , the parties compute  $[r_i \cdot r_j] = [r_i] \cdot [r_j]$ . If  $Q'(\mathbf{x}) = 1$ , namely,  $Q'_i(\mathbf{x}) = 1 \vee Q'_j(\mathbf{x}) = 1$ , we have  $r_i = 0 \vee r_j = 0$ , hence  $r_i \cdot r_j = 0$ ; otherwise,  $Q'_i(\mathbf{x}) = 0 \wedge Q'_j(\mathbf{x}) = 0$ , then both of  $r_i$  and  $r_j$  are random. Let  $E^{i,j}$  be the event that  $r_i \cdot r_j$  is random. Let  $E_0^i$  be the event  $r_i = 0$  and  $E_1^i$  be the event  $r_i \neq 0$ . We have  $\Pr[E^{i,j}] = \Pr[E^{i,j}|E_0^i] \cdot \Pr[E_0^i] + \Pr[E^{i,j}|E_1^i] \cdot \Pr[E_1^i] = 0 \cdot \Pr[E_0^i] + 1 \cdot \Pr[E_1^i] = \Pr[E_1^i] = 1 - \Pr[E_0^i]$ . To bound the correctness error by  $2^{-\sigma}$ , we require that the probability of any  $E_0^i$  occurring is negligible. By union bound,  $\Pr[\bigvee_i E_0^i] \leq \sum_i \Pr[E_0^i] = \frac{|\text{OR}|}{|\mathbb{F}|}$ . Therefore, we set the field size  $|\mathbb{F}| \geq |\text{OR}| \cdot 2^\sigma$ , where  $|\text{OR}|$  is the number of OR operators in  $Q$ .

The above correctness of implementing AND and OR operators in each step ensures the correctness of generating a relaxed predicative compound zero-sharing for  $Q$ . Then following the proof of correctness and independence in Section 4.4, the protocol satisfies the correctness and independence requirements of the standard predicative compound zero-sharing for  $Q$ .

**Privacy.** The privacy of predicative zero-sharing is straightforward to verify: All interactions happen within the invocations of blocking blocks — all relaxed simple predicative zero-sharing protocols, the secure multiplication and transformation. Therefore, given the outputs from the ideal functionality, the simulator only needs to invoke the sub-simulators for these blocking blocks in a backward-chaining manner. As long as the privacy of all relaxed simple predicative zero-sharing protocols, the secure multiplication and transformation holds, the adversary's view is indistinguishable in the ideal and real executions.

## 5 Membership Zero-Sharing

Predicative zero-sharing is the abstraction of a class of MPC protocols. With the associated first-order predicate formulas determined, predicative zero-sharing can be instantiated. To instantiate predicative zero-sharing in the context of MPSO, we introduce membership zero-sharing, each associated with a set predicate formula, which serves as the technical core of our framework.

Our goal in this section is to build membership zero-sharing protocols for any first-order set predicate formulas. At a very high level, our construction follows the recipe for predicative zero-sharing in Figure 7, with the relaxed predicative zero-sharing components awaiting instantiations. Given that any set predicate formula is only composed of two types of literals — set membership predicates  $x \in Y$  and the negations  $x \notin Y$ , the task reduces to constructing two relaxed membership zero-sharing protocols, associated with  $x \in Y$  and  $x \notin Y$  respectively, in the two-party setting. The technical route is outlined in Figure 8.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$  with inputs  $\mathbf{x} = (x_1, \dots, x_m)$ . A field  $\mathbb{F}$ . A simple/compound predicate  $Q$  composed of  $q$  literals  $Q_1, \dots, Q_q$  ( $q \geq 1$ ) and logical connectives. A Beaver triple  $([a], [b], [c])$  generated in the offline phrase.

**Protocol:**

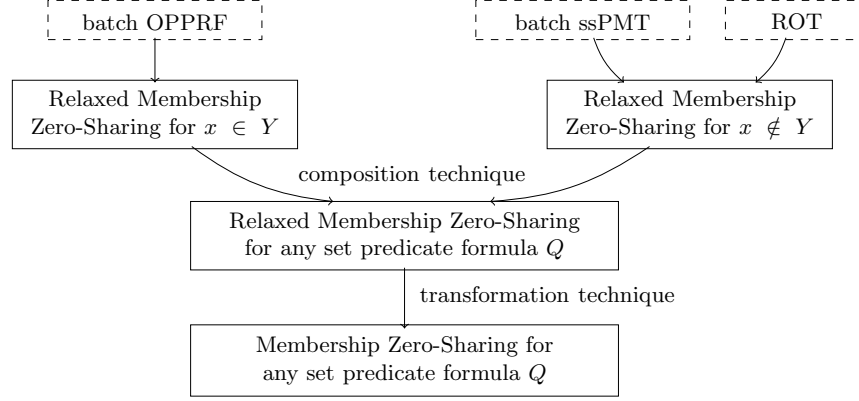
1. **The simple predicative sharing stage.** In this stage, the parties invoke  $\mathcal{F}_{\text{PZS}}^{Q_i}$  for each literal  $Q_i$ ,  $1 \leq i \leq q$ . If  $Q_i$  does not involve all the parties, then the uninvolved parties set their secret shares to 0. As a result, each  $Q_i$  has a corresponding secret-sharing among the parties.
2. **The formula emulation stage.** If  $q > 1$ , the parties collectively emulate the computation of  $Q$  in the order of operator precedence, step by step. In each step, the parties generate a secret-sharing associated with a binary clause connected by a given operator, based on the secret-sharings associated with the two contained literals  $Q'_i$  and  $Q'_j$ , which they obtain from previous steps. The actions of parties depend on the type of operator being computed:

- **AND operator:** Suppose the parties hold two secret-sharings  $[r_i]$  and  $[r_j]$  associated with  $Q'_i$  and  $Q'_j$  respectively. They want to compute a relaxed predicative zero-sharing for  $Q'$ , where  $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \wedge Q'_j(\mathbf{x})$ . All they need to do is to locally add two shares to obtain the secret-sharing  $[r_i + r_j]$ .
- **OR operator:** Suppose the parties hold two secret-sharings  $[r_i]$  and  $[r_j]$  associated with  $Q'_i$  and  $Q'_j$  respectively. They want to compute a relaxed predicative zero-sharing for  $Q'$ , where  $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \vee Q'_j(\mathbf{x})$ . Then they perform a secure multiplication  $[r_i \cdot r_j] = [r_i] \cdot [r_j]$ .

After obtaining the secret-sharing associated with  $Q'$ , the parties regard  $Q'$  as a new literal, and repeat the above process until there is only one literal in  $Q$ . The secret-sharing  $[r]$  associated with the ultimate literal held by the parties is the relaxed compound predicative zero-sharing for  $Q$ .

3. **Transformation from relaxed to standard.** All parties compute  $[s]$  by performing a secure multiplication  $[s] = [r] \cdot [b]$ , which requires one reconstruction in the online phase using Beaver triple technique (c.f. Appendix E).

**Fig. 7.** Predicative Zero-Sharing  $\Pi_{\text{PZS}}^Q$



**Fig. 8.** Technical route of building membership zero-sharing protocols for any first-order set predicate formulas. The newly introduced primitives are marked with solid boxes. The existing primitives are marked with dashed boxes.

### 5.1 Membership Zero-Sharing

A membership zero-sharing protocol allows  $m$  parties ( $m \geq 2$ ), where one party (denoted as  $P_{\text{pivot}}$ ) holds an element  $x$  while each of the others  $P_j$  holds a set  $X_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ) as input. If the associated set predicate formula  $Q(x, X_1, \dots, X_{\text{pivot}-1}, X_{\text{pivot}+1}, \dots, X_m) = 1$ , they receive secret shares of 0, otherwise they receive secret shares of a random value. The formal definition of membership zero-sharing functionality is given in Figure 9.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  is the only one holding an element instead of a set. A set predicate formula  $Q$ . A field  $\mathbb{F}$ .  
**Functionality:** On input  $x$  from  $P_{\text{pivot}}$ ,  $X_j$  from each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ), sample  $s_i \leftarrow \mathbb{F}$  for  $1 \leq i \leq m$  s.t. if  $Q(x, X_1, \dots, X_{\text{pivot}-1}, X_{\text{pivot}+1}, \dots, X_m) = 1$ ,  $\sum_{1 \leq i \leq m} s_i = 0$ . Give  $s_i$  to  $P_i$ .

**Fig. 9.** Membership Zero-Sharing Functionality  $\mathcal{F}_{\text{MZS}}^Q$

A batched version of membership zero-sharing is defined in Figure 10, where  $P_{\text{pivot}}$  holds a vector  $\mathbf{x} = (x_1, \dots, x_n)$  and each  $P_j$  holds  $n$  sets as inputs. The parties obtain  $n$  secret-sharings, where the  $i$ -th secret-sharing indicates the truth-value of the same formula  $Q$  evaluated on their  $i$ -th inputs. In particular, if  $Q$  is a conjunction of  $m-1$  set membership predicates (i.e.,  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} x \in X_j$ ), we refer to it as batch pure membership zero-sharing; if  $Q$  is a conjunction of  $m-1$  set non-membership predicates (i.e.,  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} x \notin X_j$ ), we refer to it as batch pure non-membership zero-sharing. We use  $\mathcal{F}_{\text{bpMZS}}$  and  $\mathcal{F}_{\text{bpNMZS}}$

to denote these two functionalities, respectively. The details of batch pure membership zero-sharing and batch pure non-membership zero-sharing are provided in Appendix F. We also introduce a variant of pure membership zero-sharing called pure membership zero-sharing with payloads, where  $P_{\text{pivot}}$  holds an element  $x$  while each of the others holds a set of elements and a set of associated payloads. In the end, the parties hold two secret sharings. If the conjunction of set membership predicates holds true (i.e.,  $x$  belongs to all element sets), the parties receive secret shares of 0 and secret shares of the sum of all payloads associated with  $x$ ; otherwise they receive secret shares of two random values. The ideal functionality of batch pure membership zero-sharing with payloads  $\mathcal{F}_{\text{bpMZSp}}$  is given in Figure 11, with further details also found in Appendix F.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  is the only one holding  $n$  elements instead of  $n$  sets. A set membership predicate formula  $Q$ . Batch size  $n$ . A field  $\mathbb{F}$ .  
**Functionality:** On input  $\mathbf{x} = (x_1, \dots, x_n)$  from  $P_{\text{pivot}}$  and  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  from each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ), sample  $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,n}) \leftarrow \mathbb{F}^n$  for  $1 \leq i \leq m$ , s.t. for  $1 \leq d \leq n$ , if  $Q(x_d, X_{1,d}, X_{\text{pivot}-1,d}, X_{\text{pivot}+1,d}, \dots, X_{m,d}) = 1$ ,  $\sum_{1 \leq i \leq m, 1 \leq d \leq n} s_{i,d} = 0$ . Give  $\mathbf{s}_i$  to  $P_i$ .

**Fig. 10.** Batch Membership Zero-Sharing Functionality  $\mathcal{F}_{\text{bMZS}}^Q$

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  is the only one holding  $n$  elements instead of  $2n$  sets. Batch size  $n$ . A field  $\mathbb{F}$  and payload field  $\mathbb{F}'$ . The mapping function  $\text{payload}_j()$  from element sets  $\mathbf{X}_j$  to the associated payload sets  $\mathbf{V}_j$ .  
**Functionality:** On input  $\mathbf{x} = (x_1, \dots, x_n)$  from  $P_{\text{pivot}}$ ,  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  and  $\mathbf{V}_j = (V_{j,1}, \dots, V_{j,n})$  from each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ), sample  $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,n}) \leftarrow \mathbb{F}^n$ ,  $\mathbf{w}_i = (w_{i,1}, \dots, w_{i,n}) \leftarrow \mathbb{F}'^n$  for  $1 \leq i \leq m$ , s.t. for  $1 \leq d \leq n$ , if  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} (x_d \in X_{j,d}) = 1$ ,  $\sum_{1 \leq i \leq m} s_{i,d} = 0$  and  $\sum_{1 \leq i \leq m} w_{i,d} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} v_{j,d}$ , where  $v_{j,d} = \text{payload}_j(x_d) \in V_{j,d}$ . Give  $(\mathbf{s}_i, \mathbf{w}_i)$  to  $P_i$ .

**Fig. 11.** Batch Pure Membership Zero-Sharing with Payloads Functionality  $\mathcal{F}_{\text{bpMZSp}}$

## 5.2 Relaxed Membership Zero-Sharing for Set Membership Predicate

A class of relaxed membership zero-sharing for set membership predicate  $x \in Y$  can be defined as a two-party functionality as follows: There are two parties, the

sender  $\mathcal{S}$  with a set  $Y$  and the receiver  $\mathcal{R}$  with an element  $x$ . The functionality samples  $s, r \leftarrow \mathbb{F}$  and if  $x \in Y$ , sets  $u = -r$ , otherwise  $u = s - r$ . It also generates an auxiliary information  $\text{hint}$  based on  $s$  and outputs  $r, \text{hint}$  to  $\mathcal{S}$  and  $u$  to  $\mathcal{R}$ . The  $\text{hint}$  is part of the syntax that allows for some leakage of the secret  $s$  to  $\mathcal{S}$  when  $x \notin Y$ , capturing the security relaxation in relaxed predicative zero-sharing.

We construct this protocol using OPPRF:  $\mathcal{S}$  samples a uniform  $r$ , and sets  $Y$  as the key set and  $n$  repeated values  $-r$  as the value set. Then  $\mathcal{S}$  and  $\mathcal{R}$  invoke OPPRF, where  $\mathcal{R}$  inputs  $x$  and receives  $u$ . In the end,  $\mathcal{S}$  and  $\mathcal{R}$  outputs  $r$  and  $u$  respectively. By the OPPRF functionality, if  $x \in Y$ ,  $u = -r$ , otherwise  $u$  is pseudorandom. The  $\text{hint}$  outputted to  $\mathcal{S}$  is the PRF key from OPPRF. This protocol can be naturally extended to a batched version by using batch OPPRF.

### 5.3 Relaxed Membership Zero-Sharing for Set Non-Membership Predicate

A class of relaxed membership zero-sharing protocol for set non-membership predicate  $x \notin Y$  can be defined as a two-party functionality as follows: The sender  $\mathcal{S}$  inputs a set  $Y$  while the receiver  $\mathcal{R}$  inputs  $x$ . The functionality samples  $s, r \leftarrow \mathbb{F}$  and if  $x \notin Y$ , sets  $u = -r$ , otherwise  $u = s - r$ . It also generates an auxiliary information  $\text{hint}$  based on  $s$  and outputs  $r, \text{hint}$  to  $\mathcal{S}$  and  $u$  to  $\mathcal{R}$ .

Intuitively, this functionality shares similarities with the ssPMT — both yield secret shares of 0 when  $x \notin Y$ . The key difference lies in that it outputs a zero-sharing over a field  $\mathbb{F}$ , where the opposite of a secret-sharing of 0 is a secret-sharing of a random value in  $\mathbb{F}$ , while ssPMT outputs a bit secret-sharing over  $\mathbb{F}_2$ , where the opposite of a secret-sharing of 0 is a secret-sharing of 1. Given the efficient construction for batch ssPMT in [19], our goal is to efficiently transform bit secret-sharings into zero-sharings (The batched version proceeds by first having the parties invoke batch ssPMT then execute  $n$  transformations).

Recall that in Section 4.1, ROT is considered as a relaxed simple predicative zero-sharing associated with the predicate  $e = 0$ . A variant of ROT, involving two choice bits  $e_0, e_1$  held by  $\mathcal{S}$  and  $\mathcal{R}$  respectively [40, 35, 19], is a relaxed simple predicative zero-sharing with the associated predicate  $e_0 \oplus e_1 = 0$ . After executing the protocol,  $\mathcal{S}$  receives  $r_0, r_1 \in \mathbb{F}$  while  $\mathcal{R}$  receives  $r_{e_0 \oplus e_1} \in \mathbb{F}$ .<sup>8</sup> This two-choice-bit ROT can be used to transform bit secret-sharing into zero-sharing as follows: Let  $\mathcal{S}$  set  $r = -r_0$  and  $\mathcal{R}$  set  $u = r_{e_0 \oplus e_1}$ , then if  $e_0 \oplus e_1 = 0$ ,  $r + u = 0$ ; otherwise  $r + u = r_1 - r_0$  is uniform. The  $\text{hint}$  outputted to  $\mathcal{S}$  is  $r_1$ .

### 5.4 Membership Zero-Sharing for Any Set Predicate Formulas

Using the above instantiations for the two-party relaxed membership zero-sharing protocols, we present the complete protocol of batch membership zero-sharing for any set predicate formula  $Q$  in Figure 12.

<sup>8</sup> This two-choice-bit ROT is identical to the standard 1-out-of-2 ROT, where  $e_0$  is sampled by  $\mathcal{S}$ , indicating whether to swap the order of  $r_0$  and  $r_1$ , as in Figure 12.

**Theorem 5.** *Protocol  $\Pi_{\text{bMZS}}^Q$  securely realizes  $\mathcal{F}_{\text{bMZS}}^Q$  against any semi-honest adversary corrupting  $t < m$  parties in the  $(\mathcal{F}_{\text{bOPPRF}}, \mathcal{F}_{\text{bssPMT}}, \mathcal{F}_{\text{ROT}})$ -hybrid model.*

The correctness and independence of membership zero-sharing are inherited from predicative zero-sharing, with a parameter adjustment for correctness:  $|\mathbb{F}| \geq |\text{OR}| \cdot n \cdot 2^\sigma$ , where  $|\text{OR}|$  is the number of OR operators in  $Q$ .

The privacy of membership zero-sharing is straightforward to verify: All interactions happen within the invocations of two relaxed batch membership zero-sharing protocols, which can be further decomposed into three blocking blocks — batch OPPRF, batch ssPMT and ROT. Therefore, given the outputs from the ideal functionality, the simulator only needs to invoke the sub-simulators for these blocking blocks in a backward-chaining manner. As long as the batch OPPRF, batch ssPMT and ROT protocols are secure, the adversary's view is indistinguishable in ideal and real executions, thus meeting privacy definition.

## 6 Our MPSO Framework

### 6.1 Overview

Our framework is based on the CPF representation  $\psi(x, X_1, \dots, X_m)$  of any constructible set  $Y$ . Recall that  $\psi = Q_1 \vee \dots \vee Q_s$ . Assuming that  $Q_i$  ( $1 \leq i \leq s$ ) only contains atomic propositions relevant to  $X_{i_1}, \dots, X_{i_q}$  ( $1 \leq q \leq m$ ),  $Q_i$  is set-separable with respect to  $X_j$  for some  $j \in \{i_1, \dots, i_q\}$ . Namely,

$$Q_i(x, X_{i_1}, \dots, X_{i_q}) = (x \in X_j) \wedge Q'_i(x, X_{i_1}, \dots, X_{j-1}, X_{j+1}, \dots, X_{i_q}).$$

where  $Q'_i$  is the separation formula of  $Q_i$ . We use  $Y_i$  to denote the set represented by  $Q_i$ . Our high-level idea is that for each  $Q_i$ , we designate  $P_j$  as  $P_{\text{pivot}}$  and let  $P_{i_1}, \dots, P_{i_q}$  engage in the batch membership zero-sharing protocol for  $Q'_i$ , so that for each  $x \in X_j$ , if  $Q'_i = 1$ , the parties receive secret shares of 0, otherwise secret shares of a random value.  $P_j$  adds  $x$  to its associated shares. As a result, for each  $x \in X_j$ , if  $Q_i = 1$ , i.e.,  $Q'_i = 1$ , the parties receive secret shares of  $x$ , otherwise secret shares of a random value. The process is elaborated below.

For each  $Q_i$ ,  $P_j$  uses hash functions  $h_1, h_2, h_3$  to assign elements to  $B = O(n)$  bins via Cuckoo hashing, so that each bin  $\mathcal{C}_j^b$  ( $1 \leq b \leq B$ ) has at most one item. Meanwhile, each  $P_{j'}$  ( $j' \in \{i_1, \dots, i_q\} \setminus \{j\}$ ) assigns each element  $y \in X_{j'}$  to bins  $\mathcal{T}_{j'}^{h_1(y)}, \mathcal{T}_{j'}^{h_2(y)}, \mathcal{T}_{j'}^{h_3(y)}$ . Note that if  $P_j$  maps the item  $x \in X_j$  into  $\mathcal{C}_j^b$ , then  $b \in \{h_1(x), h_2(x), h_3(x)\}$ . If  $P_{j'}$  also holds  $x$ , it must map  $x$  into  $\mathcal{T}_{j'}^b$ . This enables the remaining parties to align their input sets with respect to  $X_j$ , such that for each  $x$  in  $\mathcal{C}_j^b$ ,  $x \in X_{j'}$  if and only if  $x$  is in  $\mathcal{T}_{j'}^b$ . Thereby, we derive that

$$Q'_i(x, X_{i_1}, \dots, X_{j-1}, X_{j+1}, \dots, X_{i_q}) = Q'_i(x, \mathcal{T}_{i_1}^b, \dots, \mathcal{T}_{j-1}^b, \mathcal{T}_{j+1}^b, \dots, \mathcal{T}_{i_q}^b).$$

$P_{i_1}, \dots, P_{i_q}$  engage in the batch membership zero-sharing for  $Q'_i$ , where  $P_j$  acts as  $P_{\text{pivot}}$  with inputs  $\mathcal{C}_j^1, \dots, \mathcal{C}_j^B$  and each  $P_{j'}$  inputs  $\mathcal{T}_{j'}^1, \dots, \mathcal{T}_{j'}^B$ . In the end, they receive  $B$  secret-sharings, so that if the element  $x$  in each bin  $\mathcal{C}_j^b$  satisfies



**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  holds  $n$  elements instead of  $n$  sets. A set predicate formula  $Q$  composed of  $q \geq 1$  literals  $Q_1, \dots, Q_q$  where each  $Q_i$  ( $1 \leq i \leq q$ ) is in the form  $x \in X_j$  or  $x \notin X_j$  for some  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ .  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ , where  $[\mathbf{a}] = ([a_1], \dots, [a_n])$ ,  $[\mathbf{b}] = ([b_1], \dots, [b_n])$ ,  $[\mathbf{c}] = ([c_1], \dots, [c_n])$  and  $c_i = a_i \cdot b_i$  for  $1 \leq i \leq n$ . Batch size  $n$ . A field  $\mathbb{F}$ .

**Protocol:**

1. **The simple predicative sharing stage.** In this stage, for each  $Q_i$ ,  $P_{\text{pivot}}$  and  $P_j$  invoke the relaxed batch membership zero-sharing for  $x \in X_j$  or  $x \notin X_j$  (according to the form of  $Q_i$ ) of size  $n$ , and the remaining parties set their secret shares to 0. As a result, the parties hold a vector of  $n$  secret-sharings associated with  $Q_i$ . To be specific, if  $Q_i$  is in the form of
  - $x \in X_j$ : For the  $k$ -th instance ( $1 \leq k \leq n$ ),  $P_j$  samples  $r_{i,k}$  and sets  $K_{i,k} = X_{j,k}$  and  $V_{i,k} = \{-r_{i,k}, \dots, -r_{i,k}\}$ , where  $|K_{i,k}| = |V_{i,k}|$ . Then  $P_{\text{pivot}}$  and  $P_j$  invoke  $\mathcal{F}_{\text{BOPPRF}}$  where  $P_j$  acts as  $\mathcal{S}$  with inputs  $(K_{i,1}, \dots, K_{i,n})$  and  $(V_{i,1}, \dots, V_{i,n})$ , and  $P_{\text{pivot}}$  acts as  $\mathcal{R}$  with input  $\mathbf{x}$  and receives  $\mathbf{u}_i$ .  $P_{\text{pivot}}$  sets its shares  $\mathbf{r}_{i,\text{pivot}} = \mathbf{u}_i$ .  $P_j$  sets its shares  $\mathbf{r}_{i,j} = (r_{i,1}, \dots, r_{i,n})$ . For each  $d \in \{1, \dots, m\} \setminus \{\text{pivot}, j\}$ ,  $P_d$  sets its shares  $\mathbf{r}_{i,d} = \mathbf{0}$ .
  - $x \notin X_j$ :  $P_{\text{pivot}}$  and  $P_j$  invoke  $\mathcal{F}_{\text{BSSPMT}}$ , where in the  $k$ -th instance ( $1 \leq k \leq n$ ),  $P_j$  inputs  $X_{j,k}$  and receives  $e_{i,k}^0$ , while  $P_{\text{pivot}}$  inputs  $x_k$  and receives  $e_{i,k}^1$ . Then they invoke  $n$  instances of ROT, where in the  $k$ -th instance ( $1 \leq k \leq n$ ),  $P_j$  acts as  $\mathcal{S}$  and receives  $r_{i,k}^0, r_{i,k}^1$ , while  $P_{\text{pivot}}$  acts as  $\mathcal{R}$  with input  $e_{i,k}^1$  and receives  $r_{i,k}^{e_{i,k}^1}$ .  $P_{\text{pivot}}$  sets its shares  $\mathbf{r}_{i,\text{pivot}} = (r_{i,1}^{e_{i,1}^1}, \dots, r_{i,n}^{e_{i,n}^1})$ .  $P_j$  sets its shares  $\mathbf{r}_{i,j} = (-r_{i,1}^{e_{i,1}^0}, \dots, -r_{i,n}^{e_{i,n}^0})$ . For each  $d \in \{1, \dots, m\} \setminus \{\text{pivot}, j\}$ ,  $P_d$  sets its shares  $\mathbf{r}_{i,d} = \mathbf{0}$ .

The vector of  $n$  secret-sharings for  $Q_i$  denotes as  $[\mathbf{r}_i] = (\mathbf{r}_{i,1}, \dots, \mathbf{r}_{i,m})$ .

2. **The formula emulation stage.** If  $q > 1$ , the parties collectively emulate the computation of  $Q$  in the order of operator precedence, step by step. In each step, the parties generate a vector of  $n$  secret-sharings associated with a binary clause connected by a given operator, based on the two vectors associated with the contained literals  $Q'_i$  and  $Q'_j$ , which they obtain from previous steps. The actions of the parties depend on the type of operator being computed:
  - **AND operator:** Suppose the parties hold two vectors of  $n$  secret-sharings  $[\mathbf{r}_i]$  and  $[\mathbf{r}_j]$  associated with  $Q'_i$  and  $Q'_j$  respectively. They want to compute  $n$  relaxed membership zero-sharings of  $Q' = Q'_i \wedge Q'_j$ . Then they locally add the corresponding components of two vectors to obtain  $[\mathbf{r}_i + \mathbf{r}_j]$ .
  - **OR operator:** Suppose the parties hold two vectors of  $n$  secret-sharings  $[\mathbf{r}_i]$  and  $[\mathbf{r}_j]$  associated with  $Q'_i$  and  $Q'_j$  respectively. They want to compute  $n$  relaxed membership zero-sharings of  $Q' = Q'_i \vee Q'_j$ . Then the parties perform  $n$  secure multiplications between the corresponding components of two vectors, i.e.,  $[\mathbf{r}_i \cdot \mathbf{r}_j] = [\mathbf{r}_i] \cdot [\mathbf{r}_j]$ .

After obtaining the vector of secret-sharings associated with  $Q'$ , the parties regard  $Q'$  as a new literal, and repeat the above steps until there is only one literal in  $Q$ . The vector  $[\mathbf{r}]$  associated with the ultimate literal held by the parties is the vector of  $n$  relaxed membership zero-sharings for  $Q$ .

3. **Transformation from relaxed to standard.** All parties compute  $[\mathbf{s}]$  by performing  $[\mathbf{s}] = [\mathbf{r}] \cdot [\mathbf{b}]$ , using  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$  (c.f. Appendix E).

**Fig. 12.** Batch Membership Zero-Sharing  $\Pi_{\text{BMZS}}^Q$

$Q_i(x, X_{i_1}, \dots, X_{i_q}) = 1$ , i.e.  $x \in Y_i$ , the parties receive secret shares of 0, otherwise secret shares of a random value.  $P_j$  adds  $x$  appended with an all-zero string (for the distinction between elements and random values) to the  $b$ -th secret share (This last element addition step actually depends on the functionality, see below), so that if  $x \in Y_i$ , the parties hold a secret-sharing of  $x$ .

Given that  $\{Y_1, \dots, Y_s\}$  form a partition of  $Y$ , if the parties execute the above process with the last element addition step for all  $Q_1, \dots, Q_s$ , they will hold secret-sharings of all elements in  $Y$  (the secret-sharing of each element appears once, interspersed by random secret-sharings for elements not in  $Y$  and duplicate elements), arranged in the primary order of  $Y_1, \dots, Y_s$ . In each  $Y_i$ , secret-sharings are arranged in the secondary order of  $P_j$ 's Cuckoo hash positions, which depends on the whole set  $X_j$ . On the contrary, if the parties execute the above process without the last step, they hold secret-sharings of 0 instead of elements in the same positions. The decision to execute the last step and the subsequent process are determined by the target functionality, which is divided into three categories:

1. **MPSO.** In this functionality, the parties must reconstruct the elements in  $Y$  to  $P_1$ , thus they have to execute the last element addition step to secret-share elements. However, a straightforward reconstruction of secret-sharings leads to two types of information leakage: 1) The primary order of the reconstructed elements reveals the subset  $Y_i$  which each element belongs to. 2) The secondary order of the reconstructed elements reveals the information of  $X_j$ . The solution is to let all parties invoke the multi-party secret-shared shuffle to randomly permute and re-share secret-sharings before reconstruction.
2. **MPSO-card.** In this functionality, the parties must reconstruct the secrets without revealing the actual elements but only the cardinality. To achieve this, the parties skip the last element addition step, so that for each element in  $Y$ , the parties hold secret-sharings of 0, and for elements not in  $Y$  or repeated elements, the parties hold random secret-sharings. These secret-sharings are arranged in a specific sequence, and straightforward reconstruction would cause similar leakage as previous, thus the parties need to invoke the multi-party secret-shared shuffle as well. Afterwards, they reconstruct secrets to the leader, who counts the number of 0s as the cardinality of  $Y$ .
3. **Circuit-MPSO.** There are two ways to realize this functionality.
  - **Approach 1:** The parties skip the last element addition step for all subformulas. They feed secret-sharings along with the elements in indicated Cuckoo hashing bins into generic MPC in order, which implements a circuit identifying 0s from random values, collecting elements in the corresponding positions as  $Y$ , and computing arbitrary function on  $Y$ .
  - **Approach 2:** The parties execute the last element addition step for all subformulas. They feed secret-sharings into generic MPC, which implements a circuit first distinguishing elements from random values (by the appended 0s) to identify  $Y$ , then computing arbitrary function on  $Y$ .

In the following sections, we progressively introduce our framework in detail. Specifically, we start by constructing the simplest cases — MPSI/MPSI-card/circuit-MPSI, which are on behalf of a special case where  $\psi(x, X_1, \dots, X_m)$

is a disjunction of one subformula that is set-separable with respect to  $X_1$ , in Section 6.2. The protocols in this setting can bypass the invocation of multi-party secret-shared shuffle. In addition, we propose an MPSI-card-sum protocol as a variant. Next, we discuss another special case where  $Y$  is represented as the disjunction of several subformulas. We construct MPSU/MPSU-card/circuit-MPSU protocols as illustrations in Section 6.3. Finally, in Section 6.4, we present the complete MPSO/MPSO-card/circuit-MPSO protocols.

## 6.2 MPSI, MPSI-card and Circuit MPSI

Consider a constructible set  $Y$ , which can be represented as a set-separable formula  $Q(x, X_1, \dots, X_m)$  with respect to  $X_1$ , such as  $X_1 \cap X_2 \cap X_3$  can be represented as  $(x \in X_1) \wedge (x \in X_2) \wedge (x \in X_3)$  and  $X_1 \setminus (X_2 \cap X_3)$  can be represented as  $(x \in X_1) \wedge \neg((x \in X_2) \wedge (x \in X_3)) = (x \in X_1) \wedge ((x \notin X_2) \vee (x \notin X_3))$ . Let  $Q'(x, X_2, \dots, X_m)$  be the separation formula of  $Q$  with respect to  $X_1$ .

In this case, all elements in  $Y$  belong to  $P_1$ , and the order of secret-sharings is totally determined by  $X_1$ , so the two types of “information leakage” associated with the specific sequence of secret-sharings no longer constitute actual information leakage for  $P_1$ . Therefore, after  $P_1$  invokes batch membership zero-sharing with the other parties, acting as  $P_{\text{pivot}}$ , to realize MPSO, the parties straightforwardly reconstruct  $B$  secret-sharings to  $P_1$ . For each  $1 \leq b \leq B$ ,  $P_1$  checks whether the  $b$ -th secret is 0. If so, the element in the  $b$ -th bin is in  $Y$ . While in MPSO-card, the parties still need to invoke a multi-party secret-shared shuffle protocol before reconstruction to shuffle the correspondences between elements and secret-sharings, preventing  $P_1$  from learning the exact elements in  $Y$ .

Another and the most important benefit in this setting is that the costs of the protocols do not scale with the input length of set elements, as long as the parties pre-hash their elements into shorter strings. For correctness, we must ensure that the hashing introduces no collisions among  $P_1$  and the other parties’ input elements, so the hash function’s output length is at least  $\sigma + \log_2(m-1) + 2\log_2 n$ .

The most commonly used protocols in this case are MPSI, MPSI-card and circuit MPSI. Let  $C_{s,B,l}^1$  be a circuit that has  $sB(m\log_2|\mathbb{F}| + l)$  input wires, divided to  $s$  sections of  $B(m\log_2|\mathbb{F}| + l)$  inputs wires each. In the  $i$ -th section ( $1 \leq i \leq s$ ), the  $k$ -th group of  $B$  inputs on  $\mathbb{F}$  is associated with  $P_k$  for  $1 \leq k \leq m$ , and we denote the  $b$ -th input in this group ( $1 \leq b \leq B$ ) as  $u_{i,k,b} \in \mathbb{F}$ ; The last  $B$   $l$ -length inputs are associated with  $P_j$  for certain  $1 \leq j \leq m$ , where we denote the  $b$ -th input ( $1 \leq b \leq B$ ) as  $z_{i,b} \in \{0,1\}^l$ . The circuit first consists a subcircuit producing a bit  $w_{i,b} = 1$  if  $u_{i,1,b} + \dots + u_{i,m,b} = 0$  and 0 otherwise for  $1 \leq i \leq s, 1 \leq b \leq B$ . Then, the circuit computes and outputs  $f(Z)$  where  $Z = \{z_{i,b} | w_{i,b} = 1\}_{1 \leq i \leq s, 1 \leq b \leq B}$  and  $f$  is the function to be computed on the constructible set  $Y$ . The complete MPSI, MPSI-card and circuit MPSI protocols are described in Figure 13. Additionally, The MPSI-card-sum protocol based on pure membership zero-sharing with payloads is outlined in Figure 14.

**Parameters.**  $m$  parties  $P_1, \dots, P_m$ . Set size  $n$ . The element length  $l$ . A field  $\mathbb{F}$ . Cuckoo hashing parameters: hash functions  $h_1, h_2, h_3$  and number of bins  $B$ .

**Inputs.** Each party  $P_i$  has input  $X_i = \{x_i^1, \dots, x_i^n\} \subseteq \{0, 1\}^l$ .

1. **Hashing to bin.**  $P_1$  does  $\mathcal{C}_1^1, \dots, \mathcal{C}_1^B \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^B(X_1)$ . For  $1 < j \leq m$ ,  $P_j$  does  $\mathcal{T}_j^1, \dots, \mathcal{T}_j^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X_j)$ .
2. **Batch pure membership zero-sharing.** All parties invoke  $\mathcal{F}_{\text{bpMZS}}$  of batch size  $B$ , where  $P_1$  acts as  $P_{\text{pivot}}$  with inputs  $\mathcal{C}_1^1, \dots, \mathcal{C}_1^B$  and each  $P_j$  inputs  $\mathcal{T}_j^1, \dots, \mathcal{T}_j^B$  for  $1 < j \leq m$ . For  $1 \leq i \leq m$ ,  $P_i$  receives  $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,B})$ .

The following actions of the parties depend on the functionality:

**MPSI.**

3. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{s}_j$  to  $P_1$ .  $P_1$  computes  $\mathbf{s} = \sum_{1 < j \leq m} \mathbf{s}_j$  and sets  $Y = \emptyset$ . For  $1 \leq b \leq B$ , if  $s_b = 0$ ,  $P_1$  outputs the element in  $\mathcal{C}_1^b$ .

**MPSI-card.**

3. For  $1 \leq i \leq m$ ,  $P_i$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{s}_i$ .  $P_i$  receives  $\mathbf{s}'_i$ .
4. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{s}'_j$  to  $P_1$ .  $P_1$  outputs  $\text{zero}(\sum_{1 < j \leq m} \mathbf{s}'_j)$ .

**Circuit-MPSI (Approach 1).**

3. All parties invoke an  $m$ -party computation with circuit  $C_{1,B,l}^1$ . For  $1 \leq b \leq B$ ,  $1 \leq i \leq m$ ,  $P_i$  takes  $s_{i,b}$  as its  $b$ -th input, and  $P_1$  inputs the element in  $\mathcal{C}_1^b$ .

Fig. 13. MPSI/MPSI-card/Circuit-MPSI

**Parameters.** Same as in Figure 13.

**Inputs.** Each party  $P_i$  has input  $X_i = \{x_i^1, \dots, x_i^n\} \subseteq \{0, 1\}^l$ .

1. **Hashing to bin.**  $P_1$  does  $\mathcal{C}_1^1, \dots, \mathcal{C}_1^B \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^B(X_1)$ . For  $1 < j \leq m$ ,  $P_j$  does  $\mathcal{T}_j^1, \dots, \mathcal{T}_j^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X_j)$ .  $P_j$  defines  $\mathcal{V}_j^1, \dots, \mathcal{V}_j^B$  where for  $1 \leq b \leq B$ ,  $\mathcal{V}_j^b$  contains the associated payloads of the elements in  $\mathcal{T}_j^b$ .
2. **Batch pure membership zero-sharing with payloads.** All parties invoke  $\mathcal{F}_{\text{bpMZSp}}$  of batch size  $B$ , where  $P_1$  acts as  $P_{\text{pivot}}$  with inputs  $\mathcal{C}_1^1, \dots, \mathcal{C}_1^B$  and each of the remaining parties  $P_j$  inputs  $(\mathcal{T}_j^1, \dots, \mathcal{T}_j^B)$  and  $(\mathcal{V}_j^1, \dots, \mathcal{V}_j^B)$ . For  $1 \leq i \leq m$ ,  $P_i$  receives  $(\mathbf{s}_i, \mathbf{w}_i)$ .
3. For  $1 \leq i \leq m$ ,  $P_i$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{s}_i$ .  $P_i$  receives  $\mathbf{s}'_i = (s'_1, \dots, s'_B)$ .
4. For  $1 \leq b \leq B$ , if  $\mathcal{C}_1^b$  is not an empty bin,  $P_1$  sets  $w_{1,b} = w_{1,b} + v$ , where  $v$  is the associated payload with the element in  $\mathcal{C}_1^b$ .
5. For  $1 \leq i \leq m$ ,  $P_i$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{w}_i$ .  $P_i$  receives  $\mathbf{w}'_i = (w'_1, \dots, w'_B)$ .
6. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{s}'_j$  to  $P_1$ .  $P_1$  computes  $\mathbf{s}' = \sum_{1 < j \leq m} \mathbf{s}'_j$  and defines a bit vector  $\mathbf{e} = (e_1, \dots, e_B)$  where for  $1 \leq b \leq B$ , if  $s'_b = 0$ ,  $e_b = 1$ , otherwise  $e_b = 0$ .  $P_1$  distributes  $\mathbf{e}$  to  $P_j$ . For  $1 \leq i \leq m$ ,  $P_i$  outputs  $\text{HW}(\mathbf{e})$ .
7. For  $1 \leq i \leq m$ ,  $P_i$  computes  $u_i = \sum_{1 \leq b \leq B, \text{t.e}_b=1} w'_{i,b}$ . For  $1 < j \leq m$ ,  $P_j$  sends  $u_j$  to  $P_1$ .  $P_1$  outputs  $u = \sum_{1 \leq i \leq m} u_i$ .

Fig. 14. MPSI-card-sum

### 6.3 MPSU, MPSU-card and Circuit MPSU

Consider a constructible set  $Y$ , whose CPF representation  $\psi(x, X_1, \dots, X_m)$  is a disjunction of several subformulas, one is an atomic proposition  $x \in X_i$  for some  $1 \leq i \leq m$ . For instance,  $X_1 \cup \dots \cup X_m$  can be represented as  $(x \in X_1) \vee ((x \notin X_1) \wedge (x \in X_2)) \vee \dots \vee ((x \notin X_1) \wedge \dots \wedge (x \notin X_{m-1}) \wedge (x \in X_m))$ . In this case, the subformula  $x \in X_i$  only involves  $P_i$ , so  $P_i$  simply shares its elements among the parties. Especially, if  $i = 1$ , then the subformula can be ignored, as long as  $P_1$  finally appends its elements to the reconstructed elements to obtain  $Y_1$ .

The most commonly used protocols in this case are MPSU, MPSU-card and circuit MPSU. Let  $C_{N,l'}^2$  be a circuit with  $m$  groups of  $N$  inputs on  $\mathbb{F}$ . The  $k$ -th group is associated with  $P_k$  ( $1 \leq k \leq m$ ), where the  $i$ -th inputs is denoted by  $z_{k,i}$  ( $1 \leq i \leq N$ ). The circuit computes and outputs  $f(Z)$  where  $Z = \{z_i | z_{1,i} + \dots + z_{m,i} = z_i \parallel 0^{l'}\}_{1 \leq i \leq N}$  and  $f$  is the function to be computed on  $Y$ . The complete MPSU, MPSU-card and circuit MPSU protocols are described in Figure 15.

### 6.4 MPSO, MPSO-card and Circuit MPSO

Following the overview of our framework, we formally present the MPSO, MPSO-card, and circuit-MPSO protocols for any constructible set in Figure 16.

**Theorem 6.** *The MPSO, MPSO-card and circuit-MPSO protocols in Figure 16 are secure against any semi-honest adversary corrupting  $t < m$  parties in the  $(\mathcal{F}_{\text{bMZS}}^Q, \mathcal{F}_{\text{shuffle}})$ -hybrid model.*

**Correctness.** The correctness of these protocols comes from the existence and qualities of CPF representations in Theorem 2 and Definition 4, and the correctness of batch membership zero-sharing. To ensure the correctness of all batch membership zero-sharing protocols, the field size must satisfy  $|\mathbb{F}| \geq |\text{OR}| \cdot B \cdot 2^\sigma$ , where  $|\text{OR}|$  is the total number of OR operators in all  $Q_i$  for  $1 \leq i \leq s$ .

Let  $Y_i$  denote the set represented by  $Q_i$ . In the MPSO and circuit-MPSO (Approach 2) protocols, the parties hold  $|Y_i|$  secret-sharings of the elements in  $Y_i$ , and  $B - |Y_i|$  secret-sharings of random values after each batch membership zero-sharing for  $Q_i$ , for  $1 \leq i \leq s$ . Given that  $\{Y_1, \dots, Y_s\}$  is a partition of  $Y$ , the parties hold  $|Y|$  secret-sharings of the elements in  $Y$ , and  $sB - |Y|$  secret-sharings of random values in total. Finally,  $P_1$  and the circuit identify all set elements by checking whether the last  $l'$  bits are all zero. An error occurs when a random value collides with  $0^{l'}$ . Thereby, the overall false positive error probability is at most  $sB \cdot 2^{-l'}$ . To make this failure probability negligible, we set  $l' \geq \sigma + \log s + \log B$ ; In the MPSO-card and circuit-MPSO (Approach 1) protocols, the parties hold  $|Y|$  secret-sharings of 0, and  $B - |Y|$  secret-sharings of random values. To bound the overall false positive error probability by  $2^{-\sigma}$ , we set  $|\mathbb{F}| \geq sB \cdot 2^\sigma$ .

**Security.** This security proof is deferred to Appendix G.

**Complexity Analysis.** The computation and communication complexity are both dominated by the form of the CPF representation  $\psi(x, X_1, \dots, X_m)$  of the constructible set  $Y$  being computed, where  $\psi = Q_1 \vee \dots \vee Q_s$ .

**Parameters.**  $m$  parties  $P_1, \dots, P_m$ . Set size  $n$ . The element length  $l$ . The all-zero string length  $l'$ . A field  $\mathbb{F}$ . An encoding function  $\text{code} : \mathbb{F} \rightarrow \{0, 1\}^{l+l'}$ . Cuckoo hashing parameters: hash functions  $h_1, h_2, h_3$  and number of bins  $B$ .

**Inputs.** Each party  $P_i$  has input  $X_i = \{x_i^1, \dots, x_i^n\} \subseteq \{0, 1\}^l$ .

1. **Hashing to bin.**  $P_1$  does  $\mathcal{T}_1^1, \dots, \mathcal{T}_1^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X_1)$ . For  $1 < j \leq m$ ,  $P_j$  does  $\mathcal{C}_j^1, \dots, \mathcal{C}_j^B \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^B(X_j)$  and  $\mathcal{T}_j^1, \dots, \mathcal{T}_j^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X_j)$ .
2. **Batch pure membership zero-sharing.** For  $1 < j \leq m$ ,  $P_1, \dots, P_j$  invoke  $\mathcal{F}_{\text{bpNMZS}}$  of batch size  $B$ , where  $P_j$  acts as  $P_{\text{pivot}}$  with inputs  $\mathcal{C}_j^1, \dots, \mathcal{C}_j^B$  and each  $P_{j'}$  inputs  $\mathcal{T}_{j'}^1, \dots, \mathcal{T}_{j'}^B$  for  $j' \in \{i_1, \dots, i_q\} \setminus \{j\}$ . For  $1 \leq i \leq j$ ,  $P_i$  receives  $\mathbf{s}_{j,i} = (s_{j,i,1}, \dots, s_{j,i,B})$ .

The following actions of the parties depend on the functionality:

**MPSU.**

3. For  $1 < j \leq m$ ,  $1 \leq b \leq B$ , if  $\mathcal{C}_j^b$  is not an empty bin,  $P_j$  computes  $s_{j,j,b} = \text{code}(s_{j,j,b}) \oplus (x \| 0^{l'})$ , where  $x$  is the element in  $\mathcal{C}_j^b$ , otherwise  $P_j$  samples  $s_{j,j,b} \leftarrow \{0, 1\}^{l+l'}$ .
4. For  $1 \leq i \leq m$ ,  $P_i$  computes  $\mathbf{u}_i \in (\{0, 1\}^{l+l'})^{(m-1)B}$  as follows: For  $\max(2, i) \leq j \leq m$ ,  $1 \leq b \leq B$ ,  $u_{i,(j-2)B+b} = s_{j,i,b}$ . Set other positions to 0.
5. For  $1 \leq i \leq m$ ,  $P_i$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{u}_i$ .  $P_i$  receives  $\mathbf{u}'_i$ .
6. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{u}'_j$  to  $P_1$ .  $P_1$  computes  $\mathbf{u}' = \sum_{1 < j \leq m} \mathbf{u}'_j$  and sets  $Y = \emptyset$ . For  $1 \leq b \leq B$ , if  $u'_b = y \| 0^{l'}$  for some  $y \in \{0, 1\}^l$ ,  $P_1$  outputs  $y$ .

**MPSU-card.**

3. For  $1 < j \leq m$ ,  $1 \leq b \leq B$ ,  $P_j$  chooses  $s_{j,j,b}$  at random if  $\mathcal{C}_j^b$  is an empty bin.
4. For  $1 \leq i \leq m$ ,  $P_i$  computes  $\mathbf{u}_i \in \mathbb{F}^{(m-1)B}$  as follows: For  $\max(2, i) \leq j \leq m$ ,  $1 \leq b \leq B$ ,  $u_{i,(j-2)B+b} = s_{j,i,b}$ . Set other positions to 0.
5. For  $1 \leq i \leq m$ ,  $P_i$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{u}_i$ .  $P_i$  receives  $\mathbf{u}'_i$ .
6. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{u}'_j$  to  $P_1$ .  $P_1$  outputs  $\text{zero}(\sum_{1 < j \leq m} \mathbf{u}'_j)$ .

**Circuit-MPSU (Approach 2).**

3. For  $1 < j \leq m$ ,  $1 \leq b \leq B$ , if  $\mathcal{C}_j^b$  is not an empty bin,  $P_j$  computes  $s_{j,j,b} = \text{code}(s_{j,j,b}) \oplus (x \| 0^{l'})$ , where  $x$  is the element in  $\mathcal{C}_j^b$ , otherwise  $P_j$  samples  $s_{j,j,b} \leftarrow \{0, 1\}^{l+l'}$ .
4. For  $1 \leq i \leq m$ ,  $P_i$  computes  $\mathbf{u}_i \in (\{0, 1\}^{l+l'})^{(m-1)B}$  as follows: For  $\max(2, i) \leq j \leq m$ ,  $1 \leq b \leq B$ ,  $u_{i,(j-2)B+b} = s_{j,i,b}$ . Set other positions to 0.
5. All parties invoke an  $m$ -party computation with the circuit  $C_{(m-1)B, l'}^2$ . For  $1 \leq i \leq m$ ,  $1 \leq k \leq (m-1)B$ ,  $P_i$  inputs  $u_{i,c}$  to the circuit.

**Fig. 15.** MPSU/MPSU-card/Circuit-MPSU

**Parameters.**  $m$  parties  $P_1, \dots, P_m$ . Set size  $n$ . The element length  $l$ . The all-zero string length  $l'$ . A field  $\mathbb{F}$ . An encoding function  $\text{code} : \mathbb{F} \rightarrow \{0, 1\}^{l+l'}$ . A constructible set  $Y$  represented as a CPF representation  $\psi(x, X_1, \dots, X_m)$ . Cuckoo hashing parameters: hash functions  $h_1, h_2, h_3$  and number of bins  $B$ .

**Inputs.** Each party  $P_i$  has input  $X_i = \{x_i^1, \dots, x_i^n\} \subseteq \{0, 1\}^l$ .

1. **Hashing to bin.** For  $1 \leq i \leq m$ ,  $P_i$  does  $\mathcal{C}_i^1, \dots, \mathcal{C}_i^B \leftarrow \text{Cuckoo}_{h_1, h_2, h_3}^B(X_i)$  and  $\mathcal{T}_i^1, \dots, \mathcal{T}_i^B \leftarrow \text{Simple}_{h_1, h_2, h_3}^B(X_i)$ .
2. **Single subformula evaluation.** Let  $\psi = Q_1 \vee \dots \vee Q_s$ . For the  $i$ -th subformula  $Q_i(x, X_{i_1}, \dots, X_{i_q})$  in  $\psi$ , where  $1 \leq i \leq s, \{i_1, \dots, i_q\} \subseteq \{1, \dots, m\}$ ,
  - (a) If  $q = 1$ , suppose  $i_1 = \dots = i_q = j$ , then  $Q_i(x, X_j) = (x \in X_j)$ . For  $1 \leq b \leq B$ , if  $\mathcal{C}_j^b$  is not an empty bin,  $P_j$  sets  $s_{i,j,b} = 0$ , otherwise  $P_j$  chooses  $s_{i,j,b}$  at random. For  $j' \in \{1, \dots, m\} \setminus \{j\}$ ,  $P_{j'}$  sets  $s_{i,j',b} = 0$ .
  - (b) If  $q > 1$ , suppose  $Q_i$  is set-separable with respect to  $X_j$  for some  $j \in \{i_1, \dots, i_q\}$  and  $Q_i(x, X_{i_1}, \dots, X_{i_q}) = (x \in X_j) \wedge Q'_i(x, X_{i_1}, \dots, X_{j-1}, X_{j+1}, \dots, X_{i_q})$ . The parties invoke  $\mathcal{F}_{\text{bMZO}}^{Q'_i}$  where  $P_j$  acts as  $P_{\text{pivot}}$  with inputs  $\mathcal{C}_j^1, \dots, \mathcal{C}_j^B$  and each  $P_{j'}$  inputs  $\mathcal{T}_{j'}^1, \dots, \mathcal{T}_{j'}^B$  for  $j' \in \{i_1, \dots, i_q\} \setminus \{j\}$ . For  $1 \leq b \leq B$ , each  $P_{i'}$  receives  $s_{i,i',b} = (s_{i,i',1}, \dots, s_{i,i',B})$  for  $i' \in \{i_1, \dots, i_q\}$ , and each  $P_k$  sets  $s_{i,k,b} = 0$  for  $k \in \{1, \dots, m\} \setminus \{i_1, \dots, i_q\}$ .

The following actions of the parties depend on the functionality:

**MPSO.**

3. For  $1 < i \leq s, 1 \leq b \leq B$ , if  $\mathcal{C}_j^b$  is not an empty bin,  $P_j$  (the same  $j$  as step 2) computes  $s'_{i,j,b} = \text{code}(s_{i,j,b}) \oplus (x \| 0^{l'})$ , where  $x$  is the element in  $\mathcal{C}_j^b$ , otherwise  $P_j$  samples  $s'_{i,j,b} \leftarrow \{0, 1\}^{l+l'}$ .
4. For  $1 \leq k \leq m$ , each  $P_k$  computes  $\mathbf{u}_k \in (\{0, 1\}^{l+l'})^{sB}$  as follows: For  $1 \leq i \leq s, 1 \leq b \leq B, u_{k,(i-1)B+b} = s'_{i,k,b}$ .
5. For  $1 \leq k \leq m$ ,  $P_k$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{u}_k$ .  $P_k$  receives  $\mathbf{u}'_k$ .
6. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{u}'_j$  to  $P_1$ .  $P_1$  computes  $\mathbf{u}' = \sum_{1 < j \leq m} \mathbf{u}'_j$  and sets  $Y = \emptyset$ . For  $1 \leq b \leq B$ , if  $u'_b = y \| 0^{l'}$  for some  $y \in \{0, 1\}^l$ ,  $P_1$  outputs  $y$ .

**MPSO-card.**

3. For  $1 \leq k \leq m$ ,  $P_k$  computes  $\mathbf{u}_k \in \mathbb{F}^{sB}$  as follows: For  $1 \leq i \leq s, 1 \leq b \leq B, u_{k,(i-1)B+b} = s_{i,k,b}$ .
4. For  $1 \leq k \leq m$ ,  $P_k$  invoke  $\mathcal{F}_{\text{shuffle}}$  with input  $\mathbf{u}_k$ .  $P_k$  receives  $\mathbf{u}'_k$ .
5. For  $1 < j \leq m$ ,  $P_j$  sends  $\mathbf{u}'_j$  to  $P_1$ .  $P_1$  outputs  $\text{zero}(\sum_{1 < j \leq m} \mathbf{u}'_j)$ .

**Circuit-MPSO (Approach 1).**

3. All parties invoke an  $m$ -party computation with the circuit  $C_{s,B,l}^1$ . For  $1 \leq i \leq s, 1 \leq k \leq m$ ,  $P_k$  inputs  $s_{i,k,1}, \dots, s_{i,k,B}$  to the  $i$ -th section, and  $P_j$  (the same  $j$  as step 2) inputs the elements in  $\mathcal{C}_j^1, \dots, \mathcal{C}_j^B$  in addition.

**Circuit-MPSO (Approach 2).**

3. For  $1 < i \leq s, 1 \leq b \leq B$ , if  $\mathcal{C}_j^b$  is not an empty bin,  $P_j$  (the same  $j$  as step 2) computes  $s'_{i,j,b} = \text{code}(s_{i,j,b}) \oplus (x \| 0^{l'})$ , where  $x$  is the element in  $\mathcal{C}_j^b$ , otherwise  $P_j$  samples  $s'_{i,j,b} \leftarrow \{0, 1\}^{l+l'}$ .
4. All parties invoke an  $m$ -party computation with the circuit  $C_{sB,l'}^2$ . For  $1 \leq i \leq s, 1 \leq b \leq B, 1 \leq k \leq m$ ,  $P_k$  takes  $s'_{i,k,b}$  as its  $((i-1)B+b)$ -th input.

Fig. 16. MPSO/MPSo-card/Circuit-MPSO



In the subformula evaluation stage, the computation complexity of each party  $P_j$  includes two parts: (1)  $O(\sum_{1 \leq i \leq s} (i_q \cdot |\text{AND}_i + \text{OR}_i| \cdot n))$ , where  $Q_i$  is set-separable with respect to  $X_j$  (we use  $Q'_i$  to denote the separation formula of  $Q_i$  with respect to  $X_j$ ), while  $i_q$  is the number of literals and  $|\text{AND}_i + \text{OR}_i|$  is the total number of AND and OR operators in  $Q'_i$ ; (2)  $O(\sum_{1 \leq i \leq s} (|\text{AND}_i + \text{OR}_i| \cdot n))$ , where  $Q_i$  is not set-separable with respect to  $X_j$  while includes  $X_j$ , and  $|\text{AND}_i + \text{OR}_i|$  is the total number of AND and OR operators in the separation formula of  $Q_i$  with respect to some other set. The communication complexity of  $P_j$  can also be computed as two parts: (1)  $O(\sum_{1 \leq i \leq s} (i_q \cdot |\text{OR}_i| \cdot n))$ , where  $Q_i$  is set-separable with respect to  $X_j$  (we use  $Q'_i$  to denote the separation formula of  $Q_i$  with respect to  $X_j$ ), while  $i_q$  is the number of literals and  $|\text{OR}_i|$  is the number of OR operators in  $Q'_i$ ; (2)  $O(\sum_{1 \leq i \leq s} (|\text{OR}_i| \cdot n))$ , where  $Q_i$  is not set-separable with respect to  $X_j$  while includes  $X_j$ , and  $|\text{OR}_i|$  is the number of OR operators in the separation formula of  $Q_i$  with respect to some other set.

In the multi-party secret-shared shuffle and reconstruction steps, the leader's computation and communication complexity are both  $O(smn)$  while each client's computation and communication complexity are both  $O(sn)$ , where  $s$  is the number of subformulas in the CPF representation  $\psi$ .

A more detailed of complexity analysis for our MPSI (and its variants) and our MPSU (and its variants) protocols is provided in Appendix A.

## 7 Performance Evaluation

We demonstrate the practicality of our framework with implementations for its typical instantiations, including MPSI, MPSI-card, MPSI-card-sum and MPSU protocols. We assume a commonly used setting where Beaver triples are pre-computed offline and stored locally. This follows real scenarios where Beaver triples are pre-generated by parties themselves or with the help of a Trusted-Third Party under the Trusted Dealer model [59, 40]. We report on the online performance of our protocols in comparison with the respective state of the art:

- The state-of-the-art MPSI [58]: This work proposes two MPSI protocols, O-Ring and K-Star, with publicly available implementations [1]. We select K-Star for comparison since it is faster than O-Ring with the same total communication costs. The reported data in Table 1 is obtained by running their full protocol, as there is no correlated randomness pre-generated offline in their implementation. We set the corruption threshold  $t = m - 1$  in their code and report the leader's running time and the total communication costs of all parties for both their and our MPSI protocols.
- The state-of-the-art MPSI-card [18]: This work does not provide open-source code, thus we take the experimental results of the online phase from their paper, whose experiments was run on Intel i7-12700H 2.30GHz CPU and 28GB RAM. We report the leader's running time and communication costs for both their and our MPSI-card protocols in Table 2.

- The state-of-the-art MPSU [19]: This work proposes two MPSU protocols with publicly available implementation [2]. We compare with the symmetric-key based one for its better online performance. In the benchmark of MPSU, we set the item length as 64 bits and report the leader’s running time and communication costs in the online phase in Table 4.

To our knowledge, there is no available implementation or reported experimental data for the MPSI-card-sum protocol. We provide the first MPSI-card-sum implementation and report experimental data in the same setting as MPSI-card.

We conduct our experiments on a cloud virtual machine with Intel(R) Xeon(R) 2.70GHz CPU (32 physical cores) and 128 GB RAM. 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. The implementation details and parameter settings are detailed in Appendix H.

The experimental results, presented in Table 1-4, show that our protocols exhibit online performance that is either superior to or comparable with the state of the art in all cases. In terms of computation costs, our MPSI protocol achieves a  $2.5 - 8.3\times$  speedup (LAN) and a  $1.7 - 3.4\times$  speedup (WAN) compared to [58]; our MPSI-card protocol achieves an  $18.0 - 63.4\times$  speedup (LAN) compared to [18]. In terms of communication costs, our MPSI achieves an improvement up to  $14.4\times$  while our MPSI-card achieves an improvement up to  $20.3\times$ . The computation and communication costs of our MPSI-card-sum is only double our MPSI approximately, while realizing a richer functionality. The computation and communication costs of our MPSU are comparable with [19].

$m$	protocol	Time (second)										Comm. (MB)				
		LAN					WAN									
		$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$
3	[58]	0.064	0.188	0.860	4.389	19.81	5.283	5.733	6.680	10.24	30.10	2.109	3.418	<b>8.110</b>	<b>28.54</b>	<b>111.9</b>
	Ours	<b>0.013</b>	<b>0.047</b>	<b>0.227</b>	<b>1.533</b>	<b>7.945</b>	<b>1.540</b>	<b>2.070</b>	<b>2.912</b>	<b>5.136</b>	<b>16.42</b>	<b>0.641</b>	<b>2.551</b>	10.20	40.91	164.2
4	[58]	0.077	0.211	0.905	4.482	20.78	5.719	6.159	7.174	11.24	33.45	4.689	7.464	17.39	<b>60.20</b>	<b>235.4</b>
	Ours	<b>0.015</b>	<b>0.051</b>	<b>0.242</b>	<b>1.632</b>	<b>8.133</b>	<b>1.944</b>	<b>2.476</b>	<b>3.339</b>	<b>6.345</b>	<b>19.23</b>	<b>0.961</b>	<b>3.826</b>	<b>15.31</b>	61.36	246.2
5	[58]	0.104	0.235	1.019	4.839	22.47	6.049	6.511	7.806	12.02	37.27	8.285	13.07	30.16	103.5	403.9
	Ours	<b>0.016</b>	<b>0.053</b>	<b>0.260</b>	<b>1.724</b>	<b>8.255</b>	<b>2.348</b>	<b>2.889</b>	<b>3.770</b>	<b>6.789</b>	<b>22.04</b>	<b>1.282</b>	<b>5.102</b>	<b>20.41</b>	<b>81.81</b>	<b>328.3</b>
10	[58]	0.215	0.465	1.691	7.310	32.91	8.653	9.511	11.18	21.12	70.74	41.51	64.46	146.3	493.7	1921
	Ours	<b>0.026</b>	<b>0.075</b>	<b>0.354</b>	<b>1.910</b>	<b>8.898</b>	<b>4.363</b>	<b>4.915</b>	<b>5.769</b>	<b>9.423</b>	<b>33.38</b>	<b>2.884</b>	<b>11.48</b>	<b>45.92</b>	<b>184.1</b>	<b>738.7</b>

**Table 1.** Running time and communication costs for MPSI protocols in LAN and WAN settings.  $m$  is the number of parties.

## References

1. <https://github.com/private-panda/oring>
2. <https://github.com/real-world-cryptography/MPSU>
3. Coproto: C++ coroutine protocol library., <https://github.com/Visa-Research/coproto.git>
4. cryptoTools., <https://github.com/ladnir/cryptoTools.git>
5. Vole-PSI, <https://github.com/Visa-Research/volepsi.git>

$m$	protocol	Time (second)										Comm. (MB)				
		LAN					WAN									
		$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$
3	Ours	<b>0.015</b>	<b>0.052</b>	<b>0.237</b>	<b>1.581</b>	<b>8.050</b>	<b>1.784</b>	<b>2.800</b>	<b>3.681</b>	<b>6.498</b>	<b>19.50</b>	<b>0.761</b>	<b>3.035</b>	<b>12.15</b>	<b>48.76</b>	<b>195.8</b>
4	Ours	<b>0.016</b>	<b>0.054</b>	<b>0.252</b>	<b>1.684</b>	<b>8.300</b>	<b>2.267</b>	<b>3.780</b>	<b>5.005</b>	<b>8.565</b>	<b>24.88</b>	<b>1.122</b>	<b>4.472</b>	<b>17.91</b>	<b>71.83</b>	<b>288.4</b>
5	[18]	0.670	1.789	6.289	31.24	—	—	—	—	—	—	20.70	94.49	425.6	1894	—
	Ours	<b>0.018</b>	<b>0.056</b>	<b>0.270</b>	<b>1.735</b>	<b>8.616</b>	<b>2.753</b>	<b>4.747</b>	<b>6.076</b>	<b>10.42</b>	<b>28.87</b>	<b>1.482</b>	<b>5.909</b>	<b>23.66</b>	<b>94.90</b>	<b>381.0</b>
10	[18]	1.477	4.503	12.81	95.23	—	—	—	—	—	—	46.58	212.6	957.7	4262	—
	Ours	<b>0.026</b>	<b>0.071</b>	<b>0.375</b>	<b>2.001</b>	<b>9.226</b>	<b>5.174</b>	<b>9.578</b>	<b>11.93</b>	<b>18.43</b>	<b>50.33</b>	<b>3.285</b>	<b>13.09</b>	<b>52.42</b>	<b>210.2</b>	<b>844.0</b>

**Table 2.** Running time and communication costs for MPSI-card protocols in LAN and WAN settings. The data of [18] originates from their paper for lack of available implementation. Cells with — denote missing data that is not reported.

$m$	Time (second)										Comm.(MB)				
	LAN					WAN									
	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$
3	0.023	0.088	0.417	2.810	14.67	2.519	3.541	4.686	9.417	31.88	1.283	5.107	20.43	81.89	328.6
4	0.025	0.091	0.436	3.044	15.16	3.084	4.680	5.948	12.40	38.17	1.885	7.499	29.99	120.2	482.4
5	0.027	0.094	0.474	3.150	15.49	3.650	5.729	7.288	13.17	43.06	2.486	9.890	39.56	158.6	636.2
10	0.039	0.120	0.632	3.610	16.65	6.474	10.93	13.79	23.56	71.36	5.493	21.85	87.37	350.2	1405

**Table 3.** Running time and communication costs for our MPSI-card-sum protocol.

$m$	protocol	Time (second)										Comm. (MB)				
		LAN					WAN									
		$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$	$2^{12}$	$2^{14}$	$2^{16}$	$2^{18}$	$2^{20}$
3	[19]	<b>0.017</b>	<b>0.050</b>	<b>0.215</b>	<b>1.005</b>	<b>4.352</b>	<b>3.157</b>	<b>3.734</b>	<b>4.444</b>	<b>9.705</b>	<b>33.10</b>	<b>1.690</b>	<b>6.788</b>	<b>27.87</b>	<b>112.7</b>	<b>455.6</b>
	Ours	0.022	0.068	0.298	1.892	9.607	3.327	3.774	4.878	11.04	37.77	1.930	7.755	31.76	128.3	518.7
4	[19]	<b>0.023</b>	<b>0.071</b>	<b>0.286</b>	<b>1.393</b>	<b>5.645</b>	<b>3.976</b>	<b>4.618</b>	<b>6.507</b>	<b>17.10</b>	<b>59.21</b>	<b>3.145</b>	<b>12.81</b>	<b>51.69</b>	<b>208.8</b>	<b>843.7</b>
	Ours	0.029	0.089	0.415	2.345	11.25	3.996	4.820	6.756	17.45	64.04	3.624	14.74	59.46	240.1	969.6
5	[19]	<b>0.030</b>	<b>0.087</b>	<b>0.368</b>	<b>1.714</b>	<b>7.003</b>	<b>4.800</b>	<b>5.521</b>	<b>8.938</b>	<b>25.95</b>	<b>95.40</b>	<b>5.007</b>	<b>20.36</b>	<b>82.11</b>	<b>331.5</b>	<b>1339</b>
	Ours	0.039	0.114	0.542	2.796	13.18	4.872	5.705	8.992	26.09	101.6	5.805	23.57	95.05	383.6	1548
10	[19]	<b>0.088</b>	<b>0.286</b>	<b>1.183</b>	—	—	9.203	14.54	38.31	—	—	<b>20.48</b>	<b>82.39</b>	<b>332.0</b>	—	—
	Ours	0.110	0.337	1.483	7.167	—	<b>8.187</b>	<b>12.06</b>	<b>30.56</b>	105.7	—	24.07	96.82	390.1	1572	—

**Table 4.** Running time and communication costs for MPSU protocols in LAN and WAN settings. Cells with — denote trials running out of memory.

6. Bay, A., Erkin, Z., Hoepman, J., Samardjiska, S., Vos, J.: Practical multi-party private set intersection protocols. *IEEE Trans. Inf. Forensics Secur.* **17**, 1–15 (2022)
7. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: *Advances in Cryptology - CRYPTO '91*, 11th Annual International Cryptology Conference. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991)
8. Ben-Efraim, A., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A.: Psimple: Practical multiparty maliciously-secure private set intersection. In: *ASIA CCS '22*. pp. 1098–1112. ACM (2022)
9. Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Near-Optimal oblivious Key-Value stores for efficient PSI, PSU and Volume-Hiding Multi-Maps. In: *USENIX Security 2023*. pp. 301–318 (2023)
10. Blanton, M., Aguiar, E.: Private and oblivious set and multiset operations. In: *7th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2012*. pp. 40–41. ACM (2012)
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: 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*. pp. 291–308. ACM (2019)
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: *Advances in Cryptology - CRYPTO 2019*. Springer (2019)
13. Bui, D., Couteau, G.: 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*. Lecture Notes in Computer Science, vol. 13941, pp. 190–220. Springer (2023)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, 14–17 October 2001, Las Vegas, Nevada, USA. pp. 136–145. IEEE Computer Society (2001)
15. Chandran, N., Dasgupta, N., Gupta, D., Obbattu, S.L.B., Sekar, S., Shah, A.: Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In: *CCS '21*. pp. 1182–1204. ACM (2021)
16. Chandran, N., Gupta, D., Shah, A.: Circuit-psi with linear complexity via relaxed batch OPPrF. *Proc. Priv. Enhancing Technol.* **2022**(1), 353–372 (2022)
17. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: *Advances in Cryptology - CRYPTO 2020*. Lecture Notes in Computer Science, vol. 12172, pp. 34–63. Springer (2020)
18. Chen, Y., Ding, N., Gu, D., Bian, Y.: Practical multi-party private set intersection cardinality and intersection-sum under arbitrary collusion. In: *Information Security and Cryptology - 18th International Conference, Inscrypt 2022*. Lecture Notes in Computer Science, vol. 13837, pp. 169–191. Springer (2022)
19. Dong, M., Chen, Y., Zhang, C., Bai, Y.: Breaking free: Efficient multi-party private set union without non-collusion assumptions. *IACR Cryptol. ePrint Arch.* p. 1146 (2024), <https://eprint.iacr.org/2024/1146>
20. Eskandarian, S., Boneh, D.: Clarion: Anonymous communication from multiparty shuffling protocols. In: *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society (2022), <https://www.ndss-symposium.org/ndss-paper/auto-draft-243/>
21. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: *Theory of Cryptography, Second Theory of Cryptog-*

- raphy Conference, TCC 2005. Lecture Notes in Computer Science, vol. 3378, pp. 303–324. Springer (2005)
22. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: *Advances in Cryptology - EUROCRYPT 2004*. Lecture Notes in Computer Science, vol. 3027, pp. 1–19. Springer (2004)
  23. Frikken, K.B.: Privacy-preserving set union. In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007*. Lecture Notes in Computer Science, vol. 4521, pp. 237–252. Springer (2007)
  24. Gao, J., Nguyen, S., Trieu, N.: Toward A practical multi-party private set union. *Proc. Priv. Enhancing Technol.* **2024**(4), 622–635 (2024)
  25. Gao, J., Trieu, N., Yanai, A.: Multiparty private set intersection cardinality and its applications. *Proc. Priv. Enhancing Technol.* **2024**(2), 73–90 (2024)
  26. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: *Advances in Cryptology - CRYPTO 2021*. Lecture Notes in Computer Science, vol. 12826, pp. 395–425. Springer (2021)
  27. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: *Advances in Cryptology - EUROCRYPT 2019*. Lecture Notes in Computer Science, vol. 11478, pp. 154–185. Springer (2019)
  28. Giorgi, P., Laguillaumie, F., Ottow, L., Vergnaud, D.: Fast secure computations on shared polynomials and applications to private set operations. In: *5th Conference on Information-Theoretic Cryptography, ITC 2024*. LIPIcs, vol. 304, pp. 11:1–11:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)
  29. Goldreich, O.: *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press (2004), <http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html>
  30. Gordon, S.D., Hazay, C., Le, P.H.: Fully secure PSI via mpc-in-the-head. *Proc. Priv. Enhancing Technol.* **2022**(3), 291–313 (2022)
  31. Hazay, C., Lindell, Y.: *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography, Springer (2010)
  32. Hazay, C., Venkatasubramanian, M.: Scalable multi-party private set-intersection. In: *Public-Key Cryptography - PKC 2017*. Lecture Notes in Computer Science, vol. 10174, pp. 175–203. Springer (2017)
  33. Hazay, C., Venkatasubramanian, M.: Scalable multi-party private set-intersection. In: *Public-Key Cryptography - PKC 2017*. Lecture Notes in Computer Science, vol. 10174, pp. 175–203. Springer (2017)
  34. Inbar, R., Omri, E., Pinkas, B.: Efficient scalable multiparty private set-intersection via garbled bloom filters. In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018*. Lecture Notes in Computer Science, vol. 11035, pp. 235–252. Springer (2018)
  35. Jia, Y., Sun, S., Zhou, H., Gu, D.: Scalable private set union, with stronger security. In: *33rd USENIX Security Symposium, USENIX Security 2024*. USENIX Association (2024)
  36. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: *Advances in Cryptology - CRYPTO 2005*. Lecture Notes in Computer Science, vol. 3621, pp. 241–257. Springer (2005)
  37. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: *CCS 2016*. pp. 818–829. ACM (2016)

38. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: 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. pp. 1257–1272. ACM (2017)
39. Li, R., Wu, C.: An unconditionally secure protocol for multi-party set intersection. In: Applied Cryptography and Network Security, 5th International Conference, ACNS 2007. Lecture Notes in Computer Science, vol. 4521, pp. 226–236. Springer (2007)
40. Liu, X., Gao, Y.: Scalable multi-party private set union from multi-query secret-shared private membership test. In: Advances in Cryptology - ASIACRYPT 2023. Springer (2023)
41. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021. pp. 1151–1165. ACM (2021)
42. Pagh, R., Rodler, F.F.: Cuckoo hashing. *Journal of Algorithms* **51**(2), 122–144 (2004)
43. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from paxos: Fast, malicious private set intersection. In: Advances in Cryptology - EUROCRYPT 2020. Lecture Notes in Computer Science, vol. 12106, pp. 739–767. Springer (2020)
44. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12–14, 2015. pp. 515–530. USENIX Association (2015)
45. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: Advances in Cryptology - EUROCRYPT 2019. Lecture Notes in Computer Science, vol. 11478, pp. 122–153. Springer (2019)
46. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Proceedings of the 23rd USENIX Security Symposium, 2014. pp. 797–812. USENIX Association (2014)
47. Rabin, M.O.: How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.* p. 187 (2005), <http://eprint.iacr.org/2005/187>
48. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. In: ACM CCS 2022 (2022), <https://eprint.iacr.org/2022/320>
49. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from LPN. In: Advances in Cryptology - CRYPTO 2023. Lecture Notes in Computer Science, vol. 14084, pp. 602–632. Springer (2023)
50. Rindal, P.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library., <https://github.com/osu-crypto/libOTe.git>
51. Rindal, P., Schoppmann, P.: VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In: Advances in Cryptology - EUROCRYPT 2021. Lecture Notes in Computer Science, vol. 12697, pp. 901–930. Springer (2021)
52. Roy, L.: Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Advances in Cryptology - CRYPTO 2022. Springer (2022)
53. Sang, Y., Shen, H.: Efficient and secure protocols for privacy-preserving set operations. *ACM Trans. Inf. Syst. Secur.* **13**(1), 9:1–9:35 (2009)
54. Sang, Y., Shen, H., Tan, Y., Xiong, N.: Efficient protocols for privacy preserving matching against distributed datasets. In: Information and Communications Security, 8th International Conference, ICICS 2006. Lecture Notes in Computer Science, vol. 4307

55. Seo, J.H., Cheon, J.H., Katz, J.: Constant-round multi-party private set union using reversed laurent series. In: Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography. pp. 398–412. Springer (2012)
56. Su, J., Chen, Z.: Secure and scalable circuit-based protocol for multi-party private set intersection. CoRR **abs/2309.07406** (2023)
57. Vos, J., Conti, M., Erkin, Z.: Fast multi-party private set operations in the star topology from secure ands and ors. IACR Cryptol. ePrint Arch. p. 721 (2022), <https://eprint.iacr.org/2022/721>
58. Wu, M., Yuen, T.H., Chan, K.Y.: O-ring and k-star: Efficient multi-party private set intersection. In: 33rd USENIX Security Symposium, USENIX Security 2024. USENIX Association (2024)
59. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Optimal private set union from multi-query reverse private membership test. In: USENIX 2023 (2023), <https://eprint.iacr.org/2022/358>
60. Zhang, S.: Efficient VOLE based multi-party PSI with lower communication cost. IACR Cryptol. ePrint Arch. p. 1690 (2023), <https://eprint.iacr.org/2023/1690>

## A Theoretical Analysis and Comparison

### A.1 Complexity of Our MPSI and Its Variants

In the following analyses of asymptotic complexity, we consider the only dependency in  $n$  and  $m$ , omitting security parameters.

In Figure 13, the parties ( $P_1$  as  $P_{\text{pivot}}$ ) invoke the batch pure membership zero-sharing protocol of size  $B = O(n)$ . In this stage, the computation and communication complexity of  $P_1$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  ( $1 < j \leq m$ ) are  $O(n)$ . In MPSI/circuit-MPSI, each  $P_j$  directly sends its shares to  $P_1$ , thereby, the overall computation and communication complexity of  $P_1$  are  $O(mn)$ , while the overall computation and communication complexity of each  $P_j$  are  $O(n)$ ; In MPSI-card, the parties invoke the multi-party secret-shared shuffle protocol before the straightforward reconstruction. We use the multi-party secret-shared shuffle protocol in [20] and designate  $P_1$  as the leader. In this stage, the computation and communication complexity of  $P_1$  are  $O(mn)$ , while the computation and communication complexity of  $P_j$  are  $O(n)$ . In all, the computation and communication complexity of MPSI-card are identical to MPSI/circuit-MPSI.

In Figure 14, the parties invoke the batch pure membership zero-sharing with payloads protocol of size  $B = O(n)$ . In this stage, the computation and communication complexity of  $P_1$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  ( $1 < j \leq m$ ) are  $O(n)$ . Then, the parties invoke the multi-party secret-shared shuffle protocol twice,  $P_j$  reconstructs the cardinality to  $P_1$ , and  $P_1$  broadcasts the indicator vector for the shuffle payloads. In all, the computation and communication complexity of MPSI-card-sum remain the same as MPSI/MPSI-card/circuit-MPSI.

Notably, in the naive (insecure) solution, the clients directly sends their input sets to the leader and the leader computes the result locally, where the leader's



computation and communication complexity are  $O(mn)$  and each client's computation and communication complexity are  $O(n)$ . Therefore, our MPSI/MPSI-card/circuit-MPSI/MPSI-card-sum constructions achieve optimal complexity that matches the naive solution while ensuring security.

## A.2 Complexity of Our MPSU and Its Variants

In the following analyses of asymptotic complexity, we consider the only dependency in  $n$  and  $m$ , omitting security parameters.

In Figure 15,  $1 < j \leq m$ ,  $P_1, \dots, P_j$  ( $P_j$  as  $P_{\text{pivot}}$ ) invoke the batch pure non-membership zero-sharing protocol of size  $B = O(n)$ . Each  $P_j$  engages in  $m - j + 1$  invocations of batch pure non-membership zero-sharing protocols, acting as  $P_{\text{pivot}}$  in the first time.  $P_1$  engages in  $m - 1$  invocations of batch pure non-membership zero-sharing protocols without acting as  $P_{\text{pivot}}$ . In this stage, the computation and communication complexity of each party are  $O(mn)$ . After that, the parties hold  $(m - 1)B$  secret-sharings. Then, they invoke the multi-party secret-shared shuffle protocol (with  $P_1$  as the leader) with their  $(m - 1)B = O(mn)$  shares, and finally each  $P_j$  sends its shuffled shares to  $P_1$ . Thereby, the computation and communication complexity of  $P_1$  are  $O(m^2n)$ , while the computation and communication complexity of each  $P_j$  are  $O(mn)$ . As a result, the overall computation and communication complexity of  $P_1$  are  $O(m^2n)$ , while the computation and communication complexity of each  $P_j$  are  $O(mn)$ .

Our MPSU protocol follows the secret-sharing based MPSU paradigm, where the leader's optimal computation and communication complexity are  $O(m^2n)$ , while each client's optimal computation and communication complexity are  $O(mn)$ . This optimal complexity is determined by the core design of secret-sharing  $O(mn)$  elements among  $m$  parties, since the necessary reconstruction step requires the optimal complexity. Therefore, our MPSU construction achieves optimal computation and communication complexity of this MPSU paradigm.

## A.3 Comparison with Prior Works

Table 5 shows a theoretical comparison of the computation and communication required by various MPSI protocols. Table 6 shows a theoretical comparison between the related MPSI-card/MPSI-card-sum protocols and ours. Table 7 shows a theoretical comparison between the related MPSU protocols and ours.

## B The Proof of Theorem 1

*Proof.* We prove this theorem by constructing the predicate formula  $\varphi$  using mathematical induction.

- **Base Case.** If  $Y = X_i$  for some  $i \in \{1, \dots, m\}$ , then  $\varphi(x, X_1, \dots, X_m) = M(x, X_i) : x \in X_i$ .

Protocol	Computation		Communication		Security	Operation
	Leader	Client	Leader	Client		
[22]	$O(m^2n^2)$	$O(m^2n^2)$	$O(m^2n^2\lambda)$	$O(m^2n^2\lambda)$	standard	PK
[36]	$O(mtn^2)$	$O(mtn^2)$	$O(mtn \log U \lambda)$	$O(mtn \log U \lambda)$	standard semi-honest	PK
[33]	$O(mn^2)$	$O(n)$	$O(mn\lambda)$	$O(n\lambda)$	standard semi-honest	PK
[38]	$O(mn)$	$O(n)$	$O(mn(\lambda + \sigma + \log n))$	$O(n(\lambda + \sigma + \log n))$	augmented semi-honest	SK
		$O(tn)$		$O(tn(\lambda + \sigma + \log n))$	standard semi-honest	SK
[34]	$O(mn)$	$O(mn)$	$O(mn)$	$O(\log(m)n\sigma^2)$	augmented semi-honest	SK
				$O(mn\sigma^2)$	standard semi-honest	SK
[27]	$O(mn \log n)$	$O(n \log n)$	$O((m^2 + mn)\lambda)$		malicious	SK
[8]	$O(mn)$		$O(mn\lambda(\log(n\lambda) + \lambda))$	$O(n\lambda(\log(n\lambda) + \lambda))$	augmented semi-honest/malicious	SK
[41]	$O(mn)$	$O(tn)$	$O(mn(\lambda + \sigma + \log n))$	$O(n(\lambda + \sigma + \log n))$	augmented semi-honest/malicious	SK
[58]	$O(mn)$	$O(tn)$	$O(mn(\lambda + \sigma + \log n))$	$O(tn(\lambda + \sigma + \log n))$	standard semi-honest	SK
Ours	$O(mn)$	$O(n)$	$O(mn(\sigma + \log n))$	$O(n(\sigma + \log n))$	standard semi-honest	SK

**Table 5.** Asymptotic communication and computation costs of MPSI protocols in the semi-honest setting, where  $n$  is the set size.  $m$  is the number of parties.  $t$  is the number of colluding parties.  $U$  is the domain of elements. We use “PK” to denote the protocols based on public-key operations, and “SK” to denote the protocols based on OT and symmetric-key operations. We use  $\lambda, \sigma$  as the computational and statistical security parameters respectively. We use “augmented semi-honest/malicious” to denote the malicious protocols that implies augmented semi-honest security while is insecure in standard semi-honest model (A detailed discussion of the relations between malicious model and augmented / standard semi-honest model can be found in [31] Section 2.4.4).

Protocol	Computation		Communication		Security	Operation
	Leader	Client	Leader	Client		
[18]	$O((m-t)n + tn \log n)$	$O(tn)$	$O(((m-t)n + tn \log n)(\sigma + \log n))$	$O(tn(\sigma + \log n))$	standard semi-honest	SK
Ours	$O(mn)$	$O(n)$	$O(mn(\sigma + \log n))$	$O(n(\sigma + \log n))$	standard semi-honest	SK

**Table 6.** Asymptotic communication and computation costs of MPSI-card/MPSI-card-sum protocols in the semi-honest setting, where  $n$  is the set size.  $m$  is the number of parties. We set the number of colluding parties  $t$  as the maximum  $m - 1$ . We use “SK” to denote the protocols based on OT and symmetric-key operations. We use  $\lambda, \sigma$  as the computational and statistical security parameters respectively.

Protocol	Computation		Communication		Security	Operation
	Leader	Client	Leader	Client		
[24]	$O(mn(\log n / \log \log n))$		$\lambda mn(\log n / \log \log n)$		standard semi-honest	PK
[19]	$O(m^2n)$	$O(m^2n)$	$O(m^2n(l + \sigma + \log m + \log n))$	$O(m^2n(l + \sigma + \log m + \log n))$	standard semi-honest	SK
Ours	$O(m^2n)$	$O(mn)$	$O(m^2n(l + \sigma + \log m + \log n))$	$O(mn(l + \sigma + \log m + \log n))$	standard semi-honest	SK

**Table 7.** Asymptotic communication and computation costs of MPSU protocols in the semi-honest setting, where  $n$  is the set size.  $m$  is the number of parties.  $l$  is the length of elements. We use “PK” to denote the protocols based on public-key operations, and “SK” to denote the protocols based on OT and symmetric-key operations. We use  $\lambda, \sigma$  as the computational and statistical security parameters.

- **Induction Hypothesis.** Assume that for any sets  $A$  and  $B$  obtained from  $X_1, \dots, X_m$  through  $k$  set operations, there exist set predicate formulas  $\varphi_A$  and  $\varphi_B$  such that

$$x \in A \iff \varphi_A(x, X_1, \dots, X_m) = 1, \quad x \in B \iff \varphi_B(x, X_1, \dots, X_m) = 1.$$

- **Induction Step.** We proceed to construct  $\varphi$  for a set  $Y$  obtained from  $A$  and  $B$  through one additional set operation (intersection, union, difference), conducting  $k + 1$  set operations in total.
  1. **Union.** If  $Y = A \cup B$ , then  $\varphi(x, X_1, \dots, X_m) = \varphi_A(x, X_1, \dots, X_m) \vee \varphi_B(x, X_1, \dots, X_m)$ .
  2. **Intersection.** If  $Y = A \cap B$ ,  $\varphi(x, X_1, \dots, X_m) = \varphi_A(x, X_1, \dots, X_m) \wedge \varphi_B(x, X_1, \dots, X_m)$ .
  3. **Difference.** If  $Y = A \setminus B$ ,  $\varphi(x, X_1, \dots, X_m) = \varphi_A(x, X_1, \dots, X_m) \wedge \neg \varphi_B(x, X_1, \dots, X_m)$ .

By repeating the above steps, we construct the set predicate formula  $\varphi$  for any constructible set  $Y$ . Thus, the theorem is proven.

## C The Proof of Theorem 2

*Proof.* Given that any set predicate formula can be transformed into disjunctive normal form (DNF), Theorem 1 can be further extended to represent  $Y$  as a DNF formula

$$\varphi(x, X_1, \dots, X_m) = C_1 \vee \dots \vee C_n$$

where each  $C_i$  ( $1 \leq i \leq n$ ) is a conjunctive clause. We now show that  $\varphi$  can be transform into another DNF formula  $\psi$  with  $s$  conjunctive clauses ( $s > n$ ) such that each  $C_i$  ( $1 \leq i \leq s$ ) contains at least one atomic proposition of the form  $x \in X_j$ , i.e.,  $C_i$  can be written in the form  $C_i = (x \in X_j) \wedge D_i$  for some  $X_j$ .

The proof is by contradiction. Suppose there is a clause  $C_i$  containing no atomic propositions of the form  $x \in X_j$ , i.e.  $C_i$  is the conjunction of atomic propositions of the form  $x \notin X_j$ . Consider two cases:

- **Case 1:**  $C_i = (x \notin X_{i_1}) \wedge \dots \wedge (x \notin X_{i_t})$  where  $t < m$ . As  $C_i$  is a conjunctive clause of  $\varphi$ , the corresponding set  $Y'_i$  is a subset of the constructible set  $Y$ , and  $Y'_i$  is also a constructible set. Hence, we can augment  $C_i$  by

$$C_i = C_i \wedge ((x \in X_1) \vee \dots \vee (x \in X_m)) = (C_i \wedge (x \in X_1)) \vee \dots \vee (C_i \wedge (x \in X_m)).$$

For any clause that contains both  $x \in X_{i_d}$  and its negation  $x \notin X_{i_d}$  ( $1 \leq d \leq t$ ), it evaluates to 0 and can be discarded. The remaining formula is

$$C_i = (C_i \wedge (x \in X_{j_1})) \vee \dots \vee (C_i \wedge (x \in X_{j_{m-t}})) = C_{i,1} \vee \dots \vee C_{i,m-t}.$$

This splits each  $C_i$  into  $m - t$  conjunctive clauses where each clause contains at least one literal of the form  $x \in X_j$ . We substitute each  $C_i$  with the above equation and have the new DNF formula

$$\psi(x, X_1, \dots, X_m) = C_1 \vee \dots \vee C_s,$$

where each  $C_i$  ( $1 \leq i \leq s$ ) represents an  $X_j$ -constructible set for some  $X_j$  ( $1 \leq j \leq m$ ). Note that  $C_i$  is not set-separable with respect to  $X_j$  yet, since  $D_i$  might involve atomic propositions relevant to  $X_j$ .

- **Case 2:**  $C_i = (x \notin X_1) \wedge \dots \wedge (x \notin X_m)$ . This contradicts that the set  $Y_i$  represented by  $C_i$  is constructible from  $X_1, \dots, X_m$ , so this case is not valid.

Next we transform the new DNF formula  $\psi$  into a disjunction of subformulas that represent disjoint sets  $\{Y_1, \dots, Y_s\}$ . Since the disjunction form of  $\psi$  implies  $Y_1 \cup \dots \cup Y_s = Y$ , this will demonstrate that  $\{Y_1, \dots, Y_s\}$  form a partition of  $Y$ .

Let  $\psi_1(x, X_1, \dots, X_m) = C_1 \vee \dots \vee C_s$ , then we have

$$\psi(x, X_1, \dots, X_m) = C_1 \vee \psi_1(x, X_1, \dots, X_m).$$

We augment  $C_1 \vee \psi_1(x, X_1, \dots, X_m)$  as  $C_1 \vee ((C_1 \wedge \neg C_1) \vee \psi_1(x, X_1, \dots, X_m))$ , which can expand into  $C_1 \vee (C_1 \wedge \psi_1(x, X_1, \dots, X_m)) \vee (\neg C_1 \wedge \psi_1(x, X_1, \dots, X_m))$ . Given that  $C_1 \wedge \psi_1(x, X_1, \dots, X_m) = 1$  necessitate  $C_1 = 1$ , we have  $C_1 \vee (C_1 \wedge \psi_1(x, X_1, \dots, X_m)) = C_1$ , hence

$$\psi(x, X_1, \dots, X_m) = C_1 \vee (\neg C_1 \wedge \psi_1(x, X_1, \dots, X_m)).$$

By repeating this process for all  $C_i$  ( $1 \leq i \leq s$ ), we obtain

$$\psi(x, X_1, \dots, X_m) = C_1 \vee (\neg C_1 \wedge C_2) \vee \dots \vee (\neg C_1 \wedge \neg C_2 \wedge \dots \wedge \neg C_{s-1} \wedge C_s).$$

We denote as  $\psi(x, X_1, \dots, X_m) = C'_1 \vee C'_2 \vee \dots \vee C'_s$ . For any two distinct  $C'_i$  and  $C'_k$ , it is easy to see that  $C'_i \wedge C'_k = 0$ , so the sets  $Y_i$  and  $Y_k$  represented by  $C'_i$  and  $C'_k$  satisfy that  $Y_i \cap Y_k = \emptyset$ . Thus each  $C'_i$  represents a disjoint set.

Finally, we prove that each  $C'_i$  can be reduced to be set-separable with respect to  $X_j$ . By definition,  $C'_i$  is the conjunction of negations of previous clauses  $C_k$  ( $1 \leq k < i$ ) and  $C_i$ :

$$C'_i = \bigwedge_{k=1}^{i-1} \neg C_k \wedge C_i.$$

Since  $C_i$  can be written as  $(x \in X_j) \wedge D_i$ , where  $D_i$  is the remaining part of the conjunctive clause. Substituting this into  $C'_i$ , we get

$$C'_i = \bigwedge_{j=1}^{i-1} \neg C_j \wedge (x \in X_j) \wedge D_i = (x \in X_j) \wedge D'_i.$$

At this point,  $D'_i$  may also contain atomic propositions relevant to  $X_j$ . We now show how to eliminate these redundant atomic propositions: A key observation is that for  $C'_i = 1$  to hold, the condition  $x \in X_j$  must be true. Therefore, we reduce  $C'_i$  by assigning a truth value of 1 to all terms of the form  $x \in X_j$  and a truth value of 0 to all terms of the form  $x \notin X_j$ . After this reduction, we obtain a reduced formula  $Q_i = (x \in X_j) \wedge Q'_i$ , which is equivalent to  $C'_i$  but  $Q'_i$  contains no atomic propositions relevant to  $X_j$ . Thus,  $Q_i$  is set-separable with respect to  $X_j$ . The proof is complete.

## D The Proof of Theorem 3

*Proof.* We prove the theorem by induction on the number of parties corrupted by the adversary  $\mathcal{A}$ .

– **Base Case:  $t = m - 1$**

Assume  $\mathcal{A}$  corrupts  $t = m - 1$  parties. Denote the set of corrupted parties as  $\mathbf{P}_{\mathcal{A}} = \{P_{i_1}, \dots, P_{i_{m-1}}\}$ , leaving only one honest party  $P_h$ . According to the privacy requirement, there exists a simulator  $\text{Sim}$  such that

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}})\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x})\}_{\mathbf{x}}$$

We use  $\mathbf{r}$  (resp.  $\mathbf{r}^{\Pi}$ ) to denote the randomness in  $f(\mathbf{x})$  in the ideal (resp. real) execution. It is easy to see that in the ideal execution,  $\mathbf{r}$  is independent of  $\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}})$ . Meanwhile, by the independence requirement,  $\mathbf{r}^{\Pi}$  is independent of  $\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x})$  in the real execution, so we can obtain

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{r}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{r}^{\Pi}\}_{\mathbf{x}}.$$

We further extend the indistinguishability into

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{r}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, \mathbf{r}^{\Pi}\}_{\mathbf{x}}$$

where  $\mathbf{s}_{\mathcal{A}}^{\Pi} = (s_{i_1}^{\Pi}, \dots, s_{i_{m-1}}^{\Pi})$ , since each corrupted party  $P_i$ 's output  $s_i$  (resp.  $s_i^{\Pi}$ ) can be computed from its own view in the ideal (resp. real) execution ( $i \in \{i_1, \dots, i_{m-1}\}$ ).

By the functionality, the output of  $P_h$  satisfies  $s_h = -(\sum_{s_i \in \mathbf{s}_{\mathcal{A}}} s_i) + f(\mathbf{x})$ . By the correctness requirement,  $s_h^{\Pi} = -(\sum_{s_i^{\Pi} \in \mathbf{s}_{\mathcal{A}}^{\Pi}} s_i^{\Pi}) + f(\mathbf{x})$ . Thus, we extend the previous distributions by including  $s_h$  and  $s_h^{\Pi}$  respectively and obtain

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, s_h, \mathbf{r}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, s_h^{\Pi}, \mathbf{r}^{\Pi}\}_{\mathbf{x}}.$$

The indistinguishability holds because  $s_h$  (resp.  $s_h^{\Pi}$ ) is determined by  $\mathbf{s}_{\mathcal{A}}$  (resp.  $\mathbf{s}_{\mathcal{A}}^{\Pi}$ ), the randomness  $\mathbf{r}$  (resp.  $\mathbf{r}^{\Pi}$ ), and the parties' inputs  $\mathbf{x}$ . This implies

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, s_h\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, s_h^{\Pi}\}_{\mathbf{x}}.$$

Namely,  $\Pi$  securely computes  $f$  when  $\mathcal{A}$  corrupting  $t = m - 1$  parties. Note that the independence requirement in this case implies that  $\mathbf{r}^{\Pi}$  is independent of the joint view of any  $m - 1$  parties in the real execution, which will be used in the subsequent proof.

– **Inductive Hypothesis:  $t = m - k$**

Assume that for any adversary  $\mathcal{A}$  corrupting  $t = m - k$  parties ( $1 \leq k < m - 1$ ),  $\Pi$  securely computes  $f$ . Namely, there exists a simulator  $\text{Sim}$  such that

$$\begin{aligned} & \{\text{Sim}(\{P_1, \dots, P_m\} \setminus \mathbf{P}_{\mathcal{H}}, \{x_1, \dots, x_m\} \setminus \mathbf{x}_{\mathcal{H}}, \{s_1, \dots, s_m\} \setminus \mathbf{s}_{\mathcal{H}}), \{s_1, \dots, s_m\} \setminus \mathbf{s}_{\mathcal{H}}, \mathbf{s}_{\mathcal{H}}\}_{\mathbf{x}} \\ & \stackrel{c}{\approx} \{\{\text{View}_1^{\Pi}(\mathbf{x}), \dots, \text{View}_m^{\Pi}(\mathbf{x})\} \setminus \{\text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x})\}, \{s_1^{\Pi}, \dots, s_m^{\Pi}\} \setminus \mathbf{s}_{\mathcal{H}}^{\Pi}, \mathbf{s}_{\mathcal{H}}^{\Pi}\}_{\mathbf{x}}, \end{aligned}$$

where  $\mathbf{P}_{\mathcal{H}}$  is the set of honest parties,  $\text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x})$  denotes the joint view of  $\mathbf{P}_{\mathcal{H}}$ , while  $\mathbf{s}_{\mathcal{H}}$  and  $\mathbf{s}_{\mathcal{H}}^{\Pi}$  are the respective outputs in the ideal and real executions.

It is easy to see that in the ideal execution,  $\mathbf{s}_{\mathcal{H}}$  is independent of the joint distribution  $\{\text{Sim}(\{P_1, \dots, P_m\} \setminus \mathbf{P}_{\mathcal{H}}, \{x_1, \dots, x_m\} \setminus \mathbf{x}_{\mathcal{H}}, \{s_1, \dots, s_m\} \setminus \mathbf{s}_{\mathcal{H}}), \{s_1, \dots, s_m\} \setminus \mathbf{s}_{\mathcal{H}}\}$ . Thus, we can conclude that for any subset of parties  $\mathbf{P}_{\mathcal{H}} \subset \{P_1, \dots, P_m\}$  of size  $k$ ,  $\mathbf{s}_{\mathcal{H}}^{\Pi}$  is independent of the joint distribution  $\{\{\text{View}_1^{\Pi}(\mathbf{x}), \dots, \text{View}_m^{\Pi}(\mathbf{x})\} \setminus \text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x}), \{s_1^{\Pi}, \dots, s_m^{\Pi}\} \setminus \mathbf{s}_{\mathcal{H}}^{\Pi}\}$  in the real execution.

– **Inductive Step:  $t = m - k - 1$**

We proceed to prove the case where  $\mathcal{A}$  corrupts  $t = m - k - 1$  parties:

Let  $\mathbf{P}_{\mathcal{H}'}$  represent a subset of  $\{P_1, \dots, P_m\}$  with size  $k + 1$ . We decompose it into  $\mathbf{P}_{\mathcal{H}'} = \{\mathbf{P}_{\mathcal{H}}, P_h\}$ , where  $\mathbf{P}_{\mathcal{H}} \subset \{P_1, \dots, P_m\}$  contains exact  $k$  parties while  $P_h$  is the remaining one party. By the privacy, we have

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}\}_{\mathbf{x}}.$$

By the correctness, we also have

$$\{\mathbf{s}_{\mathcal{H}}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\mathbf{s}_{\mathcal{H}}^{\Pi}\}_{\mathbf{x}}.$$

From the inductive hypothesis,  $\mathbf{s}_{\mathcal{H}}$  is independent of the joint distribution  $\{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}\}$ , given that it is a subdistribution of  $\{\{\text{View}_1^{\Pi}(\mathbf{x}), \dots, \text{View}_m^{\Pi}(\mathbf{x})\} \setminus \text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x}), \{s_1^{\Pi}, \dots, s_m^{\Pi}\} \setminus \mathbf{s}_{\mathcal{H}}^{\Pi}\}$ . It is easy to see that  $\mathbf{s}_{\mathcal{H}}$  is independent of the joint distribution  $\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}\}_{\mathbf{x}}$ . Combining the above,

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{s}_{\mathcal{H}}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, \mathbf{s}_{\mathcal{H}}^{\Pi}\}_{\mathbf{x}}.$$

Recall that in the base case we derived that  $\mathbf{r}^{\Pi}$  is independent of the joint view of any  $m - 1$  parties in the real execution, thereby,  $\mathbf{r}^{\Pi}$  is independent of  $\{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x})\}$ . As  $\mathbf{s}_{\mathcal{A}}^{\Pi}$  and  $\mathbf{s}_{\mathcal{H}}^{\Pi}$  can be determined by  $\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x})$  and  $\text{View}_{\mathcal{H}}^{\Pi}(\mathbf{x})$  respectively,  $\mathbf{r}^{\Pi}$  is independent of  $\{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, \mathbf{s}_{\mathcal{H}}^{\Pi}\}$ . Furthermore, in the ideal execution,  $\mathbf{r}$  is independent of  $\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{s}_{\mathcal{H}}\}$ , which can extend the previous indistinguishability into

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{s}_{\mathcal{H}}, \mathbf{r}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, \mathbf{s}_{\mathcal{H}}^{\Pi}, \mathbf{r}^{\Pi}\}_{\mathbf{x}}.$$

By the functionality, the output of  $P_h$  satisfies  $s_h = -(\sum_{s_i \in \mathbf{s}_{\mathcal{A}}} s_i + \sum_{s_i \in \mathbf{s}_{\mathcal{H}}} s_i) + f(\mathbf{x})$ . By the correctness,  $s_h^{\Pi} = -(\sum_{s_i^{\Pi} \in \mathbf{s}_{\mathcal{A}}^{\Pi}} s_i^{\Pi} + \sum_{s_i^{\Pi} \in \mathbf{s}_{\mathcal{H}}^{\Pi}} s_i^{\Pi}) + f(\mathbf{x})$ . Thus, we extend the previous distributions by including  $s_h$  and  $s_h^{\Pi}$  respectively and obtain

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{s}_{\mathcal{H}}, s_h, \mathbf{r}\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, \mathbf{s}_{\mathcal{H}}^{\Pi}, s_h^{\Pi}, \mathbf{r}^{\Pi}\}_{\mathbf{x}}$$

The indistinguishability holds because  $s_h$  (resp.  $s_h^{\Pi}$ ) is uniquely determined by  $\mathbf{s}_{\mathcal{A}}$  and  $\mathbf{s}_{\mathcal{H}}$  (resp.  $\mathbf{s}_{\mathcal{A}}^{\Pi}$  and  $\mathbf{s}_{\mathcal{H}}^{\Pi}$ ), the randomness  $\mathbf{r}$  (resp.  $\mathbf{r}^{\Pi}$ ), and the parties' inputs  $\mathbf{x}$ . This implies

$$\{\text{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, s_h\}_{\mathbf{x}} \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{\Pi}(\mathbf{x}), \mathbf{s}_{\mathcal{A}}^{\Pi}, s_h^{\Pi}\}_{\mathbf{x}}.$$

Namely,  $\Pi$  securely computes  $f$  in the presence of  $\mathcal{A}$  corrupting  $t = m - k - 1$  parties. This completes the inductive step.

## E The Optimization in Section 4.4

We first recall the technique of Beaver triples. A Beaver triple consists of three secret-sharings  $([a], [b], [c])$ , where  $[a], [b]$  are random secret-sharings and  $c = a \cdot b$ . Typically, a Beaver triple is used to reduce one multiplication to two reconstructions in the online phase, while here since the multiplier  $b$  is random, a Beaver triple can be used to reduce one multiplication to one reconstruction in the online phase. Concretely,

$$s = r \cdot b = (r + a - a) \cdot b = (r + a) \cdot b + a \cdot b$$

hence we can compute

$$[s] = (r + a) \cdot [b] + [a \cdot b] = (r + a) \cdot [b] + [c]$$

The above equation suggests that we can locally compute  $[s]$  once  $u = r + a$  is publicly known. Therefore, the task of generating  $[s]$  boils down to reconstructing  $[u] = [r] + [a]$ . Let each party  $P_i$  locally compute  $u_i = r_i + a_i$  and send  $u_i$  to the leader  $P_1$ , then  $P_1$  computes  $u = u_1 + \dots + u_m$  and opens it to all parties. As we can see, the transformation only consumes one Beaver triple generated in the offline phrase and requires one opening with  $2(m-1) \log_2 \log_2 |\mathbb{F}|$  communication overhead for the leader and  $2 \log_2 \log_2 |\mathbb{F}|$  communication overhead for each client in the online phrase.

## F Membership Zero-Sharing Appendix

### F.1 Pure Membership Zero-Sharing

The (batch) pure membership zero-sharing functionality is a special case of (batch) membership zero-sharing when  $Q$  is a conjunction of  $m-1$  set membership predicates (i.e.,  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} x \in X_j$ ). The ideal functionality  $\mathcal{F}_{\text{bPMZS}}$  is formally described in Figure 17. The complete protocol is given in Figure 18.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  is the only party holding  $n$  single elements as inputs instead of  $n$  sets. Batch size  $n$ . A field  $\mathbb{F}$ .

**Functionality:** On input  $\mathbf{x} = (x_1, \dots, x_n)$  from  $P_{\text{pivot}}$  and  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  from each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ), sample  $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,n}) \leftarrow \mathbb{F}^n$  for  $1 \leq i \leq m$ , s.t. for  $1 \leq d \leq n$ , if  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} (x_d \in X_{j,d}) = 1$ ,  $\sum_{1 \leq i \leq m} s_{i,d} = 0$ . Give  $\mathbf{s}_i$  to  $P_i$ .

**Fig. 17.** Batch Pure Membership Zero-Sharing Functionality  $\mathcal{F}_{\text{bPMZS}}$

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ . Batch size  $n$ . A field  $\mathbb{F}$ .  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$  generated in the offline phrase, where  $[\mathbf{a}] = ([a_1], \dots, [a_n])$ ,  $[\mathbf{b}] = ([b_1], \dots, [b_n])$ ,  $[\mathbf{c}] = ([c_1], \dots, [c_n])$  and  $c_i = a_i \cdot b_i$  for  $1 \leq i \leq n$ .

**Inputs:**  $P_{\text{pivot}}$  inputs a vector  $\mathbf{x} = (x_1, \dots, x_n)$ .  $P_j$  inputs  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ .

**Protocol:**

1. For the  $i$ -th instance ( $1 \leq i \leq n$ ),  $P_j$  samples  $r_{j,i}$  and sets  $K_{j,i} = X_{j,i}$  and  $V_{j,i} = \{-r_{j,i}, \dots, -r_{j,i}\}$ , where  $|K_{j,i}| = |V_{j,i}|$ .
2.  $P_{\text{pivot}}$  and  $P_j$  invoke  $\mathcal{F}_{\text{bOPPRF}}$  where  $P_j$  acts as  $\mathcal{S}$  inputting  $(K_{j,1}, \dots, K_{j,n})$  and  $(V_{j,1}, \dots, V_{j,n})$ , and  $P_{\text{pivot}}$  acts as  $\mathcal{R}$  with input  $\mathbf{x}$  and receives  $\mathbf{u}_j$ .
3.  $P_{\text{pivot}}$  sets its shares  $\mathbf{r}_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} \mathbf{u}_j$ .  $P_j$  sets its shares  $\mathbf{r}_j = (r_{j,1}, \dots, r_{j,n})$ . All parties hold a vector of  $n$  secret-sharings  $[\mathbf{r}] = ([r_1], \dots, [r_n])$ .
4. All parties compute  $[\mathbf{s}]$  by performing  $n$  secure multiplications  $[s_i] = [r_i] \cdot [b_i]$  ( $1 \leq i \leq n$ ), using  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ .

**Fig. 18.** Batch Pure Membership Zero-Sharing  $\Pi_{\text{bPMZS}}$

**Complexity Analysis.** In the batch pure membership zero-sharing protocol (Figure 17), the costs of each stage are calculated as follows.

- $P_{\text{pivot}}$  executes batch OPRF of size  $n$  with each  $P_j$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ . Suppose that in the subsequent invocations of batch OPRF, each  $|X_{j,i}| = N_{j,i} = O(1)$  for  $1 \leq i \leq n, j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ , which is consistent with the use of batch membership zero-sharing protocols in our MPSO protocols (combined with hashing to bins technique). We follow the paradigm in [45] to construct batch OPRF from batch OPRF and OKVS. By leveraging the technique to amortize communication, the total communication of computing  $n$  instances of OPRF is equal to the total number of items  $3n$ . Furthermore, we utilize vector oblivious linear evaluation (VOLE) [11, 12, 49] to instantiate batch OPRF and the construction in [48] to instantiate OKVS. This ensures the computation complexity of batch OPRF of size  $n$  to scale linearly with  $n$ . Therefore, in this stage, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .
- The parties perform  $n$  secure multiplications using the optimization outlined in the previous section and designate  $P_{\text{pivot}}$  as the leader. This requires  $n$  opening with  $O(mn)$  computation/communication complexity for  $P_{\text{pivot}}$  and  $O(n)$  computation/communication complexity for each  $P_j$ .

To sum up, in the online phase of the batch pure membership zero-sharing protocol, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .



## F.2 Pure Non-Membership Zero-Sharing

The (batch) pure non-membership zero-sharing functionality is a special case of (batch) membership zero-sharing when  $Q$  is a conjunction of  $m - 1$  set non-membership predicates (i.e.,  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} x \notin X_j$ ). The ideal functionality  $\mathcal{F}_{\text{bpNMZS}}$  is formally described in Figure 19. The complete protocols is given in Figure 20.

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ , where  $P_{\text{pivot}}$  is the only party holding  $n$  single elements as inputs instead of  $n$  sets. Batch size  $n$ . A field  $\mathbb{F}$ .

**Functionality:** On input  $\mathbf{x} = (x_1, \dots, x_n)$  from  $P_{\text{pivot}}$  and  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  from each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ), sample  $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,n}) \leftarrow \mathbb{F}^n$  for  $1 \leq i \leq m$ , s.t. for  $1 \leq d \leq n$ , if  $\bigwedge_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} x_d \notin X_{j,d} = 1$ ,  $\sum_{1 \leq i \leq m} s_{i,d} = 0$ . Give  $\mathbf{s}_i$  to  $P_i$ .

**Fig. 19.** Batch Pure Non-Membership Zero-Sharing Functionality  $\mathcal{F}_{\text{bpNMZS}}$

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ . Batch size  $n$ . A field  $\mathbb{F}$ .  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$  generated in the offline phase, where  $[\mathbf{a}] = ([a_1], \dots, [a_n])$ ,  $[\mathbf{b}] = ([b_1], \dots, [b_n])$ ,  $[\mathbf{c}] = ([c_1], \dots, [c_n])$  and  $c_i = a_i \cdot b_i$  for  $1 \leq i \leq n$ .

**Inputs:**  $P_{\text{pivot}}$  inputs a vector  $\mathbf{x} = (x_1, \dots, x_n)$ .  $P_j$  inputs  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ .

**Protocol:**

1.  $P_{\text{pivot}}$  and  $P_j$  invoke  $\mathcal{F}_{\text{bssPMT}}$  where in the  $i$ -th instance ( $1 \leq i \leq n$ ),  $P_j$  inputs  $X_{j,i}$  and receives  $e_{j,i}^0$ , while  $P_{\text{pivot}}$  inputs  $x_i$  and receives  $e_{j,i}^1$ .
2.  $P_{\text{pivot}}$  and  $P_j$  invoke  $n$  instances of ROT where in the  $i$ -th instance ( $1 \leq i \leq n$ ),  $P_j$  acts as  $\mathcal{S}$  and receives  $r_{j,i}^0, r_{j,i}^1$ , while  $P_{\text{pivot}}$  acts as  $\mathcal{R}$  inputting  $e_{j,i}^1$  and receives  $r_{j,i}^{e_{j,i}^1}$ .  $P_{\text{pivot}}$  sets  $\mathbf{r}'_j = (r_{j,1}^{e_{j,1}^1}, \dots, r_{j,n}^{e_{j,n}^1})$ .
3.  $P_{\text{pivot}}$  sets its shares  $\mathbf{r}_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} \mathbf{r}'_j$ .  $P_j$  sets its shares  $\mathbf{r}_j = (-r_{j,1}^{e_{j,1}^0}, \dots, -r_{j,n}^{e_{j,n}^0})$ . All parties hold a vector of  $n$  secret-sharings  $[\mathbf{r}] = ([r_1], \dots, [r_n])$ .
4. All parties compute  $[\mathbf{s}]$  by performing  $n$  secure multiplications  $[s_i] = [r_i] \cdot [b_i]$  ( $1 \leq i \leq n$ ), using  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ .

**Fig. 20.** Batch Pure Non-Membership Zero-Sharing  $\Pi_{\text{bpNMZS}}$

**Complexity Analysis.** In the batch pure non-membership zero-sharing protocol (Figure 19), the costs of each stage are calculated as follows.

- $P_{\text{pivot}}$  executes batch ssPMT of size  $n$  with each  $P_j$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ . We utilize the construction in [19] based on batch OPPRF and secret-shared private equality test (ssPEQT) [45, 16], which achieves linear computation and communication complexity. Therefore, in this stage, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .
- $P_{\text{pivot}}$  acts as  $\mathcal{R}$  and executes  $n$  instances of ROT with each  $P_j$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ . In the offline phases,  $P_{\text{pivot}}$  and each  $P_j$  generate  $n$  instances of random-choice-bit ROT, then in the online phase,  $P_{\text{pivot}}$  only needs to send  $n$  choice bits masked by the random choice bits to each  $P_j$ . Therefore, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .
- The parties perform  $n$  secure multiplications using the optimization outlined in the previous section and designate  $P_{\text{pivot}}$  as the leader. This requires  $n$  opening with  $O(mn)$  computation/communication complexity for  $P_{\text{pivot}}$  and  $O(n)$  computation/communication complexity for each  $P_j$ .

To sum up, in the online phase of the batch pure non-membership zero-sharing protocol, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .

### F.3 Pure Membership Zero-Sharing with Payloads

Pure membership zero-sharing with payloads is an extension of the pure membership zero-sharing functionality, combined with a variant of relaxed pure membership payload-sharing, which we call relaxed pure membership payload-sharing. In this variant,  $P_{\text{pivot}}$  holds an element  $x$  while each of the others holds a set of elements and a set of associated payloads. If the conjunction of set membership predicates holds true (i.e.,  $x$  belongs to all element sets), the parties receive secret shares of the sum of all payloads associated with  $x$ ; otherwise they receive secret shares of a random value. The formal definition of batch pure membership zero-sharing with payloads functionality is in Figure 11. Note that the payload-sharing only needs to satisfy the relaxed security definition in Section 4.3.

The construction of batch pure membership zero-sharing with payloads protocol resembles the batch pure membership zero-sharing protocol in Figure 18. The core idea is to somehow encode the payload set into the senders' inputs of OPPRF, in each two-party protocol of the relaxed pure membership payload-sharing. Specifically, we start by implementing the relaxed pure membership zero-sharing with payloads in the two-party setting. Next, we show how to extend this primitive into multi-party setting.

In the two-party relaxed membership zero-sharing with payloads protocol, there are two parties, the sender  $\mathcal{S}$  with an element set  $Y$  and a payload set  $V$  and the receiver  $\mathcal{R}$  with an element  $x$ .  $\mathcal{S}$  samples two secret shares  $r, w$ , and sets  $Y$  as the key set and a set containing the pair  $(-r, v_i - w)$  for  $1 \leq i \leq n$  as the value set, where  $v_i \in V$  is the associated payload with  $y_i \in Y$ .  $\mathcal{S}$  outputs  $(r, w)$  as its

two secret shares.  $\mathcal{S}$  and  $\mathcal{R}$  invoke OPPRF, where  $\mathcal{R}$  inputs  $x$  and receives  $(r', w')$  as its secret share. By the definition of OPPRF, if  $x \in Y$ ,  $(r', w') = (-r, v - w)$ , where  $v$  is the associated payload with  $x$  in  $Y$ . Namely, if  $x \in Y$ , the parties hold one secret sharing of 0 and one secret sharing of the associated payload with  $x$ , otherwise they hold two secret sharings of pseudorandom values.

In the multi-party membership zero-sharing with payloads protocol, there are  $m$  ( $m > 2$ ) parties, where  $P_{\text{pivot}}$  holds an element  $x$  and each  $P_j$  ( $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ ) holds an element set  $X_j$  and a payload set  $V_j$ .  $P_{\text{pivot}}$  engages in the two-party version with each  $P_j$ , where  $P_{\text{pivot}}$  receives  $r'_j$  and  $w'_j$  while  $P_j$  receives  $r_j$  and  $w_j$ . By definition, we have that if  $x \in X_j$ ,  $r_j + r'_j = 0$  and  $w_j + w'_j = v_j$ , where  $v_j$  is the associated payload with  $x$  in  $V_j$ ; otherwise  $r_j + r'_j$  and  $w_j + w'_j$  are both random values.  $P_{\text{pivot}}$  sets  $r_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} t_j$  as its first secret share and sets  $w_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} u_j$  as its second secret share. Meanwhile,  $P_j$  sets  $r_j$  as its first secret share and  $w_j$  as its second secret share. Note that if and only if  $x \in X_j$  for all  $j$ ,  $\sum_{1 \leq i \leq m} r_i = 0$  and  $\sum_{1 \leq i \leq m} w_i = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} v_j$ , otherwise  $\sum_{1 \leq i \leq m} r_i$  and  $\sum_{1 \leq i \leq m} w_i$  are random values. At this point, the first secret-sharing is a relaxed pure membership zero-sharing while the second secret-sharing is relaxed pure membership payload-sharing. In order to realize the membership zero-sharing with payloads functionality, the last step is to transform the first relaxed pure membership zero-sharing into the standard. The complete batch version is provided in Figure 21.

**Complexity Analysis.** In the batch pure membership zero-sharing with payloads protocol (Figure 21), the costs of each stage are calculated as follows.

- $P_{\text{pivot}}$  executes batch OPPRF of size  $n$  with each  $P_j$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ . In this stage, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .
- The parties perform  $n$  secure multiplications using the optimization outlined in the previous section and designate  $P_{\text{pivot}}$  as the leader. This requires  $n$  opening with  $O(mn)$  computation/communication complexity for  $P_{\text{pivot}}$  and  $O(n)$  computation/communication complexity for each  $P_j$ .

To sum up, in the online phase of the batch pure membership zero-sharing with payloads protocol, the computation and communication complexity of  $P_{\text{pivot}}$  are  $O(mn)$ , while the computation and communication complexity of each  $P_j$  are  $O(n)$ .

## G Security Proof of Theorem 6

Let  $\mathbf{P}_{\mathcal{A}}$  denote the set of corrupted parties controlled by  $\mathcal{A}$ . In the MPSO protocol, the simulator receives each corrupted party's input  $X_c$  from  $P_c \in \mathbf{P}_{\mathcal{A}}$  and if  $P_1 \in \mathbf{P}_{\mathcal{A}}$ , it receives the resulting set  $Y$ . For each  $P_c$ , its view consists of its input  $X_c$ ,  $B$  secret shares  $s_{i,c}$  from each  $\mathcal{F}_{\text{bMZS}}^{Q'_i}$  for  $1 \leq i \leq s$  (if  $P_c$  belongs to

**Parameters:**  $m$  parties  $P_1, \dots, P_m$ . Batch size  $n$ . A field  $\mathbb{F}$  and payload field  $\mathbb{F}'$ . The mapping function  $\text{payload}_j()$  from element sets  $\mathbf{X}_j$  to the associated payload sets  $\mathbf{V}_j$ .  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$  generated in offline phase, where  $[\mathbf{a}] = ([a_1], \dots, [a_n])$ ,  $[\mathbf{b}] = ([b_1], \dots, [b_n])$ ,  $[\mathbf{c}] = ([c_1], \dots, [c_n])$  and  $c_i = a_i \cdot b_i$  for  $1 \leq i \leq n$ .

**Inputs:**  $P_{\text{pivot}}$  inputs a vector  $\mathbf{x} = (x_1, \dots, x_n)$ .  $P_j$  inputs  $\mathbf{X}_j = (X_{j,1}, \dots, X_{j,n})$  and  $\mathbf{V}_j = (V_{j,1}, \dots, V_{j,n})$  for  $j \in \{1, \dots, m\} \setminus \{\text{pivot}\}$ .

**Protocol:**

1. For the  $i$ -th instance ( $1 \leq i \leq n$ ),  $P_j$  samples  $(r_{j,i}, w_{j,i})$ . Suppose  $|X_{j,i}| = N_{j,i}$ ,  $P_j$  sets  $K_{j,i} = X_{j,i} = (x_{j,i,1}, \dots, x_{j,i,N_{j,i}})$  and  $V'_{j,i} = \{(-r_{j,i}, v_{j,i,1} - w_{j,i}), \dots, (-r_{j,i}, v_{j,i,N_{j,i}} - w_{j,i})\}$ , where  $|V'_{j,i}| = N_{j,i}$  and  $v_{j,i,k} = \text{payload}_j(x_{j,i,k})$  for  $1 \leq k \leq N_{j,i}$ .
2.  $P_{\text{pivot}}$  and  $P_j$  invoke  $\mathcal{F}_{\text{bOPPRF}}$  where  $P_j$  acts as  $\mathcal{S}$  inputting  $(K_{j,1}, \dots, K_{j,n})$  and  $(V'_{j,1}, \dots, V'_{j,n})$ , and  $P_{\text{pivot}}$  acts as  $\mathcal{R}$  with input  $\mathbf{x}$  and receives  $(\mathbf{r}'_j, \mathbf{w}'_j)$ .
3.  $P_{\text{pivot}}$  sets its first shares  $\mathbf{r}_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} \mathbf{r}'_j$ , and its second shares  $\mathbf{w}_{\text{pivot}} = \sum_{j \in \{1, \dots, m\} \setminus \{\text{pivot}\}} \mathbf{w}'_j$ .  $P_j$  sets its first shares  $\mathbf{r}_j = (r_{j,1}, \dots, r_{j,n})$ , and its second shares  $\mathbf{w}_j = (w_{j,1}, \dots, w_{j,n})$ . All parties hold two vectors of  $n$  secret-sharings  $[\mathbf{r}] = ([r_1], \dots, [r_n])$  and  $[\mathbf{w}] = ([w_1], \dots, [w_n])$ .
4. All parties compute  $[\mathbf{s}]$  by performing  $n$  secure multiplications  $[s_i] = [r_i] \cdot [b_i]$  ( $1 \leq i \leq n$ ), using  $n$  Beaver triples  $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ .

**Fig. 21.** Batch Pure Membership Zero-Sharing with Payloads  $\Pi_{\text{bpMZSp}}$

the set of  $Q_i$ 's involving parties  $\{P_{i_1}, \dots, P_{i_q}\}$ , shuffled secret shares  $\mathbf{u}'_c$  from  $\mathcal{F}_{\text{shuffle}}$ , and if  $P_1 \in \mathbf{P}_{\mathcal{A}}$ , reconstruction messages  $\mathbf{u}'_j$  from  $P_j$  for  $1 < j \leq m$ .

Suppose there are  $s'$  subformulas  $\mathbf{Q}_{\mathcal{A}} = \{Q_{j_1}, \dots, Q_{j_{s'}}\} \subseteq \{Q_1, \dots, Q_s\}$  without involving honest parties, and  $\mathbf{Q}_{\mathcal{H}} = \{Q_1, \dots, Q_s\} \setminus \mathbf{Q}_{\mathcal{A}}$ , containing the subformulas that involve at least one honest party. The simulator emulates each  $P_c$ 's view by running the protocol honestly with these changes:

- It simulates uniform secret shares from each  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$  for each  $Q_h \in \mathbf{Q}_{\mathcal{H}}$ .
- **Case**  $P_1 \notin \mathbf{P}_{\mathcal{A}}$ . It samples uniform secret shares  $\mathbf{u}'_c$  from  $\mathcal{F}_{\text{shuffle}}$ .
- **Case**  $P_1 \in \mathbf{P}_{\mathcal{A}}$ . After the corrupted parties honestly invoke batch membership zero-sharing protocols for all subformulas in  $\mathbf{Q}_{\mathcal{A}}$ , the parties hold  $s'B$  secret-sharings, where we denote all secrets of elements (appended with all-zero strings) as a set  $Y_{\mathcal{A}} \in Y$  and  $s'B - |Y_{\mathcal{A}}|$  random secrets as a set  $R_{\mathcal{A}}$ . Let  $Y_{\mathcal{H}} = Y \setminus Y_{\mathcal{A}}$ . The simulator samples  $(s - s')B - |Y_{\mathcal{H}}|$  random values as a set  $R_{\mathcal{H}}$ , shuffles the union  $Y \cup R_{\mathcal{H}} \cup R_{\mathcal{A}}$  with a random permutation  $\pi$  and secret-shares the shuffled union as  $\mathbf{u}'_1, \dots, \mathbf{u}'_m$ , where  $\mathbf{u}'_c$  is outputted to  $P_c$  as secret shares from  $\mathcal{F}_{\text{shuffle}}$  for each  $P_c \in \{P_{i_1}, \dots, P_{i_q}\}$ .

In the case  $P_1 \notin \mathbf{P}_{\mathcal{A}}$ , it is easy to see that  $P_c$ 's secret shares from each  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$  and  $\mathcal{F}_{\text{shuffle}}$  are uniformly distributed and independent of any other distributions in the real execution (as there exists at least an honest party holding one share), which is identical to the simulation.

In the case  $P_1 \in \mathbf{P}_{\mathcal{A}}$ ,  $P_c$ 's secret shares from each  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$  ( $Q_h \in \mathbf{Q}_{\mathcal{H}}$ ) are also uniformly distributed and independent of any other distributions in the real execution, so

$$\begin{aligned} & \{\text{Sim}^{Q_h}(\mathbf{P}_{\mathcal{A}}^h, \mathbf{X}_{\mathcal{A}}^h, \{\mathbf{s}_{h,c}\}_{P_c \in \mathbf{P}_{\mathcal{A}}^h})_{Q_h \in \mathbf{Q}_{\mathcal{H}}}, \{\mathbf{s}_{h,c}\}_{Q_h \in \mathbf{Q}_{\mathcal{H}}, P_c \in \mathbf{P}_{\mathcal{A}}^h}\} \mathbf{X} \\ & \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{Q_h}(\mathbf{X}^h)_{Q_h \in \mathbf{Q}_{\mathcal{H}}}, \{\mathbf{s}_{h,c}^{\text{II}}\}_{Q_h \in \mathbf{Q}_{\mathcal{H}}, P_c \in \mathbf{P}_{\mathcal{A}}^h}\} \mathbf{X}, \end{aligned}$$

where  $\mathbf{X} = \{X_1, \dots, X_m\}$ .  $\mathbf{P}_{\mathcal{A}}^h$  denotes the corrupted parties involving in  $Q_h$ , while  $\mathbf{X}_{\mathcal{A}}^h$  denotes the set of their inputs sets.  $\mathbf{X}^h$  denotes the set of all involved parties' inputs sets in  $Q_h$ .  $\text{Sim}^{Q_h}$  is the view emulated by the simulator of  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$ , while  $\text{View}_{\mathcal{A}}^{Q_h}$  is the real view of adversary in the batch membership zero-sharing protocol for  $Q_h$ . The distinctions with a superscript  $\text{II}$  are in the real execution, otherwise in simulation. As the corrupted parties honestly invoke batch membership zero-sharing protocols for all subformulas in  $\mathbf{Q}_{\mathcal{A}}$ , we obtain

$$\begin{aligned} & \{\text{Sim}^{Q_i}(\mathbf{P}_{\mathcal{A}}^i, \mathbf{X}_{\mathcal{A}}^i, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}_{\mathcal{A}}^i})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}^i}\} \mathbf{X} \\ & \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{Q_i}(X_{i_1}, \dots, X_{i_q})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}^{\text{II}}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}^i}\} \mathbf{X}. \end{aligned}$$

By correctness, after invoking all  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$  for each  $Q_h \in \mathbf{Q}_{\mathcal{H}}$ , the parties hold  $|Y_{\mathcal{H}}|$  secret-sharings of the elements in  $Y_{\mathcal{H}}$ , and  $(s - s')B - |Y_{\mathcal{H}}|$  secret-sharings of random values (the set of these random secrets is denoted by  $R_{\mathcal{H}}^{\text{II}}$ ). By the independence requirement of  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$ , all random values in  $R_{\mathcal{H}}^{\text{II}}$  are independent of the joint view of any  $m-1$  parties, i.e. the view of adversary, in the real execution

of batch membership zero-sharing protocols. In simulation, the random values in  $R_{\mathcal{H}}$  are sampled using independent randomness so they are also independent of the emulated view for  $\mathcal{F}_{\text{bMZS}}^{Q'_h}$ . After the execution of multi-party secret-shared shuffle, the order of elements in  $Y \cup R_{\mathcal{H}}^{\Pi} \cup R_{\mathcal{A}}^{\Pi}$  is shuffled. By the functionality of  $\mathcal{F}_{\text{shuffle}}$ , the random permutation  $\pi^{\Pi}$  is sampled independently. Thereby,

$$\begin{aligned} & \{\text{Sim}^{Q_i}(\mathbf{P}_{\mathcal{A}}^i, \mathbf{X}_{\mathcal{A}}^i, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}_{\mathcal{A}}^i})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \pi(Y \cup R_{\mathcal{H}} \cup R_{\mathcal{A}})\}_{\mathbf{X}} \\ & \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{Q_i}(X_{i_1}, \dots, X_{i_q})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}^{\Pi}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \pi^{\Pi}(Y \cup R_{\mathcal{H}}^{\Pi} \cup R_{\mathcal{A}}^{\Pi})\}_{\mathbf{X}}. \end{aligned}$$

Given that  $\mathbf{u}'_1, \dots, \mathbf{u}'_m$  and  $\mathbf{u}_1^{\Pi}, \dots, \mathbf{u}_m^{\Pi}$  are secret shares of  $\pi(Y \cup R_{\mathcal{H}} \cup R_{\mathcal{A}}^{\Pi})$  and  $\pi^{\Pi}(Y \cup R_{\mathcal{H}}^{\Pi} \cup R_{\mathcal{A}}^{\Pi})$  respectively, we derive that

$$\begin{aligned} & \{\text{Sim}^{Q_i}(\mathbf{P}_{\mathcal{A}}^i, \mathbf{X}_{\mathcal{A}}^i, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}_{\mathcal{A}}^i})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \mathbf{u}'_1, \dots, \mathbf{u}'_m\}_{\mathbf{X}} \\ & \stackrel{c}{\approx} \{\text{View}_{\mathcal{A}}^{Q_i}(X_{i_1}, \dots, X_{i_q})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}^{\Pi}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \mathbf{u}_1^{\Pi}, \dots, \mathbf{u}_m^{\Pi}\}_{\mathbf{X}}. \end{aligned}$$

By invoking the simulator for multi-party secret-shared shuffle  $\text{Sim}^{sh}$ ,

$$\begin{aligned} & \{\text{Sim}^{Q_i}(\mathbf{P}_{\mathcal{A}}^i, \mathbf{X}_{\mathcal{A}}^i, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}_{\mathcal{A}}^i})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \\ & \quad \text{Sim}^{sh}(\mathbf{P}_{\mathcal{A}}, \{\mathbf{u}_c, \mathbf{u}'_c\}_{P_c \in \mathbf{P}_{\mathcal{A}}}, \mathbf{u}'_1, \dots, \mathbf{u}'_m)\}_{\mathbf{X}} \\ & \stackrel{c}{\approx} \\ & \{\text{View}_{\mathcal{A}}^{Q_i}(X_{i_1}, \dots, X_{i_q})_{1 \leq i \leq s}, \{\mathbf{s}_{i,c}^{\Pi}\}_{1 \leq i \leq s, P_c \in \mathbf{P}_{\mathcal{A}}}, \\ & \quad \text{View}_{\mathcal{A}}^{sh}(\mathbf{u}_1^{\Pi}, \dots, \mathbf{u}_m^{\Pi}), \mathbf{u}_1^{\Pi}, \dots, \mathbf{u}_m^{\Pi}\}_{\mathbf{X}}, \end{aligned}$$

where  $\mathbf{u}_k$  is computed by  $\{\mathbf{s}_{i,k}\}_{1 \leq i \leq s}$ . We conclude that the adversary's view in real execution is indistinguishable to its view in the simulation.

The security proof for the circuit-MPSO (Approach 2) protocol is the same. The security proof for the MPSO-card and circuit-MPSO (Approach 1) protocols are similar, except that the simulator replaces all elements in the simulation with 0s, since it only obtains the cardinality rather than the set itself if  $P_1 \in \mathbf{P}_{\mathcal{A}}$ .

## H Implementation Details and Parameter Settings

### H.1 Implementation Details

Our protocols are written in C++, where each party uses  $m - 1$  threads to interact simultaneously with all other parties. We instantiate batch OPPRF with VOLE and OKVS [43, 26, 48, 9], following [13]; We instantiate batch ssPMT with batch OPPRF and ssPEQT, following [19]. We use the following libraries in our implementation.

- VOLE: We use VOLE implemented in libOTe [50], instantiating the code family with Expand-Convolute codes [49].

- OKVS and GMW: We use the optimized OKVS construction in [48]<sup>9</sup> and re-use the OKVS implementation in [5]. We also re-use the GMW implementation in [5] to construct ssPEQT.
- ROT: We use SoftSpokenOT [52] implemented in libOTe.
- Additionally, we use the `cryptoTools` [4] library to compute hash functions and PRNG calls, and we adopt Coproto [3] to realize network communication.

## H.2 Choosing Suitable Parameters

We set the computational security parameter  $\lambda = 128$  and the statistical security parameter  $\sigma = 40$ . The other parameters are:

**Cuckoo hashing parameters.** To achieve linear communication of batch ssPMT, we use stash-less Cuckoo hashing [45]. 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  $B = 1.27n$  for 3-hash Cuckoo hashing.

**OKVS parameters.** We employ  $w = 3$  scheme with a cluster size of  $2^{14}$  in [48], and the expansion rate (which is the size of OKVS divided by the number of encoding items) in this setting is 1.28.

**ROT parameters.** We set field bits to 5 in SoftSpokenOT to balance computation and communication costs.

**Length of OPPRF outputs.** According to [19], to ensure the correctness of batch ssPMT, the output length of OPPRF in batch ssPMT is at least  $\sigma + \log_2 T + \log_2 B$ , where  $T$  is the total number of the batch ssPMT invocations, which is  $(m^2 - m)/2$  in our MPSU protocol. Thereby, the lower bound of output length of OPPRF in our MPSU protocol is  $\sigma + \log_2((m^2 - m)/2) + \log_2(1.27n)$ .

**Field size and all-zero string length.** The field size and all-zero string control the probability of a spurious collision in our protocols. According to the correctness analysis in Section 6.4, for MPSI, MPSI-card and MPSI-card-sum protocols, field size of  $B \cdot 2^\sigma = 1.27n \cdot 2^\sigma$  is sufficient to bound the probability of any spurious collision to  $2^{-\sigma}$ . For MPSU protocol, the field size should meet two requirements:  $|\mathbb{F}| \geq B \cdot 2^\sigma$  and the length of elements in  $\mathbb{F}$  equals  $l + l'$ . Given that the all-zero string length  $l' \geq \sigma + \log(m - 1) + \log B$ , we have  $|\mathbb{F}| \geq 2^l + (m - 1)B \cdot 2^\sigma$  in our MPSU. Concretely, we use GF(64) for our MPSI, MPSI-card and MPSI-card-sum protocols, and GF(128) for our MPSU protocol (where  $l'$  is set as 64 bits) in our experiments.

<sup>9</sup> Since the existence of suitable parameters for the new OKVS construction of the recent work [9] is unclear when the set size is less than  $2^{10}$ , we choose to use the OKVS construction of [48].